Robuuste handdetectie en tracering in ware tijd voor monoculaire video

Robust and Real-Time Hand Detection and Tracking in Monocular Video

Vincent Spruyt

Promotoren: prof. dr. ir. W. Philips, dr. A. Ledda
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen

Vakgroep Telecommunicatie en Informatieverwerking
Voorzitter: prof. dr. ir. H. Bruneel
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014 - 2015

UNIVERSITEIT
GENT

ISBN 978-90-8578-768-6
NUR 993, 984
Wettelijk depot: D/2015/10.500/12

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Telecommunicatie en Informatieverwerking

Supervisors: Prof. Dr. ir. Wilfried Philips
Dr. Alessandro Ledda

Members of the jury:

Prof. dr. ir. Rik Van de Walle (Ghent University, chairman)
Prof. dr. ir. Wilfried Philips (Ghent University, supervisor)
Dr. Alessandro Ledda (SoftKinetic, supervisor)
Dr. ir. Hiep Luong (Ghent University)
Prof. dr. ir. Peter Veelaert (Ghent University)
Prof. dr. Jelle Saldien (Ghent University)
Prof. dr. ing. Maarten Weyn (University of Antwerp)
Prof. dr. ir. Adrian Munteanu (Vrije Universiteit Brussel)
Prof. dr. ir. Pieter Jonker (Delft University of Technology)

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Telecommunicatie en Informatieverwerking
St-Pietersnieuwstraat 41, B-9000 Gent, België

Tel.: +32-9-264.34.12
Fax.: +32-9-264.42.95

Proefschrift tot het bekomen van de graad van
Doctor in de Ingenieurswetenschappen
Academiejaar 2014–2015

Affiliations

Ghent University
Faculty of Engineering and Architecture
Department of Telecommunications and Information Processing
Research Group of Image processing and Interpretation (IPI)
iMinds

Sint-Pietersnieuwstraat 41
9000 Ghent
Belgium

University of Antwerp Faculty of Applied Engineering
Department of Electronics and ICT
Research Group of Constrained Systems (CoSys-lab)

Paardenmarkt 92
2000 Antwerp
Belgium

Dit werk kwam tot stand in het kader van een strategische onderzoeksbeurs van het IWT-Vlaanderen (Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen).

Acknowledgements

Without the extensive support, guidance and assistance of colleagues and friends, this manuscript would simply not exist. I would therefore like to express my generous gratitude to the following people:

First and foremost, I would like to thank my supervisors, prof. dr. ir. Wilfried Philips, and dr. Alessandro Ledda, for the opportunity to conduct research in the exciting field of video analysis and machine learning. They continuously provided me with much needed guidance and support, perfectly balanced with an equally needed freedom to pursue my own interests and research goals.

Secondly I would like to thank the members of my jury for being in my thesis committee, and for reading and commenting on this thesis.

I would also like to thank my colleagues at both IPI, Ghent University and CoSys-lab, University of Antwerp, for the hours of intriguing discussions on technical matters, and for the very welcomed breaks and moments of relaxation.

Finally, although explicitly mentioning names always results in the risk of forgetting some of those who matter most, I would nevertheless like to add three names to the above list:
Prof. dr. Maarten Weyn (CoSys-lab, University of Antwerp) for the many technical discussions and more importantly for the personal support and countless motivational talks. Stig Geerts, friend and former colleague, for the myriad of brainstorm sessions and discussions where science and philosophy met. Isabelle Piette, love of my life and source of endless support, understanding, and patience, for everything.

Thank you all!

August 2014, Ghent
Vincent Spruyt

Table of Contents

List of Figures

List of Tables

List of Acronyms

BHMC	Basin Hopping Monte Carlo
CPU	Central Processing Unit
CS-LBP	Center-Symmetric Local Binary Pattern
DOF	Degrees of Freedom
DoH	Determinant of Hessian
ESS	Effective Sample Size
FAST	Features from Accelerated Segment Test
FPPW	False Positives Per Window
FREAK	Fast Retina Keypoint
GLSL	OpenGL Shading Language
GPU	Graphics Processing Unit
HCI	Human-computer Interaction
HOG	Histogram of Oriented Gradients
HSV	Hue Saturation Value
ISM	Implicit Shape Model
KLT	Kanade-Lucas-Tomashi
LBP	Local Binary Pattern
LCD	Liquid Crystal Display
LDA	Linear Discriminant Analysis
LUT	Lookup Table
MCT	Modified Census Transform
MS	Mean-Shift
MSER	Maximally Stable Extremal Region
MST	Minimum Spanning Tree
OF	Optical Flow
PCA	Principle Component Analysis
PDF	Probability Density Function
QVGA	Quarter VGA
QQVGA	Quarter QVGA
RGB	Red Green Blue
SIFT	Scale Invariant Feature Transform
SIMD	Single Instruction Multiple Data
SIR	Sequential Importance Resampling
SIS	Sequential Importance Sampling
SSE	Streaming SIMD Extensions
SURF	Speeded-up Robust Features

SWAR	SIMD With a Register
TP	Transition Prior
TV	Total Variation
VOC	Visual Object Classes
VGA	Video Graphics Array

Nederlandse samenvatting

Laptops, tablets en smartphones zijn doorheen de jaren alomtegenwoordig geworden. Intelligente sensoren worden zelfs geïntegreerd in verschillende soorten gebruikersapparaten zoals brillen, horloges en televisies. Met de uitvinding van het aanraakscherm ontstond een nieuw mens-computer interactieparadigma (HCI) dat gebruikers toelaat om op een intuïtieve manier te interageren met een apparaat. Met behulp van eenvoudige gebaren, zoals een 'swipe' of 'pinch' gebaar, kunnen aanraakschermen gebruikt worden om op een directe manier een virtuele omgeving aan te sturen. Desalniettemin vormen aanraakschermen nog steeds een fysieke barrière tussen de virtuele wereld en de reële wereld.

Een onderzoeksdomein dat de afgelopen jaren enorm populair werd en dat probeert om dit probleem op te lossen, is videogebaseerde gebarenherkenning, handdetectie en handtracering. Handgestuurde interactie laat de gebruiker toe om direct te interageren met de computer op een natuurlijke wijze door een virtuele realiteit te verkennen met behulp van de eigen lichaamstaal.

In dit proefschrift onderzoeken we hoe robuuste handdetectie en -tracering bereikt kan worden, rekening houdend met de vereiste dat dit moet gebeuren in ware tijd. In de context van mens-computer interactie wordt 'ware tijd' gedefinieerd door zowel een kleine vertraging tussen invoer en uitvoer, en een lage complexiteit. Hierdoor kan een volledig videobeeld verwerkt worden alvorens het volgende beeld beschikbaar is. Verder moeten de algoritmen ook robuust zijn tegen belichtingsveranderingen, camerabeweging, en veranderende achtergrond. Ten slotte moet het systeem in staat zijn om automatisch te initialiseren en om traceringsfouten te detecteren en te herstellen. We bestuderen daarom een brede waaier aan bestaande algoritmen, en stellen significante verbeteringen en nieuwe methoden voor om een volledig detectie- en trackingsysteem te bouwen dat voldoet aan al deze vereisten.

Handdetectie, handtracering en handsegmentatie zijn conceptueel gerelateerde maar technisch erg verschillende uitdagingen. Terwijl objectdetectors proberen om objecten te herkennen en te localiseren in een statische afbeelding, proberen traceringsalgoritmen temporele informatie te interpreteren om de positie van het object te volgen doorheen de tijd in een videosequentie. Handsegmentatie gaat over het schatten van de contour van de hand, om zo de hand te onderscheiden van zijn achtergrond.

Detectie van handen in individuele videobeelden laat ons toe om vervolgens een traceringsalgoritme automatisch te initialiseren, en om traceringsfouten te detecteren en te herstellen. Menselijke handen zijn sterk gearticuleerde objecten,

bestaande uit vingerkootjes die verbonden worden door gewrichten. Hierdoor kunnen de uiterlijke kenmerken van een hand sterk variëren, afhankelijk van de aangenomen handpose. Traditionele detectiealgoritmen veronderstellen vaak dat het uiterlijk van het te detecteren object beschreven kan worden met een stug model, waarbij de relatieve positie van individuele objectonderdelen wordt verondersteld niet te veranderen. Deze algoritmen kunnen daarom niet gebruikt worden om op een robuuste manier menselijke handen te detecteren in afbeeldingen.

Om die reden ontwikkelden we een algoritme dat handen detecteert door expliciet gebruik te maken van het feit dat handen gearticuleerde objecten zijn. In de plaats van een vaak gebruikte sjabloongebaseerde aanpak toe te passen, modelleert onze oplossing de spatiale relaties tussen de verschillende handonderdelen en het centrum van de hand op een probabilistische wijze. Het detecteren van handonderdelen, zoals vingertoppen, is veel eenvoudiger dan het onmiddellijk detecteren van een volledige hand. Op basis van het spatiale model van de relatieve positie van deze onderdelen, kan elk onderdeel gebruikt worden om een schatting te bekomen van de handpositie. Om ook te voldoen aan de vereisten omtrent verwerking in ware tijd, ontwikkelden we tevens technieken om het detectieproces te versnellen door op een efficiënte manier onbelangrijke achtergrondinformatie uit het beeld te verwijderen. Experimentele resultaten tonen aan dat onze methode competitief is met de meest recente technieken in de literatuur, terwijl de computationele complexiteit verminderd wordt met een factor 1 000. Verder tonen we aan dat onze oplossing ook geschikt is voor detectie van andere objecten zoals mensen en dieren, en dus niet beperkt is tot handdetectie.

Wanneer een hand gedetecteerd werd, kan een traceringsalgoritme gebruikt worden om de hand te volgen doorheen de tijd. We ontwikkelden daarom een probabilistisch traceringsalgoritme dat kan omgaan met onzekerheid veroorzaakt door afbeeldingsruis, foute detecties, veranderende belichting, en camerabeweging. Verder is onze methode in staat om het aantal te volgen handen automatisch te bepalen, en kan ze omgaan met handen die het videobeeld binnenkomen of verlaten. We introduceren hiervoor verschillende nieuwe technieken die de robuustheid van het algoritme sterk verhogen en die tevens gebruikt kunnen worden voor het traceren van andere objecten dan handen. Om te voldoen aan de vereisten omtrent verwerking in ware tijd, onderzochten we verschillende manieren om de zoekruimte van het probleem te verkleinen, en kozen we bewust voor methoden die eenvoudig paralleliseerbaar zijn door moderne computerhardware. Experimentele resultaten geven aan dat onze methoden beter presteren dan de nieuwste technieken die beschreven zijn in de academische literatuur, terwijl een veel kleinere computationele complexiteit wordt bereikt.

Een van de technieken die door ons probabilistisch traceringsalgoritme wordt gebruikt, is optische-stroomschatting. Optische stroom is gedefinieerd als een 2D vectorveld dat de schijnbare snelheden van objecten uit een 3D scene beschrijft, geprojecteerd op het beeldvlak. Het is geweten dat dieren zoals insecten en verschillende soorten vogels gebruik maken van optische stroom om objecten te volgen en om hun eigen positie te schatten. De meeste in de literatuur beschreven methoden om optische stroom te schatten zijn echter te traag om te voldoen aan

de vereisten omtrent verwerking in ware tijd, of zijn niet robuust bij veranderende belichting of snelle bewegingen. Om die reden ontwikkelden we een algoritme voor het schatten van optische stroom dat geen last heeft van deze problemen. We introduceren ook een manier om het zoekproces te regulariseren zodat steeds een glad vectorveld wordt bekomen. Deze regularisatiemethode reduceert het aantal foute of slechte vectorschattingen, terwijl ons algoritme toch kan omgaan met bewegingsdiscontinuïteiten die veroorzaakt worden door objectgrenzen in de scene.

De bovenstaande methoden worden gecombineerd in een robuust handtraceringraamwerk dat kan werken in ware tijd en dat gebruikt kan worden voor interactieve applicaties in ongecontroleerde omgevingen. Om de mogelijkheden van handgestuurde mens-computer interactie aan te tonen, ontwikkelden we een nieuw soort computerbeeldscherm. Dit beeldscherm is volledig transparant, waardoor meerdere gebruikers het tegelijkertijd kunnen gebruiken om samen taken te vervullen, zonder oogcontact te verliezen. Het beeldscherm produceert een beeld dat lijkt te zweven in de lucht, zodat gebruikers het kunnen aanraken met de handen. Dit scherm werd gedemonstreerd op verschillende nationale en internationale evenementen en beurzen.

Het onderzoek dat wordt beschreven in deze verhandeling werd grondig geëvalueerd door de detectie- en traceringsresultaten te vergelijken met de resultaten die bekomen worden door state-of-the-art algoritmes. Deze vergelijkingen tonen aan dat het voorgestelde werk een niet eerder bereikte accuraatheid combineert met een lage computationele complexiteit waardoor verwerking in ware tijd mogelijk is. De resultaten worden telkens aan het einde van elk hoofdstuk diepgaand besproken. Het voorgestelde onderzoek resulteerde overigens in een internationale tijdschrift-publicatie; een tweede ingediende tijdschrift-paper die op het moment van schrijven onder review is; negen internationale conferentie publicaties; een nationale conferentie publicatie; een commerciële licentie overeenkomst betreffende de onderzoeksresultaten; twee hardware prototypes van een nieuw type beeldscherm; en een software demonstrator.

English summary

In recent years, personal computing devices such as laptops, tablets and smartphones have become ubiquitous. Moreover, intelligent sensors are being integrated into many consumer devices such as eyeglasses, wristwatches and smart televisions. With the advent of touchscreen technology, a new human-computer interaction (HCI) paradigm arose that allows users to interface with their device in an intuitive manner. Using simple gestures, such as swipe or pinch movements, a touchscreen can be used to directly interact with a virtual environment. Nevertheless, touchscreens still form a physical barrier between the virtual interface and the real world.

An increasingly popular field of research that tries to overcome this limitation, is video based gesture recognition, hand detection and hand tracking. Gesture based interaction allows the user to directly interact with the computer in a natural manner by exploring a virtual reality using nothing but his own body language.

In this dissertation, we investigate how robust hand detection and tracking can be accomplished under real-time constraints. In the context of human-computer interaction, real-time is defined as both low latency and low complexity, such that a complete video frame can be processed before the next one becomes available. Furthermore, for practical applications, the algorithms should be robust to illumination changes, camera motion, and cluttered backgrounds in the scene. Finally, the system should be able to initialize automatically, and to detect and recover from tracking failure. We study a wide variety of existing algorithms, and propose significant improvements and novel methods to build a complete detection and tracking system that meets these requirements.

Hand detection, hand tracking and hand segmentation are related yet technically different challenges. Whereas detection deals with finding an object in a static image, tracking considers temporal information and is used to track the position of an object over time, throughout a video sequence. Hand segmentation is the task of estimating the hand contour, thereby separating the object from its background.

Detection of hands in individual video frames allows us to automatically initialize our tracking algorithm, and to detect and recover from tracking failure. Human hands are highly articulated objects, consisting of finger parts that are connected with joints. As a result, the appearance of a hand can vary greatly, depending on the assumed hand pose. Traditional detection algorithms often assume that the appearance of the object of interest can be described using a rigid model and therefore can not be used to robustly detect human hands. Therefore,

we developed an algorithm that detects hands by exploiting their articulated nature. Instead of resorting to a template based approach, we probabilistically model the spatial relations between different hand parts, and the centroid of the hand. Detecting hand parts, such as fingertips, is much easier than detecting a complete hand. Based on our model of the spatial configuration of hand parts, the detected parts can be used to obtain an estimate of the complete hand's position. To comply with the real-time constraints, we developed techniques to speed-up the process by efficiently discarding unimportant information in the image. Experimental results show that our method is competitive with the state-of-the-art in object detection while providing a reduction in computational complexity with a factor 1 000. Furthermore, we showed that our algorithm can also be used to detect other articulated objects such as persons or animals and is therefore not restricted to the task of hand detection.

Once a hand has been detected, a tracking algorithm can be used to continuously track its position in time. We developed a probabilistic tracking method that can cope with uncertainty caused by image noise, incorrect detections, changing illumination, and camera motion. Furthermore, our tracking system automatically determines the number of hands in the scene, and can cope with hands entering or leaving the video canvas. We introduced several novel techniques that greatly increase tracking robustness, and that can also be applied in other domains than hand tracking. To achieve real-time processing, we investigated several techniques to reduce the search space of the problem, and deliberately employ methods that are easily parallelized on modern hardware. Experimental results indicate that our methods outperform the state-of-the-art in hand tracking, while providing a much lower computational complexity.

One of the methods used by our probabilistic tracking algorithm, is optical flow estimation. Optical flow is defined as a 2D vector field describing the apparent velocities of objects in a 3D scene, projected onto the image plane. Optical flow is known to be used by many insects and birds to visually track objects and to estimate their ego-motion. However, most optical flow estimation methods described in literature are either too slow to be used in real-time applications, or are not robust to illumination changes and fast motion. We therefore developed an optical flow algorithm that can cope with large displacements, and that is illumination independent. Furthermore, we introduce a regularization technique that ensures a smooth flow-field. This regularization scheme effectively reduces the number of noisy and incorrect flow-vector estimates, while maintaining the ability to handle motion discontinuities caused by object boundaries in the scene.

The above methods are combined into a hand tracking framework which can be used for interactive applications in unconstrained environments. To demonstrate the possibilities of gesture based human-computer interaction, we developed a new type of computer display. This display is completely transparent, allowing multiple users to perform collaborative tasks while maintaining eye contact. Furthermore, our display produces an image that seems to float in thin air, such that users can touch the virtual image with their hands. This floating imaging display has been showcased on several national and international events and tradeshows.

The research that is described in this dissertation has been evaluated thoroughly by comparing detection and tracking results with those obtained by state-of-the-art algorithms. These comparisons show that the proposed methods outperform most algorithms in terms of accuracy, while achieving a much lower computational complexity, resulting in a real-time implementation. Results are discussed in depth at the end of each chapter. This research further resulted in an international journal publication; a second journal paper that has been submitted and is under review at the time of writing this dissertation; nine international conference publications; a national conference publication; a commercial license agreement concerning the research results; two hardware prototypes of a new type of computer display; and a software demonstrator.

1 Introduction

This chapter serves as an introduction and as a summary of the research and contributions outlined in this dissertation. In Section 1 we discuss the research questions that will be answered in the following chapters. Section 2 presents several possible applications of our work and thereby justifies its significance. Section 3 contains an overview of the major contributions of the work that will be discussed in detail in the next chapters. In Section 4, we summarize the scientific output that has been generated during this PhD research, in the form of publications. Finally, Section 5 briefly presents the outline of this dissertation and can be used as a guideline while exploring the next chapters.

1 Research questions

Human-computer interaction traditionally occurs through the use of peripheral hardware such as a keyboard, computer mouse or joystick. However, these mechanical devices not only behave as a physical barrier between the human and the computer, but also act as a psychological barrier, in the sense that they limit the effectiveness and naturalness of the interaction.

In recent years, personal computing devices such as laptops, tablets and smartphones have become ubiquitous. Moreover, intelligent sensors are being integrated into many consumer devices such as eye glasses, wrist watches and smart televisions. With the advent of touchscreen technology, a new human-computer interaction (HCI) paradigm arose that allows users to interface with their device in an

intuitive manner. Using simple gestures, such as swipe or pinch movements, a touchscreen can be used to directly interact with a virtual environment. Nevertheless, touchscreens still form a physical barrier between the virtual interface and the real world.

An increasingly popular field of research that tries to overcome this limitation, is video based gesture recognition, hand detection and hand tracking. Gesture based interaction allows the user to directly interact with the computer in a natural manner by exploring a virtual reality using nothing but his own body language. Recent studies have shown that using a natural gestures based human computer interface significantly reduces the cognitive workload of the user involved [Cornelius 13], and allows the user to focus on the task at hand instead of focussing on the interfacing method itself.

Reviewing the early academic literature on this topic reveals an increasing interest in natural human-computer interaction, starting from the mid sixties. However, only recently robust vision based hand tracking became feasible through advances in processing speed and camera hardware. With the recent advent of inexpensive time-of-flight and structured light based depth cameras, real-time gesture based human-computer interaction moved from the academic world to the industrial scene, with practical applications being available today.

However, current hand tracking and detection methods impose specific constraints on the user, in order to overcome ambiguities in the environment. For instance, the well known Microsoft Kinect sensor [Microsoft 10] requires the user's face to be visible at all times, and can only detect hand motion if the user's body is completely contained inside the camera's field of view. Furthermore, due to hardware limitations that result from their near-infrared structured light approach, the user is expected to be positioned at a minimum distance of 60 cm from the camera, the system can not operate in direct sunlight, and tracking suffers from a high latency (60 ms for Kinect 2.0). Another well known commercial hand detection and tracking platform was developed by the Belgian company SoftKinetic which produces a consumer model time-of-flight camera. Their infrared time-of-flight camera provides low latency, high resolution depth estimates, and is used by the SoftKinetic iisu [SoftKinetic 14] software for hand tracking purposes. However, their software can not robustly separate faces from hands, and yields unstable results when hands overlap. Furthermore, their camera can not be used in direct sunlight, due to its operation in the near-infrared spectrum.

In general, range cameras, as used by the above methods, greatly simplify the problem of object tracking and detection by allowing instant removal of background pixels. However, depth cameras usually rely on infrared signals and perform poorly in direct sunlight or in environments with other infrared light sources, which limits their applicability. Additionally, these methods can not be applied to existing video datasets that lack depth information. Furthermore, although the use

of specialized hardware might indeed greatly simplify the problem at hand, from an academic point of view it is still extremely interesting to solve the problem in its original setting, as this might shed a light on how the human vision system is able to perform this task. In this dissertation, we therefore focus on intelligent algorithms and apply our methods on simple monocular video. Our focus is on the development of real-time hand detection and tracking methods, that can operate in unconstrained environments and changing illumination, and can automatically initialize and recover from failure. Our methods only rely on 2D video data, captured with an inexpensive webcam device, and are able to automatically learn from and adapt to their environment.

The research questions that are discussed and answered in this dissertation can thus be outlined as follows:

2D Hand Tracking: How can a human hand be tracked robustly and in real-time throughout a monocular, and possibly low-resolution video sequence, given its initial location? Important constraints for this problem are the possibility of sudden illumination changes in the scene, quickly changing backgrounds, moving cameras, unconstrained hand poses, and the need for a real-time solution.

Hand segmentation: Can we obtain a segmentation mask of the hand, based on the hand tracker's results? Could such segmentation mask be used to automatically adapt the tracker's internal color and texture models in order to increase its robustness during tracking?

Color modeling: Can skin color be modeled such that the resulting algorithm is independent of both ethnicity and illumination? Can the color models be made adaptive in order to cope with changing illumination?

2D Hand Detection: How can a human hand be detected automatically, and in real-time, in static images? Can such detector be designed such that it can detect hands in any pose, irrespective of scale and orientation? Can such detector be used to automatically initialize a tracking algorithm for video sequences, and can it be used for automatic error detection and recovery?

Closed-loop system: How can information be shared between the tracking, segmentation, and detection modules, such that each of those can automatically adapt their internal models as to learn about and adapt to the current environment and context?

Unsupervised context learning: If an object of interest consistently exhibits motion patterns that are correlated with those of other objects, then can we automatically model these correlations? How could such model be used to

perform automatic arm detection during hand tracking? How could context learning be used to improve the tracker's robustness to occlusions?

Real-time system: Can the complete system be implemented such that it operates in real-time on modern consumer hardware? Which sacrifices need to be made and which implementation choices are important to achieve this goal?

Floating display: How could we develop a holographic computer display, such that a virtual object appears to float in thin air? What are the important design choices and what are the production limitations? Could such a computer display be controlled interactively by means of hand gestures?

2 Relevance and Applications

Robust and real-time hand detection and tracking can be applied in a variety of industrial domains. In this section, we first discuss why 2D hand tracking is an important field of research, and then present an overview of some of the most important applications.

Obvious approaches to hand tracking, and object tracking in general, include the use of specialized hardware such as time-of-flight or structured light cameras, or the use of multiple cameras, as used in stereo-vision applications. Such solutions greatly simplify the problem since the resulting depth map or disparity map allows to quickly distinguish background objects from the object of interest. Problems such as segmentation, detection and tracking, known to be challenging for algorithms that use monocular video, often become trivial when processing depth information. Moreover, temporal ambiguity, caused by occlusions of the objects of interest, can be resolved easily if multiple camera views are available.

Nevertheless, the use of such specialized hardware poses additional constraints on the system, as discussed in section 1. These devices often fail in specific illumination, need a lot of processing power, are more expensive than traditional cameras, have a very limited range in which they operate correctly, exhibit a small field of view, and are sometimes simply not available.

Simple monocular cameras, on the other hand, introduce different problems, such as their limitations in very dark environments, and their 2D perspective of the world which makes it difficult to distinguish background from foreground objects. Most of these problems of both types of camera devices can be grouped in two disjoined sets, suggesting that the combination of both approaches could be used to develop a computer vision solution that benefits from the best of both worlds.

Ideally, we would like to be able to first solve the problem of hand detection and tracking in its most difficult setting; using only a monocular video stream,

originating from a low-end webcam device. If sufficiently robust and accurate results can be obtained in this setting, then specialized hardware can easily be added in a later stage of the design process, to handle edge cases. Depth camera's or stereo- vision approaches could then be used as a fall back when monocular tracking fails, and vice-versa. The remainder of this dissertation therefore focuses on monocular video, and discusses several novel solutions to perform hand detection and tracking.

Recently, several industries have shown significant interest in robust and real-time gestured based solutions. *A first example* resides in the medical industry. Both before and during surgery, surgeons often need access to electronic medical records and imagery. Traditional peripheral hardware, such as a keyboard and mouse, can compromise sterility of the operating room, and can spread infections. Furthermore, operating this hardware poses an additional mental burden upon the medical personnel. Research has shown that these problems can be elevated by means of gesture based human computer interfaces [Jacob 13], in which the surgeon uses natural gestures to remotely control the computer, thereby significantly reducing the cognitive load [Cornelius 13].

Another application of the proposed system is in collaborative industrial design, where multiple designers simultaneously try to accomplish a task, while manipulating the same virtual model or environment. Kraut *et al.* [Kraut 03] showed that visual information, such as body language, is of great importance when performing collaborative tasks, as this information helps maintaining situational awareness and helps achieving mutual understanding. Using hand gestures instead of traditional peripheral hardware would allow users to physically interact while performing their tasks.

Other obvious applications reside in the entertainment and multi-media sector, where several big companies, such as Microsoft [Microsoft 10], SoftKinetic [SoftKinetic 14], and Leap Motion [LeapMotion 13] are focussing on specialized hardware to solve the problem of robust hand tracking and detection. Samsung on the other hand only uses a monocular camera to allow users to control their smart television using a discrete set of hand gestures.

Finally, a variety of cross-domain research projects would greatly benefit from a robust hand tracking and detection system. Examples include the study of body language of teachers in classrooms [Cooper 13] or applicants during job interviews [Chollet 13]. The results of these studies are often based on careful, manual annotation of a large set of monocular video sequences. Robust hand tracking solutions can automate this process, allowing more data to be analyzed in a shorter time frame.

3 Overview and contributions

In this dissertation we propose several techniques for real-time, robust hand detection and tracking. Each of these methods have been designed to operate in real-time, while exhibiting robustness against unexpected environment and illumination changes.

Moreover, we developed a novel type of 3D computer display and showed how such display can serve as a HCI component when combined with hand-tracking software.

Each of these contributions are briefly introduced in the next sections and are discussed in more detail in the following chapters.

3.1 Real-time, Unconstrained Hand Detection

Literature contains many examples of hand tracking methods that require a specific hand position and pose for initial hand detection before tracking can begin. In practical applications this can be a significant limitation. On the one hand, re-initialisation is needed each time hand tracking fails (e.g. when the hand disappears from the picture). On the other hand, requesting the user to adopt a specific hand pose is annoying if it happens too frequently. It also requires additional training, e.g. using on-screen instructions, which complicates the user interface. In chapter 2 we therefore propose a real-time algorithm to detect hands irrespective of pose, rotation, scale, background and illumination.

Well known object detection methods in computer vision usually employ a template matching approach. The appearance of the object of interest is described using several features such as color or texture descriptors. The detection of these features in a new image then provides evidence about the existence of the object of interest in the image. Although this approach has been proven to be useful for a variety of problems such as face detection, it can not be used to detect highly articulated objects whose appearance can differ greatly depending on their pose.

To robustly detect hands, we therefore propose an algorithm that detects object parts instead of complete objects. Although the appearance of a hand highly depends on its posture, the appearance of hand-parts such as fingertips, is generally stable across different hand poses. Our method therefore models the relative position of these object parts in a probabilistic manner. Detected object parts then cast a probabilistic vote on the centroid of the hand. By searching for the local maxima in this voting space, hands are detected.

Although our method was specifically designed for real-time hand detection, we show that it can be useful for more general object detection tasks. While the proposed solution outperforms the state-of-the-art in hand detection, it is competitive with the state-of-the-art in people detection and animal detection while providing a decrease in computational complexity with factor 1 000.

3.2 Sparse, Regularized Optical Flow

Whereas object tracking is concerned with estimating the motion path of a specific object in a video sequence, optical flow estimation methods try to obtain an estimate of the pixel-wise motion in a video frame. For instance, if an object moves closer towards the camera, its location, when projected onto the image plane, does not change such that an object tracker would not report a change in position. On the other hand, individual pixels inside the object would appear to move in a radial fashion, due to the zoom-like motion of the object. This apparent pixel-level motion is captured by optical flow estimation methods.

Optical flow is defined as a 2D vector field describing the apparent velocities of objects in a 3D scene, projected onto the image plane. Roughly estimating the optical flow field is generally easier than estimating the motion of a specific object in the scene, mostly because the object of interest is not always easily differentiable from its background. We therefore propose an object tracking method that uses optical flow estimates, amongst other types of information, to robustly track the object throughout the video sequence.

Optical flow estimation methods try to estimate local flow vectors without considering high level information about the different objects in the scene. Therefore, the precision of individual flow vector estimates can greatly differ due to noise and ambiguity in the image. To counteract this problem, regularization is often used to impose a smoothness constraint while estimating the optical flow field. However, regularized optical flow estimation methods described in literature operate on a regular grid, estimating the flow vector for every pixel in the image. As a result, these methods are often too slow for real-time applications.

in chapter 3 we therefore propose a method that produces a sparse, yet regularized, optical flow field estimate that can be obtained in real-time. Flow vectors are only estimated for interesting points in the image, and a regularization scheme is used to increase robustness to noise. Furthermore, our method is able to apply this regularization locally such that vectors are only regularized based on other vector estimates within the same object in the image. This allows our solution to cope with motion discontinuities at boundaries between different objects in the scene.

We show that our method outperforms well known point-matching algorithms and is competitive with the state-of-the-art in optical flow estimation while being much more efficient to calculate. Our method can easily be used in a real-time setting, consuming only a fraction of the available computational power. Furthermore, we show that our method is illumination invariant, and illustrate its effectiveness in improving the sampling scheme of a particle filter based hand tracking.

3.3 Real-time, Robust Hand Tracking

To continuously track hands throughout the video sequence, we use a recursive Bayesian tracking approach as will be discussed in chapter 4. This allows us to iteratively refine the estimated object position, incorporating new information when it becomes available. A well known recursive Bayesian filtering techniques that can deal with the highly non-linear observation model and the possibly multi-modal posterior distribution encountered in hand tracking, is the particle filter.

We employ a sensor fusion approach to combine multiple types of information about motion estimation, object colors and textures, and background modeling into a robust probabilistic framework. Furthermore, we present a theoretical framework that allows us to efficiently incorporate the optical flow estimates, discussed in the previous section, into our probabilistic sampling scheme.

The object detection method described in chapter 2 is used to initialize our tracking algorithm, and allows it to automatically detect and recover from tracking failure. Furthermore, the probabilistic voting mechanism used by the detector is directly incorporated into the observation model of the particle filter to increase tracking robustness in ambiguous situations (e.g. when the hands overlap or come close to the face).

Finally, we present a simple segmentation method that allows us to quickly obtain a rough estimate of the contour of the hand. Based on this contour estimate, the color and texture models of both the detector and the tracker are automatically updated, such that our solution can quickly adapt to changing environments.

We compare our real-time method with state-of-the-art object tracking algorithms, and show that it largely outperforms these methods when applied to the task of hand detection. The complete tracking system can automatically detect hands when these enter the video frame, can recover from failure, and can dynamically detect and track multiple objects at once.

3.4 Tracking Through Occlusion by Context Learning

A major shortcoming of many state-of-the-art tracking approaches, is their tendency to be easily distracted when the object of interest is occluded by visually similar objects.

In chapter 5 we formulate a solution to this problem, based on the idea that a moving object is usually embedded in a specific context and rarely moves alone. In the case of human hands, an obvious example of such contextual link would be the arm that exhibits a motion pattern that is strongly correlated to that of the attached hand.

We propose an efficient solution to automatically and adaptively learn the arm's appearance, and to incorporate this knowledge into the tracking algorith of chapter

4. The resulting method is able to robustly track both hands and arms through in real-time, while coping with occlusion.

Furthermore, the proposed solution is not limited to arm tracking. In many practical applications, the tracked hand holds an object such as a smartphone or game controller, or attributes such as an umbrella (i.e. surveillance applications). Whereas this degrades tracking performance if a traditional tracking algorithm were to be used, it can actually improve tracking if the system is able to learn that the object's position is temporally correlated with the hand. To this end, we propose a method to exploit temporal correlation, in order to automatically learn which objects appear to be temporally connected to the tracked object.

3.5 A Transparent Floating Display

One of the most apparent applications of real-time, robust hand detection and hand tracking algorithms, can be found in human-computer interaction (HCI). An increasingly popular subdomain of HCI research is related to multiple-user environments for performing collaborative physical tasks. Examples can be found in rapid prototyping or medical research, where multiple users collaborate to perform a single task. Kraut *et al.* [Kraut 03] showed that visual information, such as the possibility of maintaining eye contact with other users, is of great importance when performing collaborative tasks. This information helps maintaining situational awareness and helps achieving mutual understanding. However, in a traditional HCI setting, where each user sits in front of his own computer display, most of these cues are lost. Even if multiple users share the same computer display, collaboration is often difficult because it is not possible to focus on the display, the environment and the other users at the same time.

In chapter 6 we therefore propose a design of a new type of display that overcomes these limitations. The proposed display is transparent and as such does not block the field of view of its user, allowing the observer to maintain eye contact with other users. Furthermore, the display effectively projects the image in thin air, such that the virtual object seems to float at a distance from the transparent medium. Therefore, the virtual object is perceived as being part of the real world, and the user can interact with it based on natural hand gestures.

Prototypes of our display have been showcased at various national and international events and tradeshows, and have been received with much enthusiasm by the public. First steps towards a spin-off company have been taken, resulting in a license agreement related to part of the results that are presented in this dissertation.

4 Summary of the Scientific Output

This dissertation resulted in ten peer-reviewed publications at international conferences, four of which as a first author [Spruyt 10a, Van den Berghe 11b, Van den Berghe 11a,Creemers 11,Heyman 12,Heyman 11,Spruyt 12,Spruyt 13b,Spruyt 13a, Bellekens 14]; one peer-reviewed publication at a national conference [Spruyt 10b]; and three international journal papers, one of which has been published [Spruyt 14], one of which was under review at the time of writing this dissertation, and one of which has been submitted.

Furthermore, the work on the development of a transparent floating imaging display has led to a commercial license agreement between an interested party, i.e. spin-off company in formation BI3D, and both Ghent University and the University of Antwerp, concerning part of this PhD research. Moreover, two hardware prototypes have been produced and showcased on several national and international events and tradeshows.

Finally, a demonstrator showing real-time, robust hand detection and tracking in video using low resolution webcam input has been developed.

5 Outline

The chapters of this dissertation each describe an important component of the complete tracking and detection system that was developed during our research. Every component receives information from other components and feeds back its output. As a result, the tracking system is able to adapt to dynamic environments and to actively update its models at run-time. To aid the reader in maintaining an overview of the relations between these components, and thus chapters, figure 1.1 presents a general overview of the modules, their relations, and the chapter numbers in which they are presented.

This dissertation is outlined as follows: Chapter 2 discusses the robust and real-time object detection algorithm that was briefly introduced in section 3.1. In chapter 3 we propose a real-time, sparse and locally regularized optical flow method, introduced in section 3.2. The object detector and optical flow estimator are then combined by the Bayesian hand tracking algorithm proposed in chapter 4 and discussed earlier in section 3.3. In this chapter, we discuss a complete hand tracking framework with automatic initialization and error recovery, and show that it outperforms the state-of-the-art while providing a significant reduction in computational complexity. Finally, in chapter 6 we discuss the design and development of a new type of holographic computer display that is able to project a virtual image in thin air. The result is an interactive human-computer interface that creates the illusion of observing a floating object, thereby creating a truly immersive experience.

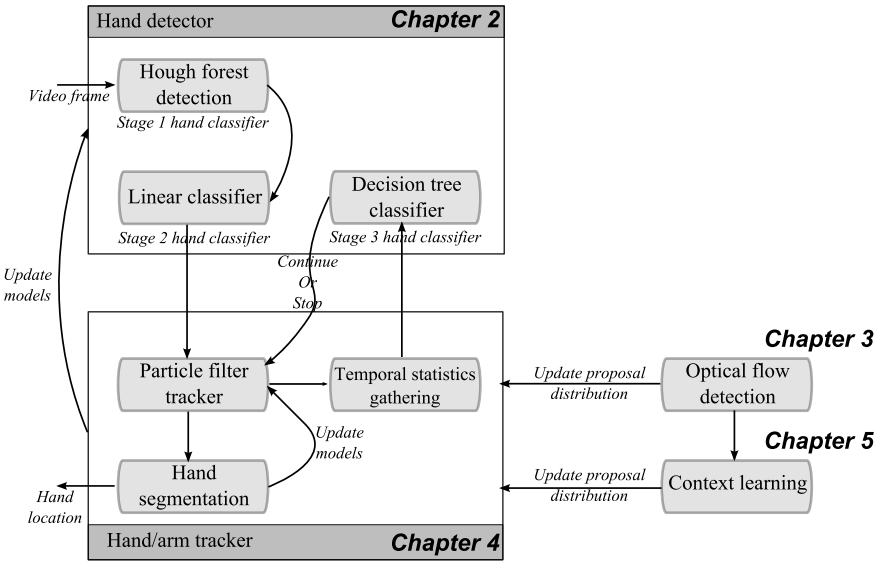


Figure 1.1: Illustration of the complete tracking and detection system, and their relations. Each module in this figure is accompanied with the chapter number in which it is presented. This figure can serve as a guideline while exploring the dissertation.

2

Real-time, Unconstrained Hand Detection

1 Introduction

Vision based hand tracking and hand detection are amongst the most challenging tasks to accomplish in modern human-computer interfaces. Although hand pose and gesture recognition are receiving an increasing amount of attention in the computer vision community, current approaches often assume that the initial hand location is known to the classifier, thereby skipping the more difficult step of initial hand detection for tracker initialization or error recovery.

Furthermore, since low cost depth cameras recently became available, focus has shifted from object detection and tracking in monocular video, to image analysis using depth maps. However, due to their dependency on infrared signals, depth cameras based on time of flight or structured light often perform poorly in direct sunlight, limiting their applicability to indoor applications with restricted lighting conditions. Additionally, these methods cannot be applied to existing image databases lacking depth information. A real-time algorithm for long-term, robust hand tracking with automatic initialization and error recovery would solve these problems and could even be combined with depth information to increase robustness.

As opposed to rigid or non-articulated objects such as faces, human hands lack clear discriminative features, making it difficult to design a robust hand detector based on low level feature detection. Additionally, the human hand has a high

number of degrees of freedom (DOF): 27 in total, six of which represent translation and rotation of the wrist [ElKoura 03]. Due to this high number of DOF, a hand's 2D silhouette can assume a wide variety of shapes, further complicating robust hand detection. As a result, even the current state-of-the-art detection methods are unable to robustly detect hands in video sequences.

Moreover, current hand detection and tracking solutions all have limitations: Some methods focus on constrained environments with a controlled background [Stefanov 05, Donoser 08, Pham 09]. Other methods require the user to wear colored gloves [Robert 09], or use specialized hardware such as an infrared depth sensor [Oikonomidis 11, Liang 12, Melax 13]. Many methods assume uniform illumination because they use a simple skin color or motion detection based segmentation.

Finally, today's state-of-the-art detection methods often need several seconds or minutes of processing time [Gall 09, Mittal 11] per video frame, or can only detect rigid objects from a predetermined viewpoint [Viola 04]. In most human-computer interfaces however, hand detection is merely a small processing task in a much larger system, and should therefore only consume a fraction of the available processing power. This real-time constraint greatly reduces the number of techniques that can be applied, and proves to be a challenging problem.

Literature contains many examples of hand detection and tracking methods that require a specific hand position and pose for initial hand detection (see Section 2). In practical applications this can be a significant limitation. On the one hand, re-initialisation is needed each time hand tracking fails (e.g., when the hand disappears from the picture). On the other hand, the need to adopt a specific hand pose is annoying to the user if it happens too frequently. Requiring a specific initial hand pose also requires additional training of the user, e.g. using on-screen instructions which complicates the user interface. Therefore in this chapter, we propose a real-time algorithm to detect hands irrespective of pose, rotation, scale, background and illumination.

Instead of directly detecting complete hands, our solution detects hand parts, such as finger tips, and models the spatial relation between these parts in order to estimate the hand location. Hand parts occur in different spatial configurations, orientations and scales, depending on the gesture and the distance from the camera. Whereas most object detection techniques try to overcome this problem by re-detecting the object in multiple scaled and rotated versions of the image, this approach is computationally too complex for real-time systems. Therefore, we introduce an object detection method that overcomes these problems, by directly incorporating scale and rotation invariance into the feature description and classifier training stage. This allowed us to efficiently use the available training data, since hand parts with different orientation and scale are not considered to be dif-

ferent object classes. Moreover, our approach allows for invariant object detection, resulting in a fast, real-time system.

Furthermore, we propose a three-stage classifier to further reduce the false positive rate of the system, and to incorporate temporal information. Based on this classifier cascade, a complete tracking system with automatic initialization and error recovery is proposed. We show that our approach outperforms the state-of-the-art in hand detection while allowing real-time processing.

Finally, an extensive evaluation of our methods is presented, by processing both images and videos, and by training and testing our detector on completely different datasets, such as pictures of horses, bikers and persons. Our results are compared with state-of-the-art detection methods, using several well known and publicly available datasets. We show that our hand detector outperforms the state-of-the-art, whereas our general detection results are competitive with the state-of-the-art in object detection. Moreover, our method provides a decrease in computational complexity with factor 1 000.

This chapter is outlined as follows: In Section 2 we describe related work. In Section 3.1, we provide an introduction into and discussion of Random Hough Forests. Section 3.2 explains our complete Hough detection method in detail, including the feature detection and description stage, the training stage and the classification stage. In Section 3.3, we discuss how a cascade of the Random Hough classifier with two additional, yet simple, classifiers increases detection performance, and allows us to detect hands in real-time video sequences. Finally, Section 4 provides a thorough evaluation and discussion of the detection results, using several publicly available datasets.

2 Related Work

Object detectors can be categorized based on the model that is used to describe the classes of interest. Traditional object detection approaches use a rigid model to assign a score to each object location hypothesis in the image. More recent object detection methods on the other hand, represent an object class as a deformable configuration of object parts. Each part represents local characteristics of the object, whereas their deformable spatial configuration describes the expected variance in the relative location of these parts.

A classical example of an object detector that describes the object using a rigid model, is the well known Viola and Jones detector [Viola 04]. Using a sliding-window approach, a cascade of Haar-like features allows the detector to quickly discard background pixels so that it can focus on the more likely object location hypotheses. This approach was used by Kölsch and Turk [Kölsch 04b], who trained a boosted Haar-cascade to detect hands in several different postures. While a reasonable detection rate between 65.8% and 92.23% was achieved, their method is

view dependent, and can therefore only be used to detect hands in a predefined pose.

A similar approach was proposed by Just *et al.* [Just 06]. However, instead of relying on Haar-like features, the Modified Census Transform (MCT) was used to describe the texture of the neighborhood of each pixel. The MCT is a feature descriptor that encodes the local texture information of a pixel as an ordered set of binary comparisons of pixel intensities between all pixels in a 3×3 neighborhood and the average intensity within this neighborhood. Calculating these features involves a simple and efficient convolution. The resulting detector outperforms the detector that was proposed by Kölsch and Turk in terms of recognition rates, but suffers from the same problem: Due to the rigid model that is represented by the boosted classifier cascade, only predefined hand postures can be recognized.

Another rigid model based hand detector was proposed by Ong and Bowden [Ong 04]. Their approach consists of a tree of boosted detectors. While the head of the tree returns hand position hypotheses, the remaining cascades focus on specific hand postures. The tree is learned by clustering a training set of hand images, based on their shape. Although this approach seems to outperform the methods that were discussed above when used on a dataset with uniform background, the system fails in case of cluttered and changing backgrounds. Furthermore, due to the rigid model that is represented by the boosted cascades at each leaf-node of the tree, only a specific set of hand postures can be detected.

The idea of using a hierarchical approach in which child-nodes at lower levels classify more specific hand postures, was taken further by Stenger *et al.* [Stenger 06b], who proposed a rigid model based hand detection and tracking system using a hierarchical Bayesian filter. A large database of hand templates is hierarchically clustered based on their Chamfer distance. Hand detection is performed by searching the hierarchical template tree, and temporal consistency is enforced during tracking, by means of Bayesian filtering. Depending on the structure of the template tree, this method either allows precise recognition of a limited set of hand postures, or coarse detection of a larger number of hand postures. However, due to the computational complexity of the Chamfer distance metric, only three frames per second can be processed on a 2.4 Ghz P4 computer.

A two-stage hand detector, in which a rigid model is learned, was proposed by the same authors [Stenger 06a]. In the first stage, the system generates hand location and scale hypotheses, based on simple color and motion cues. The color cue represents skin color likelihood, whereas the motion cue represents the difference in pixel intensity between the current frame and the previous frame in a video. Maxima are found in the distribution of these two features and are treated as hand hypotheses. Each hypothesis is then tested by the second stage classifier. The second stage classifier is a nearest neighbor classifier in which edge orientation and skin color likelihood are combined in a feature vector. A distance measure

is then calculated between the feature vectors that describe the bounding box of a hand location hypothesis, and each entry of a template database that was generated during training. Similar to the earlier discussed rigid model based approaches, this detector can only detect hands in specific postures. Nevertheless, the idea of combining a high-recall classifier with a high-precision classifier, into a two-stage classification system, is an interesting idea that is built upon in our research.

The idea of a two-stage hand detector was reused by Mittal *et al.* [Mittal 11], who proposed a parts based detector. As opposed to the earlier discussed rigid model based detectors, a parts based detector is able to cope with deformable objects by modeling the relative position of object parts, together with their expected variance. Their method uses a parts based deformable model, based on Histogram of Oriented Gradient (HoG) features. The proposed system consists of a two-stage classifier, where the first stage generates hand location hypotheses, while the second stage confirms or rejects these hypotheses, based on skin color and face detection. Due to the parts based approach, the resulting detector is able to detect hands, irrespective of their posture, in unconstrained environments with random backgrounds and illumination. However, the whole detection process takes approximately two minutes for an image of 360×640 pixels on a 2.5 Ghz quad-core computer, and is therefore not suited for real-time HCI applications.

While well known object detection approaches, such as the Viola and Jones detector, fail to capture the full range of hand poses, Mittal *et al.* showed that parts-based classifiers can yield robust hand detectors. Despite the high number of degrees of freedom, and the articulated nature of human hands, several parts of the hands, such as fingers, are usually visible throughout different hand poses. Detecting these parts, and modeling their spatial configuration, could therefore yield a robust and fast hand detector.

Leibe *et al.* [Leibe 08] proposed such a parts-based detector that learns a class-specific Implicit Shape Model (ISM). This ISM is a codebook of interest points, obtained by a feature detection method such as SIFT (Scale-invariant Feature transform). The codebook is typical for the object class under consideration. Furthermore, an offset vector from the interest point to the object's centroid in the training data is stored for each codebook entry. These vectors are used to build a probabilistic model of the spatial configuration of the interest points. At runtime, interest point descriptors in the image are matched against the codebook, and matching entries cast a probabilistic vote for the centroid of the object. The resulting detector is robust against object deformations and occlusions, but needs between 4–7 seconds of processing time for an image of size 320×240.

The approach that was proposed by Leibe *et al.* is referred to as a generalized Hough transform, since votes are accumulated after which maxima are located in this accumulator. While this generalized Hough based detector is able to cope with deformable objects, partial occlusion and complex backgrounds, the method

requires computationally expensive codebook matching at runtime, limiting its applicability in real-time applications. Moreover, in order to generate the codebook, large scale clustering problems need to be solved.

Based on the work of Leibe *et al.*, Gall and Lempitsky [Gall 09] proposed a parts-based object detector, using random forests. A random forest is trained using image patches that are selected by means of a sliding window. For each image patch, an offset vector to the object centroid is stored, describing its relative location. During classification, new image patches are classified by each tree in the forest, and the offset vectors that are stored in the leaf-nodes of each tree are used to cast a probabilistic vote for the object's center. Despite the impressive results, the random forest based detector is not scale and rotation invariant. In order to cope with different scales, several scaled versions of the image are classified sequentially, yielding high processing times. The complete detection system needs approximately 6 seconds to process a 720×576 image, and can only detect objects at four different scales within this time-frame. In addition, the detector is not rotation invariant.

Based on the work of Gall and Lempitsky, we propose a real-time, scale and rotation invariant detection method for hands in video sequences. Our detection system consists of a three-stage classifier and can robustly detect hands, independently of their posture. Based on our detector, we propose a complete real-time tracking system that can operate in unconstrained environments with changing illumination and background. We show that the detector can be used to efficiently initialize tracking algorithms, and can be used to provide automated error recovery in case of drift or tracking failure.

3 Materials and Methods

In order to detect hands in unconstrained video, we propose a cascade of three classifiers. Each classifier outputs a set of hypotheses about possible hand locations, which are then verified by the next stage.

The first-stage classifier is a parts based classifier. A region of interest detector is used to select blob-like structures in the image. These regions are classified as being part of a hand, or part of the background. For regions that are part of a hand, the spatial configuration, i.e. their offset from the hand's centroid, is modeled, such that during classification the centroid of hands can be inferred based on the detection of individual hand parts. We propose to use a Random Hough Forest as the first-stage classifier. An introduction into random forests, decision trees and Random Hough Forests, and an explanation about our choice for this approach, is provided in Section 3.1.

The parts based Random Hough Forest classifier takes a set of regions of interest as its input. To detect these regions of interest, we search for blob-like

structures in the image by detecting isotropic regions of high entropy and low self-similarity across different scales, as discussed in more detail in Section 3.2.1. Although many types of region of interest detectors, such as the well known SIFT algorithm, are available, one of the main contributions of our research is the real-time performance of our classifier. Therefore, the region of interest detector should only consume a fraction of the available CPU power. In Section 3.2, we discuss the chosen region of interest detector in detail, and describe several enhancements to the original method. Furthermore, we discuss how to incorporate scale and rotation invariance directly into the Random Hough Forest classifier, by keeping track of normalized offset vectors in the leaf-nodes of each decision tree. Finally, we discuss how the trained random forest can be used to detect hands in arbitrary images, by means of Hough voting.

The first-stage classifier is used as a high-recall, low-precision classifier. It detects almost all hands in any image, but also returns many false positive detections. To eliminate these false positive detections, we propose a second-stage classifier that classifies the earlier detected hand bounding boxes as a whole, instead of individual object parts. Finally, a third-stage classifier is proposed to exploit temporal information that is available in video data. The complete cascade is discussed in more detail in Section 3.3.

3.1 Random Hough Forests: an Introduction

A Random Hough Forest is an object detector that is largely based on the concepts of random forests. A random forest [Breiman 01], illustrated by figure 2.1, is an ensemble classifier composed of several binary decision trees, each trying to solve the same task. Therefore, before discussing Random Hough Forest classifiers, it is important to review the basics of binary decision tree classifiers and random forests first.

3.1.1 Binary Decision Trees

Decision trees are amongst the most simple classifiers to understand. Like any classifier, a decision tree takes a set of features as input, and returns a class label as its output. A decision tree consists of a set of nodes that are connected by branches. Non-leaf-nodes are called decision-nodes. In binary decision trees, each decision-node has exactly two child-nodes. Each branch that connects a decision-node with its two child-nodes, corresponds to a binary decision value. During classification, each decision-node compares a specific feature value with a threshold value, and then follows one of the two branches that corresponds to binary outcome of this test. This process is repeated until a leaf-node is reached. Leaf-nodes in a decision tree correspond to class labels, and thus represent the final decision.

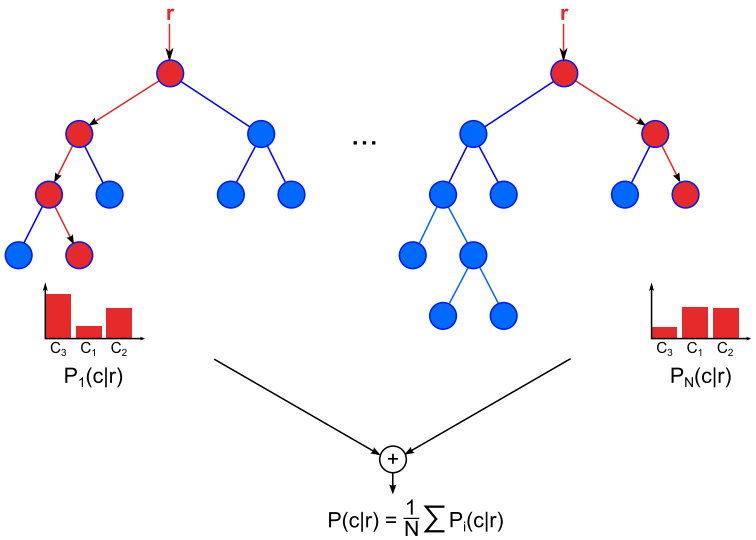


Figure 2.1: A random forest consisting of N decision trees. A feature input vector r is classified by each decision tree independently. Class probability distributions are marginalized to obtain the aggregated class decision.

Although decision tree classifiers are easy to use and can be implemented extremely efficiently, training a decision tree is a difficult problem. During tree construction, one of the several available features have to be chosen for the binary test at each decision-node. Furthermore, a threshold value has to be determined for this test. Many different choices for these parameters can result in different decision trees that represent the same set of final decisions. Simply calculating the decision tree that corresponds to the smallest tree of such an unknown set of equivalent trees that represent the same decision outcome, is an NP-hard problem.

Common methods to train decision trees are the ID3 algorithm and its successor, C4.5 [Quinlan 93], which resort to a greedy heuristic approach to determine the splitting criteria. At each decision-node, the information gain (also know as mutual information) for each possible splitting criterion is calculated, and the criterion yielding the highest information gain is chosen. The information gain is defined as the difference in entropy before and after a training set is split by a certain decision rule. In other words, maximizing the information gain reduces the uncertainty in the training data.

Binary decision trees divide the feature space into axis-parallel hyperrectangles, where each hyperrectangle is assigned a specific class label. The area of such a decision region can be arbitrary small, and depends on the depth of the decision tree. Therefore, a major disadvantage of decision trees is their tendency to overfit the training data. In theory, it would be possible to train a tree such that each de-

cision region contains a single training sample, such that the training error equals zero. However, such a classifier would not perform well on unseen data, because it does not generalize well.

Therefore, in order to increase generalization and to reduce overfitting, decision trees are usually pruned; Multiple leaf-nodes are merged with their parent decision-node. Pruning is an important aspect of decision tree training, but is very prone to error. If a tree is not pruned enough, overfitting occurs, while too much pruning would result in low classification rates.

3.1.2 Bagging and Random Forests

A random forest [Breiman 01] is an ensemble classifier composed of several binary decision trees, each trying to solve the same task. Random forests are considered amongst the most robust classifiers currently available, and have been shown to perform as well as or better than Support Vector Machines, while being much less computationally expensive to train or execute. As opposed to, e.g., neural networks, random forests largely avoid the curse of dimensionality and are therefore less prone to overfitting [Bosch 07].

A well known issue that should be considered when choosing a classifier, is the *bias-variance trade-off*. *The bias* of a classifier represents the number of samples that would be consistently misclassified, if the classifier would be trained on different subsets of the complete training population. A simple linear classifier, for instance, can be described using only a few parameters and would therefore yield approximately the same decision boundaries if it were trained on a different subset of the training population. Therefore, linear classifiers would consistently miss-classify the same samples across its training sets, and are known to be high-bias classifiers. Decision trees, neural networks or K-nearest-neighbour classifiers on the other hand, would result in decision boundaries that closely fit the chosen training data, yielding low-bias classification.

The variance of a classifier measures the variability of the number of misclassifications when different subsets of the training population are used. A linear classifier, for instance, is a low-variance classifier, since the number of misclassifications would be stable across chosen training sets. Decision trees, neural networks or K-nearest-neighbour classifiers on the other hand, are high-variance classifiers, since the number of miss-classifications strongly depends on the chosen training set.

In general, classifiers that are able to fit the data well, exhibit low bias but high variance, while classifiers that result in more general decision boundaries yield high bias but low variance.

While a low-bias, low-variance classifier is desirable, lowering the variance of a classifier, often implicitly increases the bias, and vice versa. If a low-bias classifier, such as a decision tree, is available, an obvious way to lower the variance,

is to train multiple decision trees on different subsets of the population, and then use the average decision (i.e. regression) or a voting scheme (i.e. classification). However, in practice only a limited amount of training data is available, instead of the whole population. A well know method to approximate the distribution of the complete population if only a limited number of observations are available, is to construct multiple training samples by bootstrapping the original training dataset. Bootstrapping is a resampling method that is known by statisticians as sampling with replacement.

If the original training dataset contains N samples, then we can create bootstrapped samples of this dataset, each containing N samples themselves. If we would randomly pick one sample from the original dataset using a uniform distribution, the probability that an original observation is picked is $\frac{1}{N}$. Thus, the probability that an observation will not make it into the bootstrapped dataset after N sampling attempts is $P(\neg sampled|trials = N) = (1 - \frac{1}{N})^N$. If N is large enough, this probability approaches $e^{-1} = 0.368$.

The expected fraction of observations that will make it into a bootstrapped sample, is therefore $1 - e^{-1} = 0.63$. Since sampling with replacement means that samples can be selected multiple times, this means that about 37% of each bootstrapped sample contains duplicated data. As a result, each classifier that is trained on a different bootstrapped sample, will have slightly different decision boundaries. By aggregating the resulting decisions of each of these possibly high-variance classifiers, by means of averaging or voting, a low-variance classifier is obtained. This concept of *b*ootstrap *agg*regating is called *bagg*ing [Giorgio 03].

Random forests use bagging to reduce the variance of the final ensemble classifier, compared to a single decision tree. However, bagged trees exhibit a high correlation because of the duplicates in their training data and the similarity in their training method. Highly correlated trees would therefore make the same errors in similar regions of the feature space. This means that reducing the variance by means of bagging, increased the bias of the resulting classifier. Furthermore, this bias increases further if the decision trees are pruned, because pruning corresponds to generalizing or smoothing the decision boundaries.

To decrease the bias of the ensemble classifier, random forests ensure diversity of the tree classifiers by introducing randomness into the splitting criterion that was explained in Section 3.1.1: Each time the training set is split, only a randomly selected subset of size f of all features F is considered for selecting the feature for the next decision-node (with $f \ll F$).

Also, no pruning is performed. Although pruning is essential to avoid overfitting when using a single decision tree, the randomness and bagging behavior of random forests take over this task. Random forests have been shown to be almost invariant to overfitting. Moreover, it has been shown that pruning the decision

trees in a random forest would in fact decrease its performance due to an increase in bias [Breiman 01].

Apart from avoiding overfitting, a second advantage of bagging is that it increases the classifier's robustness to noise. It has been shown that randomly changing a fraction of the labels of the training data causes a decrease in classifier performance when using boosting techniques, whereas bagging seems to be immune to the introduced noise [Breiman 01]. Intuitively this can be explained by the fact that boosting is based on weighted resampling of the dataset, assigning higher weights to samples that are difficult to classify and often represent noise, while bagging performs an unweighted resampling. Furthermore, Biau recently showed that random forests are extremely robust to noisy feature variables, whereas many traditional classifiers perform bad in case of variable noise [Biau 12].

Thus, a random forest, is a low-bias, low-variance ensemble classifier, trained by bootstrapping and random feature selection [Bühlmann 02]. It is robust to noisy labels and feature noise, and largely avoids overfitting problems. Finally, classification by means of random forests can be implemented extremely efficient, since each decision tree can simply be represented by a set of conditional statements.

3.1.3 Random Hough Forests

Creating groundtruth data for image processing tasks by manually segmenting the object of interest in a large set of images is a difficult and tedious task. Instead of precisely segmenting the object, we coarsely indicate its bounding box. However, this results in a polluted training set because each bounding box contains several background pixels. Due to its robustness to noise, a random forest classifier is an ideal candidate for object recognition in computer vision.

Traditional random forest classifiers store the distribution of class labels in each of the decision tree's leaf-nodes such that a probabilistic decision can be made during classification. A classified instance ends up in one of the leaf-nodes, after which the class label distribution of this leaf-node is used to assign the most likely label to the instance.

Gall and Lempitsky [Gall 09] introduced the Random Hough Forest for object detection in images. Apart from the class label distribution, a Random Hough Forest stores additional information in each leaf-node, such that other latent variables can be inferred jointly with the class label. When applied to parts-based object detection, simply storing the class label distribution of the object parts, represented by image patches, would only provide an answer to the question whether or not an image patch is part of the object class. However, in order to detect a complete object, the relative position to the centroid of this object needs to be known for each image patch.

Because the position of an object part relative to the object's centroid can fluctuate for non-rigid objects, a Random Hough Forest keeps track of the distribution

of relative positions. If an image patch is classified as being part of the object class, this distribution is then used to perform a probabilistic voting in the parameter space (e.g. a 2D space, representing the x and y coordinates to be estimated). Votes from all trees in the forest are accumulated, and the maxima are detected in this accumulator in order to estimate the latent variables. As such, a Random Hough Forest performs both classification and a generalized Hough transform at the same time.

This is illustrated more clearly by figure 2.2: Figure (a) shows three regions of interest as colored rectangles. The probabilistic votes of each region of interest are visualized in figure (b). Notice how the blue region of interest barely contributes to the final hand detection, since during training, most image patches that ended up in the same leaf node as this image patch, were background patches. All votes are accumulated in a Hough map, shown in figure (c). The maxima in this map then correspond to the centroid locations of the hands in the image, as shown by figure (d).

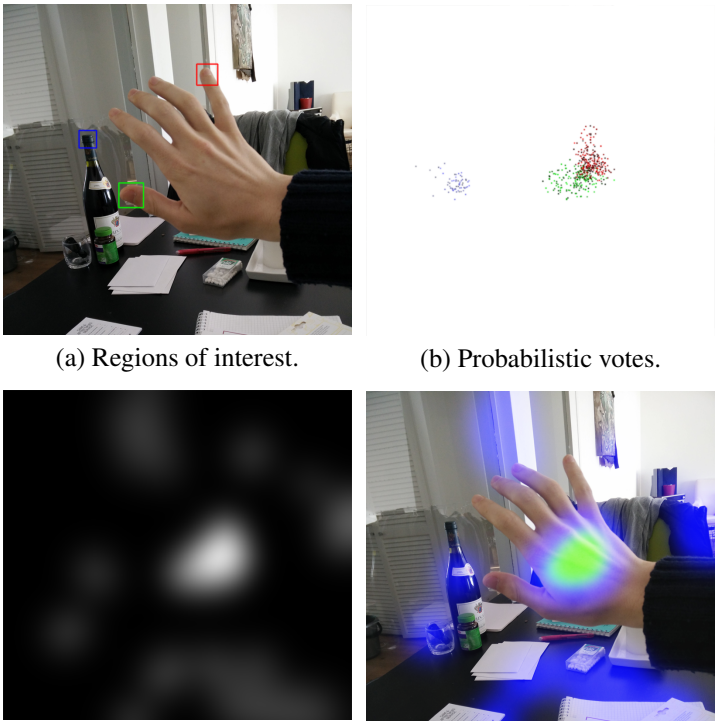
At the decision-nodes of each tree in the Random Hough Forest, the training data is split such that the image patches that are part of the object under consideration are separated from image patches that are part of the background. The result is a set of decision trees in which each leaf-node contains mostly foreground patches, or mostly background patches.

A key difference between a Random Hough Forest and a random forest, is the way the quality of binary tests is evaluated at each decision-node, during training. Traditional random forests rely on the information gain to achieve maximum separability between the class labels, thus reducing the class-label uncertainty towards the leaf-nodes. In Random Hough Forests on the other hand, also the uncertainty of the relative position, represented by an offset vector, needs to be reduced towards the leaf-nodes. The main idea is that each leaf-node will represent a code-book of image patches that is specific for a certain object part.

Random Hough Forests therefore use a splitting criterion in the decision-nodes to encourage patches representing the same part of an object to be stored under the same leaf-node. As a result, leaf-nodes containing positive class labels (i.e. labels of image patches of the object of interest as opposed to background patches) represent a codebook for a specific part of the object. Each codebook entry consists of a feature descriptor, and an offset vector to the centroid of the object.

Gall and Lempitsky measure dissimilarity between patches by the sum of squared difference between the offset vector of the patch, and the average offset vector of all patches in the leaf-node. This dissimilarity is a measure of offset vector uncertainty. The class-label uncertainty is represented by the information gain as discussed in Section 3.1.1.

During training, a random decision is made to either minimize the class-label uncertainty or the offset vector uncertainty. By randomly choosing to minimize



(a) Regions of interest. (b) Probabilistic votes.

(c) Hough voting map. (d) Maxima detection.

Figure 2.2: Each of the three patches, shown in (a) cast probabilistic votes on the hand centroid location (b) (each color corresponds to different region of interest). The probability, visualized by the color intensity, of each vote depends on the likelihood of the patch being part of a hand, whereas the location of the vote depends on the centroid distribution that is modeled by each leaf-node of the Hough Forest. Votes of all patches are accumulated into a Hough voting image (c), in which maxima are detected to obtain the hand location (d).

one of both criteria with equal probability at each decision-node, both the class-label uncertainty, and the offset vector uncertainty decrease towards the leaf-nodes. The actual splitting criterion at each decision-node is then determined by selecting the feature that minimizes the selected uncertainty measure.

The splitting criterion at a decision-node is simply the binary comparison of a feature value with a threshold value. Like in traditional random forests, a subset of features is chosen randomly from the complete set of available features at each node. From this subset, the feature that minimizes the chosen uncertainty measure is chosen to be used in the splitting criterion.

Each available feature describes a different aspect of the appearance of an image patch. Gall and Lempitsky uniformly sample patches of size 16×16 pixels from each image. The appearance of each patch is then described by 16 feature channels, such as the grayscale intensity, the gradient information, and several Histogram of oriented Gradient (HoG) channels which model the normalized counts of the number of occurrences of discretized gradient orientations in local parts of the image.

At the decision-node, a pool of N binary tests is created, by randomly selecting N pixel location pairs within the 16×16 image patch, each with a corresponding, randomly selected, feature channel. Next, for each selected pixel pair, the corresponding feature value at the first pixel location, is subtracted from the feature value at the second pixel location, and this difference is compared with a randomly generated threshold. After all N possible splitting criteria are generated, the one binary test that minimizes either the class-label uncertainty or the offset vector uncertainty is chosen as the final splitting criterion of the decision-node. This criterion splits the training data into two subsets and therefore creates two new child-nodes in the decision tree.

After training, each leaf-node contains an estimate of the posterior distribution of the class labels, and an estimate of the distribution of the offset vectors. During classification, 16×16 pixel patches are uniformly sampled from the image, and are classified by each decision tree in the forest. If a patch ends up in a leaf-node of a tree that contains foreground entries, each foreground entry casts a probabilistic vote about the existence of an object at the centroid location that is defined by the entry's offset vector.

Finally, votes from different trees are averaged and these averaged votes are accumulated additively in the Hough image. The location of the object can then be obtained by finding the maximum in this Hough image. In order to cope with rotated objects, and different scales, the detection process is repeated on rotated and scaled versions of the input image, after which the maximum is located in the resulting 3D or 4D Hough space.

3.2 Rotation and Scale Invariant Hand Detection

Building upon the work of Gall and Lempitsky, we propose a real-time Random Hough Forest based detector. It is able to detect hands in a scale and rotation invariant manner. A rotation and scale invariant feature detector is used to sample image patches, which are described by carefully chosen, scale and rotation invariant, feature descriptors. Furthermore, compared to [Gall 09] we propose an improved splitting criterion, and we revise the probabilistic voting theory to incorporate scale and rotation invariance directly into the detector instead of sequentially classifying rotated and scaled versions of the image. Finally, we introduce a normalization method to cope with the sparsity of the sampled image patches and an online learning solution to allow the detector to adapt to its environment.

3.2.1 Region of Interest Detection: Defining Saliency

During *training*, Gall and Lempitsky randomly sample fixed sized image patches from the training data. During *classification*, they use a sliding window approach to uniformly sample fixed sized image patches from the image under consideration. However, in many realistic applications, most of the processed image patches simply contain low entropy background and are therefore not relevant for the object classification task. As we focus on real-time performance, such an approach would waste a lot of processing time, without actually contributing to the final hand detection.

Instead of uniformly selecting fixed sized image patches using a sliding window approach, we propose to use a simple, rotation and scale invariant, feature detector to sample regions of interest in the image. A variety of feature detection methods have been described in literature, such as the well known SIFT detector [Lowe 04], the Maximally Stable Extremal Region (MSER) detector [Matas 02], and more advanced spatio-temporal detectors such as the method proposed by Wang *et al.* [Wang 09]. A common goal of these feature detectors, is to extract salient, blob-like regions from the image, and to robustly detect these regions, independent from their scale and rotation.

However, many of these feature detection methods exhibit a high computational complexity. In our hand detection framework, detecting salient regions is only a small step in the complete detection pipeline, illustrated by figure 2.3, and should therefore only consume a fraction of the available CPU power. Due to this constraint, we use a simple feature detection method that can be implemented efficiently using SIMD (Single Instruction, Multiple Data) instructions on modern hardware.

A feature detector that matches these constraints, was proposed by Kadir and Bradi [Kadir 01]. In the following, we extend their method such that it can cope with square regions instead of circular regions, and allows for scale-space sub-

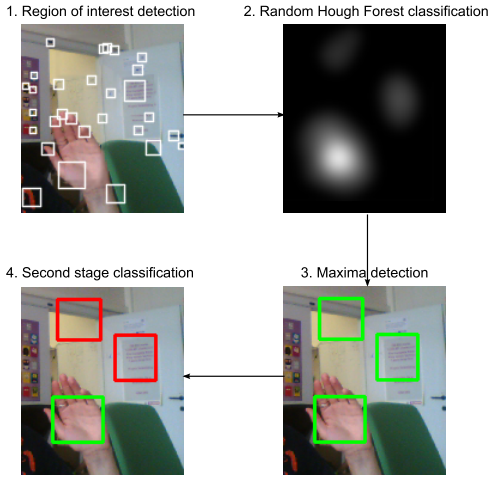


Figure 2.3: The hand detection pipeline: Regions of interest are detected in a scale and rotation invariant manner. Each region is classified as being part of a hand or part of the background by the Random Hough Forest. Hand regions cast a probabilistic vote for the hand centroid. Local maxima are considered hand hypotheses and are fed to a second stage, linear classifier which removes false positive detections.

sampling. As a result, the detector can be optimized significantly using SIMD instructions and parallelization.

The Kadir and Bradi feature detector efficiently detects blob-like structures in the image, jointly estimating their location and scale. Blob-like structures are regions that are isotropic, such as faces, hands or fingertips. On the other hand, anisotropic regions are usually only discriminative in the perpendicular direction, while there is a high degree of self-similarity in the tangential direction. This self-similarity extends over scale-space. Therefore, a simple method to detect blob-like structures, would be to search for regions that exhibit a low self-similarity in scale-space. Features that exist over a large range of scales are considered non-salient or non-discriminative for our goal. Examples of such non-salient regions with high self-similarity, are edges, corners, or regions in images that consist of noise. This is illustrated by figure 2.4.

Obviously, not all image regions with low self-similarity in scale-space, are salient regions. A single leaf in a picture of a tree for instance, would represent a blob-like structure with low self-similarity in scale-space, but would not be a salient feature due to its surroundings. Therefore, Kadir and Bradi propose to use an additional measure, that describes the local ‘rarity’ and thus discriminativeness of the region. From an information theory point of view, interesting patches are those regions whose grayscale histogram exhibit high entropy, and thus contain

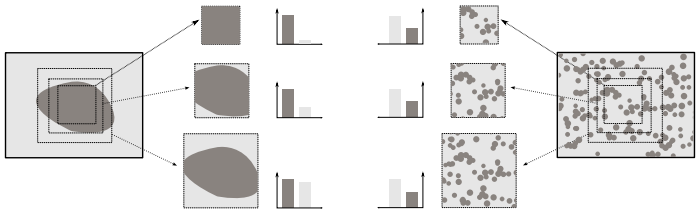


Figure 2.4: Blob-like structures exhibit a low self-similarity in scale space. On the left, a region of interest with three different scales is used to calculate the normalized histograms of a blob-like structure. Histograms across scales differ greatly. On the right, the same experiment is repeated on an image consisting of noise. In this case, self-similarity is high, resulting in similar histograms across scales.

a lot of information. Interesting patches are those whose information can be discriminative, and thus aid in the final classification task.

To understand how both entropy and self-similarity contribute to the definition of a salient region, it can be useful to consider four extreme examples. The first example would be an image patch with a uniformly colored background. Such a patch can be described with a low entropy histogram with high self-similarity, and is therefore not an interesting region. A second example on the other hand, would be an image patch that consists of a fingertip, surrounded with background. Such a patch can be described with a high entropy histogram and exhibits low self-similarity, and is therefore clearly a region of interest. A third example could be an image patch that only contains white noise. Such an image patch can be described with a high entropy histogram, but also exhibits a high self-similarity, and is therefore not a salient region. Finally, a fourth example would be an image patch that contains a smoothly changing color gradient. Such a patch exhibits low self-similarity, but would be described with a low entropy histogram and is therefore not a salient region. Figure 2.5 illustrates this idea graphically.

3.2.2 Region of Interest Detection: Scale Invariance

Kadir and Bradi define saliency of a region as a function of both entropy and self-similarity. The scale of the region is then selected such that this function is maximized. For performance reasons, we use square regions instead of circular regions. This enables an efficient implementation using SIMD instructions and integral histograms as will be discussed later, at the cost of slightly reduced rotation invariance.

In the following let $\mathbf{x} = (x, y)$ be a position, around which we consider square regions with different scale s, such that the width of the region is $2s$. We denote these regions by $R_{s,\mathbf{x}}$. Let $p_n(R; b)$ be the *normalised* n-bin gray scale histogram in an arbitrary region R of the image, where b denotes the bin number, and $h_n(R)$

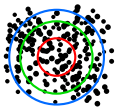
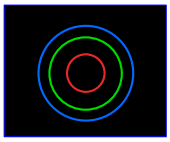
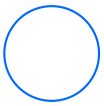
	Isotropic	Noise	Anisotropic	Uniform
Region:				
Histograms:				
Self-Similarity:	*Low*	*High*	*Medium*	*High*
Entropy:	*High*	*High*	*Medium*	*Low*
Best scale:		*None*	*None*	*None*

Figure 2.5: Discriminative features have high entropy and low self-similarity. Different scales are tested during feature selection (indicated in red, green and blue in this example). The normalized histograms for all scales are compared to measure self-similarity. If self-similarity is low, the region might be of interest, and the optimal scale is obtained by selecting the scale that corresponds to the histogram with the largest entropy.

be the corresponding entropy, defined by equation (3.1).

$$h_n(R) = -\sum_{b=1}^{n} p_n(R;b)\log_2 p_n(R;b). \tag{3.1}$$

Kadir and Bradi define the self-similarity between a region $R_{s,\mathbf{x}}$ and the region $R_{s-1,\mathbf{x}}$, as the sum of absolute differences between the grayscale histograms of the non-overlapping subregions $R_{s-1,\mathbf{x}}$ and $R_{s,\mathbf{x}} \setminus R_{s-1,\mathbf{x}}$. The self similarity is then defined by equation (3.2).

$$w(R_{s,\mathbf{x}}) = \sum_{b=1}^{n} |p_n(R_{s,\mathbf{x}} \setminus R_{s-1,\mathbf{x}};b) - p_n(R_{s-1,\mathbf{x}};b)|. \tag{3.2}$$

While there are many ways of comparing two probability density functions, many of which are considered superior to a simple sum of absolute differences (e.g. Earth mover's distance), our choice for this measure, in the context of real-time hand detection, is again driven by its computational simplicity, and the ease of which its calculation can be optimized using parallelization and SIMD instructions.

The histograms on square regions, such as $R_{s-1,\mathbf{x}}$, can be computed in constant time using integral histograms, as will be discussed in more detail on page 32. However, the region $R_{s,\mathbf{x}} \setminus R_{s-1,\mathbf{x}}$ is not square, and calculating its histogram would require more computations. Therefore, in the following, we will rewrite the above equation directly as a function of $p_n(R_{s,\mathbf{x}})$ and $p_n(R_{s-1,\mathbf{x}})$, both of which can be calculated efficiently.

Kadir and Bradi showed that equation (3.2) equals

$$\frac{\|R_{s-1,\mathbf{x}}\| + \|R_{s,\mathbf{x}} \setminus R_{s-1,\mathbf{x}}\|}{\|R_{s,\mathbf{x}} \setminus R_{s-1,\mathbf{x}}\|} \sum_{b=1}^{n} |p_n(R_{s-1,\mathbf{x}};b) - p_n(R_{s,\mathbf{x}};b)|.$$

where $\|.\|$ denotes the vector length and $|.|$ the absolute value operator.

The latter equation allows computing the similarity measure directly as a function of simple local gray value histograms in the image. As opposed to Kadir and Bradi, we chose to use square regions of interest, instead of circular regions, yielding a much less computationally expensive implementation. For square regions, $\|R_{s-1,\mathbf{x}}\| + \|R_{s,\mathbf{x}} \setminus R_{s-1,\mathbf{x}}\| = \|R_{s,\mathbf{x}}\| = 4s^2$.

Also, as opposed to Kadir and Bradi, we will allow our algorithm to subsample scale-space, thereby skipping certain scales while calculating the similarity measure. This again results in a substantial decrease in computational complexity, at the cost of lower scale-space accuracy. Let Δs be the difference between the current scale to consider, and the next scale to consider, such that $\Delta s = 1$ if no scales

are skipped and $\Delta s > 1$ otherwise. For square regions, we then find that

$$\begin{aligned}\|R_{s,\mathbf{x}} \setminus R_{s-\Delta s,\mathbf{x}}\| &= \|R_{s,\mathbf{x}}\| - \|R_{s-\Delta s,\mathbf{x}}\| \\ &= 4s^2 - 4(s-\Delta s)^2 \\ &= 4(2s\Delta s - \Delta s^2),\end{aligned}$$

and thus

$$w(R_{s,\mathbf{x}}) = \frac{s^2}{\Delta s(2s-\Delta s)} \sum_{b=1}^{n} |p_n(R_{s,\mathbf{x}};b) - p_n(R_{s-\Delta s,\mathbf{x}};b)|. \tag{3.3}$$

Therefore, the self-similarity measure can be calculated easily as the sum of absolute differences of the gray level histograms of two square regions, weighted by a factor that depends on the scale parameter. Increasing Δs results in faster computation because a lower number of scales needs to be processed, whereas a decrease in Δs results in a more accurate scale detection.

In order to determine the best scale s of a region R located at coordinates $\mathbf{x}$, a set of scale hypotheses denoted $L_{\mathbf{x}}$ is selected, consisting of the local maxima in the entropy function (3.1). This subset is calculated as defined by equation (3.4), by varying the scale parameter s.

$$L_{\mathbf{x}} = \{s : h_n(R_{s-\Delta s,\mathbf{x}}) < h_n(R_{s,\mathbf{x}}) > h_n(R_{s+\Delta s,\mathbf{x}})\}. \tag{3.4}$$

For all scales in $L_{\mathbf{x}}$, the saliency $\mathrm{sal}(R_{s,\mathbf{x}})$ is then calculated by equation (3.5).

$$\mathrm{sal}(R_{s,\mathbf{x}}) = h_n(R_{s,\mathbf{x}})w(R_{s,\mathbf{x}}). \tag{3.5}$$

Finally, the optimal scale $\hat{s}$ of the region of interest is chosen as

$$\hat{s} = \arg\max_{s} \left(\mathrm{sal}(R_{s,\mathbf{x}})\right)$$

Empirically defined thresholds are used to efficiently discard locations with very low saliency or very high self-similarity. A relative threshold, defined as a fraction of the maximum saliency of all detected regions R in the image, is then used to further reduce the amount of regions of interest. Finally, non-maximal-suppression is applied by iteratively searching for the region with the highest saliency, and removing all other detected regions within its support.

In order to reduce the computational complexity, we set $\Delta s = 2$, such that only odd scales are considered. Furthermore, to avoid a time consuming histogram calculation at every possible scale and location in the image, we use integral histograms, which is made possible by our choice of square regions of interest as opposed to circular regions of interest.

Similar to the well known integral images, proposed by Viola and Jones [Viola 04], an integral histogram is a recursive propagation method to calculate a

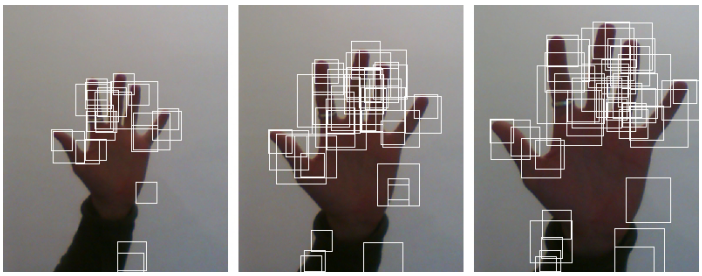


Figure 2.6: Feature detection: selecting image patches for hand detection.

cumulative histogram for each pixel in the image, by means of dynamic programming. At each pixel location, this cumulative histogram is the histogram of all pixels above and to the left of the pixel under consideration. Let $I_{\mathbf{l}}$ be the integral histogram at location $\mathbf{l} = (x, y)$. For each pixel location an integral histogram, represented by vectors of integers and with binsize n, is obtained using a single pass propagation method during a preprocessing phase.

For a square region $R_{s,\mathbf{x}}$, centered at location $\mathbf{x} = (x, y)$, let $\mathbf{a} = (x-s, y-s)$, $\mathbf{b} = (x - s, y + s)$, $\mathbf{c} = (x + s, y - s)$, $\mathbf{d} = (x + s, y + s)$ respectively be the top-left, bottom-left, top-right and bottom-right corner coordinates of the region. The histogram for this region can then be calculated easily as $p_n(R_{s,\mathbf{x}}) = I_{\mathbf{d}} + I_{\mathbf{a}} - I_{\mathbf{c}} - I_{\mathbf{b}}$.

For an n-bin histogram, this amounts in n additions and $2n$ subtractions to calculate a histogram of an arbitrary region in the image. However, the number of instructions needed can be reduced greatly by using SIMD instructions. If each bin is represented by a four-byte integer, then SSE2 (Streaming SIMD Extensions 2) instructions can be used on an IA-32 architecture to process four bins simultaneously, resulting in a total of $\frac{3}{4}n$ instead of $3n$ arithmetic instructions. On more recent CPUs that support AVX instructions, this is reduced further to $\frac{3}{8}n$ instructions, and on CPUs that support the AVX-512 extension, this reduces to $\frac{3}{16}n$. In the latter case, and if n is chosen to be $n = 16$, only three arithmetic instructions are needed to calculate the histogram of an arbitrary region in the image.

As a result, once the integral histogram is calculated during a preprocessing step, the histogram for every region, on any scale and any location in the image, can be obtained efficiently in constant time $O(1)$.

Figure 2.6 illustrates the result of the feature detector for different hand sizes. The detector clearly favors blob-like structures such as fingertips, and ignores regions with low entropy or high self-similarity. The scale of the resulting image patches varies, depending on the size of the object.

3.2.3 Region of Interest Description: Color Features

The appearance of each region of interest, often termed 'image patch' in the remainder of this chapter, is described using feature values that can be used as splitting criteria by the decision-nodes of the Random Hough Forest classifier.

Gall and Lempitsky [Gall 09] describe each image patch using 32 advanced color and gradient based features. These features, such as the Histograms of Oriented Gradients (HoG) [Dalal 05], are known to be highly discriminative, but are computationally expensive to calculate. Moreover, the features used by Gall and Lempitsky are not scale or rotation invariant.

Instead, due to our real-time constraints, we propose only seven very simple features, five of which use histograms to represent first order statistics, yielding a scale invariant, real-time detector. Furthermore, each of our features is made rotation invariant such that two regions from a different image, representing the same object part, are described with a similar feature descriptor, even if the object is rotated. This enables the detector to cope with the articulated nature and the large number of degrees of freedom of human hands.

Each image patch is resized to a fixed scale, and divided into a 3×3 grid. For each cell in this grid, five histograms are calculated. While histogram based features are inherently rotation and scale invariant locally, the grid based approach allows us to incorporate spatial information into the final descriptor. The cell ordering in the grid will be normalized by the dominant orientation in the image patch, as described in more detailed on page 40.

Three of the five histograms are color based, as skin color is one of the most distinctive features available for hand detection. The color of human skin is known to occupy a well defined region in color space [Gomez 02], often referred to as the 'skin locus'.

In earlier work however, we showed that a certain degree of illumination independence can be achieved by combining multiple color spaces [Spruyt 10a]. Additionaly, Gomez [Gomez 02] used a data analysis approach to show that the so called H-GY-Wr color space, a combination of components from multiple color spaces, yields the most stable results for skin detection. In this color space, a nearly convex area contains 97% of all skin colors, while only containing 5.16% of false positives.

In the H-GY-Wr color space, H stands for the hue, as used in the Hue-Saturation-Value (HSV) color space. In the HSV color space, the hue represents an angle, such that $H \in [0°, 360°]$. Within this interval, an increase in hue continuously corresponds to a color change from red ($0°$) to yellow ($60°$), to green ($120°$), to cyan ($180°$), to blue ($240°$), to magenta ($300°$), and back to red ($360°$).

Due to the angular representation of color, it is clear that the red range becomes discontinuous if the hue is described using a single numerical value. Since the red

range is the most prominent in skin-colored pixels, we therefore shift the hue into the range $-30^\circ < \tilde{H} < 330^\circ$, such that the red range becomes continuous.

The second component, GY, is simply obtained by subtracting the luminance of the image, defined as Y in the standard sRGB color space, from the green channel. The last component, Wr, is a non-linear combination of the original RGB channels, defined originally by Soriano *et al.* [Soriano 00]. Soriano *et al.* found that skin colored pixels, independently of the camera or illumination, generally fall within a small cluster in chromaticity space. This so called skin locus occupies only 3.2% of the color space, and its upper boundary is defined by Wr.

Thus, the three color features that are calculated for each image patch are simply normalized 1D histograms of 16 bins, accumulating the following color components:

$$\begin{cases} \tilde{H} = (H - 30) \bmod 360 \\ GY = G - (0.2126R + 0.7152G + 0.0722B) \\ Wr = \left(\frac{R}{R+G+B} - \frac{1}{3}\right)^2 + \left(\frac{G}{R+G+B} - \frac{1}{3}\right)^2 \end{cases} \tag{3.6}$$

Apart from the fact that these features are very simple to calculate, as opposed to the complicated HoG and gradient based features used by Gall and Lempitsky, a histogram based approach also greatly reduces the dimensionality of the problem and is inherently scale and rotation invariant. During training and classification of their random forest classifier, Gall and Lempitsky randomly select two pixel locations out of $16 \times 16 = 256$ possible locations in the image patch and compare these in order to split the training set at each decision-node. Instead, per image patch we simply select two random histogram bins from one of the histograms, and compare those, yielding a feature that does not depend on low level spatial information.

The previously described color histograms simply model the color of the image patch. These features can therefore be used for any object detection task, even though we carefully selected the color space to increase the discriminativeness of skin-like pixels.

Apart from these histogram based features, we introduce another color describing feature descriptor by specifically calculating the probability that a pixel in the image patch represents skin color. This feature then represents the average skin probability within each cell of the 3×3 grid in the image patch.

In order to reduce illumination dependence, most color based tracking and detection methods [Bretzner 02, Kölsch 04a, Caifeng 04, Shan 07, Asaari 10] convert the image from RGB color space to HSV (Hue-Saturation-Value) color space or to a similar color space such as YCbCr, in which the two chrominance components are separated from the luminance component. A 2D Hue-Saturation histogram is then created by dropping the luminance component, i.e. the Value, in the hope of reducing the illumination dependence.

Figure 2.7: Bayesian skin detector. Bright pixels correspond to a high skin probability, whereas dark pixels correspond to a low skin probability.

However, Phung *et al.* [Phung 05] showed that the choice of color space does not have a significant impact on skin color detection if this color space is a (piece-wise) linear transformation of the RGB space. In fact, skin classification in RGB space performs equally well as skin classification in HSV space, and outperforms color spaces in which the luminance information is dropped, if a large amount of training data is available. The reason is that most color transformations that perform a (piece-wise) linear mapping, do not introduce new information and do not guarantee uncorrelated color channels.

We used the publicly available Compaq dataset [Jones 02] to train a Bayesian skin classifier. The Compaq dataset contains over 3 000 manually segmented skin colored images and over 4 000 non-skin colored images. During training of the skin classifier, we simply model the likelihood $P(r, g, b|skin)$ of observing a certain RGB triplet in a skin region by means of a 32-bin 3D histogram of skin pixels. Similarly, the likelihood $P(r, g, b|\neg skin)$ of observing this color in a non-skin region, is modeled by means of a 32-bin 3D histogram of non-skin pixels. Furthermore, the prior probabilities $P(skin)$ and $P(\neg skin) = 1 - P(skin)$, can be obtained directly from the training data.

The posterior probability $P(skin|r, g, b)$ can then be obtained according to Bayes' rule:

$$\begin{aligned} P(skin|r,g,b) &= \frac{P(r,g,b|skin)P(skin)}{P(r,g,b)} \\ &= \frac{P(r,g,b|skin)P(skin)}{P(r,g,b|skin)P(skin) + P(r,g,b|\neg skin)P(\neg skin)} \end{aligned}$$

This probability is calculated for each pixel, as illustrated by figure 2.7. The average skin probabilities for each cell of the 3×3 grid of the image patch under consideration are concatenated into a feature vector. Therefore, the skin probability based feature vector simply contains the zero-order moments of the skin probability map.

3.2.4 Region of Interest Description: Texture Features

Although (skin) color is a general and discriminative feature that can quickly discard background objects during classification, it is also sensitive to illumination changes, and can not distinguish between hands and faces or skin-like colors in the environment. Therefore, next to the three color based histograms based on the feature values of equation (3.6), and the skin probability based feature, we also calculate two histogram based features that describe gradients and texture information.

An efficient way to describe local texture in an image is the Modified Census Transform (MCT) that was first used in the context of hand detection by Just *et al.*, as discussed in Section 2. Very similar to the MCT, only differing in the ordering of bits and the calculation of the average intensity in the window, are Local Binary Patterns (LBP) [Ojala 96]. The LBP operator was introduced approximately at the same time as the MCT operator and defines a very similar local structure feature. The advantage of the LBP operator however, is that it can be implemented efficiently using only bitwise operations, and that it can easily be modified to obtain the image gradients, as will be discussed in more detail on page 39.

LBP was originally defined as an ordered set of binary comparisons between a center pixel and each of its surrounding pixels, and was shown to have high discriminative power for texture classification problems [Ojala 96, Ojala 02]. For texture classification, LBPs of all pixels in a window can be calculated, and the resulting local binary patterns can be stored in a histogram. A disadvantage of the original LBP operator however, is that it yields an 8-bit descriptor and therefore, a 256-bin histogram. If only small image patches are to be classified, the well known 'curse of dimensionality' suggests the need for dimensionality reduction.

An extension to the original LBP operator, is the Center Symmetric Local Binary Pattern (CS-LBP) operator [Heikkilä 09], which has proven to outperform both the original LBP and the well known SIFT descriptor in various image recognition problems. The CS-LBP is obtained by using the sign of the difference between center-symmetric pairs of pixels, instead of the difference between each pixel and the central pixel. Let n_i be the grayscale value of the i^{th} pixel in a 3×3 neighborhood centered at location (x, y), where the ordering of $i \in [0, 8]$ is defined as shown by figure 2.9. Furthermore, let $1_{\mathbb{Z}^+}(.)$ be an indicator function that takes value 1 for all strictly positive integers and 0 for all others. The CS-LBP of the center pixel at (x, y) is then defined as

$$\text{CS-LBP}(\mathrm{x}, \mathrm{y}) = \sum_{i=0}^{3} 2^i \; 1_{\mathbb{Z}^+}(n_i - n_{i+4}). \tag{3.7}$$

Figure 2.8 illustrates the texture modeling capabilities of local binary patterns more clearly.

(a) Original image. (b) CS-LBP features.

Figure 2.8: For illustration purposes, each pixel in the above image is replaced by its CS-LBP value, after which the resulting image is equalized. For classification tasks, the distribution of CS-LBP values in a small region of interest is captured by a histogram which then serves as the feature vector.

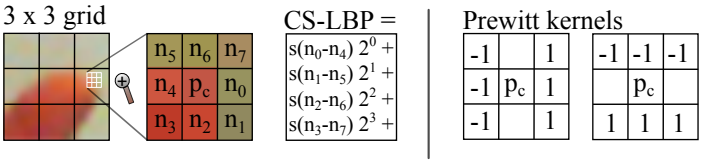


Figure 2.9: Comparison of the CS-LBP operator and the Prewitt operator.

As opposed to the original definition of Local Binary Patterns, the CS-LBP results in a value between 0 and $2^4 - 1 = 15$. Therefore, the CS-LBP values within a region can be summarized by a 16-bin histogram instead of a 256-bin histogram, which helps avoiding overfitting problems related to the curse of dimensionality.

Similar to the calculation of the color histograms, we compute a CS-LBP histogram for each of the cells in the 3×3 grid of the image patch under consideration. To further reduce the computational complexity of our method, the CS-LBP patterns are efficiently calculated using simple bit-wise operations; The sign-bits of the difference of two grayscale values can easily be extracted by masking the most significant bit of the integer, followed by bit-shifting. An additional bit-shift can then be used to shift the bit to the correct position, and a binary OR operation to combine all shifted sign-bits.

In order to ensure rotation invariance, we propose a variation on the CS-LBP operator, which we refer to as the CS-LBP^{R} operator. However, in order to obtain this rotation normalized variant, the dominant gradient orientation of the image patch needs to be computed. Since this orientation is automatically obtained during calculation of our fifth histogram based feature, yet to be discussed, it is useful to first describe this feature.

The sixth feature, used by our hand detector, is a simple orientation histogram. Orientation histograms and the closely related HoG features haven been proven to be powerful tools in gesture recognition [Mitra 07] and object classification in general [Gall 09], as they model structural and textural information. A normalized orientation histogram models the probability mass function of the gradient or edge orientations in the image patch. We propose an efficient method to obtain such an orientation histogram, by considering the close relation between the CS-LBP operator and the Prewitt gradient operator.

The Prewitt operator is a discrete differentiation operator that computes an approximation of the local gradient in an image. The kernels used to obtain the gradient in the horizontal and vertical directions are shown in figure 2.9, next to the CS-LBP definition. It is clear that the same calculations, needed to obtain the Prewitt gradients, are needed when calculating the CS-LBP values. Therefore, an approximation to the image gradients can be directly obtained from the CS-LBP calculation, without further performance loss.

Based on the image gradients ∇f_x and ∇f_y, a weighted orientation histogram that models $\theta = \arctan\left(\frac{\nabla f_y}{\nabla f_x}\right)$ is obtained. The histogram is weighted in the sense that each gradient orientation vote in the histogram is weighted by the gradient magnitude, such that the importance of the gradient is taken into account. Weighting ensures that strong, important gradients are not dominated by noise. Furthermore, orientations are quantized to 18 bins and are unsigned, which means that the orientation $\theta \in [0, \pi]$. Unsigned gradients are used as opposed to signed gradients, because the direction of the contrast in the image patch has no importance. In other words, the same orientation would be obtained whether the background has higher or lower intensity than the foreground. In the more advanced context of HoG features, Dalal and Triggs [Dalal 05] found that unsigned gradients were more discriminative than signed gradients, when applied to the task of person detection.

In order to obtain rotational invariance, the orientation histogram is normalized with respect to the dominant orientation in the image patch. To calculate this dominant orientation, an orientation histogram of the complete image patch is first obtained as the sum of the orientation histograms of each cell in the 3×3 grid of the image patch under consideration. The peak of this histogram can be considered the dominant orientation in the image patch. However, in order to obtain a more precise estimate of the dominant orientation, a parabola is fit to the peak of this histogram and its two closest neighbour bins.

Let $o_n(R_{s,\mathbf{x}}; b)$ be the n-bin orientation histogram of image patch $R_{s,\mathbf{x}}$, and let the peak of the histogram be found at bin number p, such that the maximum histogram value is $m = o_n(R_{s,\mathbf{x}}; p)$, and the values of its closest, equally spaced neighbor bins are $l = o_n(R_{s,\mathbf{x}}; p-1)$ and $r = o_n(R_{s,\mathbf{x}}; p+1)$. The peak of the fitted parabola is then located at

$$p' = p + \frac{r - l}{2(2m - l - r)}$$

The peak p' of the fitted parabola is then used as the dominant orientation of the image patch. This procedure of determining a precise estimate of the dominant orientation in a region of interest is very similar to the method used by the well known SIFT descriptor [Lowe 04].

The dominant orientation of the image patch is then used to normalize the feature histograms, in order to obtain rotation invariant descriptors. For each cell in the 3×3 grid of the image patch, the orientation histogram is shifted circularly, such that the dominant orientation always corresponds to the first histogram bin. This circular shift can be implemented efficiently and without memory copy by manipulation of the array indices. As a result, the final orientation histograms still represent the local structure of the gradients in that cell, but are invariant to global rotation of their context. This normalization process is illustrated by figure 2.10.

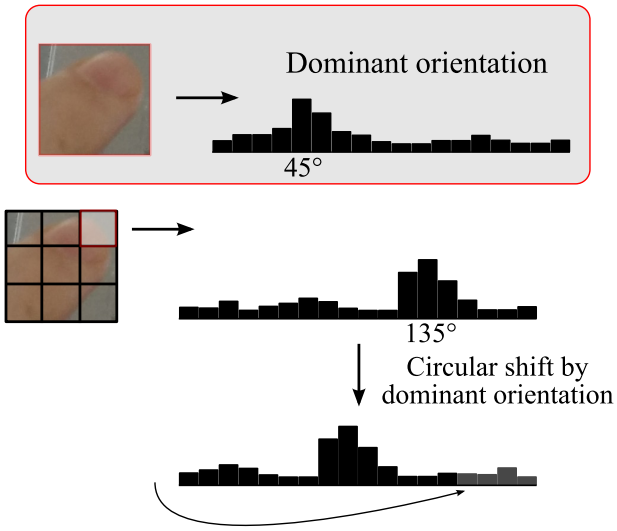


Figure 2.10: The dominant gradient orientation of the image patch is used to normalize the orientation histograms of all cells by circularly shifting these.

Similarly, the dominant orientation is used to rotate the original CS-LBP operator, shown in figure 2.9, to obtain a rotation invariant CS-LBP^{R} descriptor. The dominant orientation is first quantized into four directions, such that $\theta \in \left\{0, \frac{1}{4}\pi, \frac{2}{4}\pi, \frac{3}{4}\pi\right\}$. The CS-LBP patterns are then normalized with respect to θ. Due to the structure of Local Binary Patterns, this can be implemented efficiently by simply circularly bitwise left-shifting the original CS-LBP value.

After normalizing the CS-LBP patterns and the orientation histograms, each of the five histogram based features is locally rotation invariant. However, each histogram is calculated for a each cell in the 3×3 grid of the image patch under consideration. If the resulting histograms for each cell would then just be concatenated, the resulting feature vector would not be rotation invariant.

Therefore, the dominant orientation is also used to rotate the 3×3 grid of the image patch to achieve rotation invariance for each of the five histogram based descriptors. Instead of actually rotating the image patch itself, this can be implemented efficiently by rotating the indices or pointers used to address the feature vectors, as illustrated by figure 2.11.

Finally, apart from the previous three color based histogram features, the skin probability based feature, and the two texture describing histogram features, a seventh feature is used. This feature, namely the FREAK descriptor [Alahi 12], describes the global texture of the image patch whereas the earlier discussed CS-LBP and orientation histograms describe the local texture of the image patch.

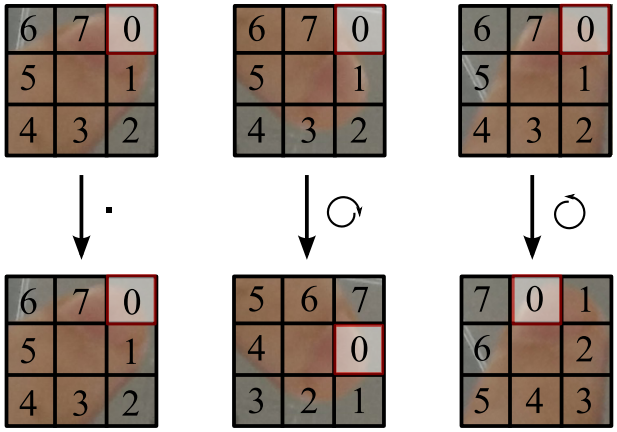


Figure 2.11: The dominant gradient orientation of the image patch is used to rotate the cell indices of the 3×3 grid, in order to achieve rotation invariance.

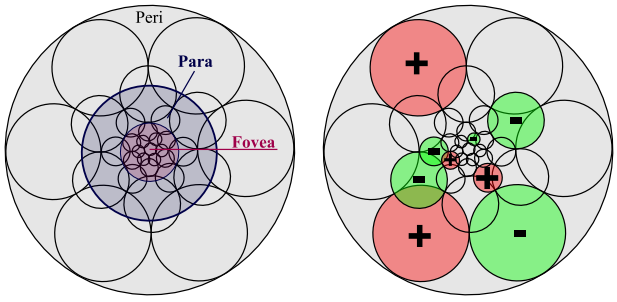


Figure 2.12: The FREAK descriptor mimics the human vision system by quantizing a series of Difference of Gaussians over a retinal sampling pattern to produce a bit string that represents the texture of the underlying scene.

The Fast Retina Keypoint (FREAK) descriptor is a cascade of binary strings, computed by efficiently comparing image intensities over a retinal sampling pattern [Alahi 12], as illustrated by figure 2.12. The FREAK descriptor could be considered an elaborate version of the LBP approach, in which pixels to be compared are sampled more densely around the center pixel, while the sampling density decreases exponentially farther away. Sampled pixels are smoothed using a Gaussian kernel, in which the variance depends on the distance from the center pixel. As a result, the FREAK descriptor mimics the human visual system, whose resolution drops exponentially towards the edges of the retina.

The FREAK descriptor has been shown to outperform the well known SIFT and SURF (Speeded Up Robust Features) descriptors, and is extremely fast to calculate. Furthermore, the resulting descriptor is a binary string, which can be easily

compared using the Hamming distance, instead of the more computationally expensive Euclidean distance. Finally, the 512-bit descriptor is inherently hierarchical, encoding the more fine grained spatial information in the most significant bits, while the least significant bits contain more coarse information.

By grouping the descriptor into sixteen 32-bit vectors, our Random Hough Forest detector can exploit this spatial topology. Whereas higher level decision-nodes in each decision tree prefer a splitting criterion based on more coarse textural information, the decision-nodes that are closer to the leaf-nodes will be able to use the vectors that represent fine grained textural information to further split the training dataset.

Since the FREAK descriptor itself is rotation and scale invariant and incorporates spatial information, we simply calculate the descriptor once per complete image patch, thus ignoring the earlier defined 3×3 grid for this feature type. The resulting descriptor therefore represents the global texture of the image patch. Moreover, we choose the size of the retinal sampling pattern such that it is larger than the image patch under consideration. This allows us to incorporate contextual information into the descriptor, such as the wrist, lower arm or body of the person.

Summarized, we propose to calculate the following seven features per image patch:

- Nine 16-bin $\tilde{H}$-histograms, encoding the shifted hue;
- Nine 16-bin GY-histograms, encoding the normalized green-channel;
- Nine 16-bin Wr-histograms, encoding the distance to the skin-locus;
- Nine 16-bin $\mathrm{CS\text{-}LBP}^{\mathrm{R}}$-histograms, encoding the local texture;
- Nine 18-bin orientation histograms, encoding the local gradients;
- Nine average skin probability values;
- Sixteen 32-bit binary FREAK strings, encoding the global structure of the image patch;

This carefully selected set of features is very simple to calculate, yielding a computationally efficient, yet robust, real-time classifier that is rotation and scale invariant. Furthermore, since in total 763 feature values per image patch are now available to the Random Hough Forest, random selection of a small subset of these features to be used in the decision-node of the trees, results in a low-variance, low-bias classifier.

Each histogram based feature is calculated per cell in the 3×3 grid. Therefore, for each arbitrary image patch R, which we will often refer to as a 'region' in the remainder of this chapter, the appearance vector $\mathbf{a}$ consists of the set of histogram

bins and skin probabilities from the earlier described features, as a function of the cell number, and by the set of FREAK strings calculated for the region.

A region is then modeled by its appearance vector $\mathbf{a}_R$, its class-label l_R, and several parameters that define the region's relative position to the centroid of the hand. These parameters, denoted $\mathbf{o}_R^n$, d_R^s, and α_R, will be defined in the next section. Each region R that is used to train the decision trees in the Random Hough Forest, is thus represented by its model D_R:

$$D_R = \{\mathbf{a}_R, \mathbf{o}_R^n, d_R^s, \alpha_R, l_R\}. \tag{3.8}$$

3.2.5 Random Hough Forest Training

The appearance vector $\mathbf{a}_R$ contains three parameters that describe the location of the image patch, relative to the centroid of the hand. In the next paragraphs, each of these parameters will be discussed in more detail.

Let $\mathbf{x} = (x, y)$ be the location of a region $R_{s,\mathbf{x}}$ in the image. Furthermore, let $\hat{\mathbf{x}} = (\hat{x}, \hat{y})$ be the centroid location of the hand in the image. We then denote $\mathbf{o}_R = \mathbf{x} - \hat{\mathbf{x}} = (\Delta x, \Delta y)$ the offset vector from the region's center to the hand's center. In the following, let superscript s indicate normalization by scale, and let superscript n indicate a complete normalization (by scale and by orientation).

Normalizing the offset vector by the region's scale, results in the scale invariant offset vector $\mathbf{o}_R^s = \mathbf{o}_R / s = (\Delta x^s, \Delta y^s)$. This normalization ensures that differently scaled image patches from different images that represent the same object part (e.g. a finger tip), correspond to the same offset vector.

To achieve rotational invariance, the offset vector is further normalized by the dominant orientation θ that was described in Section 3.2.3. Orientation normalization is performed by rotating the offset vector by θ in order to obtain the final normalized offset vector $\mathbf{o}_R^n = (\Delta x^n, \Delta y^n)$:

$$\begin{aligned} \Delta x^n &= \Delta x^s \cos(\theta) - \Delta y^s \sin(\theta) \\ \Delta y^n &= \Delta y^s \cos(\theta) + \Delta x^s \sin(\theta) \end{aligned}$$

This rotation normalization ensures that differently oriented image patches that represent the same object part (e.g. a finger tip), correspond to the same normalized offset vector.

The offset vector, $\mathbf{o}_R^n$ stores the scale and rotation invariant offset to the hand's centroid. Because of this scale and rotational invariance, regions with similar appearance and relative offset, but with a different orientation or scale, will end up in the same leaf-node of the decision tree. This allows the training process to make optimal use of the available training data, since fingertips or hand parts in different training images are correctly interpreted to be the same part of the hand, irrespective of their scaling and rotation.

However, due to the normalization of the offset vector, the discriminativeness of the classifier is reduced, since small errors in the estimation of the dominant orientation or the region's scale, would result in very different normalized offsets, yielding an incorrect vote at the classification stage. Therefore, apart from modeling the joint distribution of orientation and scale as a normalized offset value, we also model the marginals by storing the two variables that are used to obtain this normalization.

The first variable is the normalized distance, d_R^s, to the centroid of the hand. If we denote d_R the Euclidean distance from the region's centroid to the hand's centroid in the training image, then d_R^s represents this distance after normalization by the region's scale, s. This normalization allows us to store a relative distance instead of an absolute and thus scale dependent distance measure.

The second variable is the angular difference between the dominant orientation θ, and the offset vector $\mathbf{o}_R$, and is calculated as

$$\alpha_R = \arccos\left(\frac{\mathbf{o}_R}{\|\mathbf{o}_R\|} \cdot \mathbf{u}_\theta\right)$$

where $\mathbf{u}_\theta$ is the unit vector with orientation θ, and $\cdot$ denotes the dot-product.

Assuming independence between d_R^s and α_R, their likelihoods can be used to avoid the curse of dimensionality during classification stage, while at the same time reducing the influence of noisy gradients during calculation of the dominant orientation, as will be described on page 50.

Once a descriptor D_R has been calculated for each image patch, actual training of the Random Hough Forest is started. During training, a set of tests is randomly selected at each decision-node. From this set, the test that results in an optimal split of the training data is then picked as the actual splitting criterion. Instead of defining a test as the comparison of a feature value with a threshold, we define it as a decision rule that compares the difference between a feature values at two distinct locations in the image patch, with a threshold value τ. This approach implicitly incorporates spatial information into the test. Depending on the sign of this test, the image patch is passed to either the left, or the right child-node in the tree. All tests in the random set are evaluated, and the best test is used to define the splitting rule for this decision-node.

In Section 3.2.3, we defined seven feature descriptors that can be categorised into three feature types, i.e. histogram based features, a skin probability based feature, and a FREAK descriptor based feature. Each binary test in the set of randomly selected tests, corresponds to a comparison of the difference of one of these feature descriptors at two distinct locations in the image patch, with a threshold τ. This threshold value is discussed in more detail on page 46.

For each of the three feature types, a different type of binary test is needed. Tests that use histogram based features, $\mathrm{T}_{\mathrm{h_n}}(.)$, are defined by two cell numbers,

c_1 and c_2 and a bin number b. Each cell number points to a cell of the 3×3 grid in the image patch. Let $h_n(R_{s,\mathbf{x}}; c, b)$ represent the b^{th} bin of the n-bin histogram of the chosen feature descriptor, calculated at cell c. The histogram based test is then defined as:

$$\mathrm{T}_{\mathrm{h_n}}(\mathbf{a}_R; c_1, c_2, b, \tau) = \begin{cases} 0, & \text{if } h_n(R_{s,\mathbf{x}}; c_1, b) < h_n(R_{s,\mathbf{x}}; c_2, b) + \tau \\ 1, & \text{otherwise} \end{cases}$$

Tests that use the skin probability, $\mathrm{T}_{\mathrm{skin}}(.)$, are only defined by two cell numbers, c_1 and c_2. Let $\mathrm{skin(c)}$ represent the average skin probability in cell c of the 3×3 grid of the image patch. The skin probability based test is then defined as:

$$\mathrm{T}_{\mathrm{skin}}(\mathbf{a}_R; c_1, c_2, \tau) = \begin{cases} 0, & \text{if } \mathrm{skin}(c_1) < \mathrm{skin}(c_2) + \tau \\ 1, & \text{otherwise} \end{cases}$$

Finally, tests that use the FREAK based feature, $\mathrm{T}_{\mathrm{FREAK}}(.)$, are simply defined by a substring u, which represents one of the sixteen available bit strings of length 32. Let $\mathrm{d}(u_1, u_2)$ be the Hamming distance between the binary strings u_1 and u_2. For simplicity, we simply compute the Hamming distance between the FREAK string u, and the 0 value, which corresponds to a population count of the number of ones in the FREAK string. This can be implemented efficiently by means of bitwise operations, using a SWAR (SIMD With A Register) algorithm [Fisher 03]. The FREAK based test is then defined as:

$$\mathrm{T}_{\mathrm{freak}}(\mathbf{a}_R; u, \tau) = \begin{cases} 0, & \text{if } \mathrm{d}(u, 0) < \tau \\ 1, & \text{otherwise} \end{cases}$$

The handicap value τ is randomly generated by sampling from a uniform distribution, whose lower and upper bound are determined by the minimum and maximum values of the chosen feature descriptor, in all image patches from the training set at that decision-node. In our implementation, we generate 200 random tests at each decision-node. Given the large number of feature values and parameters, this is only a fraction of the amount of tests that is actually available, ensuring a low-bias classifier.

In order to select the best test out of the set of all randomly selected tests, a selection criterion is defined. As discussed in Section 3.1.3, the training phase of a Random Hough Forest tries to optimize two criteria. The first criterion is chosen to maximize the class separability, in order to separate hand patches from background regions, and therefore minimizes the 'class-label uncertainty'. The second criterion is chosen to maximize the similarity between image patches, in terms of their offset vector, and therefore minimizes the 'offset uncertainty'.

At each decision-node, one of both optimization criteria is randomly selected with equal probability, and the test, from the set of random tests, that minimizes the criterion is used as a splitting rule. Gall and Lempitsky use the information

gain as a measure for the class-label uncertainty. Information gain was originally introduced as a splitting criterion for decision trees in the ID3 algorithm [Quinlan 86]. The information gain is defined as the difference of entropy before and after a training set is split by a certain decision rule. In other words, maximizing the information gain, reduces the class-label uncertainty of the training data.

However, it has been found that the information gain is biased in favor of those splits that have the largest number of successors in the decision tree. Consider the extreme example where each sample of a training set has a unique identifier. If this identifier is used as a feature in the splitting criterion, the training data would be partitioned such that each partition contains only one training tuple. Therefore, the entropy of the class labels at each leaf-node would be zero, after the data has been split. The difference between the entropy before and after splitting, and thus the information gain, would thus be maximized by choosing this unique ID as a splitting criterion. However, it is clear that this splitting criterion is nonsensical and leads to large decision trees that overfit the data.

The information gain is therefore strongly affected by the number of possible outcomes of a test. The normalized information gain compensates for this by normalizing by a uniformity measure of the distributions of the split data, as shown by Wehenkel and Pavella [Wehenkel 91]. The normalized information gain is also the criterion used in the C4.5 decision tree algorithm [Quinlan 93], which is the successor of the ID3 algorithm. Therefore, we propose to use the normalized information gain instead of the standard information gain, as a measure of class label uncertainty.

The offset uncertainty is a measure of the variability of the offset vectors in the training data subset. The leaf-nodes in a Hough forest represent code books for parts of the object. Therefore, image patches in those leaf-nodes should preferably represent the same object part. Minimizing the offset uncertainty at the decision-nodes, results in codebook clustering towards the leaf-nodes. Similar to Gall and Lempitsky, we calculate the offset uncertainty for each subset S_i of training data, after a split, as a function of the difference between the offset vector of an image patch, and the average offset vector in the subset:

$$U_i = \sum_{R \in S_i} \alpha(\bar{\mathbf{o}}^n - \mathbf{o}_R^n)^2 + \beta(\bar{d}^s - d_R^s)^2 + \gamma(\bar{\alpha} - \alpha_R)^2$$

where α, β, γ are normalization factors.

The total offset uncertainty is then defined as the sum of the offset uncertainties in the individual subsets: $U = U_1 + U_2$. At each decision-node, either the offset uncertainty or the class-label uncertainty is chosen to be minimized, and the test, from the random subset of available tests, that corresponds to the best split is chosen as the final splitting criterion.

During training, each tree $\mathcal{T}_j$ in the random forest is grown without pruning. However, if a maximum depth is reached ($d = 20$), or if the size of the subset S_i

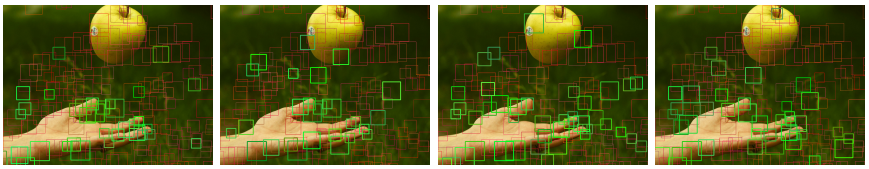


Figure 2.13: Image patch classification by four different decision trees in the random forest. Hand patches are shown in green, whereas background patches are shown in red. The brightness of the image patch region boundaries are randomized only for visualization purposes and serve no further meaning.

in a node is very small ($|S_i| = 20$), or if all samples in the subset of the node share the same class label, the node is declared a leaf-node. Each leaf-node stores the region's information D_R for positive image patches, and maintains statistics about the number of positive and negative patches that reached the node.

3.2.6 Random Hough Forest Classification

During classification, each image patch $R_{s,\mathbf{x}}$ that is returned by the region of interest detector, is classified by the Random Hough Forest as being part of a hand, or part of the background, by propagating the image patch through the decision trees until a leaf-node is reached. For a given decision tree $\mathcal{T}_j$, the probability of the image patch being part of a hand ($l_R = 1$), given its appearance $\mathbf{a}_{R_{s,\mathbf{x}}}$, can be estimated by:

$$P(l_{R_{s,\mathbf{x}}} = 1 | \mathbf{a}_{R_{s,\mathbf{x}}}; \mathcal{T}_j) = \frac{|\{R^j : l_{R^j} = 1\}|}{|\{R^j : l_{R^j} = 1\}| + \lambda |\{R^j : l_{R^j} \neq 1\}|}, \tag{3.9}$$

where $\{R^j\}$ is the complete set of regions that are stored in the leaf-node of tree $\mathcal{T}_j$. The scaling factor λ is the ratio of positive and negative patches in the original training dataset and ensures a correct weighting in case the original training set is highly unbalanced.

For illustration purposes, the probability $P(l_{R_{s,\mathbf{x}}} = 1 | \mathbf{a}_{R_{s,\mathbf{x}}}; \mathcal{T}_j)$ was calculated for each region of interest of figure 2.13, and for four different decision trees of our trained random forest. Regions of interest are indicated by a green box if $P(l_{R_{s,\mathbf{x}}} = 1 | \mathbf{a}_{R_{s,\mathbf{x}}}; \mathcal{T}_j) \geq 0.5$, and by a red box otherwise. This example clearly shows that each individual decision tree makes slightly different decisions and mistakes, due to it being trained using a bootstrapped data sample, and random features.

However, in order to detect hands in an arbitrary image, knowing whether or not an image patch is part of a hand does not suffice. Instead, the centroid location of the hand, which the image patch is part of, needs to be determined. To estimate the hand location, we need to obtain an estimate of the probability that a hand is

centered at each possible position $\mathbf{c}$ in the image. By searching for the maxima in this distribution, the hand locations can then be found. The next paragraphs discuss how this probability distribution is obtained.

As discussed in Section 3.2.5, a region $R_{s,\mathbf{x}}$ is defined by its model D_R. After propagating the image patch through a decision tree of the random forest, only one parameter of this model is known, namely the region's appearance vector $\mathbf{a}_R$.

Therefore, the probability of observing a hand at position $\mathbf{c}$, given a region's appearance vector $\mathbf{a}_R$, corresponds to the probability that the unknown parameters of D_R take on values that correspond with a hand being centered at position $\mathbf{c}$, conditioned on the region's appearance vector.

In the case that $R_{s,\mathbf{x}}$ is part of a hand centered at $\mathbf{c}$, the class label for this region $l_R = 1$ by definition. Furthermore, let the real, unknown normalized offset vector from the region to the hand centroid be $\hat{\mathbf{o}}_R^n$. Similarly, let the normalized distance be $\hat{d}_R^s$, and let the angular difference be $\hat{\alpha}_R$.

The likelihood that the appearance of region $R_{s,\mathbf{x}}$ corresponds with a hand being positioned at $\mathbf{c}$, can then be written as $p(\mathbf{o}_R^n = \hat{\mathbf{o}}_R^n, d_R^s = \hat{d}_R^s, \alpha_R = \hat{\alpha}_R, l_R = 1|\mathbf{a}_R; \mathcal{T}_j)$ for each decision tree $\mathcal{T}_j$ in the random forest. In the following, we omit the random variable names, to simplify the notations. For each decision tree, the evidence we are interested in, is then given by:

$$p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_R; \mathcal{T}_j). \tag{3.10}$$

Although it is not directly obvious how to estimate this probability density from the training data, equation (3.10) can be rewritten as follows:

$$p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_R; \mathcal{T}_j) = p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R|\mathbf{a}_R, l_R = 1; \mathcal{T}_j) p(l_R = 1|\mathbf{a}_R; \mathcal{T}_j). \tag{3.11}$$

The last factor, $p(l_R = 1|\mathbf{a}_R; \mathcal{T}_j)$, represents the probability that the image patch is part of a hand, and can easily be calculated from the leaf-node statistics as we defined earlier in equation (3.9).

The first factor represents the joint distribution of the normalized offset vector, the normalized distance from image patch to hand centroid, and the angular difference between the dominant orientation of the image patch, and the vector from image patch to hand centroid. As discussed in Section 3.2.5, these parameters depend on the scale of the region, and the dominant orientation of the image patch.

If we now assume independence between the scale estimate and the orientation estimate, we can further simplify the first factor as follows:

$$\begin{aligned} &p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R|\mathbf{a}_R, l_R = 1; \mathcal{T}_j) \\ &\quad = p(\hat{\mathbf{o}}_R^n|\mathbf{a}_R, l_R = 1; \mathcal{T}_j) p(\hat{d}_R^s|\mathbf{a}_R, l_R = 1; \mathcal{T}_j) p(\hat{\alpha}_R|\mathbf{a}_R, l_R = 1; \mathcal{T}_j) \end{aligned} \tag{3.12}$$

In equation (3.12), the probability $p(\hat{\mathbf{o}}_R^n|\mathbf{a}_R, l_R = 1; \mathcal{T}_j)$ simply corresponds to the percentage of offset vectors stored in the leaf-node that are equal to offset vector $\hat{\mathbf{o}}_R^n$. The probabilities $p(\hat{d}_R^s|\mathbf{a}_R, l_R = 1; \mathcal{T}_j)$ and $p(\hat{\alpha}_R|\mathbf{a}_R, l_R = 1; \mathcal{T}_j)$ can be extracted directly from the statistics in the leaf-node in a similar manner. During training, a Parzen window density estimation is computed for each of these probabilities, at the leaf-nodes of each decision tree. During classification, these densities are simply evaluated, yielding the desired likelihood.

While the assumption of independence in equation 3.12 is not necessarily correct, this assumption leads to a computationally efficient approximation of the desired posterior. Furthermore, the independence assumption allows us to deal with marginal likelihoods, instead of complicated joint probability distributions. As a result, the curse of dimensionality and thereby the problem of overfitting, is largely avoided. It is a well known fact in pattern recognition that a strong, possibly incorrect assumption of independence, yielding a naive Bayesian classifier, often outperforms a Bayesian classifier that tries to model the joint probability distributions [Friedman 97]. The reason is that the assumption of independence greatly simplifies decision boundaries and therefore avoids overfitting. Since a naive model can be described with much less parameters than a complicated joint distribution, the curse of dimensionality is avoided.

To aggregate the probabilistic votes coming from each tree, we marginalize equation (3.12) over all n trees in the random forest $\mathcal{F} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n\}$:

$$\begin{aligned} p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_R) &= \sum_{j \in \mathcal{F}} p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_R; \mathcal{T}_j) p(\mathcal{T}_j) \\ &= \frac{1}{n} \sum_{j \in \mathcal{F}} p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_R; \mathcal{T}_j). \end{aligned} \tag{3.13}$$

Finally, let $A = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_p\}$ be the set of appearance vectors of all q detected image patches R. To integrate the votes coming from the q different patches, we assume that all possible patches have been detected such that the elements of A are collectively exhaustive. Furthermore, we assume a uniform prior, such that each element of A has equal probability of occurrence. To integrate the votes coming from the different patches, we then marginalize equation (3.13) over all q patches:

$$\begin{aligned} p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1) &= \sum_{j \in A} p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_j) p(\mathbf{a}_j) \\ &= \frac{1}{q} \sum_{j \in A} p(\hat{\mathbf{o}}_R^n, \hat{d}_R^s, \hat{\alpha}_R, l_R = 1|\mathbf{a}_j). \end{aligned} \tag{3.14}$$

For each possible (x, y) location in the image, the probability defined by equation (3.14) can be calculated, resulting in a so called Hough accumulator image. By finding the local maxima in this image, the hands are then detected.

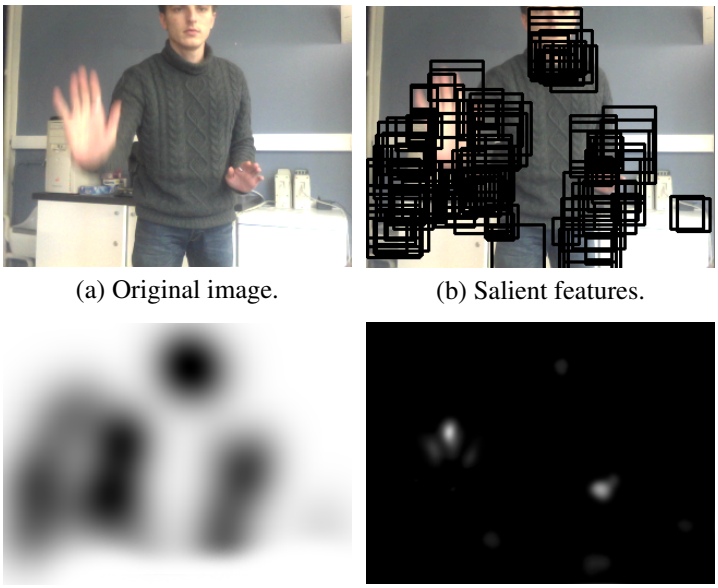
In practice however, calculating the Hough image as suggested by equation (3.14) would be very time consuming, since it would require the algorithm to iterate over all possible (x, y) locations, while calculating the normalized offset vector towards each image patch, and then computing the resulting probabilities. If the image patches were sampled on a regular grid, similar to the approach of Gall and Lempitsky, the same result, apart from a multiplicative constant, could be obtained by simply iterating over the training patches in the leaf-node of each tree that was reached after classification, casting a probabilistic vote for the hand centroid, and postprocessing the resulting Hough image by means of a Gaussian smoothing filter.

However, as opposed to the approach used by Gall and Lempitsky, our image patches are the result of a region of interest detector which only returns high entropy features. Therefore, parts of the image will contain a lot of regions of interest, while others will only have a few. If we let each region of interest cast a probabilistic vote, maxima in the Hough image would be biased towards image locations surrounded by many regions of interest. The reason for this is that these locations have a higher prior probability of obtaining a vote than locations with only few regions of interest in their neighbourhood.

Therefore, the Hough image needs to be normalized to account for the spatial distribution of detected regions of interest. The spatial distribution of votes v, cast by the image patches, can be modeled by a Gaussian mixture model, representing $P(v|R_0, R_1, ..., R_p)$, where each component of the mixture model is centered on the centroid of an image patch R_i that is returned by the feature detector. As a result, centroid locations close to an image patch have an exponentially higher probability of obtaining a vote than centroid locations farther away. Similarly, centroid locations surrounded by many regions of interest have an exponentially higher probability of obtaining a vote than centroid locations that are close to only a few image patches.

The variance of the Gaussian components is inversely proportional to the scale of the image patch, as large image patches usually contain a larger part of the object than small image patches. To normalize the Hough voting map, we simply multiply each probability as calculated by equation (3.14) by the spatial distribution of expected voting behavior, $P(\neg v|R_0, R_1, ..., R_p)$. This is illustrated in figure 2.14 (c), while figure 2.14 (d) shows the resulting Hough voting map. Figure 2.14 (b) shows the top 20% salient image regions, returned by the region of interest detector.

The mean-shift algorithm is used to detect local maxima in the Hough image, which are then considered to be detected hand locations. However, apart from binary classification in static images, the Hough image can easily be integrated into a probabilistic tracking method for video, due to its probabilistic nature. Since the classifier is trained offline, the resulting Hough image can be considered an



(a) Original image. (b) Salient features.

(c) Normalization. (d) Hough voting map.

Figure 2.14: Illustration of the Hough voting process.

appearance based prior to a probabilistic tracking algorithm such as a Kalman filter or particle filter.

However, instead of considering the Hough image as a static, appearance based prior, we propose to dynamically adapt this prior, based on past detections and misclassifications in the video. In a specific video sequence, it is likely that some leaf-nodes of the decision trees are reached more frequently than others. For instance, the appearance of objects in the background would not change very often, resulting in the same image patches to be classified or misclassified in every subsequent video frame.

This temporal consistency can then be used to adapt the random forest classifier to its specific environment. If several image patches are misclassified consistently over a period of time, our Random Hough Forest classifier can learn and adapt, in order to avoid these mistakes in subsequent video frames. To detect misclassifications however, an error detection procedure is needed. As will be explained in Section 3.3, we augment the detector with a second spatial, and a third temporal classifier in order to remove false positive hand detections. Therefore, at each time instance, we can feedback this knowledge about false positive detections to the Hough forest in order to avoid future mistakes.

To automatically adapt the Hough classifier based on this feedback, statistics are kept in each leaf-node about the number of times T_p the node was reached by

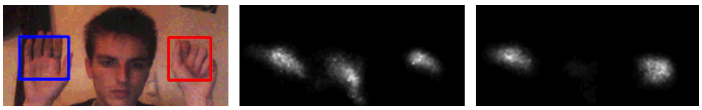


Figure 2.15: Online learning by the Random Hough Forest. The second image shows the Hough voting map in the first video frame, whereas the third image shows the voting map several seconds later. Note that no face detection algorithms were used: the Random Forest classifier automatically adapts, based on information from the second and third stage classifiers.

a positive patch, and the number of times T_n this node was reached by a negative patch. A positive patch is a patch that is classified as being part of a hand by each of the three stages of our classification system, and whose Hough probability is higher than a certain threshold δ ($\delta = 0.5$).

We define a context $\mathcal{C}$ as the set of all hand positions that have been encountered up till now, in the current video sequence. During classification of an image patch, we then consider the Hough probability resulting from equation (3.14) as the prior probability that the image patch is part of this context C. Bayesian theory then allows us to obtain the posterior probability by multiplying this prior with the likelihood, shown by equation (3.15).

$$P(R \in \mathcal{C} | \mathbf{a}_R, l_R = 1) = \frac{T_p}{T_p + \gamma T_n}, \tag{3.15}$$

where γ is the ratio of the total number of positive patches and the total number of negative patches, detected over time.

Figure 2.15 illustrates the effect of this online learning behavior of the Random Hough Forest. The first Hough image shown is the result after only a few seconds and clearly assigns a high hand probability to the face region. However, this false positive detection is corrected by the second and third stage classifiers, after which this information is fed back to the Hough detector. The second Hough image is the result several seconds later, and shows that the Hough detector learned to assign a lower probability to the face region, avoiding similar mistakes in this context.

3.3 Three-stage Classification

In the previous Section, we described an efficient and robust hand detector, based on Random Hough Forest classification. This detector is a high recall detector, which means that it can positively detect the occurrence of a human hand in an image most of the time. However, as a result, the detector yields a low precision, which means that only a fraction of the hand detections are in fact hands.

Similar to the approach described by Mittal *et al.*, we propose a cascade of classifiers to solve this problem. The first classifier in the cascade is the earlier

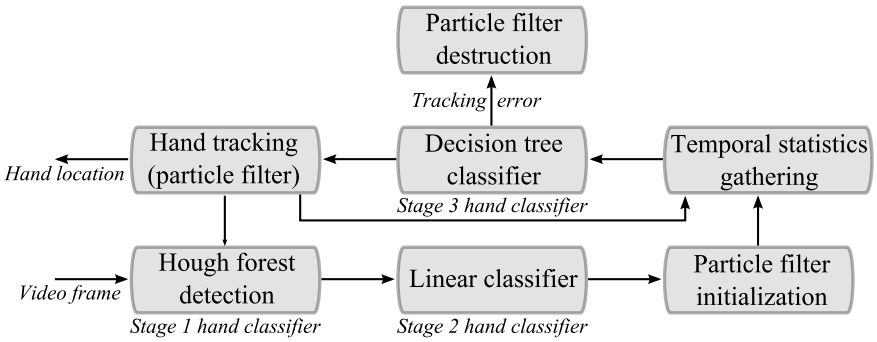


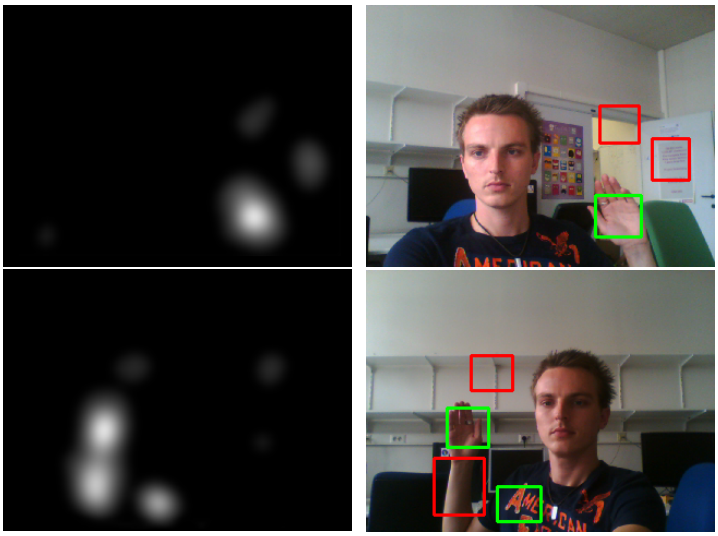
Figure 2.16: Schematic representation of the cascaded hand classification and tracking algorithm with automatic initialization and error recovery.

described Random Hough Forest classifier that solves the most difficult problem of generating high quality hand location hypotheses. The second classifier in the cascade is a simple linear classifier that quickly rejects a large amount of false positive detections by classifying the resulting hand bounding box as a whole, instead of classifying the parts that make up the hand. The third classifier in the cascade is a decision tree that uses only temporal information to further reject false positive detections in video tracking applications.

Cascading an advanced high recall classifier with two simple high precision classifiers results in a fast and robust classifier that can be used for tracker initialization and error recovery. Furthermore, results from each of the cascades can be fed back such that the previous detector can learn from its past mistakes. The next paragraphs explain each of the cascade stages, shown in figure 2.16

The Random Hough Forest classifier returns a probability map that, for each pixel, indicates the probability of being the centroid of a hand. In order to actually extract possible hand locations from this voting map, the local maxima need to be located. The variance or support of each local maximum, indicates the size of the hand. To quickly obtain a set of hand hypotheses, we use the camshift procedure [Exner 10] to obtain the location and size of each hand hypothesis. Camshift is an adapted version of the meanshift method, a discrete and iterative optimization algorithm. By using a rectangular window with uniform kernel, in combination with integral images, a camshift iteration can be executed extremely fast and with constant O(1) complexity.

The Random Hough Forest classifies image patches instead of images. Therefore, if several parts of an object have a similar appearance and spatial configuration as hand parts, a local maximum is generated in the voting map, resulting in a false positive detection. To quickly discard obvious false positives, we trained a simple classifier using linear discriminant analysis (LDA), such that the precision of the resulting classifier is higher than the precision of the Random Hough Forest



(a) Hough voting map (d) Hand detection

Figure 2.17: A second stage linear classifier rejects most of the false positive hand hypotheses (shown in red) that were generated by the first stage Random Hough Forest detector.

classifier. The linear classifier classifies hypothesized hand bounding boxes instead of image parts, and simply reuses the earlier calculated features for performance reasons. An example of the filtering capabilities of the second stage classifiers is shown in figure 2.17, where the red rectangles represent hand hypotheses that were rejected by the linear classifier.

The resulting two stage classifier is able to robustly detect hands in images captured in unconstrained environments and is independent of the hand pose, scale or rotation. However, for video sequences, even a high precision in individual video frames would lead to many false positive detections per second, due to the high framerates used nowadays. Therefore, we use a third stage classifier that further reduces the amount of false positive detections in case of video sequences.

This third stage classifier only uses features that describe the temporal behavior of the detected object. In order to gather this temporal information, for each of the hypotheses that result from the second stage classifier, a hypothesis verifying particle filter is used. This particle filter does not update any of the color models and does not assume to be tracking a real hand. Instead, it is only used as an auxiliary background process, to gather temporal information describing the tracked object, which can then be uses by the third stage classifier to verify or reject the hand hypothesis.

While the first and second stage classifier only use spatial information to detect hand hypotheses and to reject false positive detections, the third stage classifier exploits the available temporal information to further refine the set of hand hypotheses. The third stage classifier is a simple decision tree that was trained using 10-fold cross validation, and pruned afterwards. This decision tree evaluates wether an object is a hand, by examining its temporal behavior during the past N frames.

The hypothesis verifying particle filter that is used to gather temporal information is based on the method that we described earlier in [Spruyt 13b] and will be discussed in chapter 4. Once a hand has been detected by our two-stage classifier, the tracker is initialized to start tracking the hand hypothesis in the background, while gathering four distinct temporal statistics using a temporal sliding window. At each incoming video frame, the third stage decision tree classifier decides whether or not the tracked object actually is a hand. The resulting decision can then be used to initialize the final hand tracking algorithm, or to detect tracking errors and drift. Furthermore, this decision is fed back to the first stage classifier to dynamically adapt the Hough classifier to the current context.

The next paragraphs explain the four types of temporal features that are used by the third stage classifier. These features have been carefully selected based on both their discriminativeness and their low computational complexity.

A first feature that discriminates correctly tracked hands from incorrectly tracked objects, is a measure of the variance and covariance of the estimated (x, y) positions in the recent past. A small variance often indicates a stable track, while a large variance often indicates a tracking error that causes sudden, large changes in the tracked state. In order to transform this covariance matrix into a single feature value, we calculate its Frobenius norm, which is often used as a measure to compare covariance matrices. For an $m \times n$ matrix C with m rows and n columns, consisting of real elements c_{ij}, the Frobenius norm $\|C\|_F$ is defined as follows:

$$\|C\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|c_{ij}|^2} = \sqrt{\operatorname{trace}(C^\intercal C)}\,.$$

This measure allows us to efficiently combine the variances and covariances into a single feature value, thereby reducing the risk of overfitting as compared to using each matrix element as a distinct feature.

A second feature that could indicate tracking problems, is the uncertainty of the tracking algorithm itself. In case of a particle filter, this uncertainty is defined by its covariance matrix. This covariance matrix defines the spatial distribution of the particle estimates. Thus, whereas the first feature measures the temporal variance of the mode of this distribution, the second feature measures its current spatial variance. If particles are widely spread, the observations during the past few

frames were either very ambiguous, or very unlikely. Similar to the first feature, we combine all elements of this covariance matrix into a single feature value, by calculating its Frobenius norm.

The third feature that is used to detect incorrectly tracked hands, is the average amount of motion during the recent past. Motion is detected by means of frame differencing, followed by Gaussian smoothing. If the amount of motion within the tracking window is very small for a long period of time, the tracked object might be a background object instead of a hand. Furthermore, if the amount of motion, detected by frame differencing, is not proportional to the first feature value that describes the spread of the particle filter's expectation over time, there might be a tracking error.

Finally, **the fourth feature** used by the decision tree, is simply the classification consistency of the first two stages of our hand detector in the recent past. If a certain hand hypothesis is being detected consistently in every video frame, its likelihood to be an actual hand is larger than if the hypothesis is only generated sporadically.

Temporal statistics are gathered by each hypothesis verifying particle filter using a sliding window of size N, such that the classifier's decision at each point in time is based on the past N frames. If the third stage classifier decides that a hypothesis, resulting from the second stage classifier, really is a hand, the hypothesis is considered verified, and actual hand tracking is started. During hand tracking, the third stage classifier continues to verify the validity of the tracked object in each new video frame, in order to detect tracking errors or drift.

This approach allows for automatic initialization and error recovery, which is of great importance in real life applications. Finally, the tracked bounding box is fed back to the Random Hough Forest classifier. The classifier can then adapt itself to the context in which it is used as shown by equation (3.15) and explained earlier. Figure 2.16 schematically illustrates this complete process, which is repeated for every video frame.

4 Results and Discussion

In this Section, we evaluate both our two-stage detector on static images, and our three-stage detector on video sequences, by comparing the results with two state-of-the-art object detection methods.

Impressive hand detection results were recently obtained by Mittal *et al.* [Mittal 11] using a parts based deformable model. Similar to our solution, their classifier was specifically designed for the difficult task of unconstrained hand detection and is therefore used as a benchmark. We present our results on several publicly available and widely used datasets.

Furthermore, we compare our results with the results obtained by the original Random Hough Forest classifier proposed by Gall and Lempitsky [Gall 09], since our method is partially based on several concepts described in their paper. Apart from the hand detection results, we also show how our classifier performs when trained for general object detection tasks, such as person detection and horse detection. Although our classifier was specifically designed for real-time hand detection, testing its general classification capabilities allows for a fair comparison with the Gall and Lempitsky algorithm, and can serve as an indicator for future possibilities with the proposed techniques.

The main goal of our work was not to design a classifier that outperforms the state-of-the-art in terms of detection rate. Instead, our goal was to design a classifier that can be executed in real-time, without sacrificing too much precision and recall. In this section, we show that our proposed method achieves this goal and as a by-product often even outperforms the state-of-the-art in terms of detection rate, for the specific task of hand detection.

Mittal *et al.* [Mittal 11] constructed a comprehensive dataset of hand images that were collected from various publicly available sources. This dataset was made available for research purposes, together with its manually created ground truth annotations. We trained our Random Hough Forest classifier using the 4071 training images in this dataset. Additionally, 1397 images from different publicly available datasets were used to represent the non-hand class during training. Performance of our classifier is evaluated by classifying the 827 test images from the Mittal dataset.

During training, 48 213 positive patches and 50 368 negative patches were extracted, and used to train 15 decision trees. Each decision tree was grown without pruning until a maximum depth of $d = 20$ was reached. As described in the previous sections, a set of features is selected randomly at each decision-node, after which the feature that results in the best split is chosen from this set. Figure 2.18 illustrates the percentage of decision-nodes in the forest that use a certain feature.

This clearly illustrates the randomness during training, resulting in approximately half of the decision-nodes using color based features, and half of the decision-nodes using texture based features. On the other hand, certain features such as the orientation histograms and the non-linear color component, obviously are much more discriminative than features such as skin probability, when describing a hand's appearance.

To evaluate a classifier's performance, the Pascal VOC (Visual Object Classes) score [Everingham 10] is widely used in literature [Gall 09, Mittal 11]. The VOC score defines the amount of overlap between the detected bounding box, resulting from the classifier, and the given ground truth bounding box. The score can be calculated as

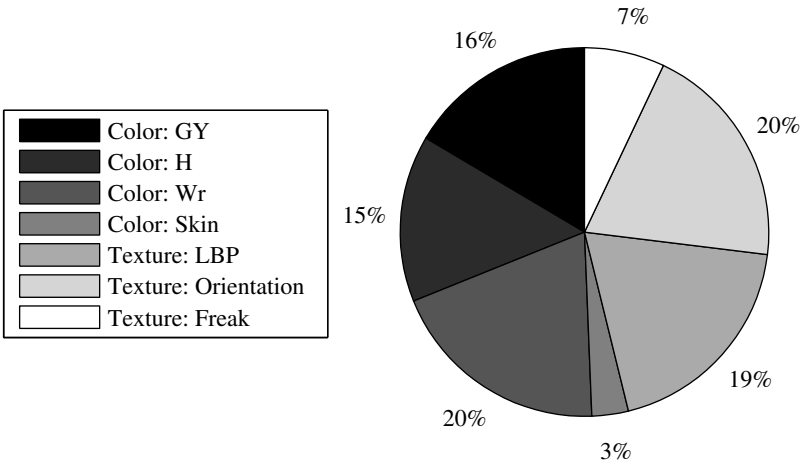


Figure 2.18: Distribution of the feature types that are used by the decision-nodes of the random forest.

$$VOC = \frac{B_g \cap B_d}{B_g \cup B_d},$$

where B_g represents the pixels within the ground truth bounding box, whereas B_d represents the pixels within the detected bounding box. Only those detections for which $VOC \geq 0.5$ are then considered *true positive* detections, while the rest are considered *false positive* detections.

While the VOC criterion allows for a standardized comparison of research results, it should be noted that a VOC score lower than 0.5 is often still very useful for practical applications. Moreover, as the VOC score depends on the bounding box definition, its quality inherently depends on the quality of the ground truth annotations. For illustration purposes, figure 2.19 shows several results that were obtained using our two-stage classifier, while evaluating the Mittal hand dataset. For each bounding box, the VOC score is listed, and boxes that correspond to a VOC score lower than 0.5 are shown in red.

We trained the classifier of Gall and Lempitsky [Gall 09], whose source code is publicly available, using the Mittal dataset in order to compare their results with ours. During evaluation, an image pyramid of three scales is used, as described in their paper, in order to allow for their algorithm to detect objects of different scale.

Mittal *et al.* report the average precision, i.e. the area under the recall-precision curve, together with their maximum recall, as a measure of their classifier quality. Table 2.1 compares these statistics with the results obtained using our classifier and

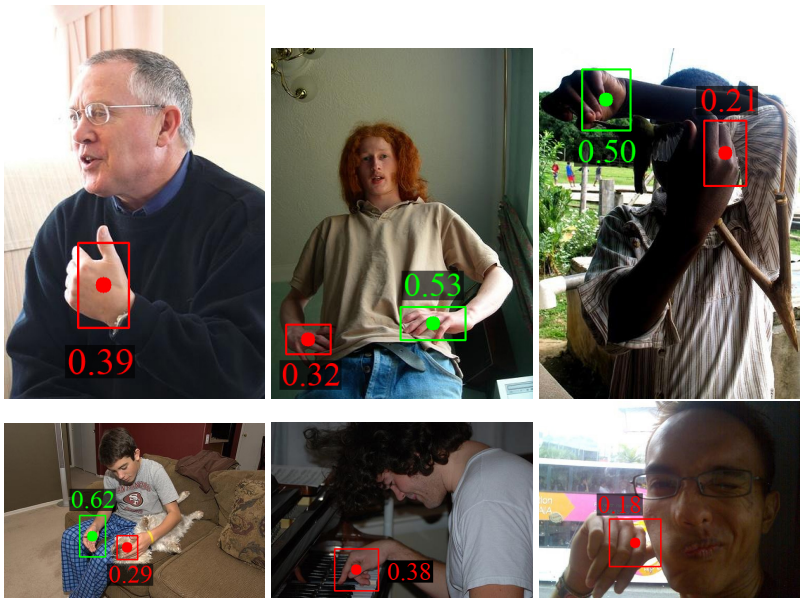


Figure 2.19: Hand detection results with a VOC score lower than 0.5 are shown in red. Detections with a VOC score higher than or equal to 0.5 are shown in green.

Table 2.1: Performance comparison for the Mittal dataset. Precision and recall are listed as percentages and detections are considered true positives if $VOC \geq 0.5$.

	Gall & Lempitsky	Mittal	Ours
Average Precision	41.02	48.20	**62.18**
Maximum Recall	79.36	85.30	**92.83**

with the results obtained using the Gall and Lempitsky algorithm, using $VOC \geq 0.5$ as an evaluation metric for each detection.

While our method seems to outperform the others, we believe that average precision in combination with maximum recall is not a very informative way of describing classification results. Instead, it is interesting to see how a classifier's precision changes when the classifier is configured to yield a certain recall. Figure 2.20 shows the recall-precision curves for each of the three algorithms.

We also show what the precision-recall curve would look like if a VOC score of 0.25 were considered to be acceptable, as will often be the case in real-life applications. This clearly shows that the method proposed by Mittal *et al.* outperforms the algorithm of Gall and Lempitsky when applied to the task of hand detection. This result is to be expected, since the classifier described by Gall and Lempitsky is a general purpose classifier, while the method described by Mittal *et al.* was

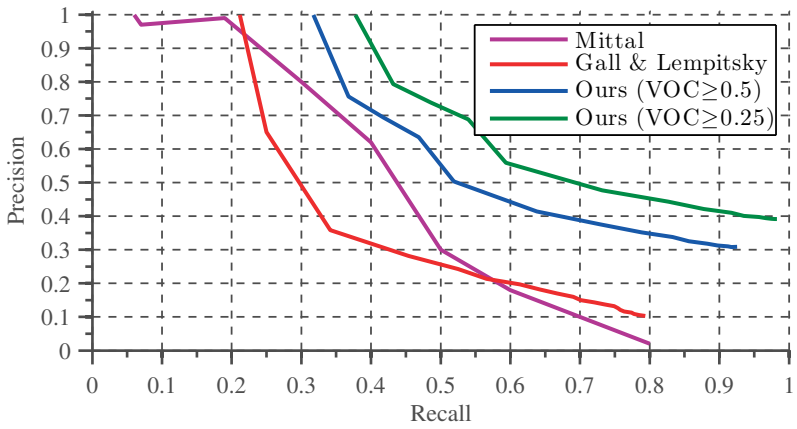


Figure 2.20: Recall-Precision curves for hand detection results on the Mittal dataset. Different recall values are obtained by varying the classifier's certainty threshold, i.e. moving the decision boundaries towards the negative class instances.

specifically designed for hand detection. Furthermore, this result shows that our proposed solution performs slightly better than the Mittal detector.

Figure 2.21 shows how the precision increases if the VOC threshold is lowered, for several fixed recall values. If a VOC threshold of 0.5 is used, a precision of 55% is obtained if the recall is 50%. For practical applications however, a VOC threshold of 0.25 can be used, as discussed earlier. In this case, a precision of 55% corresponds to a recall that is larger than 60%.

Figure 2.22 illustrates the increase in precision if the number of decision trees in the Random Forest is increased. The fluctuations in the curve can be explained by the high variance of decision tree classifiers. However, the more trees are in the forest, the less pronounced the fluctuations become. This can be explained by the variance-bias tradeoff that was discussed in Section 3.1.2. Due to the bagging principle, an overall increase in precision is observed when adding trees to the forest. Therefore, this graph clearly illustrates the strengths of a Random Forest classifier as opposed to a decision tree classifier.

However, since each decision tree is used to cast a probabilistic vote for each image patch, increasing the number of decision trees also increases the computational complexity of the classifier. Since the precision seems to converge when at least 15 decision trees are used, this is the value we chose in our configuration, as a trade-off between execution time and classifier precision.

Figure 2.23 shows several of the results obtained with our two-stage classifier. True positive detections are shown in green, while their corresponding ground truth annotation is shown in blue. False positive detections are shown in red.

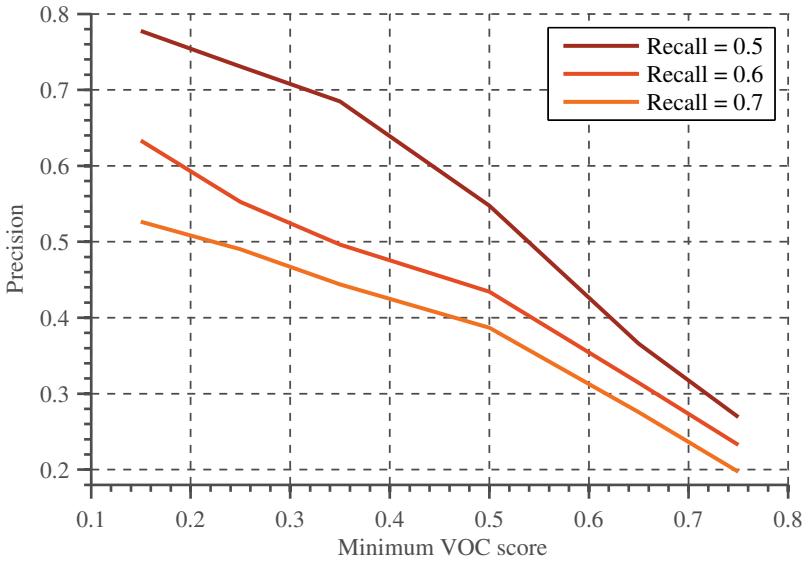


Figure 2.21: Precision values for a specific Recall value, corresponding to a minimum VOC score on the Mittal dataset. A detection is only considered to be a true positive detection, if its VOC score is equal to or greater than this minimum VOC score.

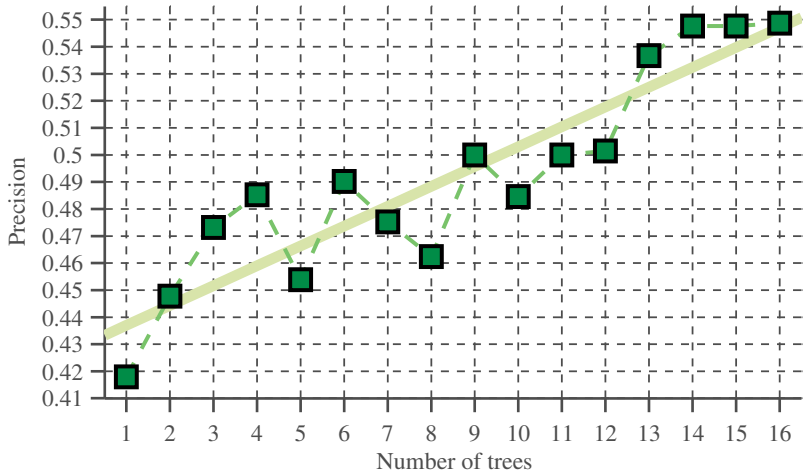


Figure 2.22: Precision of the classifier versus the number of decision trees used during classification of the Mittal dataset, corresponding to a fixed recall of 0.5.

Figure 2.23: True positive hand detections are shown in green. False positive detections are shown in red. Ground truth bounding boxes are shown in blue.

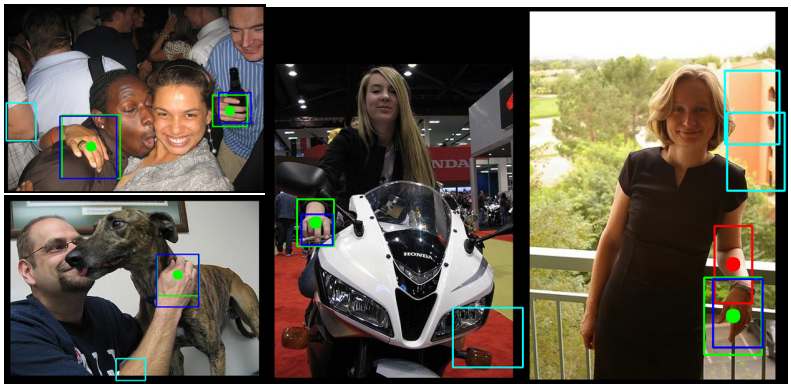


Figure 2.24: False detections, resulting from the Random Hough Forest detector, that were successfully rejected by the second stage linear classifier are shown in cyan.

Table 2.2: Average execution time in seconds, for a 320×240 image.

	Gall & Lempitsky	Mittal	Ours
Execution time (s)	19.5	40.0	**0.0248**

Figure 2.24 shows some intermediate detection results, where true positive detections that result from the first stage Random Hough Forest, but that were reclassified as being false positive by the second stage linear classifier, are shown in cyan.

In figure 2.25, the performance of our two-stage classifier is plotted against the number of pixels in an image. This graph shows that the computational complexity of our classifier is linear in the number of pixels. Also, it can be seen that small image sizes such as QQVGA and QVGA can easily be processed in real-time, allowing us to use the classifier in a hand tracking setup, where the classifier is responsible for tracker initialization and error detection.

Table 2.2 shows the average execution time for each of the evaluated algorithms on a QVGA image, which is often used in real-time video streaming applications. The results were obtained by classifying a modern quadcore 2.10Ghz i7 laptop with 8GB of memory.

In order to evaluate our complete, three-stage classifier described earlier, we tested our classifier using the Signers dataset [Buehler 08] that contains five BBC news sequences. We compare our results on this Signers dataset, with the results obtained by both Mittal *et al.* and Karlinsky *et al.* [Karlinsky 10].

Karlinsky *et al.* fit a chain model to the body of the signer in order to detect hands. Chain fitting starts from the ground truth bounding box of the face and thus only works if such ground truth is available and if the face is visible. The

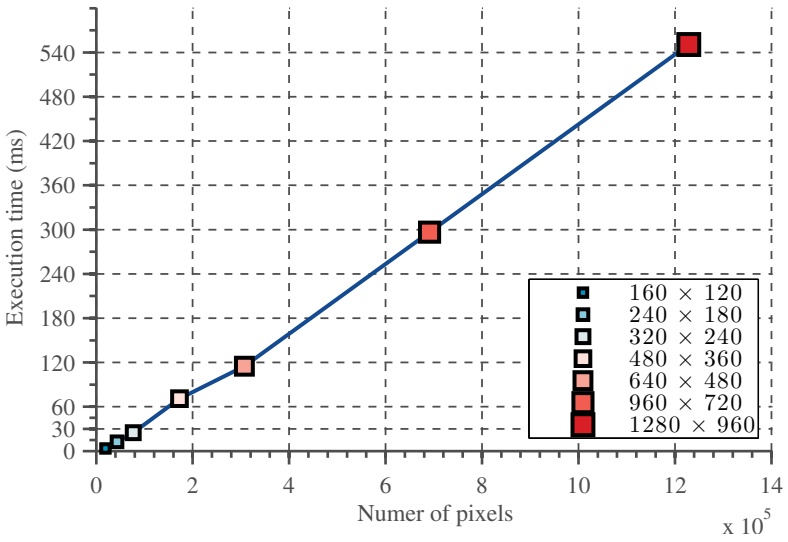


Figure 2.25: Execution time of our classifier, plotted against the number of pixels in the image.

Table 2.3: Results obtained on the Signer dataset, reported as a percentage by counting the number of times the correct detection was amongst the k-highest-confidence detections in each image.

	2 max	3 max	4 max
Mittal	90.0	95.6	97.4
Karlinsky	**92.8**	95.4	96.7
Ours (2-stage)	87.5	**96.1**	**97.3**
Ours (3-stage)	**98.7**	**98.7**	**98.7**

detected hand is considered to be correct if it is within half face width from the ground truth location of the hand. Detection performance is reported within the top k detections per ground truth hand instance. The main disadvantage of this technique is that the initial head position has to be known, and that the technique is rather computationally expensive and thus not suitable for real-time applications.

Table 2.3 shows a comparison of our results with the results obtained by Mittal *et al.* and those of Karlinsky *et al.*. These results show that our two-stage classifier slightly outperforms the state-of-the-art when $k > 2$. If $k < 2$ however, our method performs slightly worse. This can be explained by the fact that Karlinksky and Mittal perform face detection in order to eliminate false positives due to the face of the signer. In our case, the signer's face is sometimes detected as a hand with a strong Hough score, and is therefore misclassified. Even when this happens,

if k becomes larger, the second and third best detections are always the hands, which explains the better performance for $k \geq 3$.

To test our tree-stage classifier, temporal information needs to be available to the classifier. Therefore, we use the particle filter tracker that we described in an earlier publication [Spruyt 13b] and that will be discussed in chapter 4. While using this tracking algorithm was convenient to the authors, any tracking method, such as Camshift, Kalman filtering, Flocks of Features tracking and others, could be used instead. The goal of the tracking algorithm is simply to gather the necessary statistics to be used by the third stage decision tree classifier.

The third stage classifier is then responsible for making the final decision about promoting a detection hypothesis from the second stage classifier, to a final detection. Similarly, the third stage classifier is responsible for demoting an erroneous detection from the previous stages to a false positive detection. Table 2.3 shows that the addition of temporal information greatly increases detection performance. Simultaneously, this test case illustrates that a cascade of relatively weak spatial classifiers and a weak temporal classifier, can together become a robust solution for automatic filter initialization and error recovery.

Figure 2.26 shows the average execution time of each of the components of our three-stage classifier. The first stage classifier, solving the difficult task of generating hand location hypotheses consume 91% of the total detection time. The second and third stage classifiers simply need to reject false positive detections, resulting in an execution time that is negligible.

Figure 2.27 shows several classification results for the Signer dataset. Boxes that are colored cyan are early stage detections that were rejected by the second or third stage classifier.

Finally, it is interesting to evaluate our algorithm for general object detection tasks, not related to hand detection. Since our method is largely inspired by the work of Gall and Lempitsky, we evaluate our classifier on the same data they used. Note, however, that our classifier was specifically designed for hand detection, both by choice of the region of interest detector, and by choice of the feature descriptors (e.g. skin probability).

We trained our two-stage classifier on the Weizmann Horses dataset [Borenstein 02], containing side-views of horses in their natural environment. Similar to Gall and Lempitsky, we split the dataset such that 100 positive images were used for training, and 228 positive images were used for testing. As opposed to Gall and Lempitsky, we did not make special adjustments to obtain a more precise center of mass annotation before training. Also, we did not apply boosting to improve recognition rate of some of the decision trees. Figure 2.28 shows several detection results from our classifier.

We compare our results, obtained on the Weizmann horse dataset, with the method of Gall and Lempitsky, and with the classifier that was proposed by Shot-

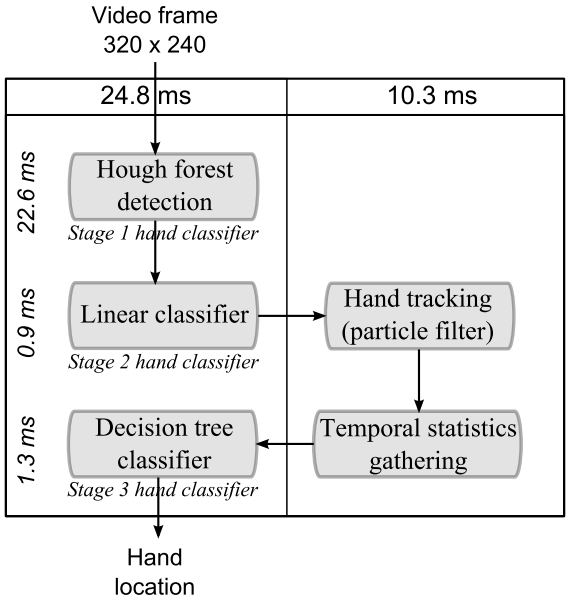


Figure 2.26: Execution timings of each component of the three-stage hand detector, obtained by averaging the timing results of 5000 video frames of size 320×240.

Figure 2.27: Detection results for the Signer dataset. True positive detections are shown in green. False positive detections are shown in red. Cyan boxes indicate false positive detections that were eliminated by the second and third stage classifier.

Figure 2.28: Detection results for the Weizmann horse dataset.

ton *et al.* [Shotton 08]. Figure 2.29 shows the recall-precision curves of these algorithms on the Weizmann Horses dataset. It is clear that our algorithm, while being outperformed by the state of the art methods, still is competitive, especially when taken into consideration that our classifier operates in real-time, and uses features such as skin probability and color components that were selected specifically for the task of hand detection, instead of more general features such as the well known HoG descriptors.

Moreover, our classifier is both rotation and scale invariant, while the Weizmann dataset only contains horses that are facing the left hand side of the picture frame. Since adding rotation invariance decreases the classifier's discriminative power, it is expected that non-invariant classifiers easily perform better on such data.

Finally, for the same reasons given above, we trained our classifier on the Inria Person database. Figure 2.30 shows several results obtained by our two-stage classifier.

Gall and Lempitsky evaluate their detection on the Inria dataset by counting the number of false positives per window (FPPW). This is a well known way of presenting results and is especially interesting for sliding window based classifiers. For comparison purposes, we report the same information in figure 2.31.

It is clear that our classifier has a lower maximum recall than the results obtained by Gall and Lempitsky on this dataset. This can be explained by the fact

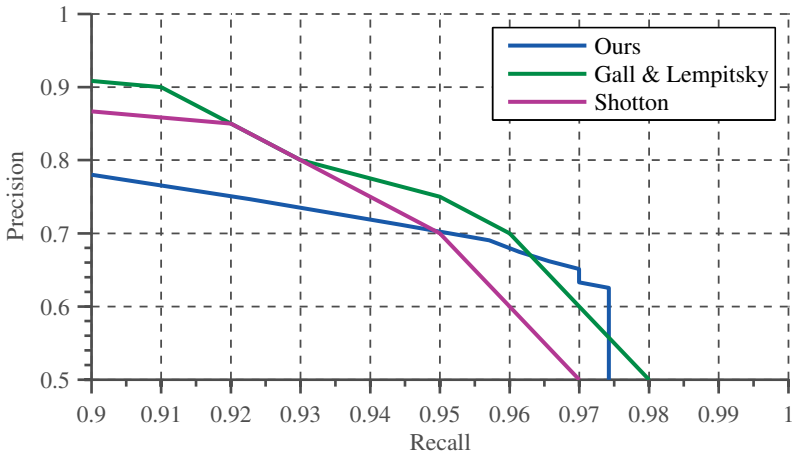


Figure 2.29: Recall-Precision curves of several algorithm when evaluated against the Weizmann Horses dataset.

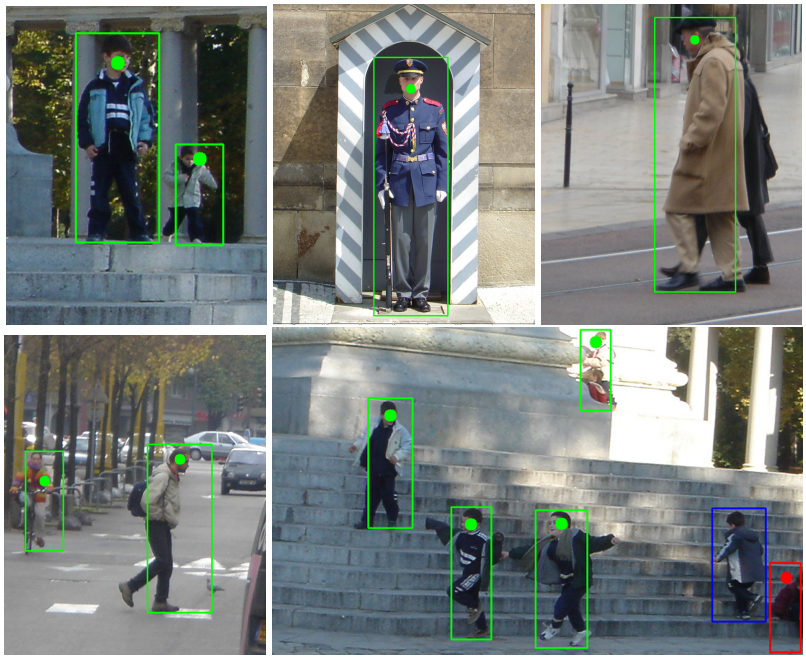


Figure 2.30: True positive detections are shown in green. False positive detections are shown in red. False negatives are shown in blue.

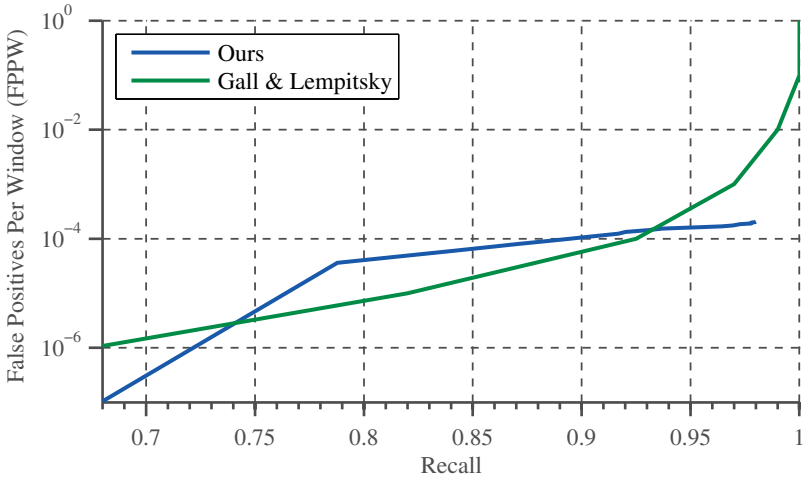


Figure 2.31: FPPW curves when evaluated against the Inria dataset.

that we incorporated scale and rotation invariance into the classifier, and only use a few computationally efficient features. For mostly symmetric objects such as people silhouettes, rotation and scale invariance is less important than for greatly deformable objects such as human hands.

5 Conclusion

We presented a real-time hand detector, capable of detecting human hands in static images, and we introduced a cascade system that incorporates temporal information to even further improve detection results in case of video sequences. Our method is based on incremental research and therefore combines several well known techniques such as the generalized Hough transform, random forest classification, and parts based object detection.

The resulting detector was compared with two state-of-the-art object detection methods on a large, publicly available dataset. We showed that our detector outperforms these methods when applied to the task of hand detection. This is of great importance because of the rising demand for interactive applications and real-time, robust hand detection and tracking.

Furthermore, we showed that our proposed solution could be of use when solving a more general object detection problem: When classifying highly articulated persons of the Inria dataset, and size varying horses of the Weizmann dataset, our method yields results that are competitive with state-of-the art classifiers, while providing a decrease in computational complexity with factor 1 000.

Finally, we presented an extensive evaluation of the computational complexity of our method which was shown to be linear in the number of pixels of the image. The resulting system operates in real-time, processing 40 images of size 320×240 per second.

3

Sparse, Regularized Optical Flow

1 Introduction

In chapter 2, we proposed a novel method to detect hands in static images. We showed that this method can be used to initialize a hand tracking algorithm, and to automatically detect tracking failures. The hand tracking algorithm used to test the detector's capabilities will be discussed in more detail in chapter 4. An important feature that is used by this tracking algorithm, is the optical flow field of the video frame. In this chapter, we present a novel method to obtain such optical flow field in real-time, thereby bridging the gap between chapter 2 and chapter 4. For an overview of the relations between the components, introduced in each chapter, please refer to figure 1.1.

Optical flow is defined as a 2D vector field describing the apparent velocities of objects in a 3D scene, projected onto the image plane. Most techniques to calculate such flow fields rely on the so called brightness constancy assumption, assuming that the intensity of an object remains constant between two frames, despite its movement. In the following, let $I(x, y)$ be the pixel intensity of image I at position (x, y), and let $(\Delta x, \Delta y)$ and Δt respectively represent the motion offset and time difference between subsequent video frames. The brightness constancy constraint is then given by:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{1.1}$$

If Δx, Δy and Δt are small, equation (1.1) can be written by its approximated Taylor expansion:

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t. \quad (1.2)$$

Therefore, equation (1.1) only holds if

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \quad (1.3)$$

$$\Leftrightarrow \frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} = -\frac{\partial I}{\partial t}. \quad (1.4)$$

Equation (1.4), contains the optical flow-vector components $\frac{\Delta x}{\Delta t}$ and $\frac{\Delta y}{\Delta t}$ as its two unknowns and is often called the optical flow equation. The optical flow equation represents an underdetermined system and is therefore an ill-posed problem. This is often referred to as the *aperture problem* which can be understood intuitively by considering the horizontal motion of a horizontal edge in the image, or the vertical motion of a vertical edge.

To solve the aperture problem, optical flow estimation methods need to introduce an additional constraint. The first optical flow estimation method in literature was proposed by Lucas and Kanade [Lucas 81] and is still widely used in the video processing domain due to its computational simplicity. Their method assumes that the flow-field is constant within a small neighborhood of the pixel under consideration. Under this assumption, equation (1.4) should hold for each pixel within this neighborhood, resulting in an overdetermined system that can be solved by the least-squares method. Furthermore, to relax the local linearity assumption of the Taylor expansion in equation (1.1), a pyramidal approach is often used such that large displacements can still be estimated reliably.

However, in uniform regions of the image, where the horizontal and vertical gradients are zero, the Lucas-Kanade method can not provide optical flow information. Therefore, the Lucas-Kanade algorithm is mostly used to calculate optical flow only at corner points, where the flow equations are well defined. The resulting flow field is then sparse, and algorithms computing such flow field are called *sparse optical flow algorithms*.

On the other hand, methods that are able to estimate the flow field at each point in the image are called *dense optical flow algorithms*. The first dense optical flow method described in literature, was proposed by Horn and Schunck [Horn 81]. In their method, the aperture problem is overcome by introducing a smoothness constraint as a regularization term instead of assuming a locally constant flow-field. The optical flow equation shown by (1.4) is extended by a regularization term, and the optimal flow estimate is obtained by minimizing this cost function. In the following, let $(I_x, I_y, I_t) \doteq (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}, \frac{\partial I}{\partial t})$, and let $(V_x, V_y) \doteq (\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$. The

optical flow field is then obtained by Horn and Schunk by minimizing a global energy functional:

$$\underset{V_x,V_y}{\operatorname{argmin}} \int\int (I_x V_x + I_y V_y + I_t)^2 + \alpha^2(\|\Delta V_x\|^2 + \|\Delta V_y\|^2)dxdy. \qquad (1.5)$$

where α is a constant that defines the regularization importance. Therefore, the Horn and Schunck method tries to minimize both the optical flow equation, and the gradients of the flow vectors, resulting in a smooth flow-field. This smoothness constraint effectively regularizes the optimization process, allowing the calculation of flow-vectors at each image point. The main disadvantage of the Horn and Schunk approach however, is that estimated flow-vectors at motion discontinuities are unreliable because of the global regularization term.

Several solutions [Xu 12, Schikora 11, Kim 13, Zhuoyuan 13] have been proposed in literature to solve the above problem, and most of these solutions only differ in the way the flow-field is regularized. However, all of these methods focus on the estimation of a dense flow-field. Dense optical flow estimation is of interest in many applications such as depth-from-motion, stereo-matching or image segmentation. On the other hand, for applications such as object tracking, the goal is usually to obtain an estimate of the motion direction and velocity of the tracked object, often represented by a bounding box or contour descriptor. In this case, a dense optical flow field does not add much value compared to a sparse optical flow field.

Furthermore, since most dense optical flow estimation techniques depend on computationally expensive optimization algorithms, these can often not be used directly for real-time applications. Although GPU implementations can reduce the computational cost, for practical tracking applications the optical flow estimation method should only consume a fraction of the available computational power. Therefore, many tracking algorithms resort to simple sparse flow-field estimators such as the Lucas-Kanade algorithm.

A major disadvantage of sparse optical flow methods, is the lack of regularization, such that the resulting flow-field is often noisy and less reliable than dense flow estimates. In this chapter, we propose an extremely efficient real-time sparse optical flow estimation method that incorporates a regularization term in its cost function, resulting in a smooth flow-field estimate. We introduce a method to regularize a sparse flow-field on a non-regular grid while preserving motion boundaries and discontinuities. The proposed algorithm combines the best of both worlds by yielding computation times that are lower than the sparse Lucas-Kanade flow algorithm and a flow estimation accuracy that is competitive with the state-of-the-art dense optical-flow algorithms.

This chapter is outlined as follows: In Section 2 we describe related work. Section 3.1 proposes an efficient feature detection and description method, resulting

in a sparse set of regions of interest, which are used to find point matches in the previous video frame. In Section 3.2 we propose an iterative regularization procedure that efficiently regularizes the obtained flow vectors in a local neighborhood. Finally, Section 4 provides a thorough evaluation and discussion of the detection results, using several publicly available datasets.

2 Related Work

The original method of Horn and Schunck suffers from the fact that its quadratic regularization term, imposing a smoothness constraint, does not allow abrupt changes in the final flow-field and can therefore not cope with motion discontinuities at the borders of moving objects in the scene. Furthermore, the L_2 norm that is used as a regularizer is not robust to outliers, resulting in a noisy flow-field.

Many authors have proposed enhancements to the work of Horn and Schunck to solve these problems. Aubert *et al.* [Aubert 99] replaced the L_2 data fidelity term by an L_1 term to increase robustness to outliers, and used several discontinuity preserving regularization terms to cope with motion discontinuities. Based on this work, Papenberg *et al.* [Papenberg 06] proposed a differentiable approximation of the L_1 norm as a regularization term, and suggested an iterative optimization scheme. The L_1 norm of the gradient, i.e. $\int |\nabla f(x)| dx$ instead of $\int \|\nabla f(x)\|^2 dx$, essentially counts the amount of variation in the data without regard to the smoothness of this variation. This norm is therefore called the total variation norm of $f(x)$, and optical flow estimation methods that use the L_1 norm as a regularizer are referred to as Total Variation (TV) minimizing optical flow algorithms. In contrast with the quadratic regularization method used by Horn and Schunck, TV-based algorithms are no longer biased towards smooth flow-fields.

However, another disadvantage of the original flow estimation method of Horn and Schunck, is its reliance on the Taylor expansion of the otherwise non-linear image intensity function $I(x + V_x, y + V_y)$. Local linearization converts the general non-convex optimization problem to a convex problem. As a result, optical flow estimates are only reliable for very small displacements for which the linear approximation is valid. To be able to cope with large object displacements, a scale-space approach is usually employed [Alvarez 99, Brox 04] such that flow estimates are obtained using a coarse-to-fine strategy. A well known publicly available dataset that is used as the de facto standard for evaluating and comparing optical flow algorithms in the computer vision literature, is the Middlebury dataset [Baker 11]. Almost all top-ranked methods on the widely referred Middlebury website use the earlier described Total Variation minimization approach, combined with a coarse-to-fine search strategy.

Although a coarse-to-fine search works well if foreground and background pixels exhibit similar motion vectors and if the moving objects are large, it tends

to fail otherwise. Small foreground object such as the finger of a human hand, tend to disappear in coarse approximations of the image. As a result, the coarse flow estimate that is used to initialize the minimization procedure at the finer level is completely based on the background flow. This can cause a violation of the linearity assumption, resulting in unstable flow estimates. Li Xu *et al.* [Xu 12] recently proposed a method that overcomes this problem by extensively computing initial flow vectors in each pyramid level, instead of solely relying on the estimate at the previous scale. Their implementation is currently top-ranked at the widely referred Middlebury optical flow evaluation dataset and is able to maintain fine object boundaries while calculating a dense flow field.

A third disadvantage of traditional optical flow estimation methods, is their dependency on the so called brightness constancy assumption. In practice, this assumption is often violated due to shadows, specular reflection, non-uniform illumination in the image, or changing illumination in the video sequence. To solve this problem, Vaudrey *et al.* [Vaudrey 10] calculate the optical flow on the normalized gradients of the image, instead of directly using the image intensities. Since normalized gradients are invariant to both multiplicative and additive intensity changes, this approach greatly increases robustness to changing illumination. Wedel *et al.* [Wedel 09] use a similar approach by decomposing the original image into a structural component and a textural component. The textural component is obtained by subtracting a smoothed version of the image from its original, and shows an increased robustness to illumination changes by only representing the high-frequency parts of the scene.

Most of the earlier described difficulties in optical flow estimation, i.e. illumination dependence, computational complexity, and large displacements, coincide with the problems that are being solved in the image registration domain of computer vision literature. However, for image registration purposes, a dense vector field is often not needed. Instead, a sparse set of point matches can be searched for, effectively avoiding the aperture problem by focussing on the well defined regions of the image. Many keypoints detector and descriptor algorithms such as SIFT [Lowe 04], SURF [Bay 06] or FREAK [Alahi 12] can be used for this task. These feature detection and description methods are often scale and rotation invariant, can cope with changing illumination, and are able to handle large displacements. However, spatial accuracy of the obtained point-matches using these techniques is generally lower than the accuracy that is obtained by dense optical flow estimation methods. The main reason for this is that point-matching methods treat all individual points as independent, and do not incorporate regularization constraints.

Brox *et al.* [Brox 09] proposed to combine both methods by using the SIFT detector and descriptor to obtain a sparse set of matching hypotheses, and then using these hypotheses to refine a dense optical flow field by means of total variation

minimization. Weinzaepfel *et al.* [Weinzaepfel 13] extended this work by designing a more robust feature point descriptor and by combining a convolutional net based matching approach with a Total Variation based optimization process to obtain a dense optical flow field.

However, all of the above Total Variation based schemes are computationally expensive due to their iterative optimization (e.g. using a primal-dual formulation) and pyramidal coarse-to-fine search. As a result, dense optical flow algorithms usually need seconds up to minutes of calculation per video frame on current hardware, before obtaining the resulting flow field and are therefore not suited for real-time applications.

For many applications such as object tracking, only a sparse flow-field is needed. In this case, optical flow computation is only a small part of the complete algorithm, and computing a dense flow field would waste a lot of processing power that could be spent more wisely. Although sparse flow methods such as the Lucas-Kanade algorithm or point-matching methods such as the SURF algorithm could be used for such purposes, both approaches often result in noisy flow estimates due to the lack of regularization.

In this chapter, we build upon the idea of Brox *et al.* [Brox 09]. However, instead of incorporating a point-matching approach into a regularized dense optical flow estimation method, we propose a method to regularize a sparse point-matching algorithm directly. The point-matching based approach allows our method to cope with large displacements, and results in illumination independence, eliminating the need for a brightness constancy assumption. Due to the sparse nature, our real-time algorithm is extremely computational efficient, consuming only a fraction of the processor time without resorting to GPU calculations. A local regularization term is included in the optimization function, yielding a smooth flow field while preserving motion boundaries. The algorithm runs faster than the pyramidal Lucas-Kanade algorithm, and outperforms more advanced algorithms in accuracy. The proposed solution can therefore replace sparse optical flow algorithms such as the Lucas-Kanade method, used in real-time applications, in order to improve their robustness without negatively impacting their computational performance.

3 Materials and Methods

To obtain a sparse optical flow field estimate, regions of interest need to be selected. These regions represent locations in the video frame that can be easily matched with corresponding regions in the previous video frame. Once a set of regions has been selected, each region is described using a feature descriptor. Both the feature detector and feature descriptor should be easy to calculate, to guar-

antee real-time performance, and should exhibit a certain degree of illumination independence, to cope with dynamic video scenes.

Two sets of feature points are obtained; one describing the current video frame, and one describing the previous video frame. Sparse optical flow estimation then reduces to an optimization problem in which an optimal match is found between the two sets. However, the sets do not necessarily contain the same number of elements because feature points can be occluded or can split, merge or disappear in subsequent video frames.

To solve this problem and to increase robustness to noise, we introduce a regularization term in the cost function that is minimized iteratively during the feature matching process. To cope with motion discontinuities at the boundaries of moving objects, this regularization term only incorporates local information. Since the feature points are sampled on a non-regular grid, a method is needed to determine the influence of near-by regions of interest.

In the following sections, each of these steps are discussed in more detail.

3.1 Feature Detection, Description and Matching

A variety of feature detection methods have been described in literature, such as the well known SIFT [Lowe 04] and SURF [Bay 06] detectors, the Maximally Stable Extremal Region (MSER) detector [Matas 02], and more advanced spatio-temporal detectors such as the method proposed by Wang *et al.* [Wang 09]. A common goal of these feature detectors, is to extract salient, blob-like regions from the image, and to robustly detect these regions, independent from their scale and rotation.

However, many of these feature detection methods exhibit a high computational complexity. Moreover, most of these detectors are designed as general-purpose feature detectors that are often also used for image registration and object classification. For optical flow estimation on the other hand, any region in the image that avoids the aperture problem can be used. In other words, any region that exhibits both a large horizontal gradient and a large vertical gradient, can be considered a region of interest.

Regions with large gradients in both directions are defined by points where a horizontal and a vertical edge intersect. These points are corner points in the image and can be extracted much more efficiently than general blob-like structures. A widely used corner detector is the Harris corner detector [Harris 88], which relies on the magnitude of the eigenvalues of a structure tensor matrix describing the local change in brightness. This solution however is rather slow and consumes a large portion of the resources available to the optical flow algorithm.

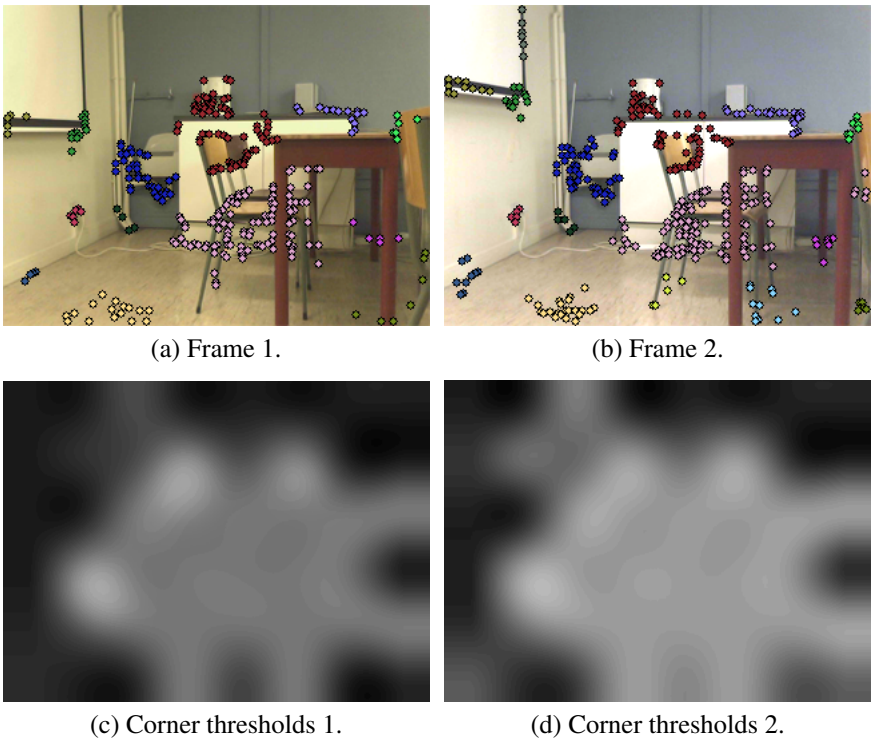
More recently, E. Rosten *et al.* [Rosten 10] introduced a machine learning based corner detector named FAST (Features from Accelerated Segment Test),

which is approximately 20 times faster than the Harris corner detector, and 40 times faster than the SIFT detector. They trained a decision tree that detects corners in the image by quickly comparing the intensity of the pixels surrounding a center pixel. The learned decision tree is then converted into computer code, yielding an extremely fast corner detector. Furthermore, the authors showed that their detector has great repeatability under various aspect changes. Due to the real-time constraints of our research, we therefore propose to use the FAST detector to quickly select a set of regions of interest in the image. A region of interest is simply defined as a square region, centered on the detected corner point.

Most corner detection methods compute a corner response function for each pixel location in the image. A threshold is then applied to this corner response. This threshold can either be static, such that only the best corners are returned, or dynamic such that a predefined number of corners is always detected. However, both approaches have obvious drawbacks. If a static threshold is used, the number of corners that are detected in subsequent frames can vary widely, especially if the scene illumination changes abruptly. On the other hand, if a dynamic threshold is used, the number of detected corners is fixed. Therefore, if a highly textured object is visible in one frame, almost all corners would be located on this object, ignoring other objects in the scene. If this object disappears in the next video frame, almost all corners would be located on the background objects, without having a corresponding match in the previous frame.

The original FAST implementation uses a fixed threshold and would therefore not be suited for point matching in scenes with changing illumination. We propose a simple and efficient change to the original FAST detector allowing it to detect an approximately fixed number of corners that are uniformly distributed over the image while automatically adapting to changing environments and illumination conditions. Instead of defining a single threshold, the image is divided into square regions of size $N \times N$, and an adaptive threshold is calculated automatically for each of these regions such that an approximately constant number of corner detections is maintained in each region. Once such threshold is obtained, bilinear interpolation is used to determine a distinct threshold for each pixel location in the image.

To obtain a threshold for each $N \times N$ block, a simple iterative procedure is used. Initially, the threshold is set to the threshold used in the previous frame. The corner response function is then calculated, after which the thresholds are decremented if less than a certain number of corners N_l is found in the block, while they are incremented if more than a maximally allowed number of corners N_h is found, with $N_l < N_h$. This approach resembles a hysteresis threshold. The lower threshold N_l is limited in order to avoid detecting noisy corners in untextured regions of constant brightness. This procedure is repeated until the desired number of corners is detected or a maximum number of iterations has



(a) Frame 1. (b) Frame 2.

(c) Corner thresholds 1. (d) Corner thresholds 2.

Figure 3.1: Per-pixel corner threshold (bottom row: darker colors correspond to lower threshold values) for two subsequent video frames (top row) with changing illumination. Detected corners are shown as colored dots and are clustered as discussed in Section 3.2.

passed. This ensures that approximately the same number of corners is found in subsequent video frames, even under extreme lighting changes, or when highly textured objects suddenly appear in the scene. This is illustrated by figure 3.1 that shows two subsequent video frames with different illumination, together with the per-pixel corner response threshold value. Detected corners are illustrated by colored dots, the color of which is defined by a clustering process that will be discussed in Section 3.2.

Each detected corner defines a region of interest, the size of which was empirically determined as 18×18 pixels. An illumination invariant descriptor is then calculated for each region such that it can be matched to a region in the previous video frame. A well known illumination independent primitive used in texture classification are Local Binary Patterns (LBP) [Ojala 96, Ojala 02], in which the sign of the difference between a center pixel and each of its neighboring pixels in a 3×3 neighborhood is multiplied with a binomial factor in order to obtain a binary number between 0 and $2^8 - 1 = 255$. This number then represents the

local texture in that area. Since only simple addition and subtraction is needed and multiplication by a power of two, which can be implemented efficiently using shifting operators, calculating a LBP is extremely fast. The histogram of the LBPs computed over a region can then be used as a texture descriptor.

If a white illumination change occurs in the scene, pixel intensities are scaled. On the other hand, if a colored illumination change occurs, pixel intensities are shifted. Furthermore, due to non-linearities in the camera capturing process (e.g. gamma correction), illumination changes in general can be approximated linearly by an intensity shift and a scaling factor. Invariance against additive intensity changes can be obtained using a gradient operator whereas invariance against multiplicative intensity changes can be obtained by only regarding the sign of the gradient instead of its real value. Local binary patterns are based on this idea and have been shown to be robust against uniform illumination changes in the scene by modeling the high-frequency components of the image [Ojala 02].

However, the classical LBP operator yields very long histograms of 256 bins, many of which would be empty if calculated on a small region, resulting in sparse histograms. Moreover, comparing two such histograms would be too slow for our application. M. Heikkilä *et al.* [Heikkilä 09] defined a modified operator which they call the Center-Symmetric LBP (CS-LBP), in which the sign of the difference between center-symmetric pairs of pixels is used instead of the difference between each pixel and the central pixel, as illustrated by equation (3.1). In a 3×3 neighborhood this results in a CS-LBP number between 0 and $2^4 - 1 = 15$ which greatly reduces the number of histogram bins, thereby avoiding sparse histograms. The authors showed that CS-LBP based descriptors exhibit superior performance when compared to classical LBP, and often outperform computationally expensive descriptors such as the SIFT descriptor.

$$\text{CS-LBP}(\mathrm{x}, \mathrm{y}) = \sum_{i=0}^{3} 2^i \; 1_{\mathbb{Z}^+}(n_i - n_{i+4}). \tag{3.1}$$

where $1_{\mathbb{Z}^+}$ is an indicator function.

In order to incorporate spatial information into the descriptor, the 18×18 region is divided into 9 cells of size 6×6, and a CS-LBP histogram is calculated for each cell. The histograms are then concatenated to obtain the final spatially enhanced CS-LBP based feature vector of size $16 \times 9 = 144$.

In the larger context of this dissertation, the choice for an LBP-based descriptor greatly reduces the computational complexity of the complete hand tracking and detection system, because the same LBP descriptors where already calculated for hand detection purposes in chapter 2 and can simply be re-used here without additional calculations.

In order to obtain the optical flow field, regions of interest from the current frame are matched with regions from the previous frame, based on their CS-LBP

descriptors. In the following, let $LBP_{144}(r;b)$ be the 144-bin CS-LBP descriptor for a region r, parameterized by its bin number b. The goodness of a match between two regions r_{t-1} and r_t can then be defined by a cost function involving the corresponding descriptors $LBP_{144}(r_{t-1})$ and $LBP_{144}(r_t)$, and the best match is found by minimizing this cost function.

Because our feature descriptor $LBP_{144}(r;b)$ represents a histogram, well known distance measures for comparing histograms could be used, such as the Chi-Square distance, Bhattacharya distance or Earth Mover's distance. However, for performance reasons, we propose to use the L_1 norm, which is much faster to calculate and can easily be implemented using SIMD instructions. The quality of a match between a region r_{t-1} in the previous video frame and a region r_t in the current video frame is thus defined as:

$$\|LBP_{144}(r_{t-1}) - LBP_{144}(r_t)\|_1 = \sum_{i=1}^{144} |LBP_{144}(r_{t-1};i) - LBP_{144}(r_t;i)|. \tag{3.2}$$

In the following, let $\mathbf{c}_t$ be the detected corner coordinates that correspond to the centroid of region r_t and let $\mathbf{c}_{t-1}$ be the corner that corresponds to the centroid of region r_{t-1}. Matches from a corner $\mathbf{c}_{t-1}$ in the previous frame to a corner $\mathbf{c}_t$ in the current frame are searched for within a certain radius s surrounding $\mathbf{c}_{t-1}$. This radius defines the maximally allowed motion vector length and was set to $s = 30$ in our experiments. Using a frame resolution of 320×240, this means that very large displacements are allowed between subsequent frames. However, while such large displacements are allowed, they are highly unlikely to occur in real video. Assuming this likelihood drops linearly with the displacement distance, the Euclidean distance between corner $\mathbf{c}_t$ and a candidate corner $\mathbf{c}_{t-1}$ is added to the cost function to be minimized. The addition of this term makes sure that large displacements are penalized more than small flow vectors.

Finally, a regularization term $\lambda(\mathbf{c}_{t-1}, \mathbf{c}_t)$ is added to steer the search process such that the obtained flow field is locally smooth. The regularization term will be defined in Section 3.2 and biasses the optimization process to prefer matches that result in a smooth and locally consistent flow-field. Local regularization increases robustness to noise and allows our method to cope with motion discontinuities at object boundaries. The complete cost function is then defined as follows:

$$C(\mathbf{c}_{t-1}, \mathbf{c}_t) = \|LBP_{144}(r_{t-1}) - LBP_{144}(r_t)\|_1 + \alpha\|\mathbf{c}_{t-1} - \mathbf{c}_t\|_2 + \beta\lambda(\mathbf{c}_{t-1}, \mathbf{c}_t). \tag{3.3}$$

where α and β are weighting factors indicating the importance of the regularization terms. In our experiments, $\alpha = 1$ and $\beta = 100$.

Equation (3.3) is optimized iteratively as will be discussed in Section 3.2. Initially, the regularization term is defined as $\lambda(.) = 0$. Therefore, during the first iteration, the non-regularized optical flow is obtained. Based on this flow field, the

regularization term for the next iteration is calculated, and the process is repeated until convergence or until the maximum number of iterations has been reached.

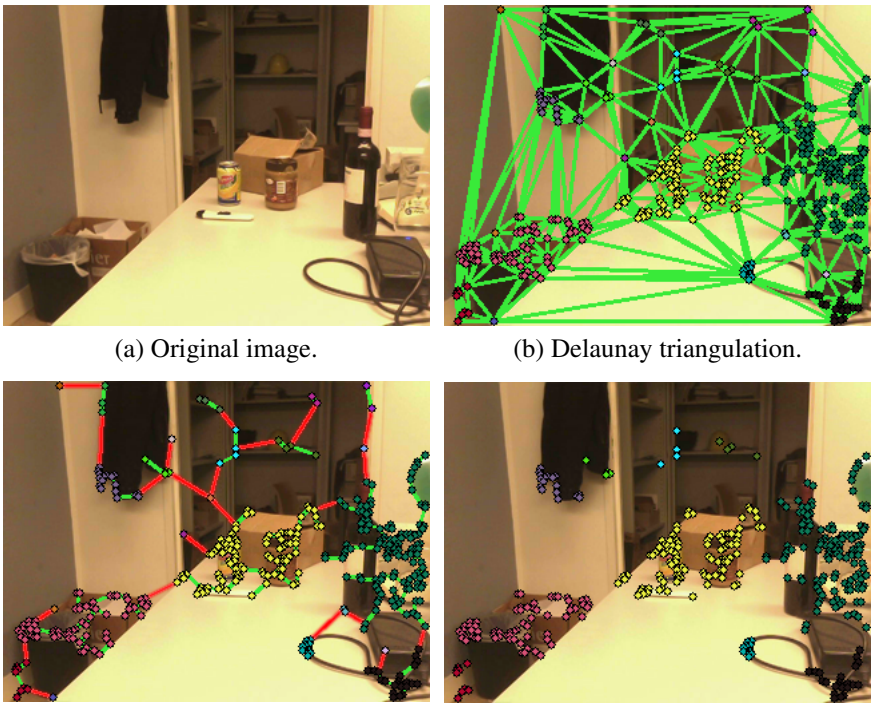
3.2 Local Regularization

Global regularization of the optical flow field, such as performed by the Horn and Schunck method, would assume a globally smooth vector field and thus cannot handle motion discontinuities at object boundaries. Instead, the optical flow field should only be regularized locally. Therefore, a method is needed to define a local neighborhood of corners for each detected corner, in an illumination invariant manner. The solution should have low complexity and should resemble a rough segmentation of the objects in an image.

Dense optical flow estimation techniques often incorporate a segmentation phase to obtain object boundaries and to guide the local regularization process [Zitnick 05, Xu 08, Deqing 12]. However, image segmentation is usually slow, and inherently depends on the scene illumination due to its use of pixel colors during the segmentation process. Instead of segmenting the image based on its lighting dependent color distribution, we propose to segment it based on its corner distribution. This means that image regions containing high spatial frequencies such as textures will be separated from regions containing only low spatial frequencies such as smoothly varying background. The main advantage of this approach is that it can be implemented extremely efficiently as a clustering operation on a sparse set of corner points. Furthermore, the clustering process is independent of the scene illumination and roughly corresponds to a texture based segmentation.

However, corner locations often exhibit clear patterns and can be almost colinear if they are part of an edge, which means that fast traditional clustering algorithms such as k-means clustering, which tries to find spherical clusters, are not suited for this goal. Although more advanced clustering algorithms such as spectral clustering exist, these are often computationally expensive, and can not be used for real-time applications. Instead, we propose to use a hierarchical clustering method that is capable of detecting clusters with irregular boundaries [Grygorash 06] and does not need to know the number of clusters in advance.

A single-link hierarchical clustering can be obtained by treating the set of corner points as a planar Euclidean graph such that each corner is represented by a vertex that is connected to its neighbors by edges with a weight that is equal to their Euclidean length. By calculating the Minimum Spanning Tree (MST) of this graph, we obtain a tree structure that connects only the corner points that are closest to each other. A clustering can then be obtained by removing inconsistent edges (i.e. edges that are significantly longer than surrounding edges) from this tree. The connected components that are defined by the resulting subtrees are considered to



(a) Original image. (b) Delaunay triangulation.

(c) Minimum spanning tree. (d) Final clusters.

Figure 3.2: The Delaunay triangulation of figure (b) is used to obtain a MST representation shown in figure (c). Inconsistent edges, shown in red, are removed from the MST, and small clusters are ignored. The final clusters are shown in figure (d).

be clusters. Finally, small clusters are considered noise, and are eliminated. This process is illustrated by figure 3.2.

In order to efficiently determine the MST, a Delaunay triangulation is calculated for the detected corners, which can be implemented very efficiently for Euclidean point sets by using a radial sweep-hull scan [Sinclair 10]. From the resulting triangulation, when considered as a Euclidean graph, the minimum spanning tree can be obtained by means of the well known Kruskal algorithm.

To obtain a clustering, inconsistent edges are removed from the MST. An edge is regarded inconsistent if one of following conditions holds:

$$\begin{cases} \|N_1 - N_2\| > \mu_{N_1} + c\,\sigma_{N_1} \\ \|N_1 - N_2\| > \mu_{N_2} + c\,\sigma_{N_2} \end{cases} \tag{3.4}$$

where $\|N_1 - N_2\|$ is de Euclidean length of the edge and μ_{N_1} and μ_{N_2} are respectively the average length of nearby edges at the side of node N_1 and the average length at the side of node N_2. Similarly, σ_{N_1} and σ_{N_2} are the standard deviations

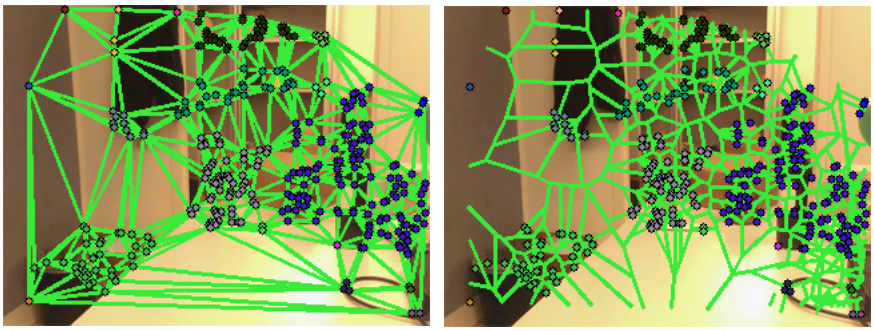
of the edge lengths at both sides. In our implementation, c was set to $c = 1.5$ such that an edge is considered an outlier if its length differs more than 1.5 standard deviations from the mean. The mean and standard deviations are calculated in a neighborhood around N_1 and N_2 such that only nearby corners are taken into account. Two nodes are considered '*nearby*' if their geodesic distance is not larger than three. Figure 3.1 illustrates the result of this clustering approach on the detected corners of a scene in two video frames. In the second video frame, illumination has changed and objects have moved compared to the first frame.

To enforce local smoothness, local regularization is based on weighted vector median filtering. Vector median filters are known to be able to smooth estimated velocity fields while maintaining motion discontinuities or edges. L. Alparone *et al.* [Alparone 99] showed that the application of a weighted vector median filter after dense optical flow calculation improves the smoothness and accuracy of the result. The disadvantage of such approach however, is that the median filter is only used as a non-linear postprocessing filter to reduce noise in a non-regularized flow field instead of incorporating the smoothness constraint directly into the cost minimization process itself.

Furthermore, while it is clear how to apply such a filter on a dense optical flow field on a regular grid, it is less obvious how to use a similar technique on a sparse flow field in a non-regular grid. In the next paragraphs, we discuss how vector weighted median filtering can efficiently be used on a non-regular grid to directly incorporate the smoothness constraint as a regularization term in the cost function defined by equation (3.3).

The vector median filter is basically just a median filter applied to each component of the vectors. In order to regularize the optical flow search process, we define the optimal flow vector for a corner $\mathbf{c}_t$ as the vector obtained in that point after weighted vector median filtering of the flow field obtained in a previous iteration, and then minimize the difference between a candidate flow vector and the optimal flow vector by means of a well chosen regularization term $\lambda(\mathbf{c}_{t-1}, \mathbf{c}_t)$.

In order to obtain the median vector within a neighborhood of corner $\mathbf{c}_t$, the concept of neighborhood must be defined for a sparse, non-regular grid. We propose to adopt the concept of natural neighbors as used in Voronoi interpolation schemes. The natural neighbors of corner $\mathbf{c}_t$ are those corners whose Voronoi cell in a Voronoi tessellation are adjacent to the Voronoi cell of $\mathbf{c}_t$. Furthermore, only corners belonging to the same cluster as corner $\mathbf{c}_t$ are considered which effectively allows us to perform regularization locally and to deal with motion discontinuities. Calculating the Voronoi tessellation incurs almost no extra performance cost, since a Delaunay triangulation was obtained already in the clustering stage, and the Voronoi diagram is nothing more than the dual graph of this Delaunay triangulation. This is illustrated by figure 3.3.



(a) Delaunay triangulation. (b) Voronoi tessellation.

Figure 3.3: The Delaunay triangulation that was used during clustering is re-used to obtain its dual graph; the Voronoi diagram. Voronoi cells define the concept of natural neighbors for a non-regular grid, and are used for local regularization.

For each corner $\mathbf{c}_t$, the weighted median flow vector can then be calculated based on its valid neighbors, which are those corners that are natural neighbors and belong to the same cluster as corner $\mathbf{c}_t$. A weighted median filter replicates each element several times, based on its weight, before finding the median value. The weighting factor for a flow-vector of a neighboring point $\mathbf{l}_t$ should depend on both the reliability r of the estimated flow vector for this neighboring corner in the previous iteration, and a distance measure d indicating how close the neighbor is to corner $\mathbf{c}_t$.

As a result, our vector median filter behaves like a bilateral filter. In the image restoration domain, bilateral filters are known as linear filters that combine two types of information into the weight calculation, and have been shown to behave both noise-reducing and edge-preserving. This concept was extended to non-linear filters by Francis *et al.* [Francis 03], resulting in the bilateral median filter. The weights of our bilateral filter are defined by a reliability measure r and a distance measure d.

The reliability r of the estimated flow vector for a neighboring point $\mathbf{l}_t$ in the previous iteration can be obtained directly from the cost $r = C(\mathbf{l}_{t-1}, \mathbf{l}_t)$ associated with the match of this point with a corner in the previous frame. For the distance measure d, we propose to use the weighting factor as used in the non-Sibsonian Voronoi interpolant for non-regular grids, described by Belikov *et al.* [Belikov 97]. In the following, let $\mathbf{s}_{cl}$ be the vector representing the edge of the voronoi diagram that separates points $\mathbf{c}_t$ and $\mathbf{l}_t$, and let $\mathbf{h}_{cl}$ be the vector from $\mathbf{c}_t$ to $\mathbf{l}_t$. The non-Sibsonian Voronoi interpolant weighting factor effectively uses the natural neighbor concept by defining the weight as the quotient of the distance between the

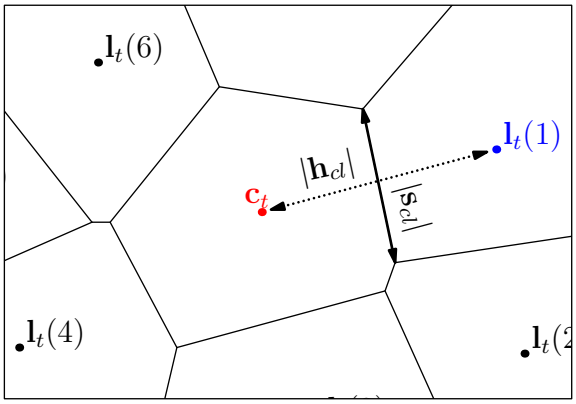


Figure 3.4: Definitions for the non-Sibsonian interpolant weight.

centroids of the cells $|\mathbf{h}_{cl}|$ and the length of the edge $|\mathbf{s}_{cl}|$ that is shared by the two Voronoi cells. This is illustrated more clearly in figure 3.4.

To calculate the weighted median flow vector for corner $\mathbf{c}_t$, all natural neighbors $\mathbf{l}_t$ that are also within the same cluster are assigned a weight. Natural neighbors for which no matching point could be found in the previous video frame are removed once the Voronoi diagram is obtained, and are therefore ignored by the weighted median calculation. Let $\mathbf{l}_{t-1}(j)$ be the corner point in the previous frame that was matched to the corner point $\mathbf{l}_t(i)$ in the current frame, where $\mathbf{l}_t(i)$ is a natural neighbor of the corner $\mathbf{c}_t$ for which we try to estimate the optical flow vector. The weight for the i-th natural neighbor, $\mathbf{l}_t(i)$, is then defined as w^i:

$$w^i = C(\mathbf{l}_{t-1}(j), \mathbf{l}_t(i)) \frac{|\mathbf{h}_{cl(i)}|}{|\mathbf{s}_{cl(i)}|}. \tag{3.5}$$

Let $\mathbf{w} = (w^1, ..., w^i, ...w^n)$ be the vector containing the bilateral filter weights for all n neighboring corners. The optical flow-vectors of all these neighboring corners, obtained in the previous iteration, are then replicated N^i times, after which the median vector $\mathbf{o}_{c_t}$ of the resulting set of flow-vectors is calculated. This flow-vector is the result of the bilateral filtering process.

$$N^i = \frac{\max(\mathbf{w}) + 1}{w^i + 1}. \tag{3.6}$$

Flow vector $\mathbf{o}_{c_t}$ is considered to be the optimal, noise-free flow vector for a corner point $\mathbf{c}_t$, and thus for its region r_t since it is obtained by weighted median filtering the u and v components of the neighbors of this corner. While searching for a better match for corner $\mathbf{c}_t$ during the next iteration, we would therefore like to encourage matches that result in a flow vector that resembles the optimal

vector $\mathbf{o}_{c_t}$. Thus, in the next iteration, the difference between a possible candidate vector and this optimal vector is minimized in order to obtain a smooth flow field. Minimization of this difference is achieved by defining the regularization term $\lambda(\mathbf{c}_{t-1}, \mathbf{c}_t)$ in the cost function shown by equation (3.3) as follows:

$$\lambda(\mathbf{c}_{t-1}, \mathbf{c}_t) = \big|\, |\mathbf{o}_{c_t}| - |\mathbf{f}_{c_t}| \,\big| \arccos\left[\frac{\langle \mathbf{o}_{c_t}, \mathbf{f}_{c_t} \rangle}{|\mathbf{o}_{c_t}||\mathbf{f}_{c_t}|}\right]^2, \tag{3.7}$$

where $\mathbf{f}_{c_t}$ is the candidate flow vector from corner $\mathbf{c}_t$ to a corner $\mathbf{c}_{t-1}$ in the previous frame.

Adding the regularization term to the cost function makes sure that the difference in magnitude between the optimal vector $\mathbf{o}_{c_t}$ and the candidate vector $\mathbf{f}_{c_t}$ is minimized, together with the squared angle difference between both vectors. Moreover, a change in direction is penalized more than a change in length. Because only corners in the same cluster as the corner under consideration are used while calculating the optimal flow vector, the minimization process is biased towards a smooth flow field but can still cope with sudden motion discontinuities at object boundaries.

Before the first iteration, $\lambda(.)$ was set to zero. Therefore, after the first iteration, the non-regularized flow-field is obtained. After each subsequent iteration, flow-vectors are biased more towards the median flow-vector in their natural neighborhood. The final flow field can be chosen to be either the result of the last iteration, or the result of the last median calculation. In the latter case, this corresponds to applying a final weighted median filter as a postprocessing step to the already regularized optical flow field.

4 Results and Discussion

Evaluating optical flow algorithms is a tedious and difficult task, mainly due to the lack of ground truth data. The most widely used dataset of ground truth optical flow vectors is the Middlebury optical flow database [Baker 11]. However, the Middlebury dataset was created for typical optical flow algorithms based on the constant brightness assumption and local linearization and therefore does not contain sequences with changing illumination or large displacements. Furthermore, the available test sequences are synthetically generated and often do not resemble real video very well as the video frames contain very fine grained details, a lot of similarity in texture, and frame to frame displacements are rather small. Also the sequences lack motion blur and realistic illumination, resulting in unfair comparison between algorithms that can deal with noisy observations, and algorithms that assume perfect data.

This led us to the creation of a real-video dataset, which is made publicly available[1] together with its carefully annotated ground truth [Spruyt 13b]. However, even though the algorithm described in this chapter was designed with changing lighting conditions and fast motion in mind, it might still be interesting to see how it performs on artificial data, allowing for an easy comparison to many of the state-of-the-art algorithms that are listed on the Middlebury website. Therefore, in the next sections, our algorithm is evaluated both on the Middlebury dataset, and on a dataset with changing illumination and fast motion for which ground truth data was manually created.

In the following experiments, the image size was 320×240, the search radius for point matching was 30 and three iterations were used for regularization.

For each sequence in the dataset, optical flow fields are calculated by our proposed algorithm, indicated as *LBP-flow*, by the pyramidal *Lucas-Kanade* algorithm, by the Total Variation based *MDP-flow* algorithm [Xu 12], and by *SURF* descriptor based matching. While Lucas-Kanade represents the most widely used motion estimation technique in real-time object tracking, MDP-flow is currently the top-ranking algorithm when evaluated on the Middlebury dataset. SURF descriptor based matching is often used for point tracking and is considered one of the most robust descriptors currently available. Since our method combines concepts from traditional optical flow estimation techniques with a point matching methodology, comparison with state-of-the-art algorithms of both types gives a clear view on its capabilities.

For the pyramidal Lucas Kanade algorithm, a window size of 10×10 and a 2-level pyramid was used, resulting in a maximally allowed displacement of $(2^2 - 1) \times 10 = 30$ pixels. This windows size was chosen such that the maximally allowed displacement is equal to the search radius used by our algorithm and the SURF algorithm. However, changing these parameters did not affect the results much. For the SURF algorithm, it is important to note that only SURF descriptors were used to describe and match the FAST feature points, and that the SURF detector itself was not used. The reason for this is that the SURF detector returns Determinant of Hessian (DoH) based features which can not be fairly compared with corners. For each of the algorithms that are tested in this section, the same sparse set of FAST keypoints are used to evaluate the performance.

The major contribution of our work is a sparse regularization scheme that allows very simple feature descriptors to yield robust point matching results. However, it should be noted that any kind of feature detector could replace the FAST detector if real-time performance is not a concern.

For evaluation purposes we report both the angular error and the endpoint error, similar to [Baker 11]. The angular error between an estimated flow-vector (u, v) and the corresponding ground truth vector (u_{GT}, v_{GT}) is commonly used to com-

[1] `http://telin.ugent.be/~vspruyt/ICME2013/`

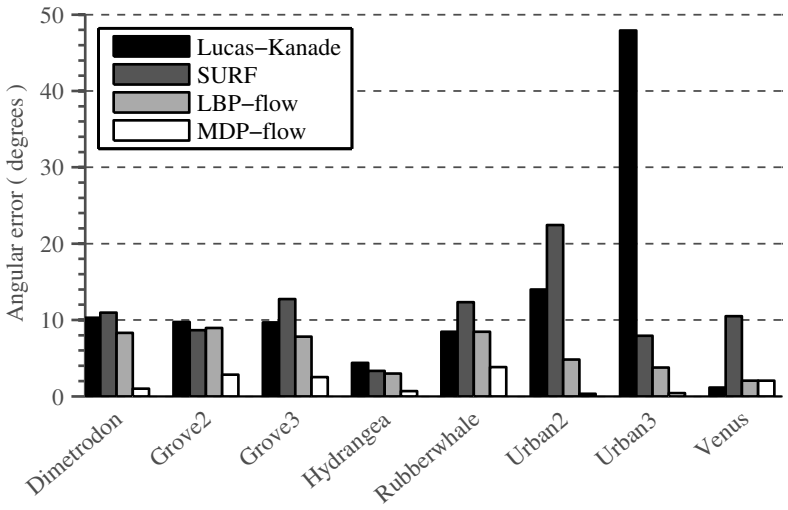


Figure 3.5: Middlebury Dataset evaluation (angular error).

pare optical flow algorithms in the computer vision community. It is defined as the angle in 3D space between the vectors $\mathbf{f} = (u, v, 1)$ and $\mathbf{f}_{GT} = (u_{GT}, v_{GT}, 1)$. The angular error AE is calculated as:

$$AE = \arccos\left(\frac{\mathbf{f} \cdot \mathbf{f}_{GT}}{|\mathbf{f}|\,|\mathbf{f}_{GT}|}\right). \tag{4.1}$$

The popularity of the angular error is due to the fact that it provides a relative error in the sense that errors in large flows are penalized less than errors in small flows. However, for practical applications both types of errors are usually equally important. Therefore, Baker *et al.* [Baker 11] introduced a second error measure, termed the *endpoint error*. The endpoint error EE simply corresponds to the Euclidean distance between the groundtruth point match and the estimated point match, and is defined as:

$$EE = \sqrt{(u - u_{GT})^2 + (v - v_{GT})^2}. \tag{4.2}$$

To allow for easy comparison of our results with results reported in literature, we report both types of error measures. Figure 3.5 shows the resulting angular deviation from the given ground truths for each of the algorithms tested on the Middlebury sequences. Figure 3.6 shows the resulting endpoint deviation for these same sequences and algorithms. These results indicate that the proposed algorithm, *LBP-flow*, outperforms both Lucas-Kanade and the SURF matching algorithm. For each of the test sequences, except for the *Grove2* sequence, a lower average angular error is obtained, accompanied by a lower endpoint-error. While

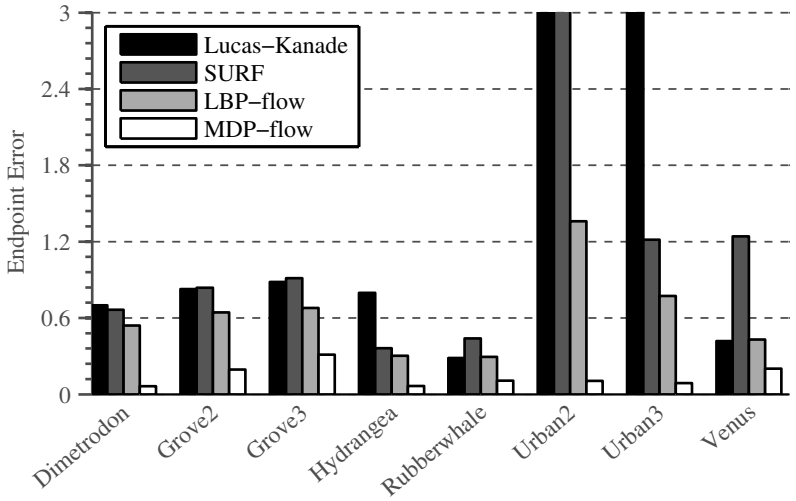


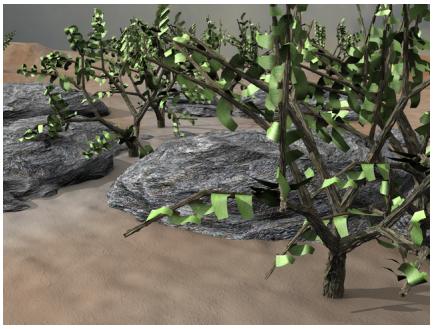
Figure 3.6: Middlebury Dataset evaluation (endpoint error).

SURF descriptors are much more descriptive than our CS-LBP descriptor and are rotation invariant, our proposed regularization scheme effectively reduces matching noise and allows the use of much simpler feature descriptors resulting in faster execution with robust matching characteristics.

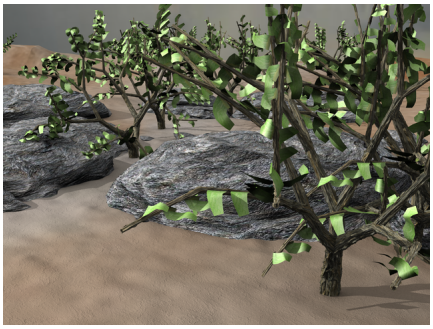
The results also show that the offline *MDF-flow* algorithm seems to be much more robust than the proposed real-time algorithm. However, it is important to note that the Middlebury dataset does not try to mimic real-life, low-quality video sequences. Real video data tends to lack sharp textures and edges, contains large and irregular displacements, a minimal amount of spatial self-similarity, and greatly violates the brightness constancy assumption. Therefore, algorithms performing well on the Middlebury dataset might not perform well on real video data and vice versa. Figure 3.7 shows frames from two sequences of the Middlebury dataset, illustrating their artificial characteristics.

Table 3.1 shows the angular errors and table 3.2 shows the endpoint errors obtained by each algorithm on the real-life video sequences we created and made public for the research community, together with their ground truth. These sequences contain large and irregular displacements up to 30 pixels, camera artifacts such as vignetting, noise and motion blur, and sudden changes in lighting conditions.

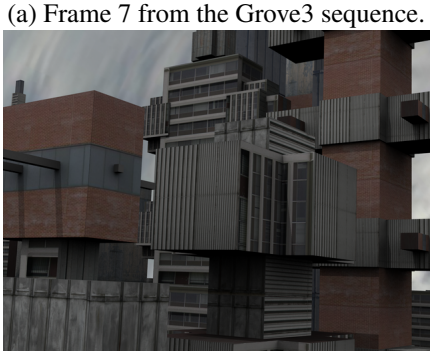
These results illustrate the robustness of the proposed approach for real-life video. The bad performance of Lucas-Kanade is easily explained by the violation of the brightness constancy constraint. While the *MDP-Flow* algorithm performs reasonably well when compared to the Lucas-Kanade algorithm, it also fails when



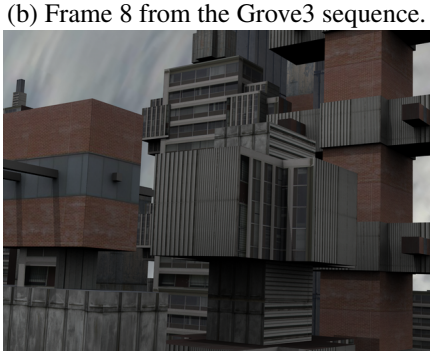
(a) Frame 7 from the Grove3 sequence.



(b) Frame 8 from the Grove3 sequence.



(c) Frame 7 from the Urban sequence.



(d) Frame 8 from the Urban sequence.

Figure 3.7: Video frames from the Middlebury dataset.

Table 3.1: Proposed Dataset evaluation (angular error).

Sequence	Algorithm	Angular error		
		Median	Avg.	Std. Dev.
Desk	Lucas-Kanade	87.6	87.9	65.0
	SURF	2.0	2.5	3.0
	LBP-Flow	**1.9**	**2.0**	**2.9**
	MDP-Flow	12.9	44.6	48.8
Drinks	Lucas-Kanade	84.7	75.1	62.0
	SURF	7.4	15.9	31.5
	LBP-Flow	**1.1**	**3.8**	**4.2**
	MDP-Flow	79.7	91.4	75.2
Office	Lucas-Kanade	7.2	41.5	54.5
	SURF	8.7	27.4	41.8
	LBP-Flow	**3.3**	**3.5**	**2.3**
	MDP-Flow	11.2	16.2	17.5
Classroom	Lucas-Kanade	6.6	21.5	37.5
	SURF	5.4	22.0	43.4
	LBP-Flow	**5.2**	**4.8**	**3.9**
	MDP-Flow	8.0	23.1	38.8

Table 3.2: Proposed Dataset evaluation (endpoint error).

Sequence	Algorithm	Endpoint error		
		Median	Avg.	Std. Dev.
Desk	**LBP-Flow**	**1.0**	**2.2**	**3.0**
	Lucas-Kanade	46.2	44.4	27.0
	MDP-Flow	26.0	32.0	26.8
	SURF	2.1	2.8	3.3
Drinks	**LBP-Flow**	**1.4**	**1.8**	**1.5**
	Lucas-Kanade	18.4	23.3	25.3
	MDP-Flow	17.9	32.0	29.6
	SURF	4.0	12.0	26.1
Office	**LBP-Flow**	**1.0**	**1.1**	**0.6**
	Lucas-Kanade	2.2	8.6	10.1
	MDP-Flow	3.9	5.5	4.3
	SURF	2.0	8.3	15.8
Classroom	**LBP-Flow**	**1.4**	**1.3**	**0.7**
	Lucas-Kanade	1.8	5.1	11.1
	MDP-Flow	3.5	7.7	11.7
	SURF	1.4	10.9	22.8

lighting conditions change. The best result for the *MDP-Flow* was obtained with the *Office* sequence, which contains the least amount of lighting change. Figure 3.8 shows the video frames and corresponding flow field for each of the sequences used for evaluation. These sequences, together with their ground truth, are made freely available for the research community.

For these challenging sequences, which represent typical scenarios in object tracking applications, point matching algorithms such as the SURF method or the proposed CS-LBP based method, outperform most optical flow approaches. These results clearly show the advantage of our regularization scheme in combination with a simple feature descriptor as opposed to a more advanced feature descriptor without regularization constrains. Figure 3.9 visually shows the flow-field obtained using each of the algorithms for the classroom sequence. The calculated flow-vectors are illustrated by green arrows, whereas the ground truth vectors are shown in black. The classroom sequence contains individually moving objects, each moving to a different direction. By visually inspecting and comparing the results, it becomes clear that the regularization helps to avoid outliers.

It is important to note that we do not claim that our algorithm outperforms the SURF method for point matching in general. For example, SURF matching would largely outperform our method when used for tasks such as image registration or object classification. Moreover, the SURF detector would yield much more stable regions of interest than the FAST detector used by our method. However, we do claim that our method outperforms the traditional SURF point matching algorithm when applied to the task of sparse optical flow estimation. This is achieved by our efficient and local regularization scheme.

Similarly, we do not claim to present an optical flow algorithm that outperforms the MDP-flow method in general. The MDP-flow algorithm is able to estimate a dense optical flow field, which makes it suitable for tasks such as motion segmentation and depth estimation. Our algorithm can not be used for such purposes. We do claim, however, that our method outperforms the MDP-flow algorithms when applied to the task of sparse optical flow estimation which is much simpler to solve because the aperture problem is largely avoided. Therefore, for specific applications such as real-time tracking in video, where only a sparse flow-estimate is needed, our method could replace more advanced optical flow algorithms to maintain real-time performance.

The major focus of our work is real-time performance. Most object tracking algorithms [Kodama 10, Lucena 03] resort to simple and sparse optical flow estimators such as the Lucas-Kanade method due to their real-time constraints. For these applications, a quick estimate of the motion direction and velocity of a region in the image is more useful than a dense and precise estimate of the complete flow-field. Therefore, the ultimate goal of our work is to present a method that

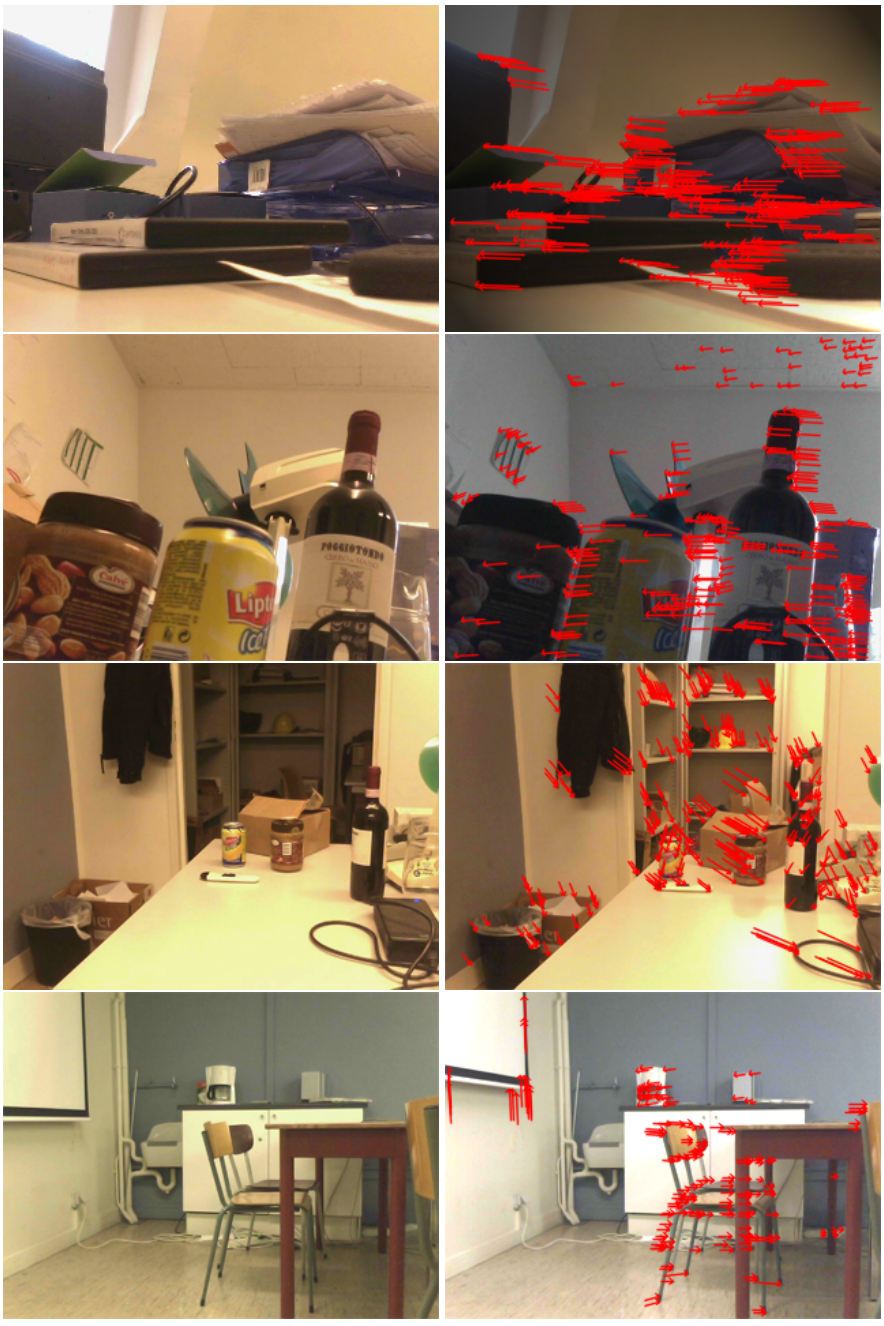
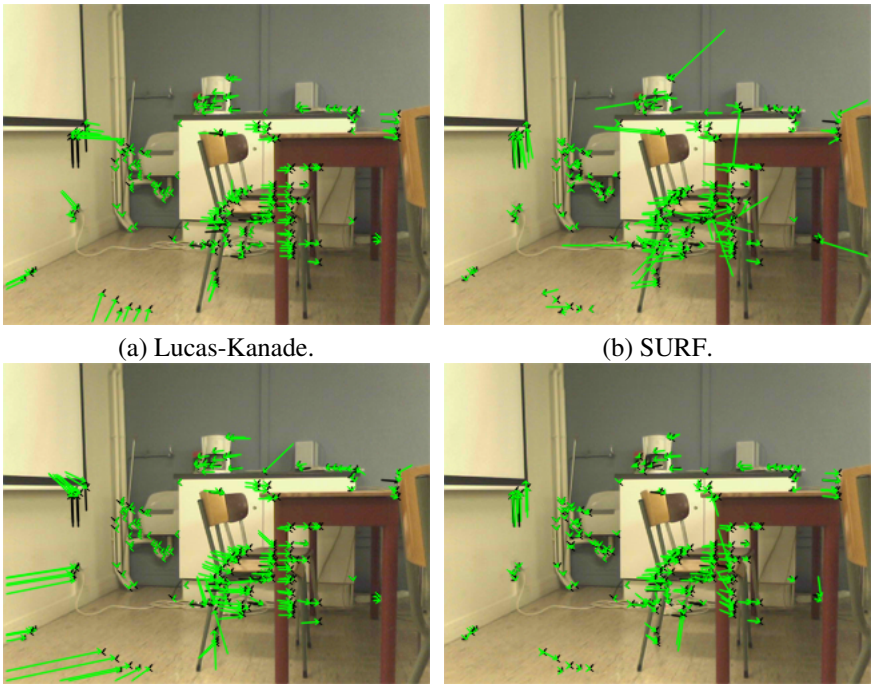


Figure 3.8: Proposed video dataset with annotated ground truth flow.



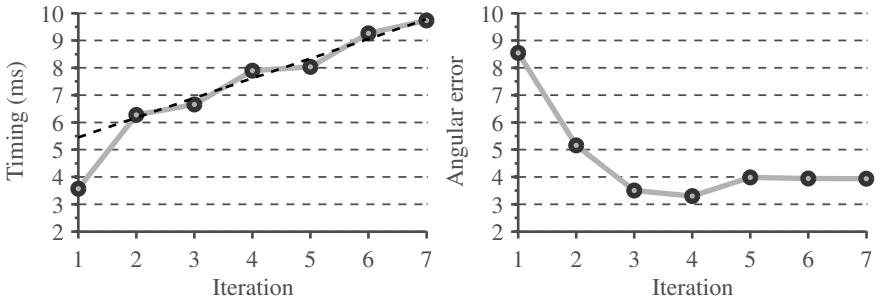
(a) Lucas-Kanade. (b) SURF.

(c) MDP-Flow. (d) LBP-Flow.

Figure 3.9: Visualization of the obtained flow-field. Calculated vectors are shown in green, whereas ground truth vectors are shown in black. For each method, the FAST corner detector was used to determine the points for which optical flow vectors are calculated. Ground-truth vectors are obtained by careful manual annotation.

Table 3.3: Average execution time per video frame.

Algorithm	Execution time (ms)	# corners
LBP-Flow	**6.6**	300
Lucas-Kanade	8.4	300
MDP-Flow	618.4	300
SURF	20.3	300



(a) Execution time (ms). (b) Angular error (degrees).

Figure 3.10: Execution times and error measurements.

outperforms the still widely used Lucas-Kanade algorithm in terms of robustness and accuracy, while having a lower computational cost.

Table 3.3 lists the average execution times for each algorithm when the flow is calculated for 300 corner points. Timings were measured on an Intel 2.4 Ghz dual core CPU. The proposed algorithm is extremely fast thanks to the simple LBP based feature descriptors used, and can thus be used in real-time tracking applications.

Figure 3.10 shows both the average execution time and the average angular error as a function of the number of regularization iterations. As indicated by the dotted line, the complexity of the proposed regularization scheme is linear in the number of iterations. Furthermore, the error clearly reduces after the first few iterations, illustrating the benefit of local regularization.

If only a single iteration is used, no regularization is performed, and the angular error would be larger than the error obtained by more advanced point matching algorithms such as SURF. However, after a few regularization iterations, the error becomes much lower than the error obtained by other, non-regularized, methods. Generally, the algorithm converges after 3 or 4 iterations which corresponds to an execution time of about 7 milliseconds on our current hardware. The small increase in error after the fourth iteration can be explained by imperfect clustering, causing local over-smoothing of the flow field.

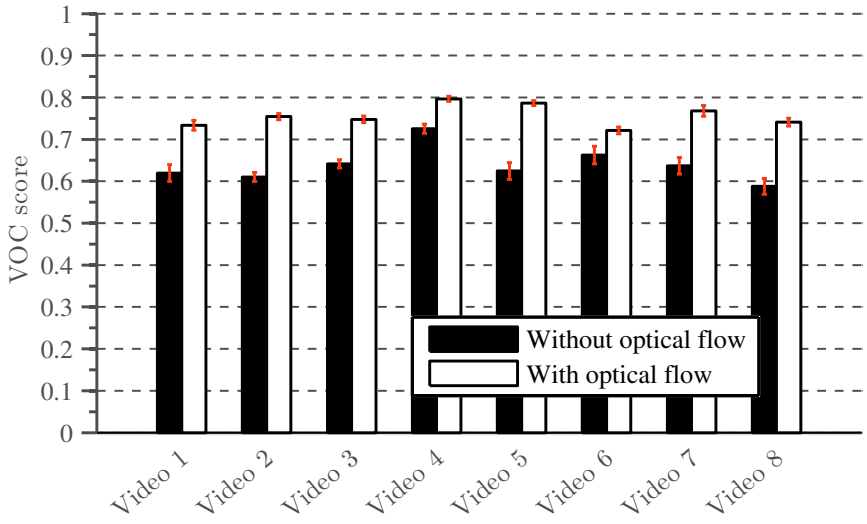
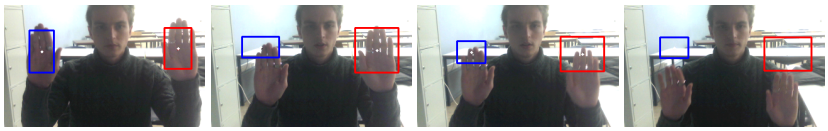


Figure 3.11: Motion model evaluation for particle filter tracking.

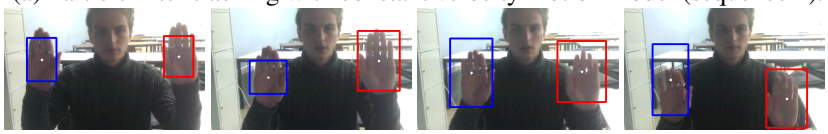
The proposed algorithm is used to improve the proposal distribution of a particle filter based hand tracker, discussed in chapter 4. Tracking results were evaluated on the publicly available dataset, proposed in [Spruyt 12], that contains eight challenging video sequences. Results are reported in terms of the achieved VOC score [Everingham 10]. The VOC score defines the amount of overlap between the tracked bounding box, and the given ground truth bounding box. The score is calculated as $VOC = \frac{B_g \cap B_d}{B_g \cup B_d}$, where B_g represents the pixels within the ground truth bounding box, whereas B_d represents the pixels within the detected bounding box.

Figure 3.11 shows the widely used VOC score for each of the 8 video sequences used for evaluation. These results show that the error is lower for each of the sequences if optical flow is used by the proposal distribution. On average, the optical flow based proposal distribution outperforms a traditional bootstrap filter by 11.8%.

Figure 3.12 shows several frames of sequence 4, used for evaluation. The top row illustrates the results obtained using a simple constant velocity motion model, while the bottom rows shows the results obtained using our proposed optical flow solution.



(a) Particle filter tracking with constant velocity motion model (sequence 4).



(b) Particle filter tracking with optical flow based motion model (sequence 4).

Figure 3.12: Motion model evaluation for particle filter tracking.

5 Conclusion

A fast algorithm was proposed to calculate a sparse optical flow field. The proposed solution combines concepts from both point matching algorithms and optical flow estimation methods. Regularization is performed iteratively, based on median filtering and Voronoi interpolation for sparse, non-regular grids. The resulting method is illumination invariant and able to handle large displacements making it a perfect candidate for incorporation into real-time applications for uncontrolled environments such as object tracking.

The method was evaluated thoroughly, and appears to be much more robust than a pyramidal Lucas Kanade implementation while yielding similar or better results then a SURF-descriptor based tracker. Furthermore, the algorithm outperforms the state-of-the-art in variational optical flow calculation, when used for sparse flow estimation on a real-life video dataset. While the accuracy and robustness of our method is comparable to state-of-the-art point matching and optical flow estimation algorithms, its computational complexity is lower than the widely used and very simple Lucas-Kanade method.

Finally, we showed that the incorporation of our optical flow method into a state-of-the-art tracker greatly improves its robustness while the computational overhead introduced by our method is negligible compared to the computation time of the tracking algorithm itself. Our optical flow dataset is made publicly available together with its carefully annotated ground truth.

4

Real-time, Robust Hand Tracking

1 Introduction

In the previous chapter, we introduced a computationally efficient method to obtain a sparse optical flow field. In this chapter, we build upon these results to introduce a real-time hand tracking algorithm. Furthermore, our hand tracking algorithm incorporates the hand detector of chapter 2 to allow for automatic initialization and error recovery. Whereas chapter 2 focussed on hand detection in static images, this chapter focusses on hand tracking throughout video sequences. For an overview of the relations between the components, introduced in each chapter, please refer to figure 1.1.

Tracking rigid objects with a stable appearance, or tracking non-rigid objects with a varying appearance but with clear, discriminative features (e.g. mouth, nose, eyes) has been largely solved in academic literature. However, the appearance of human hands varies greatly depending on their pose, and can therefore not be described easily using clearly discriminative features. Hands are non-rigid, highly articulated objects with a large number of degrees of freedom: 27 in total, six of which represent translation and rotation of the wrist [ElKoura 03]. As a result, the visual appearance of a human hand can vary widely, making hand detection and tracking a challenging task.

To reduce the dimensionality of the problem, many hand detection and tracking methods resort to detecting hands in predefined poses only [Huang 12, Kölsch 04b, Ong 04, Stenger 06a]. Such methods have limited applicability in real-life appli-

cations where the user's behavior should be restricted as little as possible. On the other hand, many hand tracking methods, allowing free hand poses, resort to a simple (skin-)color based tracking approach due to the lack of other discriminative and easily exploitable features [Asaari 10,Pham 09,Donoser 08,Kölsch 10,Shan 07,Bilal 12, Dawod 10, Soriano 00, Spruyt 10a]. These methods however, tend to fail in unconstrained environments with changing background and illumination.

With the advent of cheap, commercially available time-of-flight and structured light cameras over the past few years, many researchers circumvent the above problem by incorporating depth information into their algorithms [Huang 12, Oikonomidis 11,Melax 13]. Depth maps greatly simplify the problem by allowing instant removal of background pixels. Nevertheless, depth cameras usually rely on infra-red signals and perform poorly in direct sunlight or in environments with other infra-red light sources, which limits their applicability. Additionally, these methods can not be applied to existing datasets lacking depth data.

In this chapter, we propose a real-time hand detection and tracking system for monocular video, that could easily be combined with depth information when available, but can also operate well without it. Our approach includes methods for automatic initialization and error recovery, and yields a rough segmentation of the tracked hand.

Object tracking in general is a widely researched domain in computer vision. Object tracking can be considered a temporal extension of object detection. In each video frame, the object of interest is re-detected, and temporal information about past detections is used to overcome ambiguous observations, thereby decreasing the error rate.

An important aspect of object tracking that is generally ignored in literature, is initialization and error recovery. Robust tracking methods often model the object of interest dynamically, such that the model adapts to changing illumination or appearance. However, non-uniform illumination, caused by various unknown light sources, is difficult to model accurately. As a result, these tracking methods suffer from drift, caused by the accumulation of small estimation errors that are introduced during each model update. Academic literature often ignores this problem by evaluating the tracking methods on very short video sequences or on video sequences that are captured in controlled environments.

Due to the accumulation of tracking drift or due to unexpected environment changes, trackers tend to fail too frequently in long-term object tracking. Furthermore, even if a tracking method would be able to detect drift, current tracking algorithms are usually unable to automatically re-initialize. The reason for this is that automatic initialization requires a robust object detector that operates without temporal information. If such a detector were readily available, any object tracking problem would simply be solved by re-detecting the object in each video frame, thereby rendering the research field of object tracking useless.

Literature usually treats object tracking and object detection as two independent research domains. Yet tracking methods can not be initialized without a robust detector, and a detector is generally not robust without temporal information. It is clear that both topics are closely related, and are better solved jointly.

Similarly, object segmentation is closely related to object tracking. Object segmentation is the task of obtaining the contour of the object of interest, thereby separating the object from its background. Whereas simple background subtraction techniques are unable to cope with moving cameras, skin color based segmentation often fails in changing lighting conditions, and edge based segmentation techniques tend to fail in case of motion blur due to fast hand motion. We therefore propose a method that iteratively performs detection, tracking and segmentation, feeding back new information to automatically adapt our probabilistic models and to automatically recover from errors.

In most human-computer interfaces, all three steps (hand detection, hand tracking and hand segmentation) are merely a small part of the complete system, and should only consume a fraction of the available processing power. This low latency and low complexity constraint is not met by many techniques from literature, and proves to be a challenging problem.

In the following, we propose a complete, real-time detection, tracking and segmentation system, that is able to robustly track hands in unconstrained environments and changing illumination, against cluttered backgrounds, by using a simple monocular camera. The proposed algorithm automatically detects hands to initialize the tracker, and can cope with a dynamic number of tracked objects in the scene. Furthermore, error recovery procedures automatically detect drifted or failed tracker instances. We show that our method outperforms the state-of-the-art in hand tracking, while providing a low computational complexity, allowing real-time processing.

The chapter is outlined as follows: In Section 2, we discuss related work. Section 3.1 provides an introduction into Bayesian tracking methods such as Kalman filters and particle filters, each of which are used by our tracking system. Section 3.2 proposes a method to partition our state space in order to reduce the dimensionality of the tracking problem. In Section 3.3, the observation and likelihood models of our particle filter approach are detailed, and in Section 3.4 we discuss our motion model and propose several improvements to the traditional particle filter approach. In Section 3.5 we propose the integration of the detection, tracking and segmentation components into a closed loop tracking system with automatic initialization and error recovery. Finally, Section 4 presents an extensive evaluation of our methods, and a discussion of the results.

2 Related Work

Continuous hand tracking is a challenging task, due to the fast, non-linear motion of human hands and the changing backgrounds and illumination in unconstrained environments. Although visual object tracking in general is a well researched topic, tracking hands proves to be difficult, even for state-of-the-art tracking methods. Specifically, no single feature is able to unambiguously discriminate a human hand from its background. Nevertheless, most hand tracking methods try to model the hand appearance by means of its color distribution [Asaari 10, Pham 09, Donoser 08, Kölsch 10, Shan 07, Bilal 12, Dawod 10, Soriano 00, Spruyt 10a]. Skin color is known to occupy a well defined region in color space [Gomez 02], often referred to as the 'skin locus', and can therefore be modeled efficiently. However, pure skin-based color trackers tend to fail if other skin-like objects such as faces are present in the scene.

In [Spruyt 10a], we proposed a skin color based hand tracking method that combines multiple color spaces to obtain a certain degree of illumination independence. Skin color was modeled by means of a Gaussian Mixture Model, and a particle filter was used for temporal smoothing. While the results of this work were promising, the tracking method only relied on first order statistical information by randomly sampling color values and evaluating their likelihood. As a consequence, the particles of this particle filter implementation were only defined by their spatial location (x, y), and did not capture any information about their surroundings, such as background colors around the object, or the size of the object of interest. As a result, this method tends to be sensitive to outliers in the skin color likelihood function and the tracker is easily distracted by skin-like objects in the background.

To increase tracking robustness, Shan *et al.* [Caifeng 04] proposed the integration of the mean-shift optimization algorithm within a skin-color based particle filter framework. Mean-shift is an optimization procedure that finds the local mode of a discrete distribution. By incorporating a mean-shift iteration after assigning likelihoods to each sample in the particle filter framework, less particles are needed to efficiently model the distribution as each particle is essentially pushed towards the local maximum in the skin likelihood function. However, the approach of integrating the mean-shift algorithm into the particle filter framework is rather ad-hoc and is difficult to extend to non-conventional particle filter implementations in which a specific proposal distribution is used. In Section 3.4 we extend the idea by generating a proposal distribution using a mean-shift embedded Kalman filter. Shan *et al.* further defined a constant velocity motion model. While they clearly show the advantages of a mean-shift embedded particle filter (MSEPF), their hand tracking method itself fails to robustly track hands in unconstrained environments, due to their simple motion and observation models.

A constant velocity motion model simply predicts the next hand location based on the previous hand location and a current velocity estimate. Due to the irregularity of hand motion, this prediction is often plain wrong, resulting in tracking errors. To solve this, Kölsch and Turk [Kölsch 05] developed a deterministic tracking system that incorporates sparse optical flow estimates into the motion model. Their method, referred to as the HandVu system, tracks hands using a combination of color and Harris corner feature matching. Instead of randomly sampling image points to evaluate the color likelihood, they evaluate the likelihood at corner points as returned by a Harris corner detector. These corner points are tracked between consecutive frames, by means of the Kanade-Lucas-Tomashi (KLT) [Shi 94] tracking method and by relying on spatial constraints, defined as flocking behavior. A well known disadvantage of the KLT tracker however, is its reliance on the so called 'brightness constancy constraint', causing it to fail in environments with non-uniform or changing illumination. Furthermore, due to the lack of a distinct texture in human hands, Harris corners are usually found on the contour of a hand, describing the local gradients, and are therefore not stable enough for accurate tracking.

Each of the above methods depend on histogram based color based models that are used to estimate the posterior probability of a pixel representing skin. As a result, these algorithms can not distinguish between actual hands and skin-color like background objects. To solve this, Kölsch and Turk extended the original HandVu system by incorporating a discriminative Haar-classifier [Kölsch 10]. Instead of using the final binary decision of this classifier, Kölsch and Turk define a likelihood function, based on early-stage votes that result from the Haar cascade. This likelihood is then used as an appearance based prior in their tracking system, guiding the sampling process. Although this approach is rather sensitive to background clutter and greatly increases the tracker's computational complexity, part of our tracking method is loosely inspired by their idea of incorporating a discriminative classifier into the generative model of a probabilistic tracking algorithm.

Liu *et al.* [Liu 12] take this idea one step further by combining the HandVU tracker with a Random Hough Forest based detector. Their method integrates the flock of features tracking algorithm with the object detector proposed by Gall and Lempitsky [Gall 09]. Although this greatly increases robustness against background clutter, the proposed system can not cope with fast motion, changing illumination and hand-like objects such as human faces. Additionally, the grabcut [Rother 04] algorithm is used for segmentation purposes, such that the complete system can not be used for real-time human-computer interaction.

Independently of Liu *et al.*, we also proposed a particle filter based hand tracker that uses a Random Hough Forest detector to generate an appearance based prior that is incorporated into the likelihood model [Spruyt 12]. Our detector is scale and rotation invariant, and operates in real-time. We showed that this method

outperforms many of the state-of-the-art tracking algorithms when applied to hand tracking, while being competitive with state-of-the-art classifiers when applied to different problems such as person detection.

Liu *et al.*,showed that incorporating a segmentation step into the tracking algorithm can further increase robustness and allows the tracker to adapt to its environment. Using a similar idea, Wan *et al.* [Jun 12] combined an optical-flow based motion model with a grab-cut based observation model in a deterministic tracking framework. Like the work of Kölsch and Turk, their motion model relies on KLT feature tracking, and the Harris corner feature points are filtered using several heuristics. The estimated hand state is then used to initialize a grab-cut segmentation step, which incorporates energy terms representing skin color, motion cues and edge information. However, this method depends on the KLT feature tracking algorithm, which assumes a constant brightness, small motion vectors and spatial motion smoothness. Therefore, this method is not robust against sudden lighting changes and fast motion. Moreover, due to the use of advanced algorithms such as grab-cut and Chamfer matching, the proposed solution can only process a single frame per second, on modern hardware.

Whereas the above methods treat hands as individual objects, several solutions incorporate contextual information to increase tracking robustness. Buehler *et al.* [Buehler 08] try to fit a complete arm model to each video frame, and use this model to infer the position of the hand. The model is fitted to both frames in the past and frames in the future, and temporal tracking is applied between the keyframes in which arm detection succeeded. For this reason, their method can not be used for real-time tracking. Furthermore, the algorithm takes about two minutes of processing time per frame on modern hardware.

Based on this idea, Karlinsky *et al.* [Karlinsky 10] used a hierarchical chains model to detect and track articulated objects. The start node of this chain is configured by means of face detection, after which an iterative procedure automatically aligns the rest of the chain with the limbs of the person. Consequently, this method can only be used in situations where the face of the person is visible and can be detected robustly, and where the complete upper body of the person is contained inside the video frame.

Pfister *et al.* [Pfister 12] extended the work of Buehler *et al.* by learning a random forest classifier to find the limb joints in the image. While their method outperforms the method proposed by Buehler *et al.* in one specific dataset of video sequences, their algorithm strongly depends on a foreground-background segmentation step, which they developed and tuned specifically for these video sequences.

In this chapter, we propose a complete hand tracking framework. A major novelty is the introduction of a closed-loop detection-tracking-segmentation framework that adapts to its environment and automatically recovers from tracking failure. Secondly, we propose a theoretical framework to enhance the traditional par-

ticle filter by designing a strongly peaked proposal distribution. Our proposal distribution integrates tracking methods such as mean-shift, optical flow and Kalman filtering into the particle filter framework in a probabilistic manner. Furthermore, we propose a partitioning of the state space to avoid the curse of dimensionality when estimating both the location and the size of on object, and we introduce several enhancements to the traditional particle filtering approach in order to achieve real-time tracking. Finally, we provide an in-depth discussion of our observation model that is based on data-fusion concepts instead of relying simply on color histograms or motion detection. We show that our method can cope with background clutter and skin-like objects such as human faces. Furthermore, the proposed solution does not make assumptions about the scene and does not require the face, arms or body to be visible in the video.

3 Materials and Methods

Tracking methods such as Kalman filtering [Asaari 10], mean-shift tracking [Exner 10] and optical-flow based tracking [Spruyt 13b] each have their strengths and pitfalls. Many of these shortcomings are related to noise in the image, to the lack of a closed form solution to describe the expected hand motion, or to the difficulties in approximating the shape of the posterior distribution. Particle filters are known to be able to cope with such situations by incorporating noise in their models and by modeling the posterior distribution by a set of discrete samples.

However, instead of choosing one of the above tracking approaches, we propose a method to incorporate optical-flow, mean-shift and Kalman smoothing into the particle filter framework. Furthermore, we do this in a way that this combination also solves the largest problem with conventional particle filters; i.e. inefficient use of the particles by sampling regions of the image that are unlikely to contain the object of interest.

Incorporating the above methods in the observation model of the particle filter would not solve this problem. On the other hand, using these methods in the motion model of the particle filter would result in particle weights that have no probabilistic meaning, and would force us to abandon the Bayesian framework. Hence, we propose to incorporate these techniques into the importance sampling stage of the particle filter, by designing a new proposal distribution. To do this, we need to redefine the particle filtering problem in the context of importance sampling, instead of the widely used prediction-update concept of traditional bootstrap particle filters.

In the following section, we provide an introduction into Bayesian filtering methods such as Kalman and particle filtering to further develop the above idea.

3.1 Bayesian Filters: an Introduction

Tracking methods can be coarsely classified as being either deterministic or probabilistic. The Camshift algorithm [Exner 10] is a well known example of a deterministic method for visual tracking. Deterministic methods, such as the Camshift algorithm or the flocks of feature based tracker [Kölsch 10], typically return a predictable result that only depends on the system's input. These methods assume that their model of the state to be estimated is deterministic and complete. Probabilistic trackers such as particle filters on the other hand, incorporate randomness into their model and thus yield results that are a function of both the input and a random sample from a predefined probability distribution. These methods assume that their model of the state to be estimated is incomplete or non-deterministic, and accordingly model this uncertainty by a random noise variable from a known distribution.

Although deterministic tracking methods may intuitively seem to be preferable, they often exhibit a hit-or-miss characteristic. If the model used by the tracker is accurate, the result will be correct, while an inaccurate model automatically results in tracking failure. Due to the difficult problem of hand tracking, it is often impossible to take all possible parameters that describe the environment into account. Observations are influenced by camera parameters, illumination, background colors and dynamics, motion velocity and the user's appearance, amongst others. A probabilistic tracking algorithm explicitly assumes that the model used to describe the object of interest is insufficient, and that the influence of unknown parameters can be modeled as an additive noise term that is sampled from a chosen distribution.

For example, if the motion of an object would be modeled by a first order autoregressive model, while its actual motion follows an n-th order autoregressive model, then deterministic tracking methods would simply fail, while probabilistic tracking methods would be able to model the higher order additive terms by means of a white noise term.

Moreover, in visual tracking, observations about the object of interest, such as colors or gradient information, are almost always polluted by clutter and noise. Probabilistic tracking methods are able to incorporate this noise into their models, while deterministic methods assume perfect observations.

In the following, let $\mathbf{x} = (x_1, x_2, ...x_n)$ be the n-dimensional state vector to be estimated (e.g. position of a hand), and let $\mathbf{y}$ be an observation (e.g. an image). From a probabilistic tracking perspective, the problem of object tracking can be described by the following state space representation:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) : \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{m}_{t-1}) \tag{3.1}$$

$$p(\mathbf{y}_t|\mathbf{x}_t) : \mathbf{y}_t = h(\mathbf{x}_t, \mathbf{n}_t), \tag{3.2}$$

where $\mathbf{x}_t$ represents the state of the random variable of interest at time t, $\mathbf{y}_t$ represents the corresponding observation or output, $\mathbf{m}_t$ is the system noise, and $\mathbf{n}_t$ is the measurement noise. Whereas the system noise is a random variable that models the inaccuracies of the prediction model $f(.)$, the measurement noise models the discrepancy between an observed object position and the true object position and thereby represents the sensor noise. Thus, the state to be estimated is a function of the previous state and a noise component, whereas the observed output is a function of the current state and a noise component.

The goal of visual tracking is then to estimate the state $\mathbf{x}_t$, such that the belief in this state is maximized, given the past observations $\mathbf{y}_{1:t}$. To that end, Bayesian tracking methods estimate the posterior density function $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ based on equations (3.1) and (3.2). In theory, this estimate can be obtained recursively, in two stages: prediction and update.

In the prediction step, only the previous estimate $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ is known and no observations are available yet. In this step, the previous state estimate is propagated into the current time instance, by means of the state transition prior $p(\mathbf{x}_t|\mathbf{x}_{t-1})$:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{y}_{1:t-1}, \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1} \tag{3.3}$$

$$= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1}. \tag{3.4}$$

The equality between equation (3.3) and (3.4) is valid because equation (3.1) models a first order Markov process, such that the current state is independent of the previous observations given the previous state.

Once a new observation $\mathbf{y}_t$ becomes available, the update step is executed. During the update step, Bayes' rule is used to combine the latest observation y_t with the predicted state. This amounts to the computation of:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_{1:t}|\mathbf{x}_t) p(\mathbf{x}_t)}{p(\mathbf{y}_{1:t})} \tag{3.5}$$

$$= \frac{p(\mathbf{y}_{1:t-1}|\mathbf{x}_t) p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t)}{p(\mathbf{y}_{1:t-1}) p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \tag{3.6}$$

$$= \frac{p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) p(\mathbf{y}_t|\mathbf{x}_t)}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})}. \tag{3.7}$$

The equality between equation (3.5) and (3.6) again depends on the Markov assumption. This implies that observations are independent given the current state.

In the numerator of equation (3.7), the first factor is computed by equation (3.4) and is the result of the prediction step. The second factor corresponds to the likelihood function that is computed from an observation model and indicates how likely the current measurements y_t are, given the hypothesized state x_t. Finally, the

normalization factor in the denominator can be written as a function of these two known distributions, again using the Markov assumption such that observations are independent given the current state:

$$p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})d\mathbf{x}_t. \tag{3.8}$$

If this probability density function (PDF) is known to be normally distributed, and if both the motion model as defined by equation (3.1) and the observation model as defined by equation (3.2) are linear functions of their input, then the corresponding prior and likelihood distributions are inherently Gaussian too. In such case, where all distributions are Gaussian, and all transformations are linear, the integrals shown in equations (3.4) and (3.8) can easily be calculated analytically. The resulting algorithm is called a Kalman filter and will be of importance in Section 3.4.2.

However, in many practical problems the PDFs are not Gaussian, and the motion model or observation model is not linear. In visual tracking, for instance, multiple locations in the image might have an equal likelihood of representing the object, resulting in a multimodal distribution. Furthermore, motion models are often quadratic, or piecewise linear. In such case, an exact solution for these high dimensional integrations and thus for the posterior density, is hard to obtain. Instead, an approximate solution can be found by means of Monte Carlo sampling.

Particle filters do not try to solve the above equations but instead use Monte Carlo simulation to directly represent the posterior PDF as a weighted sum of N discrete samples (a.k.a. particles) as shown by equation (3.9).

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta(\mathbf{x}_t - \mathbf{x}_t^i), \tag{3.9}$$

where $\mathbf{x}_t^i$ is a random sample from this distribution $\mathbf{x}_t^i \sim p(\mathbf{x}_t|\mathbf{y}_{1:t})$, δ represents the Dirac delta function, and $w_t^i = \frac{1}{N}$. In practice, drawing samples from the posterior is impossible because the posterior distribution is exactly what we are trying to estimate. On the other hand, for a given observation $\mathbf{y}_t$, the likelihood $p(\mathbf{y}_t|\mathbf{x}_t^i)$ can easily be obtained from the observation model defined by equation (3.2).

If a distribution can not be sampled directly, but its likelihood can easily be evaluated, an approximation of this distribution can be obtained by means of importance sampling. Instead of sampling the posterior distribution, samples are drawn from any other distribution, called the proposal distribution, the support of which must include the support of the true posterior. The weights w_t^i of the samples from this proposal distribution are then obtained by evaluating these samples using the likelihood function, such that the weighted set of samples approximates the true posterior distribution.

Let the proposal density, also called importance density, be $q(.)$. According to Bayes' rule, we can write the posterior as $p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \alpha p(\mathbf{y}_{1:t}|\mathbf{x}_t)p(\mathbf{x}_t)$, where α is a normalization factor that is equal for all the samples drawn from $q(.)$. Then the importance weight of each particle can be calculated as shown by equation (3.10).

$$w_t^i = \frac{p(\mathbf{y}_{1:t}|\mathbf{x}_t^i)p(\mathbf{x}_t^i)}{q(\mathbf{x}_t^i|\mathbf{y}_{1:t})}. \tag{3.10}$$

Moreover, under the Markov assumption, these weights can be estimated recursively [van der Merwe 01] as

$$w_t^i = w_{t-1}^i \frac{p(\mathbf{y}_t|\mathbf{x}_t^i)p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)}{q(\mathbf{x}_t^i|\mathbf{x}_{1:t-1}^i, \mathbf{y}_{1:t})}. \tag{3.11}$$

The numerator of equation (3.11) is then simply the product of the likelihood (i.e. observation model) $p(\mathbf{y}_t|\mathbf{x}_t^i)$ and the state transition prior (i.e. motion model) $p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)$. The only remaining problem now is to choose the proposal distribution in the denominator of equation (3.11).

Particle filters that use recursive importance sampling to obtain a weighted, discrete approximation of the posterior distribution are called Sequential Importance Sampling (SIS) particle filters.

The optimal proposal distribution has been shown to be the one that minimizes the variance of the importance weights, conditioned on $\mathbf{x}_{1:t-1}$ and $\mathbf{y}_{1:t}$ [van der Merwe 01]. Nevertheless, in practice it is usually impossible to calculate this optimal distribution. Most current particle filter implementations, such as the well known CONDENSATION algorithm [Isard 98], therefore simply use the state transition prior $p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)$, returned by the motion model, as the proposal distribution. Particle filter implementations that employ this strategy are called bootstrap filters and are the most widely used particle filter variant.

Bootstrap filters ignore the fact that the proposal distribution is conditioned on the latest observation. Instead, they assume that the current state is only a function of the previous state and is independent of the latest observation. In other words, the posterior is assumed to change smoothly over time thereby closely resembling the transition prior at each time step t. Plugging the state transition prior into equation (3.11) results in a greatly simplified weight calculation:

$$w_t^i = w_{t-1}^i \frac{p(\mathbf{y}_t|\mathbf{x}_t^i)p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)}{p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)} = w_{t-1}^i p(\mathbf{y}_t|\mathbf{x}_t^i). \tag{3.12}$$

The proposal distribution is cancelled out in the bootstrap filter equation, and as a result only a prediction and an update needs to be performed. Although bootstrap filters are the most widely used type of particle filter, failing to introduce the latest observation into the proposal distribution has several problematic disadvantages.

In Section 3.4.2 we propose a solution to these by abandoning the bootstrap filter idea, and designing a more appropriate proposal distribution.

The complete set of particles represents a probability mass function, such that the final state estimate $\bar{\mathbf{x}}$ can be obtained by searching for the mode of this discrete distribution, or by calculating the average state of all particles if the distribution is Gaussian:

$$\bar{\mathbf{x}} = \mathrm{E}(\mathbf{x}_i) = \frac{\sum_i^N w_i \mathbf{x}_i}{\sum_i^N w_i}. \tag{3.13}$$

At each time instance t, some particles that represent a close approximation to the actual state will be assigned a high weight, whereas others will be assigned a low weight. At the next time instance, $t+1$, particles with a low weight will probably be assigned an even lower weight, since the new estimate is based on the bad state approximation of the previous time instance t. As a result, after a while, only few particles with a corresponding high weight will model the mode of the posterior, whereas most of the particles with corresponding low weight will model the tails of the posterior distribution.

This problem is called particle depletion [Shan 07]. Sparse sampling of the mode of the posterior distribution causes the filter to get easily distracted by noise towards the tails of the distribution. To solve this problem, resampling techniques are often used to replicate particles with high weights, while removing particles with a low weight. SIS particle filters that employ resampling are called Sequential Importance Resampling (SIR) filters.

A widely used resampling scheme is multinomial resampling. In multinomial resampling, the set of N new particles is sampled with replacement from the set of N old particles. The probability that a particle i is picked at each sampling round is defined by its weight w_i. After resampling, all weights are set to $w_i = \frac{1}{N}$ such that the new sample set represents the same distribution as the original sample set before resampling.

Although resampling solves the particle depletion problem, it introduces a new problem called particle impoverishment [Shan 07]. After resampling, most samples are located on or around the peaks of the distribution, while few samples are available to represent the tails. As a result, the tracker fails to explore the complete search space and easily gets stuck on local maxima. As a compromise, most SIR filters only perform resampling when needed, instead of resampling at every time instance t.

In order to determine when resampling should be performed, an estimate is needed for the performance of the particle filter state estimation at each time step. Neal [Neal 01] showed that the variance of the estimator can be approximated as follows:

$$\mathrm{Var}(\hat{\mathbf{x}}) = \frac{\mathrm{Var}(\mathbf{x}_i)(1 + \mathrm{Var}(w_i))}{N}. \tag{3.14}$$

If we were able to draw N samples directly from the posterior distribution, instead of the proposal distribution, all weights would be $w_i = \frac{1}{N}$ and the variance of the estimator would be:

$$\mathrm{Var}(\hat{\mathbf{x}}) = \frac{\mathrm{Var}(\mathbf{x}_i)}{N}. \tag{3.15}$$

Although we can not directly sample from the posterior, an estimator with the variance of equation (3.15) could be obtained by increasing the number of samples N in equation (3.14). A good measure of the performance of the particle filter would thus be the number of samples N_1 that would be needed by a particle filter that would draw its samples directly from the posterior, in order to obtain a variance that is equal to the variance observed in the current particle filter with N_2 particles:

$$\frac{\mathrm{Var}(\mathbf{x}_i)(1 + \mathrm{Var}(w_i))}{N_2} = \frac{\mathrm{Var}(\mathbf{x}_i)}{N_1} \tag{3.16}$$

$$\Leftrightarrow N_2 = \frac{N_1}{1 + \mathrm{Var}(w_i)}. \tag{3.17}$$

This number N_2 of independent and identically distributed (i.i.d.) particles that would be needed by an ideal sampler to obtain the current variance is called the Effective Sample size (ESS) and can be further simplified as follows:

$$ESS = N_2 = \frac{\left(\sum_{i=1}^{N} \hat{w}_t^i\right)^2}{\sum_{i=1}^{N} (\hat{w}_t^i)^2} = \frac{1}{\sum_{i=1}^{N} (\hat{w}_t^i)^2}, \tag{3.18}$$

where $\hat{w}_t^i$ is the normalized weight, such that the sum of all weights equals 1. If all particles are assigned the same weight, the effective sample size equals the real sample size. If a single particle has a weight of 1 and all other particles have a weight of 0, then the effective sample size is 1. In order to avoid impoverishment, most SIR filters therefore only resample if $ESS < \alpha\, N$ with $\alpha \in [0, 1]$.

In general, we want to resample as little as possible, while still being able to model the mode of the posterior density accurately. Hence, a proposal distribution that closely matches the posterior distribution is extremely important as it greatly reduces the need for resampling. In Section 3.4.4 we further extend our proposal distribution to accomplish this.

In the next sections we discuss our observation and motion models, and we propose solutions to several of the problems described above. Moreover, we propose several design choices to reduce the computational complexity of the system, in order to accomplish real-time tracking.

3.2 State Space Definition and Partitioning

Since hands are highly articulated and non-rigid objects, a complicated model would be needed if a deterministic tracker were to be used. Instead, we use a prob-

abilistic particle filter, and propose to coarsely model the human hand by means of its bounding box. Using such a simple model allows the tracker to operate in real-time, since many calculations can be simplified, as will be explained later. The orientation of this box, and a fine grained segmentation of the hand, can easily be obtained during a postprocessing step. The bounding box, and thus the unknown state $\mathbf{b}$, is parameterized as follows:

$$\mathbf{b} = (x, y, w, h), \tag{3.19}$$

where x and y represent the hand location, whereas w and h represent its width and height respectively.

However, even with the coarse simplification of representing the human hand by its bounding box, the search space is still 4-dimensional. Since particle filters depend on random sampling, the accuracy of the estimated posterior distribution heavily depends on the sample density in this search space. Due to the curse of dimensionality, the number of samples needed to achieve a certain sample density, increases exponentially with each added dimension. If 30 particles would suffice to sample a 1-dimensional unit interval, then $30^4 = 810\,000$ samples would be needed to sample a 4-dimensional unit hyperspace with the same sample density.

MacCormick *et al.* [MacCormick 00] proposed a technique to reduce this dimensionality in case a hierarchical relation exists between the parameters to be estimated. As an example, they showed how the finger position, hand location and arm orientation can be tracked sequentially instead of simultaneously. We propose to use this concept of partitioned sampling to reduce our 4D tracking problem to two connected 2D tracking problems.

Partitioned sampling can be considered the statistical equivalent of a hierarchical search, in which the child node in a tree depends on the state of its parent. Instead of simultaneously estimating the joint state of parent and child, partitioned sampling proposes to first estimate the parent's state, and to use this estimate to obtain an estimate of the child's state. After estimating the parent's state, the state space is resampled using the parent's likelihood as an importance density, in order to populate the regions of highest likelihood in the known state partition. The child's state is then estimated, using this resampled particle set.

To be able to apply partitioned sampling, the state space $\mathbf{b}$ must be partitionable into a Cartesian product $\mathbf{b} = \mathbf{b}_1 \times \mathbf{b}_2$. Furthermore, the dynamics f or the motion model, must be decomposable as $f = f_1 * f_2$, where $*$ represents convolution, and where f_1 acts on $\mathbf{b}_1$, while f_2 acts on $\mathbf{b}_2$. Finally, a factorization of the observation model h must be found, such that each factor is peaked in the same region as the posterior of the corresponding partition $\mathbf{b}_i$.

In other words, independent subsets of state variables must exist in the original state $\mathbf{b}$. In our case, it is safe to assume that the 2D location (x, y) of the hand is independent of the hand's size (w, h). Consequently, we can split the state space

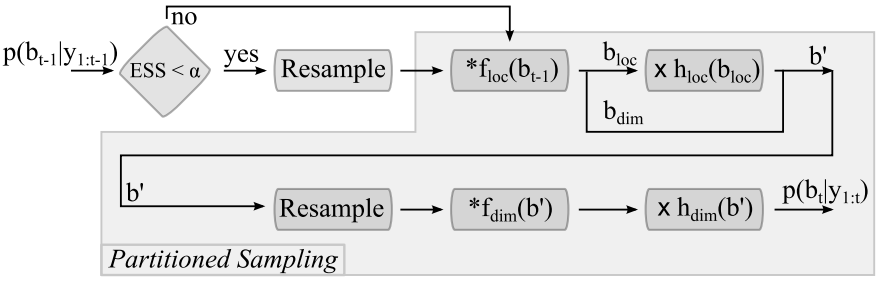


Figure 4.1: Schematic representation of the partitioned sampling approach.

$\mathbf{b}$ into $\mathbf{b}_{loc} = (x, y)$ and $\mathbf{b}_{size} = (w, h)$. We also define a motion model f_{loc} that acts on $\mathbf{b}_{loc}$, and a motion model f_{size} that acts on $\mathbf{b}_{size}$. Similarly, an observation model h_{loc} is defined to represent the likelihood of $\mathbf{b}_{loc}$, and an observation model h_{size} is defined to represent the likelihood of $\mathbf{b}_{size}$.

The first state partition, $\mathbf{b}_{loc}$, is predicted by motion model f_{loc}, after which the resulting distribution is resampled such that the majority of samples occupy the region of maximum likelihood according to the corresponding observation model h_{loc}. This weighted resampling can easily be implemented by first assigning weights to each particle based on the observation model h_{loc}, and then executing resampling like any conventional SIR particle filter.

Resampling results in a new set of samples describing state $\mathbf{b}'$. These samples are then used by the second motion model, f_{size}, to predict the width and height of the hand. After prediction, the corresponding observation model, h_{size}, is used to assign weights to each of the particles, and finally the resulting particle set is resampled again if the effective sample size (ESS) is smaller than a certain threshold.

This process of partitioned sampling corresponds to factorization of the likelihood function as shown in equation (3.20).

$$h(\mathbf{b}_t) = h_{loc}(\mathbf{b}_{loc}) h_{size}(\mathbf{b}'), \tag{3.20}$$

where $\mathbf{b}'$ is the state estimate from the first motion model f_{loc}.

Furthermore, the motion model is decomposed hierarchically, such that the second motion model, f_{size} acts upon the resampled particles representing $\mathbf{b}'$, as shown by equation (3.21).

$$f(\mathbf{b}_{t-1}) = f_{loc}(\mathbf{b}_{t-1}) * f_{size}(\mathbf{b}'). \tag{3.21}$$

This complete process of partitioned sampling is illustrated graphically by figure 4.1.

Summarized, we converted the 4-dimensional problem into a sequence of two 2-dimensional problems which can be solved much easier. This enables our tracker

to robustly track hands using only 50 particles while maintaining a sample density in each partition that is equal to the density that would be obtained when using $(\sqrt{50})^4 = 2500$ particles in the original 4-dimensional problem. Thus, this approach greatly reduces the computational complexity of the tracking algorithm.

3.3 Observation Model

In this section, the observation models h_{loc} and h_{size} that are needed to assign weights to the particles are discussed. An observation model is a function of the current state, as defined by equation (3.2), and is used to calculate the likelihood of this state, given the observation.

3.3.1 Skin Detection and Color Space Selection

The most distinctive cue that can be used to describe the appearance of human hands is color. Skin color has been shown to occupy a small cluster of possible values in the complete color space [Soriano 00], often called the skin locus, and can be used to quickly discard background pixels in the video frame.

To illustrate this, figure 4.2 shows a scatter plot of skin-pixels in a 2D color space. The skin pixels were gathered from over 4000 images using the well known Compaq dataset [Jones 02]. These images have been segmented manually and were randomly selected using a web-crawler. The dataset contains pictures of persons of different ethnicities and skin colors.

For visualisation purposes, the 3D RGB color space has been projected onto a 2D linear subspace. This subspace is spanned by the two largest eigenvectors of the covariance matrix of the skin-pixel data, and is obtained using PCA. Figure 4.2 shows the convex hull of the projected RGB cube, together with the skin-pixels themselves, and clearly illustrates the concept of the 'skin locus'.

Phung *et al.* [Phung 05] showed that skin color of different races (black, yellow, brown, white) have very similar chrominance, whereas only the skin color luminance varies widely. This means that the skin color in a picture of a white person, captured in a dark scene, is similar to the skin color of a black person, whose picture was captured in a bright scene. As a result, all skin color occurrences, across races, form a tight and continuous region within the RGB color space, as can be seen clearly in figure 4.2.

However, color is inherently sensitive to illumination changes, causing a shift and scaling of the skin color of a specific individual, and should therefore be combined with other cues to achieve robust hand tracking in unconstrained environments.

A well known method to model color in images, taking into account the illumination, is based on the Lambertian reflectance assumption, which states that the surface reflects light with equal intensity in all directions [Van de Sande 10]. The

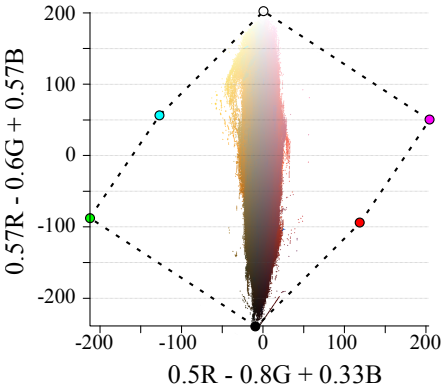


Figure 4.2: Skin locus (RGB color space projected onto a linear subspace).

image $I(x, y)$ can then be represented by equation (3.22).

$$I_i(x, y) = \int_\lambda c(\lambda) s_i(\lambda) r(x, y, \lambda) d\lambda, \tag{3.22}$$

where λ is the light's wavelength, $c(\lambda)$ represents the color of the light source, $s_i(\lambda)$ with $i \in \{R, G, B\}$ defines the sensitivity of the camera sensors, and $r(x, y, \lambda)$ is the reflectivity of the object at location (x, y). While more advanced models exist, this simple model illustrates two of the most important problems in color based object tracking.

The first problem is that the color in the image $I(x, y)$ depends on the sensitivity $s_i(\lambda)$ of the camera sensors. In practice, the sensitivity curves of the red, green and blue sensors tend to overlap, which introduces significant correlations between the color channels. Accordingly, the R, G and B channels of the image should not be modeled by three marginal probability distributions, but with a single joint distribution instead. Modeling the colors by their joint distribution would require exponentially more training data than modeling three independent 1D distributions, due to the curse of dimensionality.

The second problem that can be noticed in equation (3.22), is that the color of a pixel clearly depends on the color and intensity $c(\lambda)$ of the light source, and not only on the reflectivity of the object itself. Moreover, the intensity of the light source is a function of the wavelength. This means that an increase in white illumination results in a scaling of the R, G and B components with a single scaling factor. An increase in illumination that is not white however, results in different, unknown scaling factors for the R, G and B components. Moreover, although not modeled by a simple Lambertian model, changes in diffuse light can cause color and intensity shifts. As a result, the color of a pixel consists of three dependent random variables that are all scaled and shifted by unknown parameters, due to environmental illumination.

To solve the second problem, i.e. illumination dependence of the color values, most color based tracking implementations [Asaari 10, Bretzner 02, Kölsch 04a, Shan 07, Caifeng 04] convert the image from RGB color space to HSV (Hue-Saturation-Value) color space or to a similar color space such as YCbCr, in which the two chrominance components are separated from the luminance component. A 2D Hue-Saturation histogram is then created by dropping the luminance component, i.e. the Value, in the hope of reducing the illumination dependency.

The first problem, i.e. the need for a joint PDF and the corresponding dimensionality issue, is often solved in a similar manner: Dropping the luminance component reduces the dimensionality of the problem, making it possible to calculate the joint Hue-Saturation histograms using a limited amount of training data. The chrominance components are often quantized to further reduce the number of histogram bins, in an effort to avoid the curse of dimensionality.

However, Phung *et al.* [Phung 05] showed that the choice of color space does not have a significant impact on skin color detection. In fact, skin classification in RGB space performs equally well as skin classification in HSV space, and outperforms color spaces in which the luminance information is dropped if a large amount of training data is available. The reason is that standard color transformations that perform a one-to-one mapping, do not introduce new information and do not guarantee uncorrelated color channels.

If only a limited amount of training data is available on the other hand, skin detection in HS space (dropping the Value component) outperforms detection in HSV or RGB space. However, as opposed to what is often assumed, this performance increase is not due to an increased robustness to illumination changes. Instead, it can be explained simply by the curse of dimensionality, since the reduced dimensionality avoids sparse histogram bins that would result in overfitting of the classifier. This statement is supported by the fact that skin detection in a quantized RGB space (reducing the number of histogram bins) yields the same detection results as skin detection in the non-quantized HS space.

Summarized, although conversion to another color space such as HS(V) or (Y)CbCr could accidently result in uncorrelated color channels, and can certainly help in reducing the number of dimensions, the general assumption that these color spaces are illumination invariant is incorrect.

Color normalization techniques such as white patch normalization or gray world normalization [Rizzi 02, Ebner 07] could increase illumination independence in some cases, but usually introduce new problems because these techniques only try to remove either the multiplicative or the additive illumination component. Additionally, normalization inherently decreases the discriminativeness of the color feature.

3.3.2 Adaptive Skin Detection

Based on the above, we propose the combination of three different skin classifiers. The first classifier is a static skin classifier that was trained offline using a large set of images in quantized RGB space. The second classifier is a slowly adapting classifier that models skin color appearance in the past few video frames using the quantized HS color space (dropping the Value component). The final classifier is a very adaptive classifier that models the skin color in the previous video frame using only the quantized Hue component of the HSV color space. Due to the angular representation of color in the HSV color space, it is clear that the red range becomes discontinuous if the hue is described using a single numerical value. Since the red range is the most prominent in skin-colored pixels, we therefore shift the hue into the range $-30^\circ < \tilde{H} < 330^\circ$, such that the red range becomes continuous: $\tilde{H} = (H - 30) \bmod 360$.

Thus, our skin classifiers that have more training data at their disposal use a more fine grained model than the classifiers that only have few training pixels. The second, dynamic, classifier is constantly re-trained using the skin pixels within the bounding box of the previous hand detections. Since these bounding boxes can be small, only few pixels are available in each hand bounding box. Therefore, modeling the skin color for this classifier in full RGB space, instead of quantized HS space, would increase the risk of overfitting and would result in sparse histograms. Similarly, using quantized HS space instead of the quantized Hue component for the third, dynamic, classifier would again result in sparse histograms because it is only trained on the skin pixels of the previous video frame.

For the RGB skin classifier, a 3D histogram of skin pixels and a 3D histogram of non-skin pixels is created, each consisting of 32 uniformly spaced bins. While a larger number of bins could result in a more discriminative classifier, a smaller number of bins results in less memory storage and faster computation times. Phung *et al.* [Phung 05] showed that 32 bins is an acceptable compromise between memory storage and recognition rate. For the online HS classifier, we use a 16-bin 2D histogram, because less training data is available to populate the histogram. Similarly, a 16-bin 1D Hue histogram is used for the third classifier.

While the choice for the Hue-Saturation color space, when learning the online histogram, is mostly driven by the need for fast dimensionality reduction, an added advantage is that the combination of multiple color spaces such as RGB and HSV, leads to superior skin classification, as we showed in earlier work [Spruyt 10a].

To train the offline static skin classifier, we used the well known Compaq database [Jones 02], consisting of over 4000 manually segmented skin colored images, and over 8000 images that do not contain any skin segments. We define the random variable $skin \in \{1, 0\}$ that represents the skin-membership of a pixel $\mathbf{i} = (r, g, b)$, where r, g, b are the red, green and blue color values in RGB color space.

In the following, let $\neg$ denote the logical complement of a binary random variable. By simply counting the number of skin pixels N_{skin} and the total number of pixels N in the training dataset, we obtain the prior probabilities shown by equations (3.23) and (3.24).

$$P(skin) = \frac{N_{\text{skin}}}{N} = 0.16 \tag{3.23}$$

$$P(\neg skin) = 1 - P(skin) = 0.84 \tag{3.24}$$

Furthermore, a normalized 32-bin histogram $H_{\text{skin}}(r, g, b)$ is created by counting the number of occurrences $N_{\text{skin}}(r, g, b)$ of each (r, g, b) triplet in the skin-regions of the training dataset, and a similar normalized histogram, $H_{\neg skin}(r, g, b)$, is created by processing all non-skin pixels in the dataset. These histograms represent the probability density functions shown by equations (3.25) and (3.26).

$$P(r, g, b|skin) = \frac{H_{\text{skin}}(r, g, b)}{N_{\text{skin}}(r, g, b)} \tag{3.25}$$

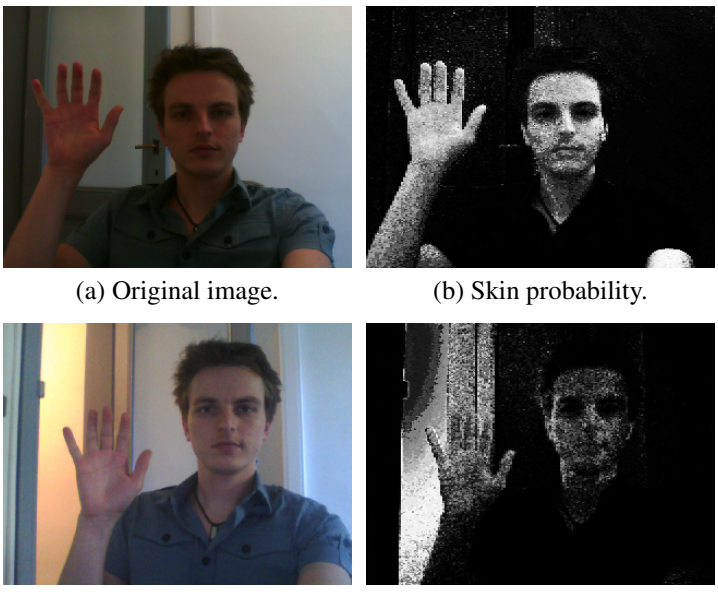
$$P(r, g, b|\neg skin) = \frac{H_{\text{nonskin}}(r, g, b)}{N_{\text{nonskin}}(r, g, b)} \tag{3.26}$$

Using Bayesian theory, we can now calculate the posterior probability that a certain (r, g, b) triplet, found in a video frame, represents skin color, as shown in equation (3.27).

$$\begin{aligned} P(skin|r, g, b) &= \frac{P(r, g, b|skin)P(skin)}{P(r, g, b)} \\ &= \frac{P(r, g, b|skin)P(skin)}{\sum_{\text{skin}} P(r, g, b|skin)P(skin)} \\ &= \frac{P(r, g, b|skin)P(skin)}{P(r, g, b|skin)P(skin) + P(r, g, b|\neg skin)P(\neg skin)}. \end{aligned} \tag{3.27}$$

Although more advanced skin classification methods exist, such as those based on Markov Random Fields, Phung *et al.* [Phung 05] showed that a simple Bayesian skin classifier outperforms more advanced algorithms if a very large training dataset is available. More importantly, both the likelihoods and the normalization factor of equation (3.27) can be precalculated and stored efficiently in lookup tables, yielding an extremely fast classifier, suited for real-time applications.

Figure 4.3 (b) illustrates the result of the Bayesian skin classifier in normal illumination conditions, with a neutral background. Figure 4.3 (d) shows the result when the foreground lights are dimmed, and when skin-like colors occur in the background. The second result clearly shows the need for other features and online histogram adaption, while the first model illustrates the discriminativeness of the skin classifier in average illumination conditions.



(a) Original image. (b) Skin probability.

(c) Original image. (d) Skin probability.

Figure 4.3: The second column shows the result of Bayesian skin classification, where a bright pixel corresponds to a high skin probability and a dark pixel corresponds to a low skin probability. Top row: an example of successful Bayesian skin classification. Bottom row: an example of classification failure due to background illumination.

Apart from the offline trained static RGB skin classifier, two online classifiers are created and constantly updated. For the first online classifier, the histograms are updated slowly, while the histograms of the second online classifiers are updated every frame. The fast adapting histogram allows the tracking algorithm to cope with sudden illumination changes, while the slow adaptive histogram models gradual changes in appearance and stabilizes the tracker. The static RGB histogram helps to avoid drift by incorrect updates of the online histograms, as will be explained on page 122.

The online histograms are initialized by a Haar-cascade based face detector, or by the first detected hand that results from the Random Hough Forest detector (explained in Section 3.3.4), in case no face detection result is available. The slowly adapting histograms are updated based on previous tracking results, by only incorporating pixels that fall within the hand segmentation region, as explained in Section 3.5.1, and that have a high skin probability. The histograms are only updated once a certain number of observations is available, in order to avoid overfitting to a single observation. The fast adaptive histogram, on the other hand, is updated every frame, by incorporating all pixels within the bounding box of the hand from the previous frame.

Each skin classifier results in a skin probability map. The final probability image, P_{final}, used in the observation model of the particle filter, is a linear combination of the three individual probability maps, P_{static}, P_{slow} and P_{fast} as shown by equation (3.28).

$$P_{final} = \alpha_1 P_{static} + \alpha_2 P_{slow} + \alpha_3 P_{fast}, \tag{3.28}$$

where $\sum_1^3 \alpha_i = 1$.

When tracking starts, or after a tracking error was detected, the online histograms are not trained well yet, and the static histogram is the only reliable color cue. The longer the tracker is able to correctly track the hands, the better the online histograms represent their appearance. Therefore, the weights are initially defined as $\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 0$. The weight of the fast updated histogram, α_3, is incremented after each successfully tracked frame, while the weight of the static histogram, α_1 is decremented after each successfully tracked frame. The weight of the slowly adapting histogram is then calculated as $\alpha_2 = 1 - (\alpha_0 + \alpha_1)$. The weights are capped, such that they satisfy the condition $\alpha_0 > \alpha_1 > \alpha_2$. This condition helps avoiding tracker drift that can be caused by incorrect updates of the online models.

Figure 4.3 shows that a simple skin probability map, representing $P(skin|r, g, b)$, is not sufficient to cope with skin-like background pixels. To solve this, we would like to obtain a probability map that represents $P(skin, FG|r, g, b)$ for each pixel, where $FG \in \{1, 0\}$ is a random variable that indicates whether or

not the pixel is part of a foreground object:

$$P(skin, FG|r, g, b) = P(FG|skin, r, g, b)P(skin|r, g, b). \tag{3.29}$$

Hence, a new probability map S_{FG}, that represents $P(FG|skin, r, g, b)$, is needed. If we ensure that FG is conditionally independent of (r, g, b) given $skin$, equation (3.29) reduces to equation (3.30).

$$S_{FG} \sim P(skin, FG|r, g, b) = P(FG|skin)P(skin|r, g, b). \tag{3.30}$$

To obtain the foreground probability map, representing $P(FG|skin)$, we first assume that all skin-like pixels are part of the background. Consequently, the probability that a pixel is part of the foreground is calculated as $P(\neg skin|r, g, b) = 1 - P(skin|r, g, b)$ and stored in a foreground map. Since pixels that are part of the tracked hand region are definitely foreground, all pixels within the bounding box of each tracked hand in the previous video frame are assigned a probability of one in this foreground map. The resulting map represents the probability $P(FG|skin)$ that a pixel is foreground, conditioned on its skin-likelihood.

To cope with noise and motion, this foreground probability map is slowly adapted in time by exponentially weighted temporal averaging over several frames. This is a simple technique that is often used by background subtraction methods. The smoothed model thus contains high probability values for skin pixels that are part of a hand, and low probability values for skin pixels that are not part of a hand, and can therefore be considered to model $P(FG|skin)$. The skin-foreground probability map S_{FG} can now be obtained as shown by equation (3.30), by multiplying the skin probability map $P(skin|r, g, b)$ and the foreground probability map $P(FG|r, g, b)$.

Figure 4.4 (b) illustrates the skin probability map representing $P(skin|r, g, b)$. Figure 4.4 (c) shows the foreground probability map, representing $P(FG|skin)$. Figure 4.4 (d) shows the combined skin-foreground probability map, representing $P(skin, FG|r, g, b)$. The background model clearly helps the particle filter to successfully track the hand in cluttered environments with skin-like backgrounds. Figure 4.4 (b) illustrates the skin probability map representing $P(skin|r, g, b)$.

3.3.3 Motion Detection and Dynamic Scenes

To further increase tracking robustness in environments with changing illumination or skin-like backgrounds, the observation model is designed such that moving objects are assigned a higher likelihood than static objects with the same skin probability. To detect motion in the video frame, we perform temporal frame differencing, which is simple to calculate and can easily be parallelized. The resulting frame difference is smoothed by a Gaussian filter, and thresholded to obtain a motion mask that indicates the moving regions in the image.

(a) Tracking result. (b) Skin probability map.

(c) Foreground probability map. (d) Combined probability map.

Figure 4.4: Combining foreground modeling and skin-color modeling. Figure (b) shows the original skin probability map, where a bright pixel corresponds to a high skin probability and a dark pixel corresponds to a low skin probability. Figure (c) shows the foreground probability map representing $p(FG|skin)$. All skin-like pixels (e.g. face) that are not part of the foreground have a low foreground probability, whereas skin-like pixels (e.g. hands) that are part of the foreground have a high foreground probability.

Similar to the approach of Shan *et al.* [Shan 07], a skin-motion map, S_{motion} is calculated by multiplying the skin probability map with the motion mask. This skin-motion map represents the joint probability that a pixel is part of a foreground skin region and part of a moving region in the image:

$$\begin{aligned} S_{motion} &\sim P(skin, FG|r, g, b)P(motion|r, g, b) \\ &= P(skin, FG, motion|r, g, b). \end{aligned} \tag{3.31}$$

The final skin probability based cue C_{skin}, used in the observation model of the particle filter, is then calculated as a linear combination of the skin-motion map S_{motion}, and the original skin-foreground probability map, S_{FG}:

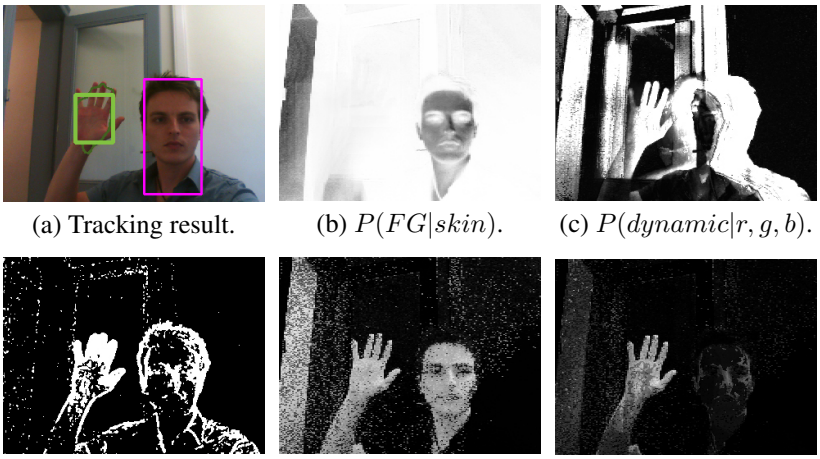
$$C_{\text{skin}} = \alpha S_{motion} + (1 - \alpha)S_{FG}, \tag{3.32}$$

where α is proportional to the velocity of the tracked hand. As a result, the likelihood for particles that are tracking a static hand is mostly determined by S_{FG}, while the likelihood of particles that track a moving hand, is determined more by S_{motion}, depending on the hand velocity. As a result, the particle filter is able to robustly track the hand through time, even if the moving hand comes close to static or slowly moving objects with a higher skin probability. Furthermore, it should be noted that even very fast moving objects that are part of the background, often appear to move much slower than the hand itself, due to the parallax principle.

The motion mask indicates which pixels cover objects that are currently moving. However, objects that have moved in the recent past should also be of more interest to the tracker than objects that are known to be static, since static objects are most probably part of the background. To this end, we use the probabilistic background subtraction method, introduced by Zivkovic [Zivkovic 04] to obtain a background model. Yet instead of thresholding the background probability map, we directly combine the obtained likelihood map with the skin-motion map. Using the likelihood map instead of a thresholded foreground mask, allows the tracker to also consider static objects and to cope with changing backgrounds and incorrect background models.

The Zivkovic foreground likelihood map is an image in which each pixel represents the likelihood of observing a specific (r, g, b) triplet, given the background model that is represented by a mixture of Gaussians. By assuming uniform and equal priors, we obtain $P(BG|r, g, b)$. If the complement of the BG random variable describes how dynamic an object has been in the recent past, we easily find $P(dyn|r, g, b) = 1 - P(BG|r, g, b)$, where $dyn \in \{1, 0\}$ is a random variable that indicates whether the pixel represents a dynamic or a static object.

Assuming independence between the different cues that have been calculated up till now, we can combine this background subtraction based model multiplicatively with the skin and motion based cue C_{skin}. Each value of the resulting like-



(a) Tracking result. (b) $P(FG|skin)$. (c) $P(dynamic|r,g,b)$.

(d) $P(motion|r,g,b)$. (e) $P(skin|r,g,b)$. (f) Final cue L

Figure 4.5: Different likelihood maps used in the observation model.

lihood map L then represents a sample from an approximation of the joint probability distribution that is shown in equation (3.33).

$$L \sim P(skin, FG, motion, dyn|r,g,b). \tag{3.33}$$

Finally, if a face has been detected by the Haar-cascade that was also used to initialize the online learning skin histograms, the values of the probability map within the bounding box of the face are reduced, to avoid particles to get trapped on the local maximum that is caused by the similar appearance of faces and hands. This operation can be considered as incorporating an informed prior spatial distribution into L.

Figure 4.5 shows the different likelihood maps that have been discussed so far, and Figure 4.5 (f) illustrates the combination of these likelihood maps into the final cue, L. This example clearly shows that the probabilistic fusion of all cues increases the discriminativeness of the final likelihood function. While each of the individual cues clearly has its pitfalls, the final cue is able to combine the strengths of each individual feature.

3.3.4 Discriminative Hand Detection

Although the observation model as described above results in a fast tracker that is able to track hands in a variety of environments, the model heavily depends on color information. While several steps have been taken to reduce this color dependence, as discussed earlier, tracking still fails in very dark or very light illumination

environments. Additionally, tracking tends to fail if hand-sized, skin-like objects appear in the background, especially when these objects are moving.

To solve this, we propose to combine our generative skin color based model, with a discriminative Random Hough Forest based model that we proposed in earlier work [Spruyt 12]. Whereas the previously discussed cue L mainly depends on color information, the Random Hough Forest classifier incorporates structural and textural features which greatly increases its robustness against illumination changes.

The Random Hough Forest classifier is a parts-based classifier, that outputs a probability map indicating the likelihood of representing a hand's centroid for each pixel coordinate in the image. As opposed to traditional rigid model based detectors that try to model the object as a whole (e.g. template matching), a parts based detector is able to cope with deformable objects by modeling the relative position of small object parts, together with their expected variance.

In [Spruyt 12], we proposed such a parts based classifier that is able to robustly *detect* hands in real-time. Small blob-like structures are first detected using an extended version of the Kadir and Brady [Kadir 01] feature detector. Several decision trees are then trained that classify these image patches as being part of a hand or part of the background. For patches that are part of a hand, the decision trees store the offset vector from the image patch to the hand's centroid. This offset vector is normalized to ensure rotation and scale invariance.

Each decision tree is trained using a randomly selected subset of features. Furthermore, each decision tree is trained on a bootstrapped sample of the original training data. As a result, all decision trees are distinct. During classification, blob-like regions of interest are detected again, and for each detected region, all decision trees cast a probabilistic vote on the centroid location of the hand. These votes are accumulated in a so called Hough voting map, representing the probability that each pixel coordinate corresponds to the centroid location of a hand.

The complete set of features, from which a subset is randomly selected by each decision tree, is made up of four color based histograms, and three texture based features. The texture based features include the local binary pattern descriptor [Ojala 96], gradient orientation histograms, and the FREAK descriptor [Alahi 12].

Incorporating this classifier into our observation model greatly increases the tracker's robustness to illumination changes, due to the detector's dependence on texture based features. The reason is that texture is mainly defined by high-frequency information such as image gradients, which are known to be invariant to several types of color transformations.

The Random Hough Forest classifier is a discriminative classifier, whereas our previously described models used to generate L are generative classifiers. Both types of classifiers aim to solve the same task, namely obtaining the conditional density $P(x|z)$, where x represents the class label and z represents the observation.

However, the methodology used to obtain an estimate of this density differs for each classifier type. A generative approach models both the joint density $P(x, z)$ and the prior distribution of the observations $P(z)$, and combines both using the Bayes rule to obtain the posterior $P(x|z)$. In practice, the joint distribution is often obtained indirectly by estimating the conditional likelihood of the observation $P(z|x)$, and the prior distribution of class labels $P(x)$. Discriminative models on the other hand, directly try to estimate the conditional PDF $P(x|z)$, and thus the classifier's decision boundaries, thereby skipping the difficult step of estimating the likelihood distributions.

Discriminative models do not make assumptions about the shape of the likelihood functions and therefore often outperform generative models if a lot of labeled training data is available. Generative models on the other hand, make assumptions about the shape of the likelihood function and the marginal distributions, allowing an accurate model to be learned more quickly if few training samples are available. While both approaches have their merits and pitfalls, it has been shown that combining generative and discriminative methods in a classifier can greatly improve the detection rates [Kölsch 10].

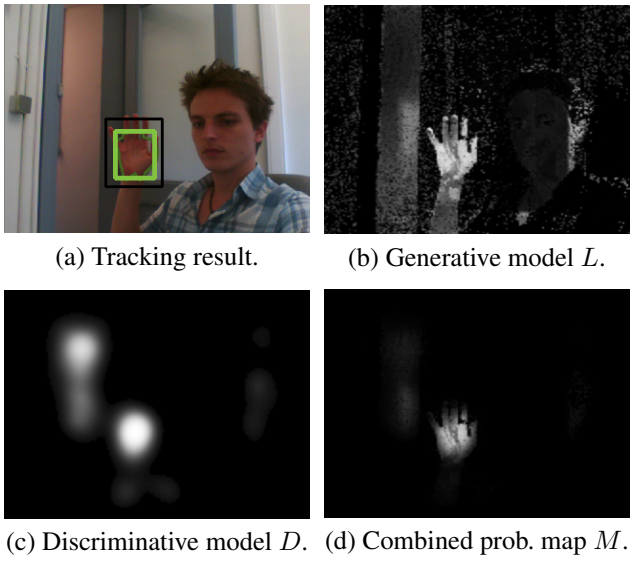
Our particle filter's observation model combines the probability map L that was obtained using the generative skin models, described in earlier sections, with the hand probability map that was obtained using a discriminative Random Hough Forest classifier. Since each model is trained on different features, we can assume a weak form of orthogonality and can simply combine both models multiplicatively. Let D be the probability map obtained by the discriminative Random Hough Forest classifier. In the next section, the combined probability map will then be referred to as $M = D\,L$.

Figure 4.6 illustrates the combination of both probability images, resulting in the final likelihood map that is used to weigh the random samples of the particle filter. While each model has its own shortcomings, combining them clearly increases the difference between the peaks of the local maxima, and the global maximum in the likelihood.

3.3.5 Likelihood Model

The final cue M contains a probability value for each pixel in the image. This probability map is considered to be the observation at time instance t. In this section, we explain how each particle of the particle filter is assigned a weight, based on this observation. The model that calculates the likelihood of a state estimate, based on the available observation, is often called the likelihood model.

As discussed in Section 3.2, and shown by equation (3.19), the state to be estimated is the bounding box of the hand, defined by its position (x, y) and its dimensions (w, h). After state prediction by the motion model $f(\mathbf{b}_{t-1})$, the observation model $h(\mathbf{b}_t)$ is used to assign weights to each particle representing a



(a) Tracking result. (b) Generative model L.

(c) Discriminative model D. (d) Combined prob. map M.

Figure 4.6: Combining the generative skin likelihood map with the discriminative Random Hough Forest likelihood map increases the discriminativeness of the final probability map.

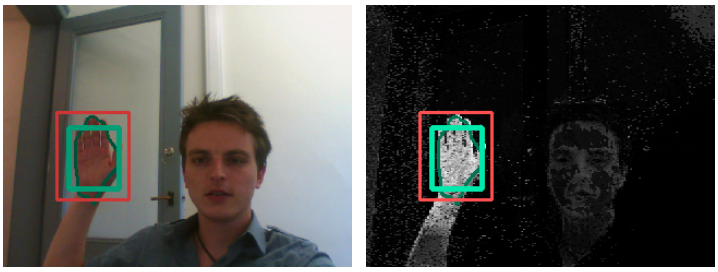
different state estimate. In our case, the observation corresponds to the likelihood map, $y \doteq M$, and the weight of a particle therefore depends on this likelihood map.

One way to assign a weight to a particle, represented by a bounding box, would be to calculate the average of the values of the probability map inside the bounding box and use this as a weight. Such an approach would also assign height weights to particles that completely cover a small part of a larger skin-like background region. Furthermore, particles that cover part of the arm of a person would be weighted similar to particles that cover the hand of a person.

Instead, a particle should be assigned a larger weight if it covers a blob-like structure, which is a region of the probability image in which the average probability is significantly larger than its surrounding. As such, we propose a simple operator that results in a high response for blob-like structures and a low response for uniform regions in the probability map.

The proposed operator resembles a simplified version of the Laplacian operator, which is known for its capabilities for blob-detection, and is used by the well known SIFT and SURF detectors. We define an outer bounding box $\mathbf{b}_o$, surrounding the bounding box $\mathbf{b}$ of a particle. The dimensions of $\mathbf{b}_o$ are chosen such that its area is twice the area of $\mathbf{b}$. Hence, $\mathbf{b}_o$ is defined by equation (3.34).

$$\mathbf{b}_o = (\mathbf{b}[x], \mathbf{b}[y], \sqrt{2}\mathbf{b}[width], \sqrt{2}\mathbf{b}[height]). \tag{3.34}$$



(a) Tracking result. (b) Probability image.

Figure 4.7: Bounding boxes for calculating the blob-response.

Figure 4.7 shows the bounding boxes $\mathbf{b}$ in green and $\mathbf{b}_o$ in red.

In the following, let B be the set of all (x, y)-coordinates within the rectangular region defined by $\mathbf{b}$ and let B_o be the set of all (x, y)-coordinates that fall within the region defined by $\mathbf{b}_o$.

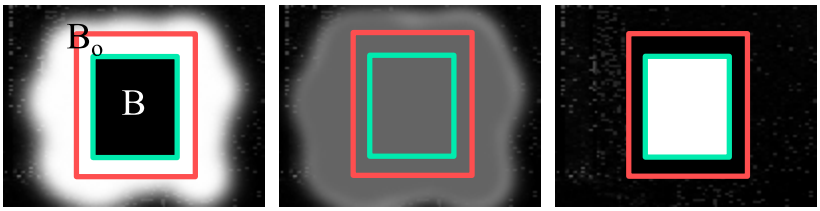
Similar to the well known approximation of a Laplacian operator as the difference of two Gaussians kernel functions, we define the blob operator simply as the normalized difference between the sum of all pixel inside region B, and those inside the region surrounding this box, defined by the pixel coordinates $B_o \setminus B$. Alternatively, the blob response, $R(B)$, can be calculated efficiently for each particle by subtracting the sum of all pixels inside region B_o from twice the sum of all pixels inside region B, as shown by equation (3.35).

$$R(B) = \frac{2 \sum_{x,y \in B} M(x,y) - \sum_{x,y \in B_o} M(x,y)}{\mathbf{b}[width] \times \mathbf{b}[height]}. \tag{3.35}$$

Directly calculating the blob response as proposed by equation (3.35) would require the execution of computationally expensive nested loops and out of order memory access for each particle. Moreover, the complexity would depend on the size of the object to be tracked and would thus increase if a hand approaches the camera. To avoid these issues, we use integral images, originally proposed by Viola and Jones [Viola 04].

An integral image, also known as a summed area table, is an image in which each pixel represents the sum of all values to the upper-left side of this pixel. This integral image can be calculated easily by a single pass algorithm with $O(width \times height)$ complexity.

The integral image $I_M(x, y)$ is calculated based on the probability image $M(x, y)$, before the next iteration of the particle filter is executed. Based on the integral image, the sum of all pixels within any rectangular area of $M(x, y)$ can be computed in constant time $O(1)$. Let $\mathbf{j}$, $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ represent the coordinates of the



(a) $R(B) = -1$. (b) $R(B) = 0$. (c) $R(B) = 1$.

Figure 4.8: Blob filter response for three special cases.

left-top corner, the right-top corner, the left-bottom corner and the right-bottom corner of region B respectively. Similarly, let $\mathbf{j}_o$, $\mathbf{x}_o$, $\mathbf{y}_o$ and $\mathbf{z}_o$ be the corner coordinates of region B_o. Then equation (3.35) can be rewritten as follows:

$$R(B) = \frac{2\left(I_M(\mathbf{z}) - I_M(\mathbf{x}) - I_M(\mathbf{y}) + I_M(\mathbf{j})\right)}{\mathbf{b}(width) \times \mathbf{b}(height)} - \frac{\left(I_M(\mathbf{z}_o) - I_M(\mathbf{x}_o) - I_M(\mathbf{y}_o) + I_M(\mathbf{j}_o)\right)}{\mathbf{b}(width) \times \mathbf{b}(height)}. \tag{3.36}$$

As a result, once the probability map M has been calculated, the likelihood of each particle can be evaluated in constant time. This great reduction in complexity further encourages the use of rectangular areas and a rectangular blob detector as opposed to shapes that more accurately capture the contour of a human hand.

The response of the blob detector is defined by the interval $\{R(B) \in \mathbb{R} | -1 \leq R(B) \leq 1\}$. This can easily be understood by calculating the response for the interval boundaries. If all pixel values inside the inner region B equal zero, and all pixel values in the region defined by $B_o \setminus B$ equal one, then $R(B) = -1$. In this case, the bounding box definitely does not represent a hand, because its surroundings have a much higher probability. If all pixel values inside the inner region B equal one, and all values in the region defined by $B_o \setminus B$ equal zero, then $R(B) = 1$. In this case, the blob is very likely to represent a hand. Finally, if all pixels inside region B_o are equal, then $R(B) = 0$. In this case the region is not a blob, but might still represent a hand with equal probability as its background. Figure 4.8 illustrates each of these three special cases visually.

The weight w_i, assigned to a particle, represents the likelihood of the observed blob response, given the current state. We define a truncated Gaussian probability density function by its parameters $\mu = 1$ and $\sigma = \frac{(1-(-1))}{3} = \frac{2}{3}$, such that approximately 98% of the observed data can be explained by this distribution. Assuming a non-informative prior, we obtain the probability mass function. Figure 4.9 illustrates the truncated Gaussian function that is used to map blob detection responses to particle weights. Each of the special cases shown in figure 4.8 are indicated.

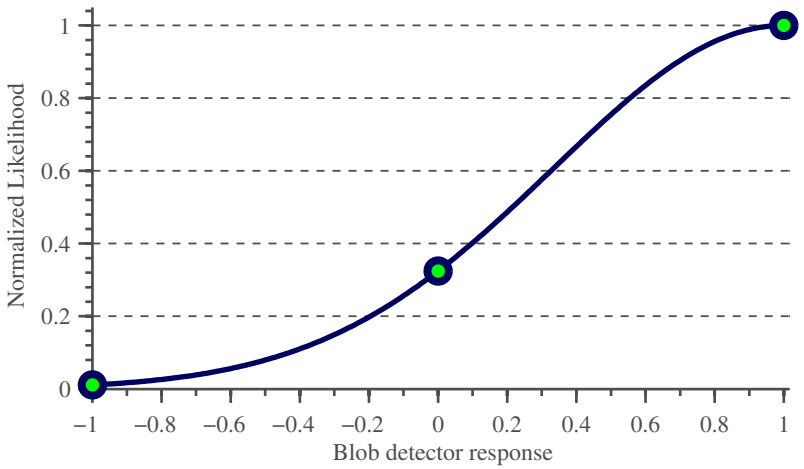


Figure 4.9: Particle filter likelihood function.

This shows that a blob response of $R(B) = -1$ corresponds to a particle weight $w_i \approx 0$. Higher blob responses result in an exponentially increasing observation likelihood. A blob response of $R(B) = 0$ corresponds to a particle weight $w_i \approx 0.3$, and a blob response of $R(B) = 1$ corresponds to a particle weight of $w_i = 1$.

3.4 Motion Model and Proposal Distribution

In this section, the motion models f_{loc} and f_{size} that were defined in section 3.2 are discussed in more detail. The motion models are functions of the previous state, as defined by equation (3.1), and predict the next state, given this previous state.

The most widely used particle filter variant is the bootstrap filter, which is also the most prevalent in visual tracking literature. The bootstrap filter is a SIR particle filter, where the state transition prior $p(x_t^i|x_{t-1}^i)$, as returned by the motion model, is used as the proposal distribution. Although this greatly simplifies the importance sampling process, the main disadvantage of this approach is that the proposal distribution does not take the most recent observation into account. In the second part of this section, we propose to incorporate optical-flow information and a mean-shift optimization iteration directly into the proposal distribution to enhance the importance sampling step.

3.4.1 Motion Model

As discussed in Section 3.1, a particle filter can be considered a discrete time, non-linear, non-Gaussian dynamic system, defined by its motion model and observation model, shown by equations (3.1) and (3.2). In visual tracking applications, it is usually difficult to accurately predict and model the expected motion of an object. Apart from specific applications such as avionics, learning an accurate motion model based on training data would result in tracking failure when unexpected motions are encountered.

Instead of learning an accurate motion model, we proposed a robust observation model to compensate for a simple motion model that only roughly approximates the object's expected motion. Non-linear or unknown components of the hand's motion are simply modeled by a Gaussian process, such that prediction errors are explained by a Gaussian white noise variable. In time series prediction, this is often accomplished by means of an autoregressive model. Equation (3.37) shows the definition of a general autoregressive model of order n, where c is a constant, α_i are the model parameters, and ε_t is a white noise component.

$$\mathbf{b}_t = c + \sum_{i=1}^{n} \alpha_i \mathbf{b}_{t-i} + \varepsilon_t. \tag{3.37}$$

Probably the most widely used motion model in visual tracking, is the random walk that models Brownian motion. A random walk model is a first order autoregressive model, and assumes that the difference between the previous state and the current state can be modelled as a Gaussian process, such that $(\mathbf{b}_t - \mathbf{b}_{t-1}) \sim \mathcal{N}(\mu, \Sigma)$. Usually, a white process is assumed, such that the motion model becomes $\mathbf{b}_t = \mathbf{b}_{t-1} + \varepsilon_t$, where ε_t is a white noise component. The result is a first order autoregressive model with parameters $c = 0$, $n = 1$ and $\alpha_1 = 1$.

In Section 3.2 we partitioned the state space, such that $\mathbf{b} = (\mathbf{b}_{loc}, \mathbf{b}_{size})$. Partition $\mathbf{b}_{size}$ represents the width and height of the bounding box of the tracked object. These dimensions can quickly and arbitrarily change. For instance, if the user shows his hand with open palm at time instance $t-1$, but quickly makes a fist pose in time instance t, the width and height of the bounding box will change due to the apparent disappearance of the fingers. These abrupt changes in $\mathbf{b}_{size}$ are difficult to predict but can be expected to behave as white Gaussian noise and are therefore modeled by a random walk.

However, hand motion, described by $\mathbf{b}_{loc}$, is not completely random. If a hand travels in a specific direction, there is a high likelihood that the hand will follow that same direction in the next frame. Modeling such motion with a random walk model, would result in autocorrelated estimation errors. Therefore, we propose to use a second order autoregressive motion model to predict this state partition, which incorporates one extra 'lag' into the prediction equation.

The parameters of the model are chosen as $c = 0, n = 2, \alpha_1 = 2$ and $\alpha_2 = -1$, such that our motion model is defined as

$$\begin{aligned} \mathbf{b}_t &= 2\mathbf{b}_{t-1} - \mathbf{b}_{t-2} + \varepsilon_t && (3.38)\\ &= \mathbf{b}_{t-1} + (\mathbf{b}_{t-1} - \mathbf{b}_{t-2}) + \varepsilon_t && (3.39)\\ &= \mathbf{b}_{t-1} + \Delta V + \varepsilon_t. && (3.40) \end{aligned}$$

where ΔV represents the velocity of the object in the previous video frame.

Equation (3.40) shows that this prediction equation assumes that the velocity of the object remains constant between two subsequent time steps. Consequently, this motion model is often termed a 'constant velocity' model in literature. The Gaussian noise component, ε_t, models non-linearities or unexpected accelerations as a zero-mean random process. Next to the random walk model, a constant velocity model is probably one of the most popular motion models used in probabilistic visual tracking [Asaari 10, Shan 07].

In equation (3.19), we defined the state vector $\mathbf{b} = (x, y, w, h)$. As illustrated by equation (3.38), at time instance $t-1$, the prediction of the state at time t also depends on the previous state at time $t-2$. However, as defined by equation (3.1), the motion model is a function of only the state at time $t-1$, due to the Markov principle. Therefore, the originally defined state vector $\mathbf{b}$ is augmented with (x_{t-1}, y_{t-1}) and denoted $\dot{\mathbf{b}}$.

The complete motion model then becomes $\dot{\mathbf{b}}_t = f(\dot{\mathbf{b}}_{t-1}, \mathbf{M}_{t-1}) = A\,\dot{\mathbf{b}}_{t-1} + \mathbf{M}$, where M is a column vector containing white noise components, and matrix A and augmented vector $\dot{\mathbf{b}}_{t-1}$ are defined as follows:

$$A = \begin{bmatrix} 2 & 0 & -1 & 0 & 0 & 0 \\ 0 & 2 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \dot{\mathbf{b}}_{t-1} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ x_{t-2} \\ y_{t-2} \\ w \\ h \end{bmatrix}. \qquad (3.41)$$

3.4.2 Proposal Distribution: Optical-Flow

A major shortcoming of the bootstrap particle filter is that it only uses the latest available information after the state prediction has been made. The latest observations are then used to assign a likelihood (i.e. weight) to each particle. However, in case of irregular motion, many of the predicted states will be inaccurate, such that the resulting set of particles mainly model the tails of the posterior distribution. In this case, only a few particles will be assigned a high likelihood, and tracking will be lost if the irregular motion continues.

In contrast, many successful deterministic tracking approaches use the latest observations to predict the next state, instead of relying on a simple motion

model that only depends on the previous state estimate. As an example, Kolsch *et al.* [Kölsch 04b] use the Kanade-Lucas-Tomasi (KLT) tracker to estimate optical-flow in the video, and use the obtained flow vectors to guide their deterministic sampling process.

Optical-flow is the apparent motion of objects in a 3D scene, projected onto the 2D camera plane. Intuitively, such an optical-flow vector field would represent the actual motion in a scene much more accurately than a state prediction that is based simply on a constant velocity model.

Several approaches can be found in literature to incorporate optical-flow into the motion model of a particle filter. Belgacem *et al.* [Belgacem 12] directly use the optical-flow vectors to steer the motion model. Yao *et al.* [Yao 10] use a weighted combination of the flow vectors and the vector resulting from their constant velocity model, as a motion model. Lucena *et al.* [Lucena 03] use optical-flow in the observation model of the particle filter instead of the motion model.

However, while these methods have been shown to increase tracking robustness, using optical-flow in the observation model does not solve the earlier mentioned problem of inaccurate state prediction. Directly incorporating the optical-flow results into the particle filter's motion model on the other hand, is mathematically incorrect and would force us to abandon the Bayesian framework which forms the theoretical foundation of particle filters. As shown by equation (3.1), the motion model of a particle filter should only depend on the previous state estimate, such that the state transition prior $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is obtained.

Although the state transition prior is not allowed to depend on the latest observation, the opposite is true for the proposal distribution $q(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i, \mathbf{y}_{1:t})$ shown in the denominator of equation (3.11). Conventional bootstrap particle filters ignore this, and use the state transition prior as the proposal distribution. Although this greatly simplifies the importance weight calculations because the prior distribution in the numerator and the proposal distribution in the denominator cancel each other out, it is the cause of the above described problem.

Figure 4.10 illustrates this problem graphically. The proposal density, which equals the state transition prior $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ in bootstrap filters, determines the particle density, whereas the observation likelihood function $p(\mathbf{y}_t|\mathbf{x}_t)$ defines the particle weights. As a result, in this example most particles model the tails of the posterior density. In this illustration, particles are drawn with a radius that is proportional to the assigned likelihood.

To solve this problem, we propose to incorporate optical-flow estimates in the proposal distribution, such that the proposal distribution depends both on the previous state estimate, and on the latest measurements. In this case, it is important to note that the transition prior in the numerator of equation (3.11) is no longer cancelled out by the denominator. Therefore, not only do we need an analytic ex-

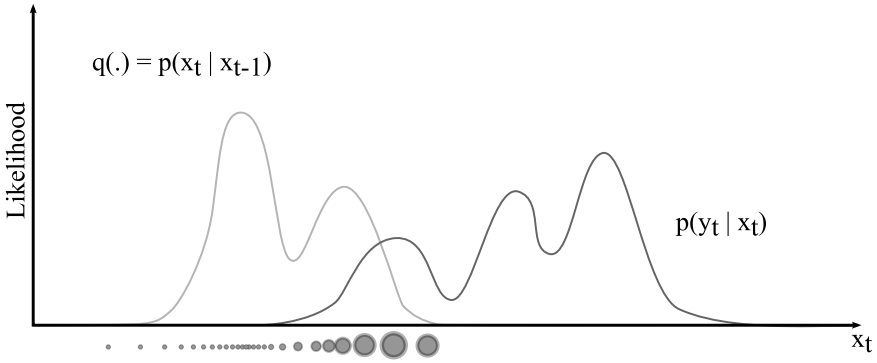


Figure 4.10: Importance sampling issues in bootstrap filters: If the proposal distribution $q(.)$ equals the state transition prior, overlap with the observation likelihood function is small. As a result, the particles with the largest weight only model the tail of the true, but unknown, posterior distribution instead of its mode.

pression of the proposal distribution such that it can be sampled, we also need to calculate the actual transition prior likelihood during weight assignment.

Van der Merwe *et al.* [van der Merwe 01] proposed to generate the proposal distribution for each particle using an Unscented Kalman filter [Julier 02]. Their method solves the above problem for non-linear filtering problems. However, the unscented transform is performance intensive, and unnecessary in our case, because the motion model that was defined in Section 3.4.1 is a linear model.

Based on this idea, we propose to generate the proposal distribution for each particle using a standard Kalman filter. For each particle, the Kalman filter allows us to generate a proposal distribution that incorporates both the optical-flow estimate, and the earlier defined constant velocity estimate. Due to the Kalman filter's properties, the resulting proposal distribution represents an optimally weighted average of the constant velocity motion model and the observed optical-flow vectors. The resulting particle filter suffers less from particle depletion, and can cope with peaked and narrow likelihood functions, even if these lie in the tails of the transition prior.

In Section 3.1, we discussed Bayesian filters in general, and concluded that Kalman filters assume Gaussian likelihoods and linear models. However, it is important to note that our particle filter solution, with Kalman filter proposal distributions, does not require the posterior to be normally distributed and does not require linear observation models. Instead, a Kalman filter is used per individual particle, hence only the distribution of each single particle is assumed to be Gaussian. During importance sampling, the particle is drawn from this normal distribution that serves as the proposal distribution, as shown by equation (3.42).

$$q(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i, \mathbf{y}_{1:t}) \sim N(\bar{\mathbf{x}}_t^i, P_t^i) \qquad i = 1, ..., N, \tag{3.42}$$

where $\bar{\mathbf{x}}_t^i$ represents the Kalman filter's state estimate and P_t^i represents its covariance matrix.

Thus, at time t, the latest observation, which is the optical-flow field in our case, is used to calculate the mean, $\bar{\mathbf{x}}_t^i$, and covariance P_t^i of the proposal distribution of each particle i, using the Kalman filter measurement and update equations. Next, the i^{th} particle is sampled from this distribution, and is assigned a weight by the observation model, as in a traditional particle filter. This method requires us to propagate the covariance P_t^i for each particle since multiple particles might share the same covariance matrix after resampling.

Because optical-flow mostly contains information regarding the displacement of the object of interest and is not very informative about its size, we only apply this method to the first state partition $\mathbf{b}_{loc}$. The second state partition $\mathbf{b}_{size}$ is then treated in a conventional bootstrap filtering manner, using the transition prior as the proposal density. Like in Section 3.4.1, we define the augmented state vector $\dot{\mathbf{b}}_{loc}^t = (x_t, y_t, x_{t-1}, y_{t-1})$. In the following, we denote $\mathbf{b}_{loc}$ as $\mathbf{b}$ to simplify further notations. We introduce the subscripts $(.)_0$ and $(.)_1$ to indicate the prediction (no measurement has been incorporated yet) and update (the optical-flow measurement has been incorporated) stage respectively.

3.4.3 A Kalman Filter Embedded Proposal Distribution

For the Kalman filter, several important matrices have to be defined. Let Q be the process noise covariance matrix that defines the noise component $\mathbf{m}$ from equation (3.1). Let R_1 be the measurement noise matrix that defines the noise component $\mathbf{n}$ from equation (3.2). Let $\mathbf{y}_1$ be the state estimate that is obtained by the optical-flow observation. Let H be the observation model matrix that describes how the predicted state corresponds to the observed state, such that $\mathbf{y}_1 = H\,\mathbf{b}_{loc}(t|t-1) + \mathbf{n}$, where n is a white noise component. Finally, let F be the motion model matrix that defines how the next state is predicted based on the previous state, such that $\mathbf{b}_{loc}(t|t-1) = F\,\mathbf{b}_{loc}(t-1|t-1) + \mathbf{m}$, where $\mathbf{b}_{loc}(t|t-1)$ is the state estimate at time t, given observations up to and including time $t-1$, and m is a white noise component.

The motion model matrix F then corresponds to the 4×4 matrix whose elements are the first four rows and columns of matrix A, defined by equation (3.41). The observation matrix H is defined by equation (3.43), since the state that is observed using the optical-flow field is described by a coordinate pair $\mathbf{y}_1 = (x, y)$. Each optical-flow vector represents a mapping of a pixel location in the previous video frame, to a corresponding location in the current video frame. The average of all flow vectors within the bounding box $\mathbf{b}_{t-1}$ is used to obtain the observed state estimate $\mathbf{y}_1$.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.43}$$

For each particle i in the set of N particles, the Kalman filter estimates the mean $\mu^i = \hat{\mathbf{b}}_1(t|t)$ and covariance matrix $P^i = P_1(t|t)$ of the multivariate Gaussian proposal distribution $q(.) \sim N(\mu^i, P^i)$. The state prediction at time t, given observations up to and including time $t-1$ is:

$$\hat{\mathbf{b}}_0(t|t-1) = A\,\hat{\mathbf{b}}_1(t-1|t-1), \tag{3.44}$$

and the corresponding covariance is

$$P_0(t|t-1) = AP_1(t-1|t-1)A^\intercal + Q. \tag{3.45}$$

The new estimate, after incorporating the optical-flow measurement, is then:

$$\hat{\mathbf{b}}_1(t|t) = \hat{\mathbf{b}}_0(t|t-1) + K_1(t)[\mathbf{y}_1(t) - H\hat{\mathbf{b}}_0(t|t-1)], \tag{3.46}$$

with covariance

$$P_1(t|t) = P_0(t|t-1) - K_1(t)S_1(t)K_1(t)^\intercal, \tag{3.47}$$

where the Kalman gain is

$$K_1 = P_0(t|t-1)H^\intercal S_1^{-1}(t), \tag{3.48}$$

and the innovation covariance is

$$S_1(t) = HP_0(t|t-1)H^\intercal + R_1. \tag{3.49}$$

The particle filter then draws N new particles, each from their corresponding proposal distribution. In the following, let $\hat{\mathbf{b}}_t^i$ be the state of the particle that is sampled from this proposal distribution of particle i at time instance t. Let $\dot{\mathbf{b}}_t^i$ be the state estimate of the same particle, obtained by the constant velocity model which is discussed in Section 3.4.1. The state transition prior for particle i is then obtained by evaluating the likelihood of the sample $\hat{\mathbf{b}}_t^i$ by the motion model. If the motion model's likelihood function is a Gaussian distribution, centered at the state that was predicted by the constant velocity model, then the state transition prior is:

$$p(\hat{\mathbf{b}}_t^i|\hat{\mathbf{b}}_{t-1}^i) \sim \exp\left(-\frac{1}{2}(\hat{\mathbf{b}}_t^i - \dot{\mathbf{b}}_t^i)^\intercal Q^{-1}(\hat{\mathbf{b}}_t^i - \dot{\mathbf{b}}_t^i)\right). \tag{3.50}$$

The proposal density likelihood for sample i is defined by the Gaussian posterior that is estimated by the Kalman filter, and can be obtained as follows:

$$q(\hat{\mathbf{b}}_t^i|\hat{\mathbf{b}}_{1:t-1}^i, \mathbf{y}_{1:t}) \sim \frac{1}{\sqrt{|P_t^i|}} \exp\left(-\frac{1}{2}(\hat{\mathbf{b}}_t^i - \mu_t^i)^\intercal (P_t^i)^{-1}(\hat{\mathbf{b}}_t^i - \mu_t^i)\right). \tag{3.51}$$

Finally, the observation likelihood $p(\mathbf{y}_t|\hat{\mathbf{b}}_t^i)$ can be obtained as explained in Section 3.3.5. Based on the previous weight, the state transition prior, the proposal

density likelihood, and the observation likelihood, the particle weights w_t^i are then calculated as follows:

$$w_t^i = w_{t-1}^i \frac{p(\mathbf{y}_t|\hat{\mathbf{b}}_t^i)p(\hat{\mathbf{b}}_t^i|\hat{\mathbf{b}}_{t-1}^i)}{q(\hat{\mathbf{b}}_t^i|\hat{\mathbf{b}}_{1:t-1}^i, \mathbf{y}_{1:t})}. \tag{3.52}$$

As opposed to the widely used bootstrap filter, our approach makes use of the latest observations in the proposal distribution, resulting in a more efficient use of the samples. As a result, relatively few particles (50 in our experiments) are needed for robust tracking.

The system noise and measurement noise matrices of a Kalman filter are usually chosen by hand during filter design, and are usually chosen to be diagonal matrices if no informative choice can be made. Nevertheless, we can estimate the measurement noise matrix R_1, based on the local uniformity of the optical flow field. The measurement noise matrix R_1 is obtained by calculating the covariance matrix of the components of all flow vectors inside the bounding box that is defined by $\mathbf{b}_t^i$. If all vectors are similar, the observation noise is assumed to be small, whereas a large error is assumed if the optical flow field is noisy.

Similarly, we dynamically estimate the process noise matrix Q, based on the amount of predicted displacement $\Delta\mathbf{v} = |\mathbf{b}_t^i - \mathbf{b}_{t-1}^i|$. If the hand is moving slowly, the prediction error is likely to be smaller than when it is moving fast. Furthermore, if a hand suddenly moves to the opposite direction with the same speed $\Delta\mathbf{v}$, the error of the predicted state is twice the predicted displacement. Because approximately 99.7% of normally distributed data ($N(\mu, \sigma^2)$) lies within the interval $\mu \pm 3\sigma$, we simply set the variances of Q equal to $(\|\Delta\mathbf{v}\| + u)^2$, where $\|.\|$ denotes the vector norm and u is an empirically defined constant. This slight overestimation of the variance allows the model to cope with unexpected non-linearities such as accelerations. The off-diagonal elements are set to zero.

While many optical-flow algorithms exist, and any of them could be used in our tracking framework, most optical-flow algorithms adhere to the so called brightness constancy assumption. As a result, these algorithms, such as the KLT method used by Kolsch *et al.* [Kölsch 04b], can not be used in unconstrained environments with unexpected illumination. Moreover, most regularized optical-flow algorithms are too slow to be incorporated into a real-time tracking system.

To cope with changing illumination, a real-time, sparse regularized optical-flow method that we proposed earlier in [Spruyt 13b] is used in our framework. This optical-flow algorithm uses a fast feature detector and a Local Binary Patterns descriptor for point matching. A sparse regularization method based on the non-Sibsonian interpolant, described in chapter 3, is used to obtain a smooth flow field, and the average of all flow vectors within the bounding box, defined by $\mathbf{b}$, is used as the observation vector of the Kalman filter. An added advantage of using our optical-flow method in the context of our tracking algorithm, is that the Local

Binary Patterns that were already calculated during hand detection, as explained in Section 3.3.4, can simply be reused without additional computations.

3.4.4 Proposal Distribution Based on Mean-shift

Incorporation of the optical-flow estimate into the proposal distribution largely counteracts the shortcomings of the simple constant velocity motion model. However, integration of the latest observations into the proposal density could be taken even further. As an example, the tracking approach of Kolsch *et al.* [Kölsch 04b] guides its deterministic sampling process based on the color likelihood map of the image. As discussed in the previous section, directly incorporating such information into the motion model of the particle filter would yield particles that are meaningless from a Bayesian point of view. In the next paragraphs, we propose a method to use this information to further fine-tune the proposal distribution instead.

A well known deterministic tracking approach that tries to model the mode of a color based PDF, is the Camshift algorithm [Exner 10]. The Camshift method uses the mean-shift optimization algorithm to iteratively shift the previous state estimate closer to the local maximum of the PDF. However, due to its deterministic approach and the gradient based search, the mean-shift procedure on its own only finds one local maximum, and is therefore not suited for tracking objects in a complicated environment, where the approximated PDF has many local maxima.

If many skin-like background objects are present in the scene, we would like the proposal distribution of our particle filter to be based mostly on the constant velocity motion model and the optical-flow estimation. If only few skin-like background objects are present however, we would like the proposal distribution to be determined mainly by the skin likelihood in the neighborhood of the particle. The skin likelihood map was defined in Section 3.3.1.

Shan *et al.* [Caifeng 04] proposed to integrate mean-shift into their bootstrap particle filter tracker. They draw samples from the state transition prior and perform a mean-shift iteration after particle prediction, and before likelihood assignment. Additionally, they model the particle distribution before mean-shift correction using a Gaussian mixture model (GMM), and create a second GMM to model the distribution after the mean-shift iteration. Finally, they weight the particle likelihood by the ratio of both Gaussian mixture models to correct for the fact that the mean-shift iteration in fact altered their proposal distribution.

Shan *et al.*,'s method corresponds to generating a proposal distribution based on the mean-shift process, and then using the local maxima of this distribution directly as the particle states instead of sampling from it. As a result, this approach suffers from particle impoverishment, since many particles will end up representing the same state. Moreover, this method implicitly models a single proposal distribution for the complete particle set, instead of designing the proposal distri-

bution specifically for each particle. Because of this, it can not cope with situations where the mean-shift observation is unreliable. In order to deal with the fact that the mean-shift iteration might in fact degrade the result sometimes, Shan *et al.* heuristically propose to only use this for 50% of the particles. Finally, this approach is tailored to bootstrap filters and can not be easily combined with different proposal distributions.

In the following paragraphs, we propose to incorporate the mean-shift result directly into the proposal distribution instead of performing the mean-shift iterations on the samples of the transition prior. The mode of our final proposal distribution will be a weighted average of the constant velocity motion model, the optical-flow estimate, and the mean-shift result. The weights are determined dynamically and optimally, based on the performance of each of these predictions in the recent past. Thus, our method automatically adapts its parameters such that the most reliable prediction (i.e. constant velocity, optical flow based or mean-shift based) dominates the final proposal PDF. Our method avoids the particle impoverishment problem by sampling the particles from their proposal distribution instead of using the proposal PDF's maxima directly as the particle state.

The effect is that each particle is shifted closer towards the mode of the posterior distribution before assigning weights to the particles, compared to conventional bootstrap filter approaches. This greatly reduces the need for resampling as will be shown in Section 4, and helps to avoid both the depletion and the particle impoverishment problems. Moreover, particles are used more efficiently, such that only a limited number of particles is needed for robust tracking.

In Section 3.4.2, the mode of the optical-flow based proposal distribution for each particle was defined as μ_t^i. This mode, together with the corresponding covariance matrix P_t^i, was obtained by means of Kalman filtering. In the following, let the mode of the skin-likelihood based proposal distribution be $\mathbf{s}_t^i$, and let its corresponding covariance matrix be S_t^i.

Let $\epsilon_t = \mu_t^i - \mathbf{s}_t^i$ be the difference between the mode locations of the two distributions. The normal proposal distribution that optimally combines both types of information, would then be a distribution with its mode chosen such that the error $\sum_{n=1}^{t} \epsilon_n^2$ is minimized. This corresponds to a least-squares regression problem, and can be solved using a recursive least-squares filter. Thus, the mode of the optimal proposal distribution at time instance t is a weighted linear combination of the modes $\mathbf{s}_t^i$ and μ_t^i, where the dynamic weight depends on the errors in the past. The covariance matrix of this proposal distribution is a measure of the spread of this error. If an exponential weighting scheme is used such that old data is gradually discounted, the optimal algorithm to calculate this recursive least-squares regression is the Kalman filter.

In fact, the above problem could be seen as a sequential data fusion problem [Durrant-Whyte 01]. At time instance t, optical-flow information becomes

available, and is used to update the Kalman prediction, yielding μ_t^i and P_t^i. This state estimate μ_t^i allows us to measure the skin-likelihood at that position, and should be further updated based on this likelihood. Also, the covariance matrix P_t^i needs to be updated, to incorporate the uncertainty of the skin-likelihood observation.

Since both observations are obtained synchronously, the state does not evolve between observation of the optical-flow field, and observation of the skin likelihood. Therefore, we can simply use the optical-flow based Kalman prediction itself as the prediction for the next update which will incorporate the skin likelihood.

Like in Section 3.4.2, we need to define the Kalman matrices. Let R_2 be the measurement noise matrix for the skin likelihood measurement. Let $\mathbf{y}_2$ be the state estimate that is obtained by the mean-shift procedure. The mean-shift algorithm uses μ_t^i as its initial estimate, and returns a new state estimate $\mathbf{y}_2 = (x, y)$. Consequently, the observation matrix H from Section 3.4.2 can be re-used. Finally, we introduce the subscript $(.)_2$ to indicate a Kalman update stage that is based on the second observation, namely the skin likelihood.

The parameters of the final proposal distribution are then calculated recursively as follows: In the prediction stage, we calculate the state estimate based on the constant velocity model using equation (3.44). In the update stage, we update this prediction by the optical-flow measurement using equation (3.46). Finally, we update the resulting state estimate to incorporate the skin likelihood observation by considering $\hat{\mathbf{b}}_1(t|t)$ to be the prediction for this update step. The final state estimate, is then:

$$\hat{\mathbf{b}}_2(t|t) = \hat{\mathbf{b}}_1(t|t-1) + K_2(t)[\mathbf{y}_2(t) - H\hat{\mathbf{b}}_1(t|t-1)], \tag{3.53}$$

where the covariance matrix is calculated similar to equation (3.47), with P_1, K_2 and S_2 replacing P_0, K_1 and S_1 respectively.

The above equations show that a different gain matrix K is used for each observation type. As a result, the final state estimate is a weighted combination of the constant velocity motion model, the optical-flow observation, and the skin likelihood observation. The weights are dynamically adapted, based on the estimation errors in the recent past. Instead of sequentially incorporating the new evidence (i.e. optical flow and mean-shift optimization), once could opt to group all observations such that a single Kalman gain matrix is calculated. The advantage of this approach would be that correlations between the observations can be modeled. However, if many observations are grouped, the size of the matrices grows exponentially, resulting in slow computation times.

The final proposal distribution, used by the particle filter, is the normal distribution that is defined by its parameters $\mu_t^i = \hat{\mathbf{b}}_2(t|t)$ and covariance matrix $P_t^i = P_1(t|t)$. Plugging these parameters into equation (3.51) yields the final pro-

posal density for the particle filter, used to sample new particles before weights are assigned.

The skin likelihood based observation is obtained by performing the mean-shift procedure after an initial optical-flow based state estimate $\hat{\mathbf{b}}_1(t|t-1)$ is obtained. Mean-shift is a gradient based optimization method that, given a coarse estimate of the local maximum, finds the peak of a discrete PDF. We use the result of the optical-flow based Kalman state prediction as the initial estimate for the Mean-shift algorithm.

Let $L(\mathbf{x})$ be the skin probability map that was defined in Section 3.3.1, with $\mathbf{x} = (x, y)$. Let W be the set of all pixel coordinates within the bounding box defined by $\hat{\mathbf{b}}_1(t|t-1)$. Finally, let $\mathbf{c} = (x, y)$ be the vector with as its components the coordinates of the center of this bounding box. Then the skin likelihood observation $\mathbf{y}_2$ is obtained by performing a single mean-shift iteration:

$$\mathbf{y}_2 = \hat{\mathbf{b}}_1 + \Delta\hat{\mathbf{b}}_1 \quad : \quad \Delta\hat{\mathbf{b}}_1 = \frac{\sum_{\mathbf{x}\in W} K(\mathbf{x}-\mathbf{c})\, L(\mathbf{x})\, (\mathbf{x}-\mathbf{c})}{\sum_{\mathbf{x}\in W} K(\mathbf{x}-\mathbf{c})\, L(\mathbf{x})}, \tag{3.54}$$

where $K(.)$ defines a smoothing kernel.

Often a Gaussian kernel is used such that points that are closer to the initial estimate receive more weight than points farther away. Nevertheless, we propose to use a simple uniform kernel, the bandwidth of which is defined by the state of the particle under consideration. This allows us to re-use the previously calculated integral images to execute the mean-shift iteration in constant time.

The measurement noise matrix of a Kalman filter is usually chosen by hand during filter design. However, we can estimate the measurement noise matrix R_2, based on the skin likelihood map, by considering all (x, y) coordinates in the bounding box defined by $\mathbf{y}_2$ as weighted samples, where the weights are again defined by the skin likelihood. We then find the weighted mean μ^* by running a second mean-shift iteration. Finally, if W represents the set of all coordinates within the bounding box defined by $\mathbf{y}_2$, we find the weighted covariance matrix [Price 72] as follows:

$$R_2 = \frac{\sum_{\mathbf{x}\in W} L(\mathbf{x})}{\left(\sum_{\mathbf{x}\in W} L(\mathbf{x})\right)^2 - \sum_{\mathbf{x}\in W} L(\mathbf{x})^2} \sum_{\mathbf{x}\in W} L(\mathbf{x})\, (\mathbf{x}-\mu^*)^T\, (\mathbf{x}-\mu^*)\,. \tag{3.55}$$

This covariance matrix is used as the measurement noise matrix for the Kalman filter and represents the uncertainty about the skin likelihood based measurement.

3.5 The Complete Tracking System

In this section, we introduce an efficient segmentation method to obtain a rough approximation of the contour of the tracked hand. This segmentation is then used to dynamically update the color models of the particle filter, and to introduce a learning stage in the Random Hough Forest classifier.

Furthermore, an error detection and recovery method is discussed, which allows the object tracker to automatically and dynamically determine the number of objects to be tracked.

3.5.1 Hand Segmentation

Segmenting the tracked hand from the video frame, allows us to feed back the hand contour information to the particle filter, in order to refine the color models and to ease the detection of tracking failure. Additionally, many tasks that are outside the scope of this dissertation, such as hand pose recognition, benefit from a hand segmentation result.

Although image segmentation in general is an ill posed problem with many possible solutions, our segmentation problem can be solved relatively easily, because the location and size of the hand is already known from the particle filter tracking stage. The parameters $\hat{\mathbf{b}} = (x, y, w, h)$ of the bounding box of the tracked object are obtained by calculating the expected value of the posterior distribution $p(\mathbf{b}_t|z_{1:t})$ that is represented by the sample set of the particle filter, as illustrated by equation (3.56).

$$\bar{\mathbf{b}}_t = E[p(\mathbf{b}_t|z_{1:t})] = \sum_{i=1}^{N} w_t^i \mathbf{b}_t, \tag{3.56}$$

where w_t^i are the normalized particle weights such that $\sum_{i=1}^{N} w_t^i = 1$.

During tracking, a hand was represented by its bounding box, mainly because such an approach allows us to greatly reduce the computational complexity by using integral images. Also, an upright bounding box can be represented by only four distinct parameters, which reduces the dimensionality of the real problem. A more accurate description of the size and location of the hand could be obtained by representing the hand by its surrounding ellipse, instead of its bounding box. Moreover, an even more precise description would also model the in-plane rotation of the hand, apart from its location and size. Therefore, before trying to solve the segmentation problem, we propose to fit an ellipse to the tracked hand, based on knowledge that originates from the particle filter. The oriented ellipse represents a more precise approximation of the hand contour than the bounding box, and is used to obtain a rough segmentation as explained in the next paragraphs.

The observation model of the particle filter, described in Section 3.3, already provides a probabilistic description of the most likely hand locations, as illustrated

in figure 4.6. This probability image is an integration of skin detection, background modeling, motion detection and discriminative Hough detection, and is a more discriminative base for segmentation than the original RGB video frame.

To obtain an elliptical description of the hand, starting from its bounding box and the probability image that resulted from the particle filter's observation model, the Camshift algorithm [Exner 10] is used. In essence, the Camshift algorithm consists of a mean-shift optimization step, and a size and orientation estimation step. During the mean-shift optimization, the bounding box is iteratively shifted towards the expected value, or normalized first moment, within its support. After convergence, The size of the bounding box is adapted as a function of the zeroth moment which has already been calculated by the mean-shift operation. The orientation of the bounding box is determined as a function of the second moments, which describe the variance within the bounding box.

The Camshift iteration thus yields an oriented ellipse that more closely fits the hand than the original bounding box, especially if the hand is rotated. We then propose a simple active contour based approach to obtain the final segmentation. While complicated and robust active contour like algorithms exist, these are often computationally expensive due to their need for a global optimization algorithm or due to their iterative nature. Instead, we propose a very simple algorithm that corresponds to an active contour approach where the internal energy is set to zero. Smoothness is then imposed by a postprocessing step on the resulting contour.

First, the ellipse, obtained by the Camshift iteration, is approximated by a finite set of discrete points. Each of these points, $(p_1, p_2, ...p_k)$, serves as a control point of the active contour, and is only allowed to move along the normal vector of the ellipse at that point. The normal vector $\mathbf{n} = (x, y)$, at a point $p_i = (x_i, y_i)$ and with length M, is defined by equation (3.57).

$$\mathbf{n} = M\frac{(y_{i+1} - y_i, x_i - x_{i+1})}{|\mathbf{p}_{i+1} - \mathbf{p}_i|}, \tag{3.57}$$

where $|.|$ is the length of the vector.

The length M of the normal vector defines the maximum distance in pixels that the point can deviate from its original location on the ellipse. In our implementation, we set $M = 20$. The line along which the control point p_i is allowed to travel, is then defined by equation (3.58).

$$\mathbf{N} = \mathbf{p}_i + s \cdot \mathbf{n} \qquad s \in [-1, 1]. \tag{3.58}$$

By varying parameter s, a cost function can be calculated at each location on this line, after which the location that corresponds to the smallest cost is chosen as the final location of the control point. The cost function is based on the probability image that was calculated earlier and used in the observation model of the particle filter. However, due to non-uniform illumination, cluttered backgrounds and motion, and camera noise, this probability image is often very noisy.

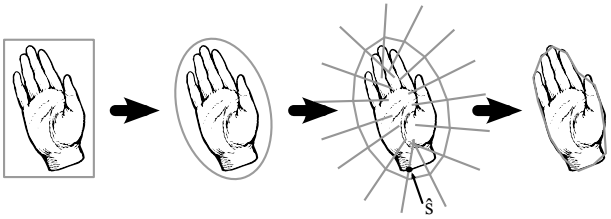


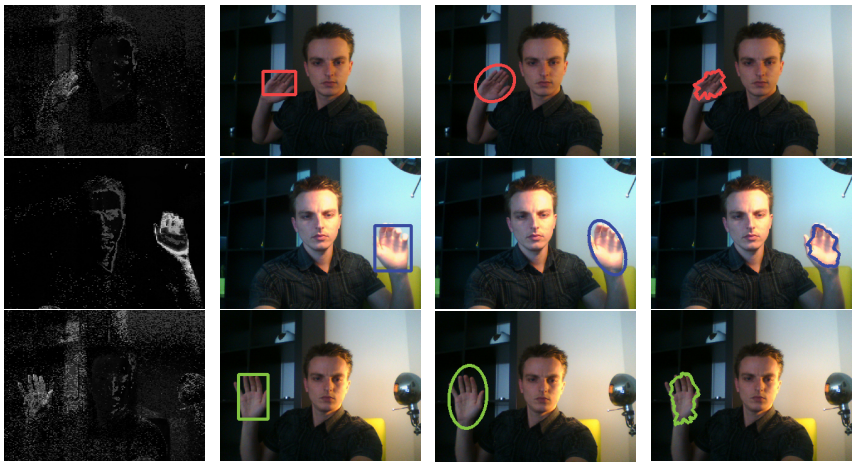
Figure 4.11: Conceptual illustration of the segmentation process: The bounding box, returned by the particle filter tracker, is used to initialize a Camshift iteration. The Camshift algorithm yields an oriented ellipse, surrounding the hand. This ellipse serves to initialize an active contour segmentation algorithm which relocates a set of contour points in the direction defined by their normal vector.

For this reason, traditional cost functions that are often used in active contour approaches, such as the image gradient, can not be used directly. Instead, we assume that the optimal value $\hat{s}$ of parameter s should be chosen such that difference between the average skin probability on this line for $s < \hat{s}$ and the average probability for $s > \hat{s}$, is maximized. This can be interpreted as finding the maximum gradient on a line segment, where the gradient is approximated using finite differences with a window size that equals $2M$, and where the boundaries of the line segment are extended by the average value between the boundary and the window center. Due to the smoothing introduced by this operator, the cost function is robust against noise and errors in the probability image. The complete process is illustrated conceptually by figure 4.11.

After each control point has been relocated to its optimal position, the resulting contour is smoothed by a Gaussian kernel. Figure 4.12 illustrates the probability image that is used as the input for the segmentation algorithm, and each of the subsequent segmentation steps for several very challenging video frames where most traditional segmentation methods would fail.

To significantly decrease the computational complexity of finding the optimal value $\hat{s}_i$ for each control point $\mathbf{p}_i$, we extend the concept of integral images to lines, resulting in a simple two pass algorithm. In the first pass, the line integral $I(s)$ is calculated for each line $\mathbf{N}_i$ that corresponds to control point $\mathbf{p}_i$. In the second pass, the optimal value $\hat{s}_i$ can then easily be computed, since the sum of all skin probabilities on this line for $s < \hat{s}$ equals $I(s-1)$, while the sum of all probabilities for $s > \hat{s}$ equals $I(M) - I(s)$.

The resulting algorithm is extremely efficient, its computational complexity being negligible when compared to the complexity of the tracking and detection algorithms. Also, the segmentation method does not depend on sharply defined edges or smooth gradients, and does not directly use color information, making it robust against motion blur, illumination changes and camera noise.



(a) Input image (b) Tracking result (c) Camshift result (d) Segmentation

Figure 4.12: Hand segmentation in challenging illumination conditions.

3.5.2 Initialization and Error Recovery

The proposed tracking algorithm is able to overcome local maxima that are caused by background clutter, moving objects, illumination changes, skin-like colors or noise. This robustness is mainly achieved by assuming temporal consistency, expressed by means of a motion model. However, at some point in time, the tracking algorithm will inevitably make a mistake, causing tracking failure.

Since our tracking system continuously learns about and adapts to its environment, tracking failure results in incorrect models, inducing even more tracking failures in the future. This is often referred to as 'drift'. To avoid such behavior, error recovery procedures are necessary, allowing the particle filter to recognize tracking errors and to re-initialize.

Before tracking can even begin, the tracking algorithm needs to be initialized since the distribution $p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)$, in the numerator of equation (3.11), is unknown before initialization at time instance $t = 0$. Therefore, tracking initialization corresponds to choosing a prior distribution $p(\mathbf{x}_0^i)$ over the filter state variables.

Both the error detection and recovery problem, and the (re)initialization problem, could be solved if a real-time hand detector were available that is more reliable than the tracking algorithm. However, if such detector were available, then particle filtering itself would not be needed, as tracking could then be achieved simply by re-detecting the object in each video frame. A reliable detector would be a detector yielding both a high precision, and a high recall. Precision measures the fraction of true positive detections in the set of positive detections returned by

the classifier, while recall measures the fraction of true positive detections in the set of all detections that should have been positive.

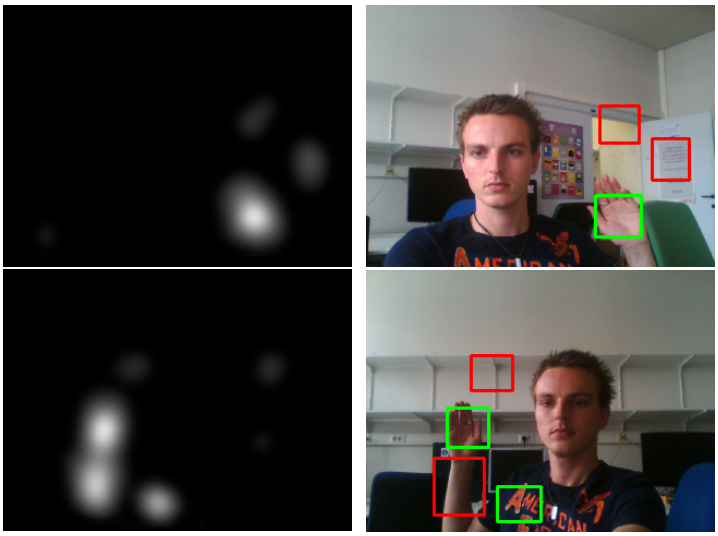
In Section 3.3.4, we introduced an efficient and robust hand detector, based on Random Hough Forest classification. This detector is a high recall detector, which means that it can positively detect the occurrence of a human hand in an image most of the time. As a result, the detector yields a low precision, which means that only a fraction of the hand detections are in fact hands.

The Hough voting map that is generated by the classifier, is used in the likelihood map of the particle filter, were a high recall is more important than a high precision, as a particle filter can easily overcome noisy observations due to its probabilistic nature. For reliable tracking initialization or error detection and recovery, both recall and precision should be low. Moreover, since the detection process needs to run in parallel with the tracking algorithm, the classifier needs to be efficient and should only use a fraction of the available CPU power.

Similar to the approach described by Mittal *et al.* [Mittal 11], we propose a cascade of classifiers to solve this problem. The first classifier in the cascade is the earlier described Random Hough Forest classifier that solves the most difficult problem of generating hand location hypotheses. The second classifier in the cascade is a simple linear classifier that quickly rejects a large amount of false positive detections by classifying the resulting hand as a whole, instead of classifying the parts that make up the hand. The third classifier in the cascade is a decision tree that uses only temporal information to further reject false positive detections. This temporal information includes the temporal variance of the estimated hand location amongst others, and will be discussed on page 149. Cascading an advanced high recall classifier with two simple high precision classifiers results in a fast and robust classifier that can be used for particle filter initialization and error recovery in video. The next paragraphs explain each of the cascade stages.

The Random Hough Forest classifier returns a probability map that, for each pixel, indicates the probability of being part of a hand. In order to actually extract possible hand locations from this voting map, the local maxima need to be located. The variance or support of each local maximum, indicates the size of the hand. To quickly obtain a set of hand hypotheses, we first locate the maxima in the probability map, and then use the Camshift procedure that was described earlier to obtain the location and size of each hand hypothesis.

The Random Hough Forest classifies object parts instead of complete objects. As a result, if several objects have a similar appearance and spatial configuration as hand parts, a local maximum is generated in the voting map, resulting in a false positive detection. To quickly discard obvious false positives, we trained a simple linear classifier such that its precision is higher than the precision of the Random Hough Forest classifier. The linear classifier classifies hypothesized hand bounding boxes instead of image parts, and simply reuses the earlier calculated



(a) Hough voting map (d) Hand detection

Figure 4.13: Hand hypotheses, generated by the Random Forest classifier, are illustrated in green, while several false positives, shown in red, are rejected by the second stage linear classifier.

features for performance reasons. An example of the filtering capabilities of the second stage classifiers is shown in figure 4.13, where the red rectangles represent hand hypotheses that were rejected by the linear classifier.

For each video frame, the first stage of the hand detector yields a set of hand hypothesis which is then refined by the second stage linear classifier. For each of these hypotheses, a hypothesis verifying particle filter is then used to gather temporal statistics during the next N frames. This particle filter does not update any of the color models and does not assume to be correctly tracking a hand. Instead, it is only used to gather temporal information describing the tracked object, which can then be uses by the third stage classifier to verify or reject the hand hypothesis.

Initializing the particle filter corresponds to choosing the prior distribution $p(\mathbf{x}_0^i)$ at time $t = 0$ in equation (3.11). Because the hand detector returns bounding boxes, the location and dimension of these bounding boxes correspond to the state variables used by the particle filter. Hence, we can directly use the bounding boxes obtained by the detector, to generate the prior distribution from which initial samples are drawn by the particle filter. We define the prior distribution as a four dimensional normal distribution, the means of which are defined by the detected hand's bounding box parameters. A diagonal covariance matrix is used, with empirically defined variances that represent the uncertainty of the hand detector.

While the first and second stage classifier only use spatial information to detect hand hypotheses and to reject false positive detections, the third stage classifier exploits the available temporal information in video, to further refine the set of hand hypotheses and to detect tracking errors. The third stage classifier is a simple decision tree that was trained using 10-fold cross validation, and pruned afterwards. This decision tree evaluates whether the tracked object is a hand, by examining its temporal behavior during the past N frames. This classifier uses four simple temporal features that are discussed in the next paragraphs.

A first feature that often discriminates correctly tracked hands from incorrectly tracked objects, is the variance of the estimated (x, y) positions in the recent past. A small variance often indicates a stable track, while a large variance often indicates a tracking error. For instance, when the hand leaves the camera's field of view, the particle filter tries to find a new local maximum in the probability image, resulting in random behavior and an increased temporal variance. In order to transform this covariance matrix into a single feature value, we calculate its Frobenius norm, which is often used as a measure to compare covariance matrices. This measure allows us to combine the variances and covariances instead of using each matrix element as a distinct feature, thereby reducing the risk of overfitting.

A second feature that could indicate tracking problems at time instance t, is the uncertainty of the particle filter itself at this time instance. The current uncertainty of the state estimator, is defined by the covariance of the current particle set. Although this covariance is calculated at a single time instance, it implicitly contains temporal information because of the recursive nature of the particle filter algorithm. If particles are widely spread, the observations during the past few frames were either very ambiguous, or very unlikely. Alternatively, if the particles have a small variance, the observations closely matched the predictions in the past few frames, indicating stable tracking. Similar to the first feature, we also combine all elements of this covariance matrix into a single feature value, by calculating its Frobenius norm.

The third feature that is used to detect incorrectly tracked hands, is the average amount of motion during the recent past, as detected by our frame differencing motion detector that was described earlier. If the amount of motion is very small for a long period of time, the tracked object might be a background object instead of a hand. Furthermore, if the amount of motion, detected by frame differencing, is not proportional to the first feature value that describes the spread of the particle filter's expectation in the past, there might be a tracking error.

The fourth feature used by the decision tree, is simply the Random Hough Forest's detection consistency in the recent past. If a certain hand hypothesis is being detected consistently in every video frame, its likelihood to be an actual hand is larger than if the hypothesis is only generated sporadically.

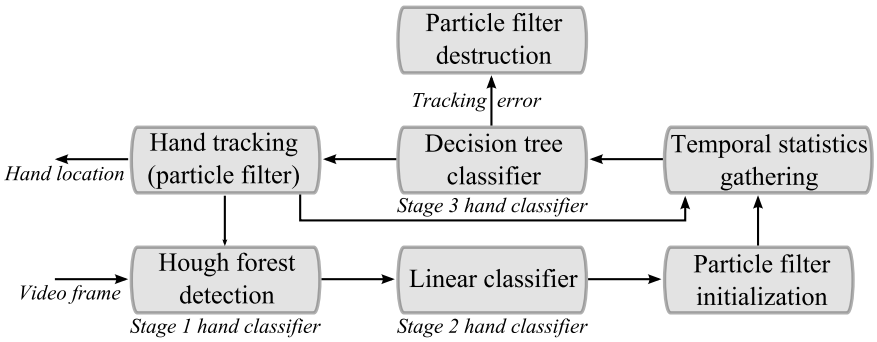


Figure 4.14: Schematic representation of the cascaded hand classification and tracking algorithm with automatic initialization and error recovery.

Finally, **the fifth feature** is related to the segmentation algorithm. If an object resulted in a very inconsistent segmentation during the recent past, this might indicate a tracking error. For instance, if a hand is occluded or outside the video frame, the particle filter might get stuck on a skin-like surface in the background. During segmentation, the contour will then try to expand as much as possible. However, its radius can not be larger than M, due to the nature of our segmentation approach. Therefore, if the particle filter gets stuck on a skin like background surface, the segmented contour will have a radius close to M such that its area is similar to the area of the original bounding box. Accordingly, the average ratio of contour area and bounding box area in the recent past is used as a feature.

Temporal statistics are gathered by each hypothesis verifying particle filter using a sliding window of size N, such that the classifier's decision at each point in time is based on the past N frames. If the third stage classifier classifies a hypothesis to be hand, the hypothesis is considered verified, and actual hand tracking is started. During hand tracking, the third stage classifier continues to verify the validity of the tracked object in each new video frame, in order to detect tracking errors or drift. This approach allows for automatic initialization and error recovery, which is of great importance in real life applications. Figure 4.14 schematically illustrates this process, that is repeated for every video frame.

Each particle filter is described by one of three states: 'correct', 'uncertain' or 'incorrect'. A hypothesis verifying particle filter is assigned the state 'uncertain'. Based on its decisions in the next frames, the state then transitions to either 'correct' or 'incorrect'. Incorrect filters are removed, whereas correct filters are continuously verified for n time instances, and can transition to 'uncertain' and eventually 'incorrect', based on the three-stage classifier's decisions during this timespan. The effect of the 'uncertain' state is similar to that of hysteresis thresholding. A hypothesis is only considered verified when a lot of evidence is available,

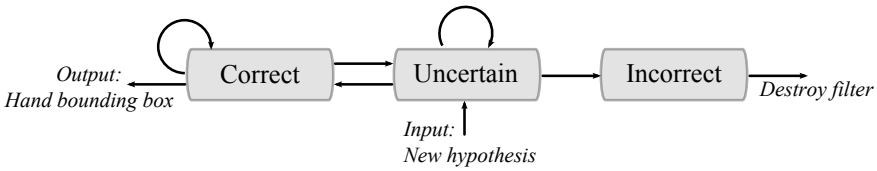


Figure 4.15: Each particle filter transitions through three possible states, based on the results of the three-stage hand classifier.

but needs little evidence to be kept alive. This state transition scheme is illustrated by figure 4.15.

Based on the known position and dimensions of the tracked hands, each hand is segmented using the algorithm described in Section 3.5.1. This segmentation is then used to adapt and improve the classifiers and tracking models in three ways.

First, the ratio of the area of the segmented object and the area of the tracked bounding box is used as a feature by the third stage hand classifier. If the area of the segmented object is much larger or much smaller than the area of the bounding box, a tracking error likely occurred.

Second, the segmented contour is used to learn a context for the first stage Random Hough Forest classifier. In the first stage classifier, object patches cast a probabilistic vote on their learned centroid location. For each hand hypothesis that is extracted from the resulting probability map using the Camshift algorithm, we know the set of patches that contributed to this hypothesis. For each hand hypothesis, all contributing patches that correspond to centroid locations falling within the segmented area, are then considered positive patches, while patches whose centroid location falls outside of the segmented area, are considered negative patches. In each leaf node of each decision tree in the random forest, statistics are kept about the number of times T_p the node was reached by a positive patch, and how many times T_n the node was reached by a negative patch. This allows us to generate a context specific prior that steers the classifier, such that patches that are consistently classified incorrectly are assigned a lower probability in the future. For more information about our Random Hough Forest classifier, we refer to chapter 2.

The **third** way in which the segmented contour is used to adapt the tracking system is related to the color models. The segmented area is used to update the color models that are used in the observation model of the particle filter. Pixels that fall within the contour are used to update the skin histograms, while pixels falling outside the contour are used to update the non-skin histograms. As discussed in Section 3.2, three different Bayesian skin classifiers are used. The offline trained skin classifier operates in RGB space, and is considered static, in that its models are not updated online. This makes sure that the classifier still

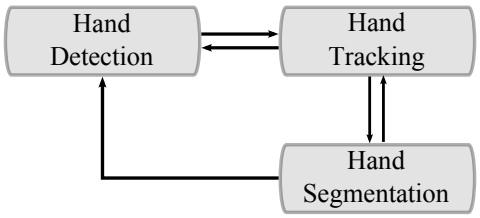


Figure 4.16: Each module feeds back information to previous modules, in order to allow for error detection and automated learning about a specific environment.

yields reasonable results in case of tracking drift. The second classifier operates in the Hue-Saturation color space and its skin and non-skin models are updated, based on the segmentation results. To update these histograms, only those pixels are used that fall within the segmented area, and that have a high skin probability. The histograms are accumulated and smoothed over a certain number of frames, to allow for gradual illumination changes, while maintaining stability. The third classifier is updated every video frame, and allows the tracker to cope with sudden illumination changes.

Figure 4.16 illustrates this feedback concept. The hand detection module feeds bounding boxes to the hand tracking module, which feeds information back to the third stage hand classifier to allow for error detection. Furthermore, the tracking module delivers bounding box information to the segmentation module, which feeds back segmented contour information, to allow the particle filter to update its color models. Finally, the segmentation module feeds back segmentation information to the detector module, to enable an online learning process in the first stage hand classifier, and to improve error detection by the third stage hand classifier.

4 Results and Discussion

In this section, we evaluate the proposed tracking scheme on two challenging datasets, and we compare our results with state-of-the-art tracking solutions. The first dataset was proposed in [Spruyt 12] and contains eight video sequences, each of a duration of approximately 80 seconds. The video sequences are extremely challenging, containing fast motion, occlusion, changing illumination, cluttered backgrounds, and a moving camera. Table 4.1 summarizes the characteristics of each video sequence. These sequences are publicly available[1] for research purposes and contain carefully annotated ground truth locations and sizes for each hand.

[1] `http://telin.ugent.be/~vspruyt/icip2012/`

Table 4.1: Test dataset specification: Each video sequence used to evaluate the tracking algorithm can be described by a set of characteristics such as whether or not hand occlusions (Occl. in this table) occur, or whether the person is wearing short or long sleeves.

#	Scene	Lighting	Motion	Sleeves	Camera	Occl.	Length
1	cluttered	dark	normal	long	fixed	no	83 s
2	cluttered	bright	normal	long	fixed	no	82 s
3	normal	normal	fast	long	fixed	no	85 s
4	normal	bright	normal	long	moving	no	88 s
5	normal	normal	normal	long	fixed	no	89 s
6	cluttered	normal	normal	short	fixed	no	84 s
7	normal	changing	normal	long	fixed	no	85 s
8	normal	normal	normal	short	fixed	yes	83 s

To evaluate a tracker's accuracy, the Pascal VOC score [Everingham 10] is widely used in literature [Gall 09, Mittal 11]. The VOC score defines the amount of overlap between the detected bounding box, and the given ground truth bounding box. The score can be calculated as $VOC = \frac{B_g \cap B_d}{B_g \cup B_d}$, where B_g represents the pixels within the ground truth bounding box, whereas B_d represents the pixels within the detected bounding box. Therefore, detected bounding boxes that completely contain the ground truth bounding box, are penalized if their size is much larger. On the other hand, detected boxes that are completely contained within the ground truth bounding box, are penalized if their size is much smaller. In general, only those detections for which $VOC \geq 0.5$ are considered 'true positive' detections.

In the following, we compare our tracking solution with six well known tracking methods from literature. The first method is the Camshift algorithm [Exner 10]. Although the Camshift algorithm might not be considered a state-of-the-art tracking solution, it is often used as a baseline when evaluating tracking methods in computer vision. The second method is the HandVU system [Kölsch 10] that was discussed in Section 2. The HandVU algorithm tracks hands using a combination of color and Harris corner feature matching, by propagating random samples that are spatially constrained by flocking behavior rules. Third, we compare our method with the RTseg solution that was proposed in [Spruyt 10a], and discussed in Section 2. This method combines multiple color and motion cues in a Gaussian Mixture Model to track hands in unconstrained environments.

Whereas the above methods could be considered generative tracking approaches, we also compare our results with three state of the art object tracking algorithms based on discriminative template matching. The Predator tracker [Kalal 12] is a recent patch-based tracking solution that is able to automatically learn the object's appearance over time in order to adapt to its environment. The adaptive basin hopping Monte Carlo filter (BHMC) proposed in [Jun-

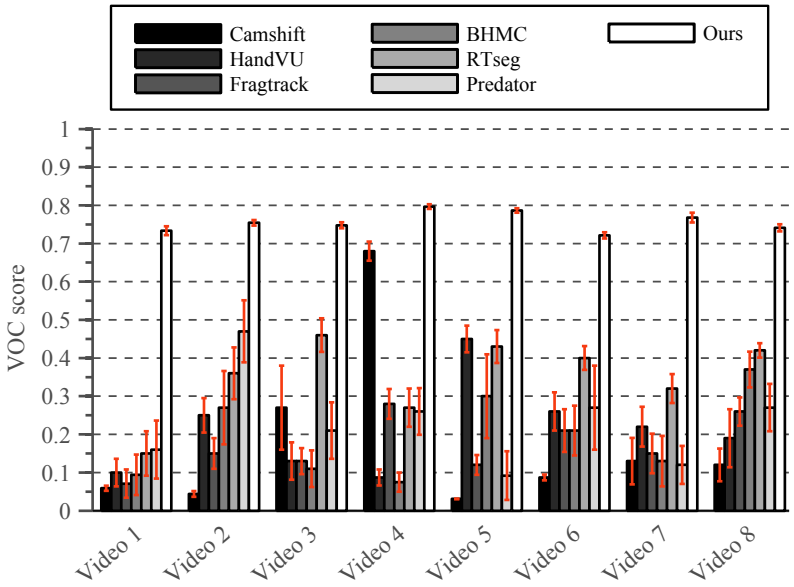


Figure 4.17: VOC score comparison for different tracking algorithms.

seok 09] is another adaptive patch based tracker, specifically designed for tracking non-rigid objects. Finally, the Fragtrack algorithm [Amit 06] is an adaptive patch based tracker using a Hough voting scheme and integral histograms.

For each video sequence, all of the above methods were initialized manually based on the ground truth bounding box in the first video frame. Figure 4.17 shows the average VOC scores that were obtained with each of these methods on every video sequence. The variances of the VOC scores are indicated as error bars.

This result illustrates the difficulty of robustly tracking hands in unconstrained environments. Although methods such as the Predator tracker and the BHMC tracker are sophisticated tracking solutions that are able to robustly track object such as cars or persons, they fail to accurately track human hands due to the lack of clear discriminative features. Furthermore, the variance of the VOC scores obtained with our method is much smaller, indicating very stable tracking results.

Figure 4.18 shows the VOC score of the three top performing algorithms, plotted against time, for the first 2000 frames of several videos of this dataset. Whereas most methods tend to fail when the hands overlap with the face (e.g. Video 2, frame number 600) or with each other (e.g. Video 8, frame number 700), or in case of sudden changes in hand size (e.g. Video 8, frame number 1200), our solution is able to cope with these situations and to quickly recover from failure. Moreover, while the optical flow estimates, used in our proposal distribution, allow the filter to cope with challenging (Video 2) and changing (Video 7) illumination, the

mean-shift iteration in the proposal distribution enables it to overcome incorrect flow estimates in case of fast camera motion (Video 4).

The advantage of sampling from a proposal distribution that incorporates the latest observations, instead of using the transition prior as a proposal distribution, is illustrated more clearly by figure 4.19. In this experiment, we first changed the proposal distribution of our method such that it corresponds to the transition prior, resulting in a bootstrap particle filter (Proposal: TP). Next, we added the mean-shift optimization step using the Kalman filter approach discussed in Section 3.4.4 (Proposal: TP+MS). Finally, the optical flow estimates were added as discussed in Section 3.4.2 (Proposal: TP+MS+OF). These results clearly show that incorporating the latest observations into the proposal distribution greatly increases the tracker's accuracy and stability.

The main reason for this increase in performance, is that the limited number of samples (50 samples in our experiments) is used much more efficiently. Instead of wasting particles by sampling from the transition prior and assigning a low weight to most of them, our proposal distribution yields samples that closely model the final posterior. As a result, our method suffers less from particle depletion and therefore does not depend on the resampling process as much as traditional bootstrap filters. By limiting the number of resampling steps, the algorithm is able to explore the tails of the posterior distribution while at the same time concentrate most of its computational power on the mode of the distribution.

To illustrate this, we disabled resampling and generated figure 4.20 by drawing particles in red and the mode of the distribution in white. For illustrative purposes, only the tracking results for a single hand are shown. The first and the third row show the sample set when using our proposal distribution, whereas the second and fourth row show the sample set when using a bootstrap filtering approach. In the traditional bootstrap filter case, the particle set quickly diverges, resulting in particle depletion where few particles have a high likelihood. Therefore, this method heavily depends on the resampling process which in turn tends to cause particle impoverishment.

Figure 4.21 shows the tracking result on several randomly selected video frames of each of the videos in the dataset.

Finally, table 4.2 shows the processing times for each of the tested algorithms on a single 320×240 video frame of the dataset. Our method operates in real-time, processing approximately 24 frames per second on modern hardware (I7 quadcore CPU, 8 GB RAM). This includes the hand detection, segmentation, automatic initialization, error recovery and online learning procedures.

Furthermore, figure 4.22 shows that the computational complexity of our method is linear in the number of pixels, and that video resolutions up to QVGA can easily be processed in real-time. Moreover, the algorithm was not parallelized

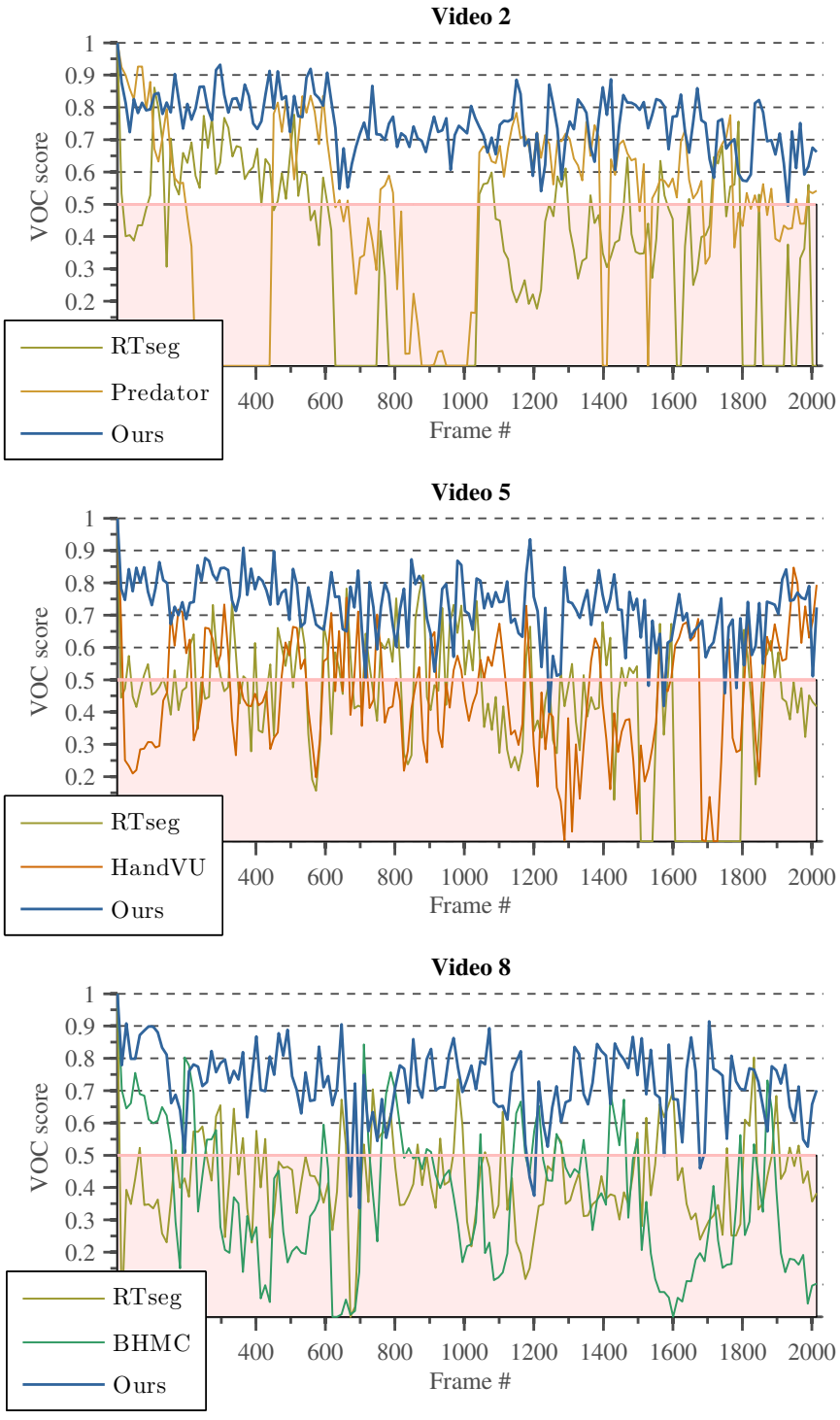


Figure 4.18: Time evolution of VOC scores for the top performing algorithms.

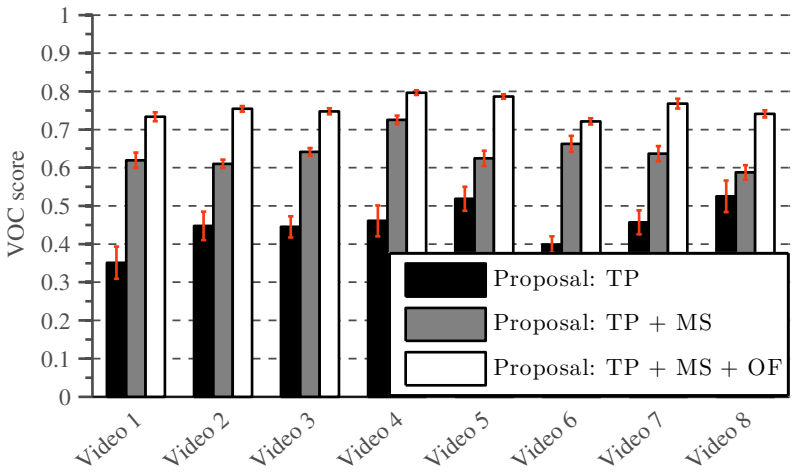


Figure 4.19: Comparison of different proposal distributions (Transition Prior, Mean-Shift, Optical Flow).

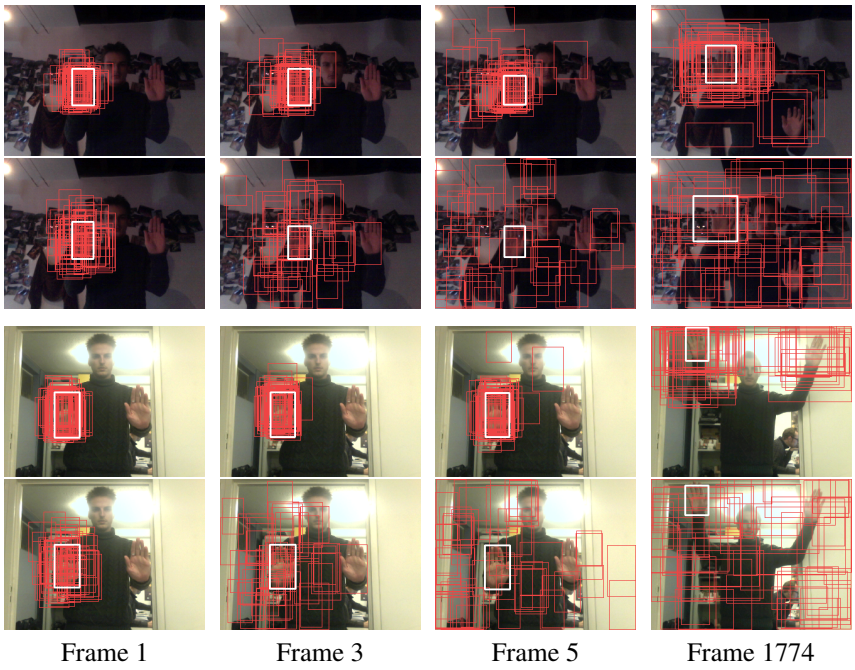


Figure 4.20: Particle depletion in traditional bootstrap filters (second and bottom row).

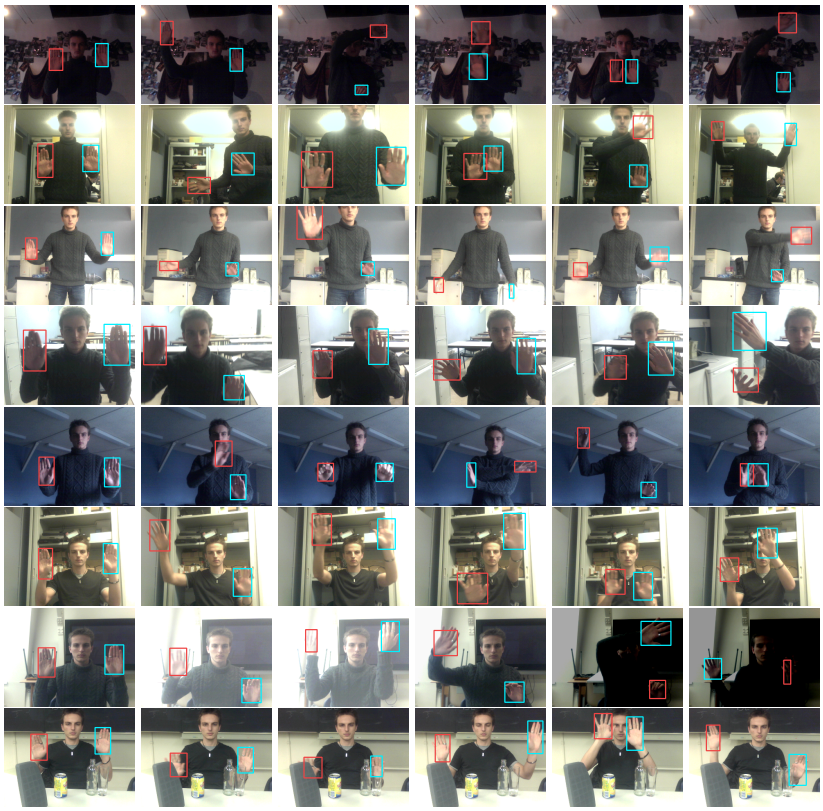


Figure 4.21: Tracking results on eight challenging video sequences.

Table 4.2: Average processing times in milliseconds per video frame of size 320×240 pixels.

Method	Processing time (ms)
Camshift	13.56
Ours	**41.69**
HandVU	53.18
Predator	45.62
Fragtrack	79.98
RTseg	95.72
BHMC	1036.85

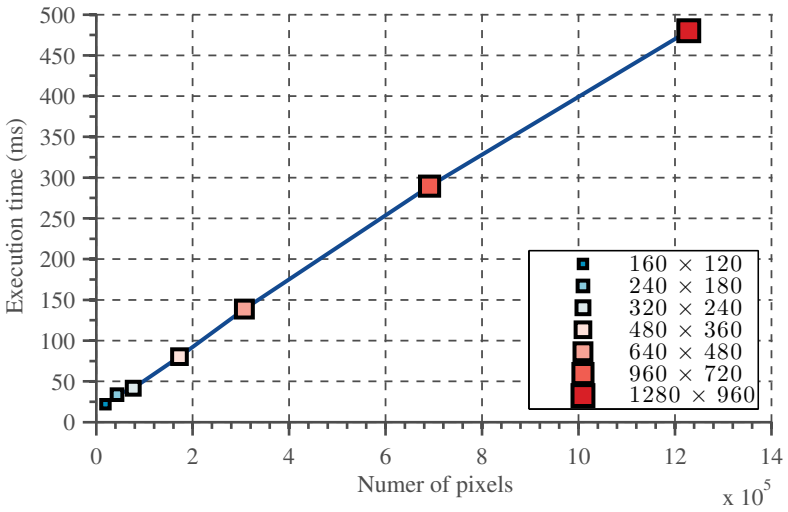


Figure 4.22: Execution time, plotted against the number of pixels in an image.

such that it did not benefit from a multi-core system. Parallelizing parts of our algorithm would therefore reduce processing times even further.

Figure 4.23 shows the execution times of each the components of the tracking and detection system, obtained by averaging the timing results of 5000 video frames of size 320×240.

Whereas the above results clearly illustrate the superior accuracy and robustness and the real-time characteristics of our tracking solution, it does not necessarily prove its error recovery capabilities. Therefore, a second experiment was used to test the long-term tracking behavior and the error recovery procedures of our method. For this experiment, we used the publicly available 'Signers dataset' [Buehler 08] that contains five BBC news sequences. We compare our results on this Signers dataset with the results obtained by Karlinsky *et al.* [Karlinsky 10].

As discussed in Section 2, Karlinsky *et al.* fit a chain model to the body of the signer in order to detect hands. Chain fitting starts from the ground truth bounding box of the face and thus only works if such ground truth is available and if the face is visible. The main disadvantage of this technique is that the initial head position has to be known, and that the technique is rather computationally expensive and thus not suitable for real-time applications. The detected hand is considered correct if it is within half face width from the ground truth location of the hand. Detection performance is reported within the top k detections per ground truth hand instance.

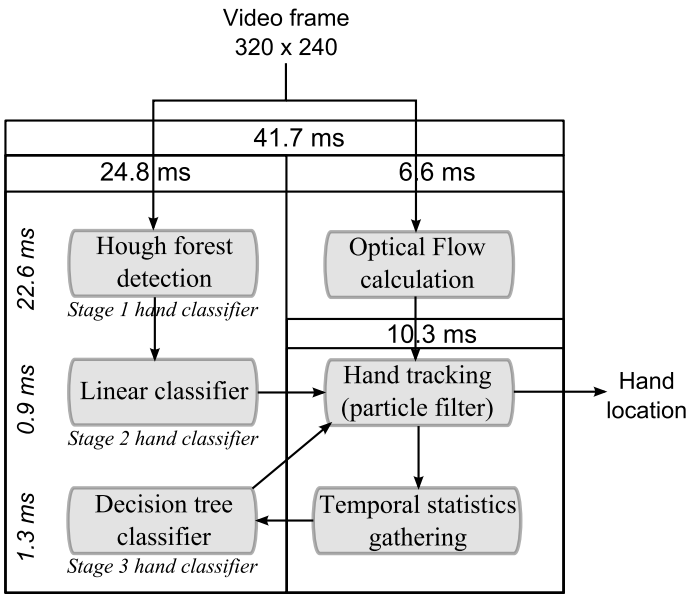


Figure 4.23: Execution timings of each component of the tracking and detection system, obtained by averaging the timing results of 5000 video frames of size 320×240.

Table 4.3: Results obtained on the Signer dataset, reported as a percentage by counting the number of times the correct detection was amongst the k-highest-confidence tracking hypotheses in each image.

	2 max	3 max	4 max
Karlinsky	**92.8**	95.4	96.7
Ours	**98.7**	**98.7**	**98.7**

Table 4.3 shows a comparison of our results with the results obtained by Karlinsky *et al.*, whereas figure 4.24 illustrates our results on several video frames from the Signer dataset.

Although tracking fails several times while processing the Signer dataset, our algorithm is able to automatically recover an re-initialize. To illustrate this, figure 4.25 shows an example of tracking error and recovery. The video frames have been cropped for illustrative purposes.

5 Conclusion

We proposed a real-time hand tracking method that robustly tracks hands in unconstrained video. Our solution automatically recovers from failure and does not need a manual initialization step.

Figure 4.24: Tracking results for the signer dataset.

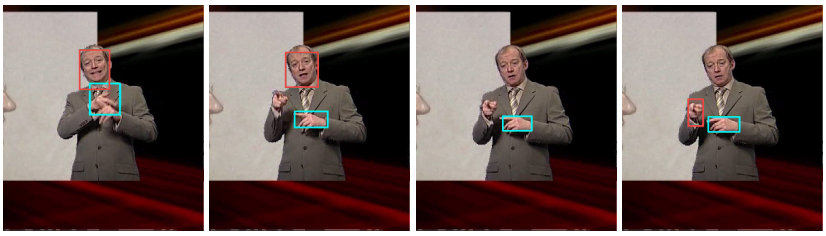


Figure 4.25: Automatic error recovery while processing the Signer dataset.

The traditional bootstrap particle filter is extended by a Kalman filter driven proposal distribution that incorporates the latest optical flow and skin likelihood observations. Furthermore, partitioned sampling is used to greatly reduce the search space dimensionality such that only 50 particles are needed to accurately track hands in challenging conditions.

We integrated a discriminative hand detector with several generative cues to further increase the tracker's robustness, and to provide a method for automatic initialization. A simple yet efficient segmentation method is proposed, and several types of feedback are introduced such that both the tracker and the detector are able to adapt to the video under consideration while avoiding error drift.

We showed that our method outperforms several state of the art tracking algorithms, and we provided an extensive evaluation on three publicly available datasets. Furthermore, the error recovery capabilities of the tracker were evaluated and shown to enable long-term tracking in challenging conditions.

5

Tracking Through Occlusion by Unsupervised Context Learning

1 Introduction

Whereas the task of tracking rigid objects with a known appearance has been largely solved in academic literature, tracking non-rigid objects remains a difficult problem. Specifically, the appearance of human hands greatly depends on their pose and can not be easily described using discriminative features.

In chapter 4 we proposed to combine a particle filter based tracking framework with a discriminative classifier to overcome these difficulties. By combining different cues such as skin-color probability, motion likelihood and optical flow estimates into a probabilistic framework using a sensor-fusion approach, local maxima in the likelihood function can be overcome. We showed that our method can track hands robustly in unconstrained environments and greatly outperforms the state-of-the-art solutions. However, a major shortcoming of the proposed technique is its tendency to be easily distracted when the hand occludes the face, another hand, or skin-colored background objects that exhibit a hand-like visual appearance.

In this chapter we formulate a solution to the above problem, based on the idea that a moving object is usually embedded in a specific context and rarely moves alone. In the case of human hands, an obvious example of such contextual link would be the arm which exhibits a motion pattern that is strongly correlated to that of the attached hand. Although an arm detector could be trained offline based on color cues [Stefanov 05], edge detection [Metaxas 06] or background subtraction

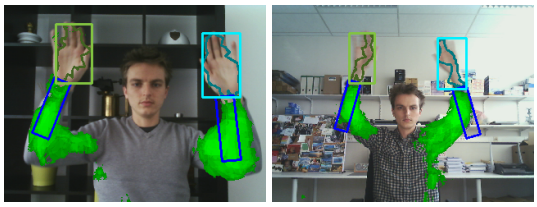


Figure 5.1: Illustration of contextual learning. Objects colored in green exhibit a temporally correlated behavior to the hands, and are learned automatically. Hand and arm bounding boxes are estimated by partitioned sampling particle filtering, and the hand segmentation mask is obtained by active contour modeling.

[Goncalves 95], these methods tend to fail in unconstrained environments where the appearance of the subject's arm is unknown before tracking starts. Instead, we propose a method to exploit temporal correlation, in order to automatically learn which objects appear to be temporally connected to the hand. These objects can then be used to increase tracking performance in case of occlusion of the hand itself.

Furthermore, the proposed solution is not limited to arm tracking. In many practical applications, the tracked hand holds an object such as a smartphone or game controller, or attributes such as an umbrella (i.e. surveillance applications). Whereas this degrades tracking performance if a traditional hand tracking algorithm were to be used [Spruyt 13a], it can actually improve tracking if the system is able to learn that the object's position is temporally correlated with the hand.

Apart from increasing tracking robustness, our temporal learning scheme automatically results in a rough segmentation of the supporting object (e.g. human arm). This could be of interest in many applications such as surveillance or human-computer-interaction. Figure 5.1 illustrates this by drawing the supporting pixels in green, where the intensity of the color indicates the probability of correct classification. The segmentation mask shown by Figure 5.1 and all following figures was obtained by means of a simple and efficient active contour segmentation of the likelihood image described in [Spruyt 13a]. The internal energy was set to zero, and smoothing is imposed by a postprocessing step on the resulting contour. The segmentation mask does not directly contribute to the context learning method described in this chapter and is only shown for visualisation purposes.

It is important to note that the context segmentation, shown in green, is obtained using a single, static video frame, and is not the result of motion detection. Therefore, our method is able to detect and segment the supporting objects even in video frames that do not contain any motion.

We introduce a technique to automatically learn this context in real-time. Based on this continuous learning process, we propose two methods to increase object tracking performance. The first method models the spatial relationship between

context and object of interest, and incorporates this information into the proposal distribution of a particle filter. Therefore, this method naturally extends to object tracking in general. The second method extends the state space of the hand tracker with an extra parameter representing the arm and uses a context-based observation model to obtain the likelihood estimates. This not only yields an estimate of the arm location, but also increases tracking robustness in case of occlusion.

The remainder of this chapter is outlined as follows: In section 2 we briefly discuss related work from literature. Sections 3.1 and 3.2 provide an extensive discussion of the online learning process. In section 3.3 we discuss how the contextual information is incorporated into the proposal distribution of a particle filter to improve tracking robustness, and in section 3.4 we propose an efficient method to track the arm together with the corresponding hand. Finally, in section 4 we evaluate our algorithm by comparing the results with state-of-the-art solutions using three publicly available datasets.

2 Related Work

Learning the context of an object of interest is known to increase detection performance of object detectors. For instance, Mittal *et al.* [Mittal 11] explicitly train a parts based deformable model to capture the arm's appearance while designing a hand detector. Their evaluation shows that including contextual information yields a 10% increase in recall.

Similarly, Torralba [Torralba 03] proposes to use a statistical model of low-level features to obtain a prior that is used to improve object detection rates. However, since object detectors are trained offline and should be able to detect objects embedded in an unknown context, only statistically significant relations between the context of the training data and the objects of interest can be learned. In contrast, object trackers are embedded in one specific context and could learn this context online. Furthermore, tracking can benefit from learning temporary links between the context and the tracked object. These links might appear and disappear at any time but can greatly increase robustness against occlusion and background clutter when available.

Therefore, our goal is to automatically learn a model of the current context that supports the object of interest in an unsupervised manner, based on few and unreliable measurements. The model should adapt online and should be able to quickly learn and forget new information without overfitting.

The idea of unsupervised learning of unlabeled data was explored by Kalal *et al.* [Kalal 10, Kalal 12], resulting in the well known 'predator' tracking algorithm. Kalal *et al.* introduce the concept of structured unlabeled data which is based on the observation that the labels of data in computer vision often exhibit strong spatio-temporal dependencies. Structured unlabeled data is then defined as unlabeled

data for which the label is restricted if the label of related data is known. In a video sequence, the location of an object defines a trajectory in time. Unlabeled image patches close to this trajectory are likely to be samples from the same object class and are therefore considered structured data that can be used to train an online classifier.

However, although the predator algorithm is able to quickly learn an appearance model for the object of interest, it was shown to fail when applied to the task of hand tracking [Spruyt 12]. The main reason for this is that the appearance of hands varies widely throughout the video sequence, making it difficult to learn an accurate model. In this chapter, we therefore extend the concept of structured unlabeled data by considering trajectories of other objects in the scene that show a temporal correlation to the trajectory of the object of interest. Instead of trying to learn the appearance of the object of interest, we propose to learn the appearance of surrounding objects that move coherently with the object of interest, as illustrated by Figure 5.2. These surrounding objects can then be used to steer the tracking algorithm in case of uncertainty due to occlusion or cluttered backgrounds.

A related idea was explored by Cerman *et al.* [Cerman 09]. In their work, objects are tracked using an affine system model. A foreground model representing all coherently moving objects is learned online. Although their approach illustrates that contextual learning can greatly improve tracking performance, this method takes approximately 10 seconds to process a single video frame on modern hardware. Furthermore, a background model is learned offline by processing the complete video before actual tracking starts. Therefore, this solution is not suited for real-time tracking applications.

Grabner *et al.* [Grabner 10] extended the work of Cerman *et al.* by defining so called 'supporters' instead of trying to obtain a complete model of the foreground and background. Supporters are regions in the image that seem to move coherently with the object of interest. An online learned implicit shape model is used to describe the object of interest and its supporting regions by means of SIFT descriptors. A generalized Hough transform is then employed such that each supporting region votes for the object's position. However due to the use of the implicit shape model and several computationally expensive feature descriptors, this approach can not be used for real-time purposes. Furthermore, if supporting regions are anisotropic they can often not be used to obtain an estimate of the object's location. An obvious example are regions on a human arm which are not discriminative in the tangential direction. Finally, the method proposed by Grabner *et al.* simply performs tracking by detection in each video frame and therefore does not fully exploit temporal consistency.

Inspired by the work of Grabner *et al.*, we propose an online learning scheme to model image regions that move coherently with the hand, and we incorporate this information probabilistically in a particle filter framework. Our solution operates

in real-time and as a side-effect also produces a rough segmentation of the supporting objects. Supporting regions provide an estimate of the object's location, which is used to steer the particle filter. We propose a novel proposal distribution that greatly increases sampling efficiency, resulting in robust tracking with only a few particles to represent the posterior distribution of the state estimate.

3 Materials and Methods

Online contextual learning is a continuous process, such that the tracking algorithm becomes more robust as time progresses. At each time instance, parts in the scene that are close to and move coherently with the hand are used to train a classifier in an online manner. Both the process of selecting these training samples, and the classifier that is trained based on these samples, are discussed in the next sections.

At each time instance, every pixel in the image is labeled by this classifier as being either part of the context, or part of the background. Pixels that are part of the hand's context are then used to steer the particle filter, thereby increasing its robustness against occlusion and cluttered backgrounds. In the next sections, we propose two methods to incorporate contextual information into a probabilistic tracking scheme. Whereas the first method can be generally applied to any object tracking problem, the second method is specifically focussed on hand-tracking, and involves estimating the arm's position.

3.1 Image Patch Selection Using Structural Constraints

The remainder of this chapter is based on the hand detection and tracking algorithm that we proposed earlier in [Spruyt 12, Spruyt 13a] and discussed in chapter 4. This method was shown to be able to robustly track hands in unconstrained environments, and to automatically initialize and recover from errors.

Given the hand location and trajectory, we define positive image patches as those patches that appear to move coherently with the tracked hand, whereas negative patches are defined as those image regions that do not exhibit such correlation. This is illustrated by Figure 5.2 in which positive patches are shown in blue, while randomly sampled negative patches are indicated by red rectangles.

We define positive image patches as those patches that are close to the trajectory of the tracked hands. These patches are therefore defined by spatio-temporal constraints. Only those patches within a neighborhood of the hand that move coherently with the hand are considered positive. The labels of patches within this neighborhood that do not exhibit coherent movement, and the labels of patches outside this neighborhood that do move coherently with the hand, are considered unknown. Finally, patches outside this neighborhood that do not move coherently

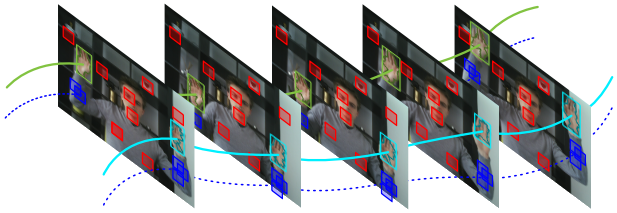


Figure 5.2: Structured data, shown in red and blue, is defined by the known hand trajectories shown in cyan and green. Blue patches exhibit motion that is correlated to the motion of the labeled hands, whereas red patches are negative samples.

with the hand, are labelled as negative patches. This concept of using structured data for unsupervised learning was introduced by Kalal *et al.* [Kalal 10, Kalal 12] and is known as PN-learning. Kalal *et al.* showed formally that iterative PN-learning can converge to a zero-error classifier if certain assumptions are met such that errors that are introduced by incorrectly labeled positive patches and incorrectly labeled negative patches cancel out each other. Although this assumption does not necessarily hold in practice, it was shown that PN-learning using imperfect constraints greatly boosts classification performance.

Since we use spatio-temporal constraints to exploit the structural nature of the unlabeled data, training data is only available at time instances where the tracked hands actually move. To obtain a measure of motion coherence, the real-time, sparse optical flow algorithm that we proposed earlier in [Spruyt 13b] is used. This method yields a sparse, regularized flow-field and was used to steer the motion model of the hand tracking particle filter in [Spruyt 12, Spruyt 13a]. Based on the optical flow estimates, positive samples can easily be defined as those samples in a neighborhood of the hand, whose optical flow vector is similar to the average of all flow vectors within the hand region itself.

In the following let $\mathbf{x} = (x, y)$ be a position in the image, around which we consider a square region with scale s, such that the width of the region is $2s$. We denote these regions by $R_{s,\mathbf{x}}$. Let $\mathbf{f}(\mathbf{x}) = (u, v)$ be the optical flow vector that represents the velocity and direction of the motion of this region from the previous frame to the current frame. Finally, let $\mathbf{h} = (x, y)$ represent the position of the hand in the video frame, and let $\bar{\mathbf{f}}(\mathbf{h})$ represent the average flow within the bounding box of the tracked hand.

A spatial constraint can then be defined simply by considering only patches for which $\epsilon = \|\mathbf{x} - \mathbf{h}\|_2 < \tau$, where τ is a parameter that defines the size of the neighborhood. Similarly, the temporal constraint is defined simply by $\theta = \|\mathbf{f}(\mathbf{x}) - \bar{\mathbf{f}}(\mathbf{h})\|_2 < \delta$, where δ is a parameter that defines the temporal coherence of two optical flow vectors.

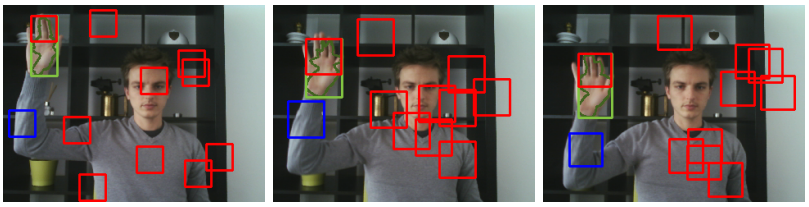
Each image patch $R_{s,\mathbf{x}}$ can then be assigned a label $l(R_{s,\mathbf{x}}) \in \{-1, 0, 1\}$, where $l = -1$ represents the fact that no informed decision can be made, whereas $l = 0$ represents a background patch and $l = 1$ represents a patch that moves coherently with the hand. Image patches are labeled as follows:

$$l(R_{s,\mathbf{x}}) = \begin{cases} 0 & \text{if } \epsilon \geq \tau \text{ and } \theta \geq \delta \\ -1 & \text{if } \epsilon \geq \tau \text{ and } \theta < \delta \\ 1 & \text{if } \epsilon < \tau \text{ and } \theta < \delta \\ -1 & \text{if } \epsilon < \tau \text{ and } \theta \geq \delta \end{cases} \tag{3.1}$$

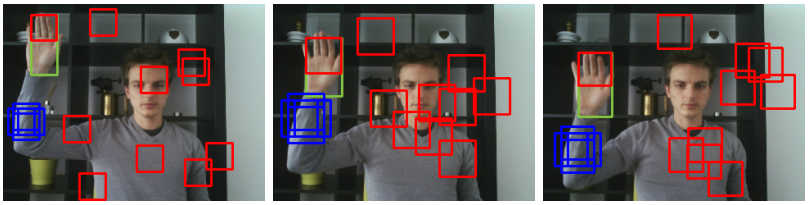
The scale s of the square image patches at time instance t is chosen such that $2s = \min(\mathbf{r})$, where $\mathbf{r} = (w, h)$ represents the width and height of the hand in that video frame. Dynamic scale selection results in a more efficient usage of the available training data and increases scale invariance of the final classifier.

Although real-time, dense optical flow algorithms exist [Newcombe 11, Sundaram 10], in the context of our particle filter framework the optical flow estimation stage should only consume a fraction of the available CPU power. The sparse optical flow estimation method used in this chapter was proposed in [Spruyt 13b] and discussed in detail in chapter 3, and only needs 6 milliseconds of processing time on QVGA video. Similar to most dense optical flow algorithms, this method could easily benefit from GPU parallelization to decrease processing times even further. Due to the nature of the sparse optical flow algorithm used, optical flow vectors are often found around the edges of an object. As a result, positive regions centered at those edge points tend to contain a lot of background pixels, slowing down the learning process. Therefore, we propose to slightly relocate each positive image patch such that it contains as much foreground pixels as possible. Assuming the foreground contains moving pixels while the background consists of static objects, this corresponds to shifting the region's centroid such that the amount of moving pixels within the region is maximized. Since training data is only gathered if the hands are moving, the first assumption is valid per definition. The second assumption does not necessarily hold. However, in case both the background and foreground contain moving pixels, the current position of the positive region already maximizes the amount of motion within the bounding box such that no new position will be found. Therefore, in the worst case the region is not moved at all, whereas the best case scenario moves the region closer towards the center of the supporting object.

For performance reasons we simply detect moving pixels by temporal frame differencing. Since the frame difference image T is also used by the particle filter framework during hand tracking [Spruyt 12], no additional computational cost is incurred. Each positive region is then pushed towards its closest local maximum



(a) Patches are selected automatically, based on the optical flow estimates.



(b) Patches are pushed towards the center of the arm.

Figure 5.3: Positive (context) and negative (background) patches are sampled automatically to train an online Random Forest classifier. Positive samples are shown in blue, whereas negative samples are shown in red.

within the frame difference image T by executing a mean-shift iteration:

$$\hat{\mathbf{x}} = \mathbf{x} + \Delta\mathbf{x} \quad : \quad \Delta\mathbf{x} = \frac{\sum_{\mathbf{p}\in W} K(\mathbf{p}-\mathbf{x})\, T(\mathbf{p})\, (\mathbf{p}-\mathbf{x})}{\sum_{\mathbf{p}\in W} K(\mathbf{p}-\mathbf{x})\, T(\mathbf{p})}, \tag{3.2}$$

where $K(.)$ defines a smoothing kernel and W represents the set of pixels within a window surrounding $\mathbf{x}$. We use a simple uniform kernel, the bandwidth of which is defined by the scale s of the region under consideration. This allows us to re-use the integral images that are also used by the hand tracking particle filter, to execute the mean-shift iteration in constant time.

Finally, several slightly shifted versions of each region are added to the set of positive image patches, in order to increase translational invariance of the final classifier. Figure 5.3 illustrates the PN-learning process graphically. The top-row shows the positive ($l(R_{s,\mathbf{x}}) = 1$) and negative ($l(R_{s,\mathbf{x}}) = 0$) patch candidates respectively by blue and red rectangles. The bottom-row shows the positive patch candidates after mean-shift iteration and the introduction of small spatial shifting.

Each region $R_{s,\mathbf{x}}$ is described using several simple feature descriptors. Although robust spatial descriptors such as Histograms of oriented Gradients (HoG) [Dalal 05], SIFT [Lowe 04] and SURF [Bay 06] or spatio-temporal descriptors as discussed in [Wang 09] could be used to further improve our detection rates, these

descriptors introduce a high computational complexity. Therefore, to allow real-time processing, we use several simple feature descriptors to model the color and texture of each image patch.

Color is represented simply by three 16-bin histograms. To reduce the computational complexity of our solution, we first calculate the integral histograms after which the histogram of each arbitrary region in the image can be obtained in constant time. Similar to the well known integral images, proposed by Viola and Jones [Viola 04], an integral histogram is a recursive propagation method to calculate a cumulative histogram for each pixel in the image, by means of dynamic programming. At each pixel location, this cumulative histogram is the histogram of all pixels above and to the left of the pixel under consideration. For each pixel location an integral histogram, represented by vectors of integers and with bin size 16, is obtained using a single pass propagation method during a preprocessing phase.

Although color can be a discriminative cue, it is inherently sensitive to illumination changes and can not represent local structure. Therefore, we model fine texture details by a 16-bin local binary pattern (LBP) [Ojala 96] histogram. The local binary pattern was originally defined by Ojala *et al.* [Ojala 96] as an ordered set of binary comparisons between a center pixel and each of its surrounding pixels, and was shown to have high discriminative power for texture classification problems. The LBPs are calculated for each pixel in the image, after which the histogram of LBPs within a region can be used to describe the texture of this region. However, since each pixel is surrounded by eight neighbors, an 8-bit LBP descriptor results in a 256-bin histogram, many of which are empty if small regions have to be described. Therefore, to avoid the well known 'curse of dimensionality', we propose to use the Center Symmetric Local Binary Pattern (CS-LBP) operator [Heikkilä 09], which has been proven to outperform both the original LBP and the well known SIFT descriptor in various image recognition problems. The CS-LBP is simply obtained by using the sign of the difference between center-symmetric pairs of pixels, instead of the difference between each pixel and a central pixel, resulting in a 4-bit descriptor and thus a 16-bin histogram. The CS-LBP can be implemented efficiently using only bit-shifting operations.

While the LBP descriptor models fine grained texture details, we use several Haar-like features [Viola 04] to model larger structural properties of the image patch. Haar-like features are defined as the normalized difference between distinct subregions of the image patch under consideration and can therefore model larger structural properties of the object. Haar-like features can be calculated efficiently and in constant time by first obtaining the so called integral image, similar to the concept of integral histograms as was explained above.

Finally, these Haar-like features are not only calculated using the grayscale version of the video frame, but also based on a skin probability map in which each pixel contains the probability of it representing human skin. We used the

Figure 5.4: The original image, the skin probability map and the CS-LBP image.

publicly available Compaq dataset [Jones 02] to train a Bayesian skin classifier. The Compaq dataset contains over 3 000 manually segmented skin colored images and over 4 000 non-skin colored images. During training of the skin classifier, we simply model the likelihood $P(r, g, b|skin)$ of observing a certain RGB triplet in a skin region by means of a 32-bin 3D histogram of skin pixels. Similarly, the likelihood $P(r, g, b|\neg skin)$ of observing this color in a non-skin region, is modeled by means of a 32-bin 3D histogram of non-skin pixels. Furthermore, the prior probabilities $P(skin)$ and $P(\neg skin) = 1 - P(skin)$, can be obtained directly from the training data. The posterior probability $P(skin|r, g, b)$ can then be obtained according to Bayes' rule:

$$\begin{aligned} P(skin|r,g,b) &= \frac{P(r,g,b|skin)P(skin)}{P(r,g,b)} \\ &= \frac{P(r,g,b|skin)P(skin)}{P(r,g,b|skin)P(skin) + P(r,g,b|\neg skin)P(\neg skin)} \end{aligned}$$

Figure 5.4 shows the skin probability map and the CS-LBP image together with the original video frame for illustrative purposes.

3.2 Robust Online Learning and Forgetting

Since the training data becomes available only incrementally as time progresses, an online learning scheme is needed. Furthermore, the chosen classifier needs to be extremely robust to noise in the training data, since it is expected that errors occur during the image patch selection process described in the previous paragraph. Finally, the classifier should be able to automatically select the most discriminative features from the set of available features, and should be able to forget outdated information that was learned in the past.

A recently proposed classification approach that is known to be robust against overfitting, exhibits a low variance and a low bias, and can cope with noisy labels and missing values in the training data, are random forests. A random forest is an ensemble of decision tree classifiers, each of which are trained using a bootstrapped sample of the original training data. Random forests were discussed in detail in section 3.1 of chapter 2.

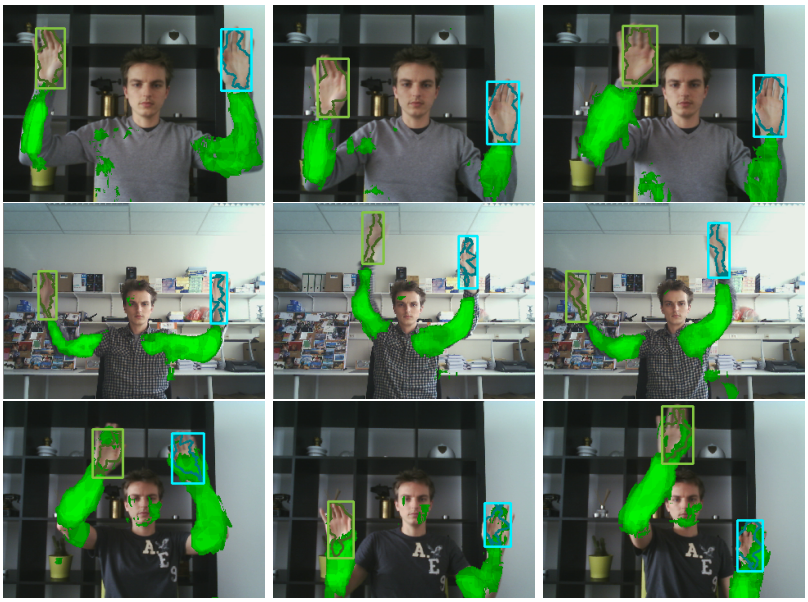


Figure 5.5: Pixels that are classified as being part of the hand's context are shown in green. The brightness indicates the posterior probability (classifiers certainty) that the label assignment was correct.

Saffari *et al.* [Saffari 09] recently reformulated the problem of training random forests to an incremental learning process using an online decision tree growing procedure. Furthermore, their method allows for a temporal weighting scheme to automatically discard old and irrelevant trees while adding new trees to the ensemble when needed. Saffari *et al.* further showed that the performance of the online random forest algorithm converges to that of an offline trained random forest.

In this chapter, we therefore use the online random forest algorithm proposed by Saffari *et al.* to dynamically learn a representation of the supporting regions of the tracked hand. In our experiments we use 10 decision trees that are grown to a maximum depth of 20. Furthermore, 200 random tests are executed during each feature selection iteration while growing the trees.

Figure 5.5 shows several classification examples. The center pixel of each region $R_{s,\mathbf{x}}$ is indicated in green if classified as being part of the hand's context. The brightness represents the posterior probability of being part of this context and thus indicates the classifier's certainty.

Furthermore, although we are mostly interested in learning the appearance of the arm, due to our related research on hand tracking, the proposed context learning scheme can also be used to learn the appearance of other attributes. This is illustrated by Figure 5.6, in which the person holds a magazine. The classifier learns

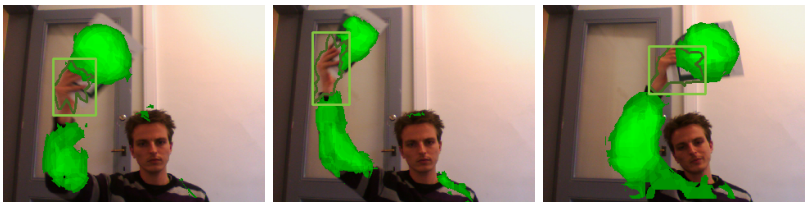


Figure 5.6: Context learning allows our hand tracker to automatically learn the appearance of any object that exhibits coherent motion with the hand itself. In this image, the person is holding a magazine.

that the magazine exhibits coherent motion with the hand and therefore learns its appearance in real-time.

3.3 A Context Based Proposal Distribution for Particle Filtering

The random forest classifier yields a probabilistic estimate about the class label of each pixel in the image. In Figure 5.6, these probabilities for the positive class labels are represented by the intensity of the green overlay. This set of probabilities therefore represents a spatial distribution. Throughout the video frame, the modes of this distribution will shift due to the motion of the arm or the supporting objects. Therefore, instead of tracking the hand directly, we can track the changes in the context classification distribution around the hand, and use these changes to predict the location of the hand in the next video frame. Although robust histogram comparison methods, such as the Earth Mover's Distance, are available, many of these have a high computational complexity. Instead, we propose to simply use the normalized cross-correlation as a measure of correspondence between the spatial context-histograms of subsequent frames, since this can be implemented efficiently using integral images [Lewis 95]. On a side node, normalized cross-correlation is often used for template matching. Therefore, our approach of finding the hand in the next frame could also be interpreted as a template matching solution, where the template is defined by the context classification result of the previous frame. This is illustrated more clearly by Figure 5.7, in which the position of the left hand is estimated.

Normalized cross-correlation is defined as

$$r(u,v) = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})(t(x-u,y-v) - \bar{t})}{\sqrt{\left(\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2 \sum_{x,y}(t(x-u,y-v) - \bar{t})^2\right)}}, \qquad (3.3)$$

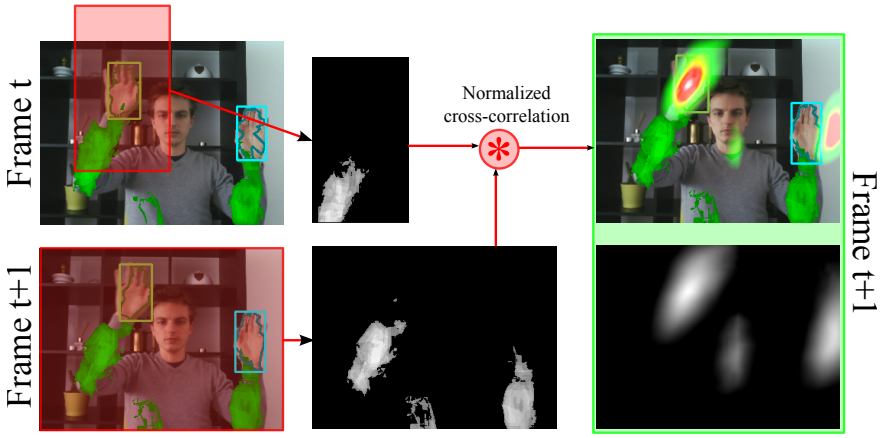


Figure 5.7: In this illustration, the location of the left hand is estimated in frame t+1, based on its context in frame t. Cross-correlation based template matching is used to match a region of interest in the context classification distribution of frame t, with the context classification distribution of frame t+1. The result is a prediction of the hand location, purely based on static, contextual information.

where $f(x, y)$ represents the image under consideration, $t(x, y)$ is the template, $\bar{f}_{u,v}$ represents the mean of $f(x, y)$ in the region of the template centered at (u, v), and $\bar{t}$ is the mean of the template image $t(x, y)$.

To obtain an estimate of the hand location we simply want to maximize this correlation. Therefore, the estimate $\hat{\mathbf{y}} = (\hat{x}, \hat{y})$ is obtained as

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} r(\mathbf{y}). \tag{3.4}$$

This state estimate is complementary to the skin-color and motion based state estimate as obtained by the hand tracker used [Spruyt 13a]. The question now remains how to incorporate this estimate into the particle filter framework to improve tracking performance. The skin-color and motion based likelihood, used by the hand tracker's observation model [Spruyt 12], represents $p(\mathbf{y}_t|\mathbf{x}_t^i)$, where $\mathbf{y}$ is the latest observation and $\mathbf{x}$ is the state to be estimated, i.e. the hand location hypothesis of particle i. In earlier work, we showed that this likelihood function is extremely accurate if a good estimate $\mathbf{x}_t^i$ is available. Therefore, instead of incorporating the contextual information in the observation model of the particle filter, we propose to use it to improve the estimate $\mathbf{x}_t^i$ of each particle, which is then simply assigned a skin-color and motion based likelihood.

One way of improving the estimate $\mathbf{x}_t^i$ would be to adapt the motion model of the particle filter. Several approaches can be found in literature that incorporate the latest measurements directly into the motion model of a particle filter [Belgacem 12, Yao 10]. However, from a Bayesian perspective, the motion model of the

particle filter, yielding the state transition prior $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, should only depend on the previous state estimate and not on the current observations. Therefore, directly incorporating the latest measurements into the particle filter's motion model, is mathematically incorrect and would force us to abandon the Bayesian framework which forms the theoretical foundation of particle filters.

In section 3.4 of chapter 4 we proposed a method to directly incorporate the latest observations into the proposal distribution of a particle filter by means of Kalman filtering. In this section, we reuse this method to incorporate contextual information into the proposal distribution.

Although Kalman filters assume Gaussian likelihoods and linear models, it is important to note that our particle filter solution, with Gaussian proposal distributions, does not require the posterior to be normally distributed and does not require linear observation models. The reason for this is that a distinct Kalman filter is used per individual particle, from which the new particle state will be sampled during the importance sampling process. Hence only the distribution of each single particle is assumed to be Gaussian. During importance sampling, the particle is drawn from the normal distribution that serves as the proposal distribution, as shown by equation (3.5).

$$q(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i, \mathbf{y}_{1:t}) \sim N(\bar{\mathbf{x}}_t^i, P_t^i) \qquad i = 1, ..., N, \tag{3.5}$$

where $\bar{\mathbf{x}}_t^i$ represents the Kalman filter's state estimate and P_t^i represents its covariance matrix.

Thus, at time t, the latest observation $\hat{\mathbf{y}}$, defined by equation (3.4), is used to obtain the mean, $\bar{\mathbf{x}}_t^i$, and covariance matrix P_t^i of the proposal distribution of each particle i, using the Kalman filter measurement and update equations. Next, the i^{th} particle is sampled from this distribution, and is assigned a weight by the observation model, as in a traditional particle filter. This method requires us to propagate the covariance P_t^i for each particle since multiple particles might share the same covariance matrix after resampling.

In the following, let $\mathbf{x}(t|t-1)$ be the state estimate at time instance t, given observations up to and including time $t-1$. We need to define the motion model matrix A that is used to predict the next state based on the previous state estimate as $\mathbf{x}(t|t-1) = A\,\mathbf{x}(t-1|t-1) + \mathbf{m}$, where $\mathbf{m}$ is the noise component. In our case, A corresponds to the 4×4 matrix that represent the following constant velocity transformation:

$$\mathbf{x}_t = 2\mathbf{x}_{t-1} - \mathbf{x}_{t-2} \tag{3.6}$$

$$= \mathbf{x}_{t-1} + (\mathbf{x}_{t-1} - \mathbf{x}_{t-2}) \tag{3.7}$$

$$= \mathbf{x}_{t-1} + \Delta V, \tag{3.8}$$

where ΔV models the velocity of the object in the previous video frame. For state vector $\mathbf{x}_t = (x_t, y_t, x_{t-1}, y_{t-1})$, the motion model matrix A then becomes:

$$A = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \mathbf{x}_{t-1} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ x_{t-2} \\ y_{t-2} \end{bmatrix}. \tag{3.9}$$

Similarly, the observation matrix H needs to be defined, such that $\hat{\mathbf{y}} = H\,\mathbf{x}(t|t-1) + \mathbf{n}$, where $\mathbf{n}$ is a white noise component and where $\hat{\mathbf{y}}$ is the context classifier based observation defined by equation (3.4). Since both the state estimate $\mathbf{x}$ and the observation $\hat{\mathbf{y}}$ consists of a coordinate pair (x, y) representing the hand location, the observation matrix H simply reduces to:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.10}$$

In the following, let Q be the covariance matrix that defines the white Gaussian process noise component $\mathbf{m}$, and let R be the covariance matrix that defined the white Gaussian measurement noise component $\mathbf{n}$. For each particle i in the set of N particles, the Kalman filter estimates the mean $\mu^i = \mathbf{x}(t|t)$ and covariance matrix $P^i = P(t|t)$ of the multivariate Gaussian proposal distribution $q(.) \sim N(\mu^i, P^i)$. The state prediction at time t, given observations up to and including time $t-1$ is:

$$\mathbf{x}(t|t-1) = A\,\mathbf{x}(t-1|t-1), \tag{3.11}$$

and the corresponding covariance is

$$P(t|t-1) = AP(t-1|t-1)A^\intercal + Q. \tag{3.12}$$

The new estimate, after incorporating the context classification result $\hat{\mathbf{y}}$, is then:

$$\mathbf{x}(t|t) = \mathbf{x}(t|t-1) + K(t)[\hat{\mathbf{y}}(t) - H\mathbf{x}(t|t-1)], \tag{3.13}$$

with covariance

$$P(t|t) = P(t|t-1) - K(t)S(t)K(t)^\intercal, \tag{3.14}$$

where the Kalman gain is

$$K = P(t|t-1)H^\intercal S^{-1}(t), \tag{3.15}$$

and the innovation covariance is

$$S(t) = HP(t|t-1)H^\intercal + R. \tag{3.16}$$

If N particles are used to represent the posterior distribution, then also N Kalman filters are used to generate particle-specific proposal distributions. The particle filter then draws N new particles, each from their corresponding proposal distribution. In the following, let $\mathbf{x}_t^i$ be the state of the particle that is sampled from this proposal distribution of particle i at time instance t. Let $\dot{\mathbf{x}}_t^i$ be the state estimate of the same particle, obtained by the motion model of the particle filter. The state transition prior for this particle i is then obtained by evaluating the likelihood of the sample $\mathbf{x}_t^i$ by the particle filter's motion model. If the motion model's likelihood function is normally distributed, centered at the state that was predicted by the constant velocity model, then the state transition prior is:

$$p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i) \sim e^{-\frac{1}{2}(\mathbf{x}_t^i-\dot{\mathbf{x}}_t^i)^\intercal Q^{-1}(\mathbf{x}_t^i-\dot{\mathbf{x}}_t^i)}. \tag{3.17}$$

The above equation shows that, although we do not assume a linear observation model for the particle filter, and we do not assume a Gaussian posterior distribution, we do assume a Gaussian state transition prior and a linear motion model. However, most particle filter implementations use simple linear motion models like the constant velocity or the random walk model, such that this constraint does not pose a practical problem.

The proposal density likelihood for sample i is defined by the Gaussian posterior that is estimated by the Kalman filter:

$$q(\mathbf{x}_t^i|\mathbf{x}_{1:t-1}^i, \hat{\mathbf{y}}_{1:t}) \sim \frac{1}{\sqrt{|P_t^i|}} e^{-\frac{1}{2}(\mathbf{x}_t^i-\mu_t^i)^\intercal (P_t^i)^{-1}(\mathbf{x}_t^i-\mu_t^i)}. \tag{3.18}$$

Finally, the observation likelihood $p(\mathbf{y}_t|\mathbf{x}_t^i)$ is obtained using the skin-color and motion based measurement model of our particle filter as proposed in [Spruyt 13a] and discussed in chapter 4. Substituting equations (3.17), (3.18) and the observation likelihood into equation (3.11) allows us to calculate the particle weights w_t^i as follows:

$$w_t^i = w_{t-1}^i \frac{p(\mathbf{y}_t|\mathbf{x}_t^i)p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)}{q(\mathbf{x}_t^i|\mathbf{x}_{1:t-1}^i, \hat{\mathbf{y}}_{1:t})}. \tag{3.19}$$

In contrast to the widely used bootstrap filter, our approach is able to incorporate the latest observations into the proposal distribution to greatly improve sampling efficiency. As a result, relatively few particles (50 in our experiments) are needed for robust tracking. Furthermore, since the proposal density is based on the context classification result, particles are samples around the maxima of the cross-correlation map shown by Figure 5.7 and are therefore less sensitive to distractions caused by hand-like objects in the neighborhood.

3.4 Context Based Arm Tracking Using Partitioned Sampling

In the previous section we proposed a method to incorporate contextual information into a particle filter tracking framework. Contextual information in this setting could be the arm if tracking a hand, but might as well be a shadow when tracking vehicles [Mosabbeb 07] or a limb when tracking articulated objects in general.

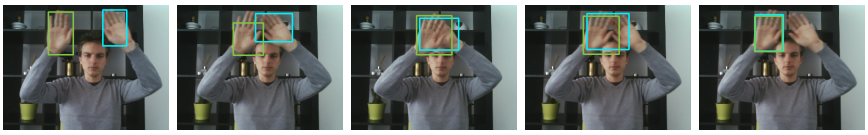
In this section we introduce a second method to improve tracking performance. This solution however is specific to hand tracking, and improves robustness against occlusion by explicitly tracking the arm. In the previous section, the state vector was defined as $\mathbf{x} = (x, y)$ and represented the coordinates of the hand to be estimated. We now extend the state vector with a parameter that describes the angle θ of the arm. The state vector then becomes $\mathbf{x} = (x, y, \theta)$.

Directly estimating the joint probability $p(x, y, \theta|\mathbf{y}_{1:t})$ would require exponentially more particles to be used compared to estimating the previous posterior $p(x, y|\mathbf{y}_{1:t})$. Intuitively this can be explained as follows: If 10 samples would be needed to sample a 1-dimensional unit interval with sufficient sample density, then $10^2 = 100$ samples would be needed to model a two-dimensional problem with the same density, and $10^3 = 1000$ samples would be needed in the 3D case. In the above problem, 50 particles are used such that $(\sqrt{50})^3 = 354$ particles would be needed to maintain the same sample density in the current 3D state space. However, this would greatly increase the computational complexity of the particle filter, preventing real-time operation.

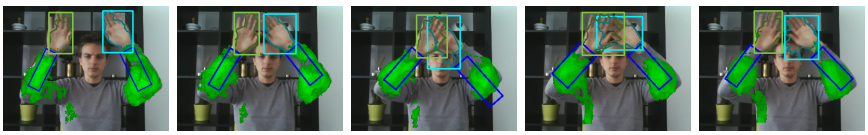
To avoid the curse of dimensionality, we partitioned our search space based on the assumption that the arm angle is independent of the (x, y) location of the hand. Particle filtering in a partitioned state space is called partitioned sampling and was proposed by MacCormick *et al.* [MacCormick 00]. They showed that different state partitions can be solved for sequentially instead of simultaneously if a hierarchical relation exists between the subspaces. We use this concept to transform our 3D search space to a sequence of a 2D and a 1D estimation problem.

Partitioned sampling was discussed in chapter 4, where it was used to efficiently estimate the hand's position and bounding box size. In the current setting, we first estimate the hand coordinates, and use this estimate to obtain an estimate of the arm angle if contextual information is available. An additional advantage of this approach is that the second partition, i.e. estimating θ, can easily be omitted if no contextual information is available at a time instance t.

A valid question is now how estimating θ would improve the estimate of the hand location (x, y), since the hand location is estimated before and independently from the arm angle. However, it is important to note that particle filters employ a resampling strategy to avoid the particle depletion problem. When the variance of the particle weights becomes large, the sample set is replaced by a new sample set by resampling with replacement. This means that particles with a bad θ estimate and corresponding low arm-likelihood, get replaced by duplicates of



(a) Hand tracking as proposed in [Spruyt 13a].



(b) Arm and hand tracking using partitioned sampling.

Figure 5.8: Top row: Hand tracking fails due to occlusion. Bottom row: Arm tracking helps to disambiguate in case of clutter or occlusion.

particles with a good θ estimate with corresponding high arm-likelihood. In the next iteration, the (x, y) location of the hand is then estimated for this new set of particles. Therefore, arm tracking makes sure that particles with a high skin-color and motion likelihood are deleted from the sample set if they correspond to a low arm-likelihood. Furthermore, resampling only occurs if the variance of the particle weights becomes large and thus when the covariance of the particles increases. Therefore, the correcting effect of the arm tracking partition is negligible during stable hand tracking (almost no resampling occurs), whereas it becomes more apparent in case of clutter, and occlusion (resampling occurs often). This is illustrated in Figure 5.8, where the top row shows the tracking result if only the first partition is estimated, whereas the bottom row shows the result if also the arm angle is tracked.

Whereas a custom proposal distribution was used to estimate the parameters of the first state partition, i.e. the hand location, a simple bootstrap filter was used for the second partition, i.e. the arm angle. The state transition prior is therefore used as a proposal distribution, and the motion model for this partition is a random walk model.

Our choice of observation model used to obtain the likelihood for each estimate θ_i of the arm location is again driven by the real-time constraints of our tracking solution and is therefore computationally inexpensive. For each particle, the arm is represented by a straight line segment, the length of which is proportional to $\max(w, h)$, where (w, h) defines the width and height of the hand's bounding box. Each point on this line corresponds to a context probability $p(l = 1|x, y)$, indicated in green in Figure 5.8. We obtain the probability that a line represents the arm by marginalizing this conditional density such that $p(l = 1; \theta) = \sum_{x,y} p(l = 1|x, y; \theta)$, where θ is the arm angle that defines which points (x, y) are considered.

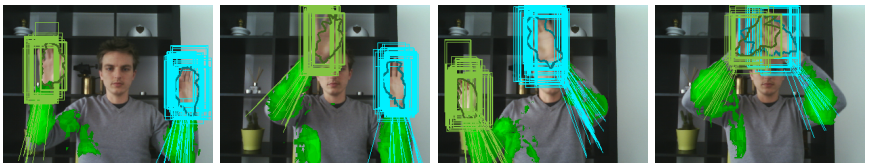


Figure 5.9: Visual representation of the particle filter's posterior distribution. The $(x, y, width, height)$ parameters of the particles are represented by colored rectangles, whereas the arm angle θ is represented by line segments.

The actual arm angle then corresponds to

$$\hat{\theta} = \arg\max_{\theta} p(l = 1; \theta).$$

Iterating over all pixel coordinates (x, y) defined by the line segment with angle θ can be implemented extremely efficiently using the well known Bresenham line algorithm. Figure 5.9 shows all particles of each particle filter visually for illustrative purposes.

4 Results and Discussion

In this section, we compare our hand tracking results with several state-of-the-art tracking algorithms using three publicly available datasets. The first dataset was created to specifically test tracking robustness against occlusion and cluttered backgrounds. This dataset contains eight video sequences with an average duration of 45 seconds, and is made publicly available online[1] for research purposes to allow for fair comparison with future methods. The sequences contain cluttered backgrounds, fast motion, both long and short sleeved arms, and several cases of occlusion. Tables 5.1 and 5.2 list the characteristics of each video in this dataset.

To evaluate the increase in robustness by incorporating contextual information into the original hand tracking method described in [Spruyt 12], we simply count the number of tracking errors in each video sequence. A tracking result is considered erroneous if the Pascal VOC score [Everingham 10] is below 0.5. The VOC score defines the amount of overlap between the detected bounding box, and the given ground truth bounding box and is calculated as $VOC = \frac{B_g \cap B_d}{B_g \cup B_d}$, where B_g represents the pixels within the ground truth bounding box, whereas B_d represents the pixels within the detected bounding box.

Table 5.3 compares the number of tracking errors for the original algorithm of [Spruyt 12] with the number of tracking errors obtained using the approach proposed in this chapter based on the new proposal distribution and our arm tracking solution.

[1] `http://telin.ugent.be/~vspruyt/sensors/`

Table 5.1: Test dataset specification, part 1.

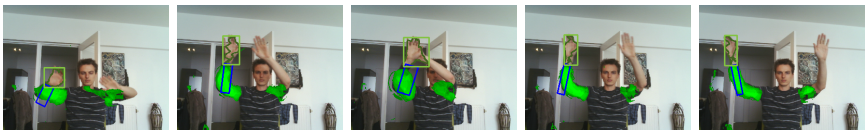
	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Seq. 5
Scene clutter	no	yes	yes	yes	yes
Fast motion	no	yes	no	no	yes
Short sleeves	no	no	no	no	yes
Hand occlusion	yes	yes	yes	yes	yes
Face occlusion	yes	yes	yes	yes	yes
Arm hidden	no	no	no	no	no
Hand hidden	no	no	no	no	no
Textured clothing	yes	yes	yes	no	no
Background motion	no	no	no	no	no
Length	34s	46s	67s	35s	35s

Table 5.2: Test dataset specification, part 2.

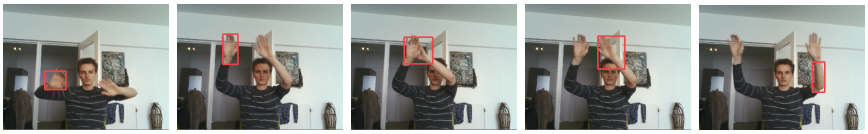
	Seq. 6	Seq. 7	Seq. 8	Seq. 9
Scene clutter	yes	yes	no	yes
Fast motion	yes	no	no	no
Short sleeves	yes	no	no	no
Hand occlusion	yes	yes	yes	yes
Face occlusion	yes	yes	yes	no
Arm hidden	no	no	yes	no
Hand hidden	no	no	yes	no
Textured clothing	no	yes	no	no
Background motion	no	no	no	yes
Length	56s	47s	45s	45s

Table 5.3: Evaluation results on the novel dataset. For each algorithm, the number of tracking errors is reported. Column *Proposal* corresponds to the results where the custom proposal distribution was used. Column *Arm tracking* shows the results where only the partitioned sampling based arm tracking solution was used. Column *Combined* lists the results where both approaches are combined. Both the custom proposal distribution and the partitioned sampling based arm tracking reduce the number of tracking errors on average.

Sequence #	[Spruyt 12]	Proposal	Arm tracking	Combined
Video 1	2	0	0	**0**
Video 2	1	0	1	**0**
Video 3	1	0	0	**0**
Video 4	4	1	2	**0**
Video 5	3	1	0	**1**
Video 6	2	1	0	**0**
Video 7	6	3	1	**1**
Video 8	4	2	0	**0**
Video 9	6	1	1	**0**



(a) Our result (proposal + arm tracking).



(b) Result of [Spruyt 12].

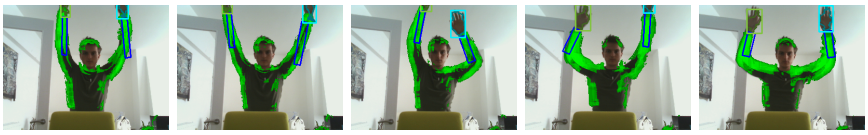
Figure 5.10: Several video frames from sequence 7, in which only the left hand is tracked such that the right hand can serve as a distraction to test robustness against occlusion with unknown hand-like objects.

These results illustrate that both methods proposed in this chapter to incorporate contextual information into the tracking framework contribute to an increased tracking robustness. Both methods are complementary as can be seen by the decrease in tracking errors when combined, e.g. in video 2, video 4, video 7 and video 8.

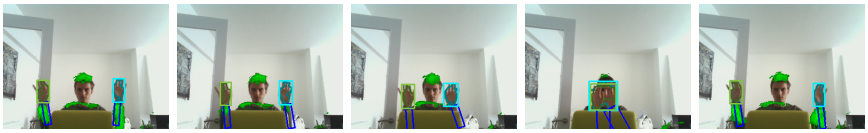
The first six video sequences are meant to test the tracker's robustness against occlusion. Results for these sequences are shown by Figure 5.18.

Video 7 serves as a special case. In this sequence we deliberately tracked only one hand, such that the other hand and its corresponding arm can serve as a distraction as if it was any moving, hand-like object. This allows us to evaluate the behavior of the tracking algorithm in case of occlusion with unknown, hand-like objects. Figure 5.10 compares the tracking results of this sequence by the algorithm described in [Spruyt 12] with the results obtained using the proposed method. These results clearly show the benefits of incorporating contextual information into the tracking process. Whereas the original hand tracking algorithm is unable to distinguish between the tracked left hand and the unknown right hand, context learning allows the particle filter to use spatial dependencies to resolve ambiguous situations.

Video 8 contains several cases of severe occlusion of both the object of interest, i.e. the hand moves out of picture, and its context, i.e. the arm moves behind a chair, as shown in Figure 5.11. When the hand is completely occluded, the context based proposal distribution allows the particle filter to overcome the local minimum of the observation likelihood, thereby increasing tracking robustness. When the context itself is completely occluded, it does not contribute to the proposal distribution, such that the particle filter falls back to its normal bootstrap filter behavior. However, the partitioned sampling based arm tracking stage still predicts



(a) Tracking hands through occlusion.



(b) Tracking context (i.e. arms) through occlusion.

Figure 5.11: Several video frames from sequence 8, in which both the hands (a) and the arms (b) are completely occluded. Incorporating context in the proposal distribution resolves ambiguity when the object of interest is occluded. On the other hand, if the context itself is occluded, the proposal distribution becomes non-informative and does not hurt tracking performance. The predicted arm location in this case still increases robustness of the hand tracker.

Figure 5.12: Several video frames from sequence 9, in which the hand only moves perpendicular to the camera plane while background motion serves to test the tracker's robustness.

the most likely position of the arm, which in turn helps the hand tracking stage to overcome ambiguity due to occlusion between the left and right hands.

Sequence 9 contains a lot of background motion. The video sequence was captured while projecting a movie clip on the background wall. The hand only moves perpendicular to the camera plane, i.e. towards and away from the camera. Therefore, this sequence challenges the capabilities of the context learning method which depends on optical flow similarity between the tracked hand and its context. Figure 5.12 shows several frames from sequence 9. Although the context segmentation is less accurate than in the other videos, due to the challenging background motion, the results illustrate the robustness of our method in unconstrained environments.

Figure 5.13 shows the percentage of frames that is tracked correctly in each video sequence. Whereas Table 5.3 listed the number of times a tracking error begins, Figure 5.13 also incorporates the duration of each tracking failure by reporting the fraction of video frames that was tracked correctly.

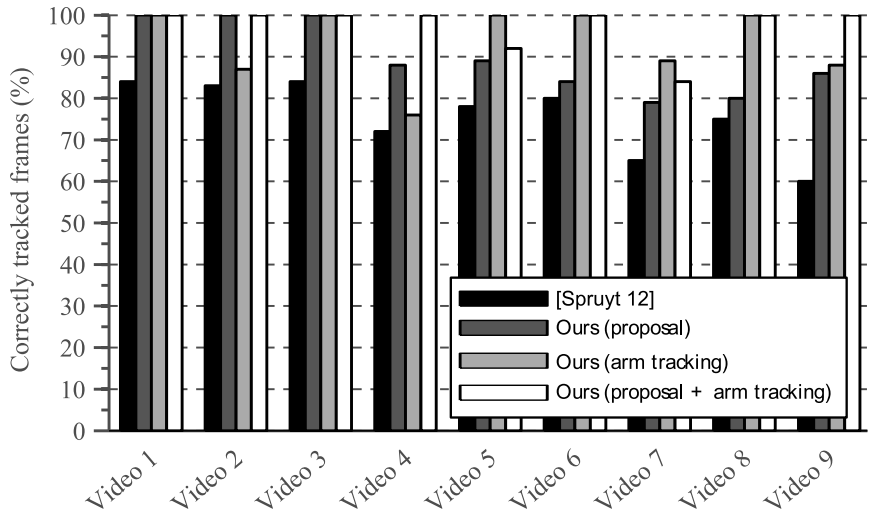


Figure 5.13: Percentage of correctly tracked frames in each video sequence. When tracking failure occurs (Table 5.3), all frames are considered incorrectly tracked until the VOC score exceeds 0.5 again.

These results consider tracking to be failed when the resulting bounding box corresponds to a VOC score lower than 0.5. However, many practical applications would benefit from a lower threshold, since even a VOC score of 0.2 corresponds to a reasonable tracking results. Therefore, Figure 5.14 shows the percentage of correctly tracked video frames, plotted against the corresponding VOC score. A VOC threshold of 0.5 is often used in object detection research to indicate correct detections, whereas a VOC threshold of 0.25 is common in face detection literature.

Although the focus of this chapter is context based hand tracking, we showed that the context can be used to obtain an estimate of the arm location. To evaluate the arm tracking capabilities, we compared our method with the pictorial structure based upper body pose detector described by Eichner *et al.* [Eichner 12]. This method estimates the spatial structure of the upper body. Although no temporal information is used by their method, the algorithm requires a bounding box of the upper body as input and can therefore be compared fairly with our tracking algorithm.

In [Eichner 12], the algorithm is evaluated by reporting the Percentage of correctly estimated body Parts (PCP). An estimated body part, i.e. the arm, is considered correct if its segment endpoints lie within a fraction f of the length of the ground-truth segment. In the following, we adopt this evaluation measure and

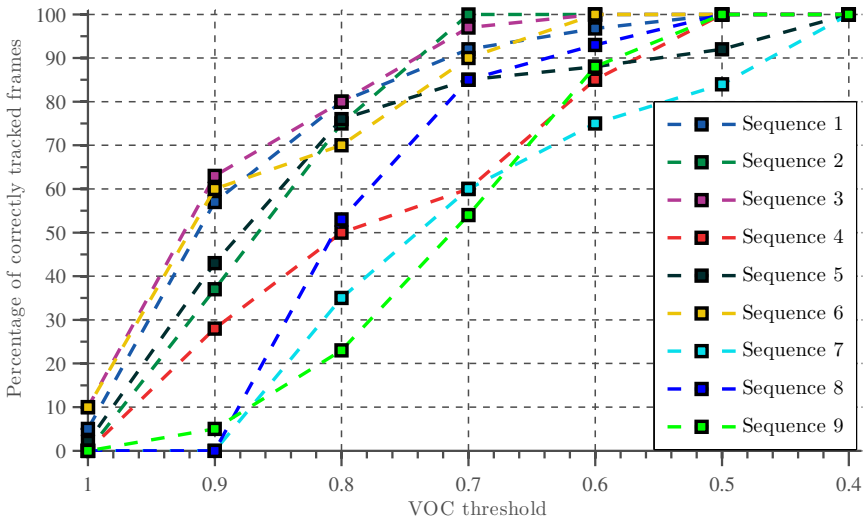


Figure 5.14: Percentage of correctly tracked frames in each video sequence, plotted against the VOC score. Perfect tracking is obtained if results with a VOC score of 0.4 and higher are considered to be correctly tracked instances. A VOC threshold of 0.5 is often used in object detection research to indicate correct detections.

compare their method with ours by reporting the PCP while varying the threshold $f \in (0, 1.5)$. The results are illustrated by Figure 5.15.

Although our method appears slightly less accurate when the PCP threshold is low, its accuracy increases rapidly and outperforms [Eichner 12] for higher PCP thresholds. Our method is able to provide a rough estimate of the arm pose, even in cluttered scenes with complicated interactions, whereas the algorithm proposed by [Eichner 12] provides an accurate estimate in simple sequences but quickly breaks down in cluttered scenes. Furthermore, [Eichner 12] requires approximately 1.5 seconds of processing time per video frame, whereas our solution processes 17 frames per second. Figure 5.16 illustrates some of the results obtained by both methods.

Figure 5.17 shows the execution times of each the components of the complete tracking and detection system, including context learning. This figure summarizes and combines the previous chapters and is based on figures 1.1, 2.26 and 4.23.

Finally, Figure 5.18 shows several results on the proposed dataset for illustrative purposes.

Whereas evaluation results on this challenging dataset clearly indicate the advantages of our solution, a second question of interest is whether or not context learning would degrade tracking performance in the cases where near-perfect tracking can already be accomplished without contextual information.

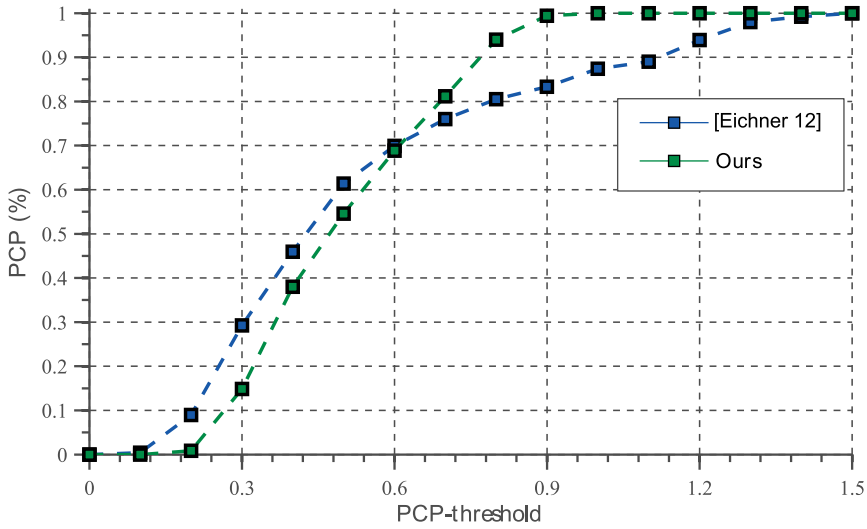
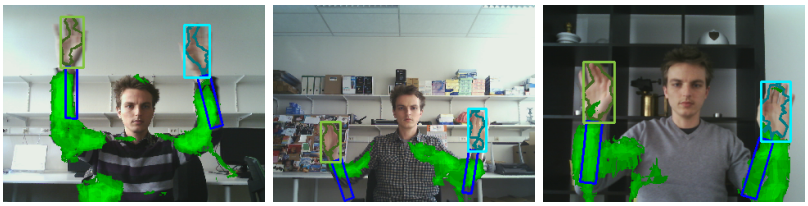
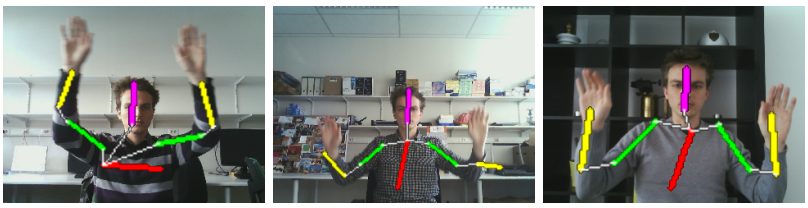


Figure 5.15: Comparison of the arm tracking results of our method with the results obtained with the pictorial structure based algorithm proposed by [Eichner 12]. The PCP score represents the percentage of correctly tracked arm instances, whereas the PCP-threshold defines when a detected instance is considered correct.



(a) Arm tracking by context.



(b) Pictorial structure based pose estimation [Eichner 12].

Figure 5.16: Some example frames comparing our arm tracking method with the pictorial structure based algorithm proposed by [Eichner 12].

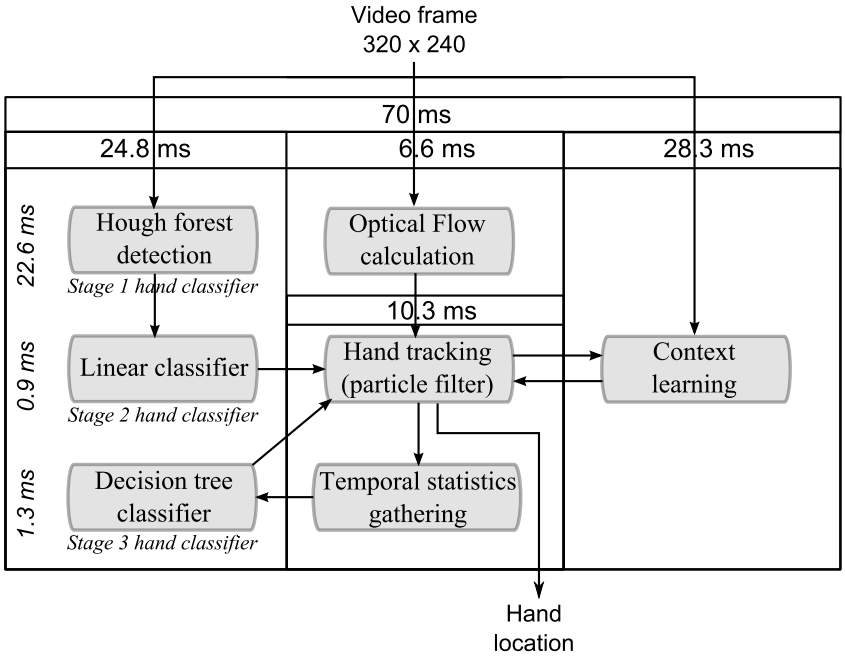


Figure 5.17: Execution timings of each component of the complete tracking and detection system, obtained by averaging the timing results of 5000 video frames of size 320×240.

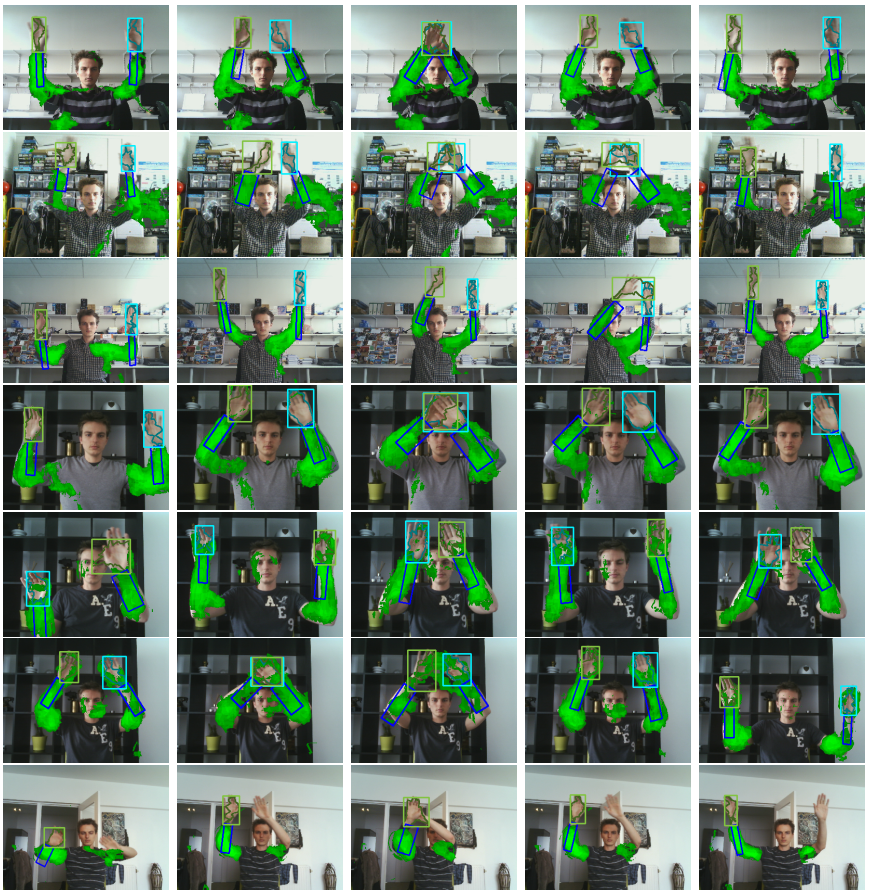


Figure 5.18: Several hand tracking results for the newly proposed publicly available dataset. Each row corresponds to a different video sequence in this dataset.

Table 5.4: Test dataset ([Spruyt 12]) specification.

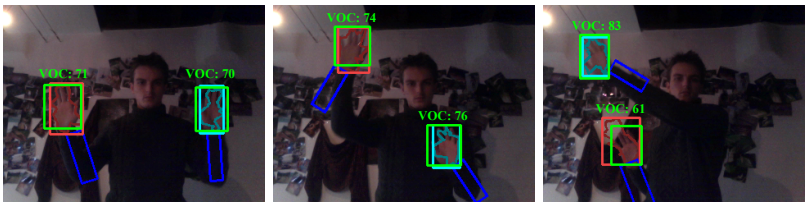
#	Scene	Lighting	Motion	Sleeves	Camera	Occlusion	Length
1	cluttered	dark	normal	long	fixed	no	83 s
2	cluttered	bright	normal	long	fixed	no	82 s
3	normal	normal	fast	long	fixed	no	85 s
4	normal	bright	normal	long	moving	no	88 s
5	normal	normal	normal	long	fixed	no	89 s
6	cluttered	normal	normal	short	fixed	no	84 s
7	normal	changing	normal	long	fixed	no	85 s
8	normal	normal	normal	short	fixed	yes	83 s

To evaluate this question, we performed additional tests using the publicly available dataset from [Spruyt 12]. This dataset contains eight video sequences each spanning approximately 90 seconds. These sequences contain changing illumination, fast motion, camera motion, and both long and short sleeved arms. The dataset was proposed in [Spruyt 12] and is publicly available for academic research, together with manually annotated groundtruth bounding boxes for both hands. Table 5.4 summarizes the characteristic of each video sequence in this dataset.

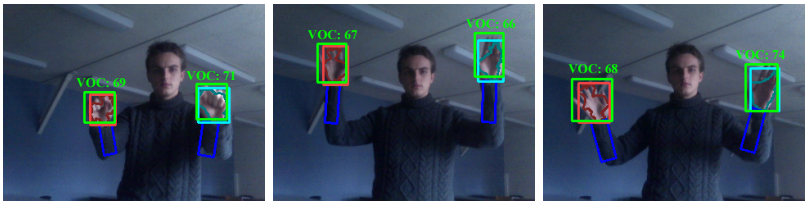
As listed by Table 5.4, this dataset does not contain occlusions. Even for sequence 8, the occlusions that occur are partial occlusions behind semi-transparent objects. The hands never completely overlap in these sequences, whereas the major novelty introduced in this chapter specifically increases robustness in case of occlusion. Furthermore, the results obtained by the original hand tracker upon which the work presented here is based, are already impressive; yielding an average VOC score of 76%. This means that the tracker of [Spruyt 12] never looses track of the target object and is able to accurately model its appearance at all times.

Figure 5.19 shows the VOC scores obtained on these sequences using the method proposed in this chapter. These results clearly show that a VOC score around 75% represents almost perfect tracking in the sense that higher VOC scores would probably not be very meaningful because even the groundtruth annotations might not be that perfect. Therefore it is obvious that no tracker could largely outperform the near perfect results reported in [Spruyt 12] on this dataset. Nevertheless, it would be interesting to see if the addition of contextual information would degrade the results in this case, especially since this dataset contains abruptly changing illumination, moving camera's and very fast motion.

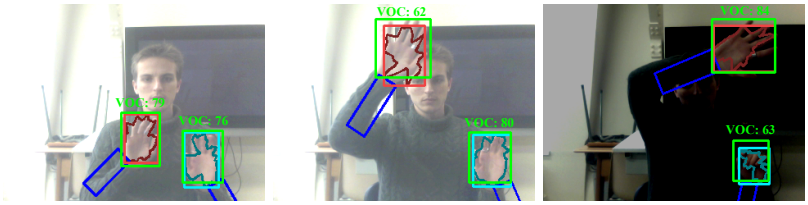
Figure 5.20 shows the VOC scores for several state-of-the-art algorithms, applied to this dataset. Each method was initialized manually based on the ground truth bounding boxes in the first video frame to allow for a fair comparison. The first method is the Camshift algorithm [Exner 10] which is often used as a baseline when evaluating tracking methods in computer vision. The second method is



(a) Sequence 1.



(b) Sequence 5.



(b) Sequence 7.

Figure 5.19: Several hand tracking results for the publicly available dataset from [Spruyt 12] with corresponding VOC score (expressed as a percentage). The groundtruth is indicated by a green rectangle. VOC scores higher than 0.5 are usually considered correct detections in object detection literature.

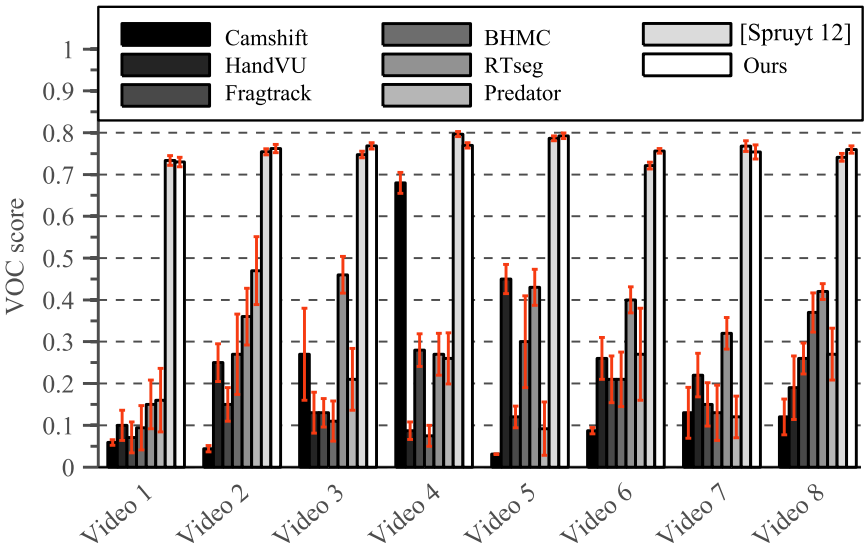


Figure 5.20: VOC score comparison for several state-of-the-art tracking algorithms on the publicly available hand dataset from [Spruyt 12].

the HandVU system [Kölsch 10], a well known hand-tracking method based on a combination of color cues and Harris corner features that are spatially constrained using flocking behavior rules. Third, we compare our method with the RTseg solution that we proposed earlier in [Spruyt 10a]. The RTseg method combines multiple color spaces using a Gaussian Mixture Model, in an attempt to decrease illumination dependence. The fourth algorithm is the Predator tracker [Kalal 12] which was discussed in section 2 and is a patch-based tracking solution that employs PN-learning to automatically learn the object's appearance over time. Whereas the predator tracker is designed to track rigid objects and is therefore expected to fail when the hand undergoes large deformations, the adaptive basin hopping Monte Carlo filter (BHMC) proposed in [Junseok 09] is an adaptive patch based tracker designed for tracking non-rigid objects. Finally, the Fragtrack algorithm [Amit 06] implements the generalized Hough transform to perform patch based tracking and uses integral histograms to efficiently represent the object's appearance. The method labelled [Spruyt 12], is the hand tracking algorithm we proposed in [Spruyt 12] and forms the base of the solution described in this chapter. This method use a combination of color and motion cues, optical flow and a discriminative Hough voting scheme.

These results show that our solution maintains the near perfect tracking results on this dataset, with the added value that the location and a segmentation of the arm is obtained automatically.

Table 5.5: Results on the Signer dataset, reported as the percentage of video frames for which the VOC score exceeds a predefined threshold.

VOC threshold	Buehler [Buehler 11]	Spruyt [Spruyt 12]	Our method
$VOC \geq 0.2$	**100.0%**	96.0%	**100.0%**
$VOC \geq 0.5$	94.3%	91.9%	**96.8%**
$VOC \geq 0.6$	**83.4%**	75.8%	82.6%

Finally, similar to the HandVU and Predator algorithms, our non-optimized C++ code processes QVGA data in 70 milliseconds, resulting in a framerate of 14 fps on a quadcore Intel I7 architecture with 8 gigabytes of RAM. The original hand tracking algorithm proposed in [Spruyt 12] runs at 18 fps such that the overhead, introduced by the contextual learning, is negligible compared to the benefit of increased robustness. As a comparison, both the BHMC and the Fragtrack algorithms need several seconds per video frame and are unable to robustly track hands.

Finally, it would be interesting to compare our hand tracking results with tracking algorithms that employ pictorial structure to track articulated objects. To this end, we compared our method with two well know upper body tracking solutions [Buehler 11, Karlinsky 10], applied to the publicly available and widely used Signers dataset [Buehler 08]. The Signers dataset contains five real and challenging BBC news sequences with signers performing a sign language translation.

Pictorial structure based approaches employ prior knowledge about the articulated nature of an object of interest to infer its joint configuration. Buehler *et al.* [Buehler 11] proposed an efficient method to sample from a pictorial picture proposal distribution to perform upper body tracking. Their method searches for key frames in which accurate detection can be achieved, and performs temporal tracking between these key frames based on HOG features and color cues. Their solution is able to accurately detect and track upper and lower arms, and uses this information to infer the hand location.

Buehler *et al.* [Buehler 11] report the percentage of video frames in which hands were correctly tracked. Hands are considered to be tracked correctly, if the VOC score is above a certain threshold. Results are reported for VOC scores of $VOC \geq 0.2$, $VOC \geq 0.5$ and $VOC \geq 0.6$. Table 5.5 compares the results obtained by Buehler *et al.* with the results obtained by [Spruyt 12] and with the results obtained by the method proposed in this chapter.

These results show that the incorporation of context clearly improves tracking performance for this real-life dataset, when compared to the original tracking method, proposed in [Spruyt 12]. This can be explained by observing the first row of Table 5.5: Without context based information, the target hand is lost during

Table 5.6: Results on the Signer dataset, reported as the percentage of video frames for which the detected hand location is less than one hand-width from the ground truth location.

[Karlinsky 10]	[Kumar 09]	[Spruyt 12]	Ours
84.9 %	78.95%	97.3%	**100.0%**

tracking several times. When context is added however, tracking accuracy still varies, but the hand is never truly lost.

Furthermore, our method provides a higher precision (row two of Table 5.5) than the method proposed by Buehler *et al.*, when applied to the task of hand tracking. On the other hand, Buehler's approach yields a slightly higher accuracy (last row of Table 5.5) than ours, and has the added advantage of returning an accurate estimate of both the upper and lower arm position. However, their approach requires approximately 100 seconds of processing per video frame, whereas our proposed solution operates in real-time, processing approximately 17 frames per second.

A related, pictorial structure based approach was proposed by Karlinsky *et al.* [Karlinsky 10], who detect hand and arm configurations using an ensemble of feature chains leading from the face location to the hand location. In their method, a face detector is used to estimate the face location, after which the chains model infers the upper body configuration. Karlinsky *et al.* assume the hand width equal to half the face size, and consider a hand detection to be correct if it is within one hand width from the ground truth location.

Finally, Kumar *et al.* [Kumar 09] proposed a discriminative parts based classifier to detect the upper body of a person by embedding pictorial structure. Although their method performs object detection and therefore does not incorporate temporal information, we include their results on the Signers dataset for completeness.

Table 5.6 shows the results for both methods on the Signer dataset, compared to the results obtained by our method with and without contextual information.

Although the chains model proposed by Karlinsky does not employ temporal information, it does require the face location to be known for initialization purposes in each video frame. As such, it can be compared fairly to tracking algorithms. The results illustrated by Table 5.6 show the advantage of incorporating contextual information into a tracking framework, considering the real-time performance of our solution whereas the pictorial structure based approach of Kumar *et al.* needs approximately 15 seconds of processing time per video frame.

Figure 5.21 shows several frames from the Signer data set for illustrative purposes.

Figure 5.21: Illustration of hand tracking on the Signer dataset. Although context modeling is not perfect due to the uniform color of arms and body of the signer, the tracker clearly benefits from the local contextual information.

5 Conclusion

In this chapter we proposed a novel method to automatically learn the context of a tracked object in an unsupervised manner. Furthermore, we introduced two approaches to incorporate this information into a particle filter tracking framework to improve the tracker's robustness against occlusion and distracting background objects. Our algorithm operates in real-time, and as a by-product can output a rough segmentation of the object's context, i.e. the arm in the case of hand tracking. We showed that our solution outperforms the state-of-the-art when dealing with occlusions, whereas tracker performance is not degraded when no occlusions are present. Finally, our method runs in real-time, processing approximately 14 frames per second on modern hardware.

6

A Transparent Floating Display

1 Introduction

One of the most apparent applications of real-time, robust hand detection and hand tracking algorithms, can be found in human-computer interaction (HCI). Traditional HCI occurs through the use of peripheral hardware such as a keyboard, computer mouse, touchpad or joystick. However, these mechanical devices behave not only as a physical barrier between the human and the computer, but also as a psychological barrier, in the sense that they limit the effectiveness and naturalness of the interaction.

Gesture based interaction allows users to directly interact with the computer in a natural manner, e.g. to manipulate objects in a virtual reality space. Recent studies have shown that using a natural gestures based human computer interface significantly reduces the cognitive load on the user [Cornelius 13]. However, a well known problem with gesture based interaction, is the so called 'gorilla' effect: If users are required to raise their arms for a long period of time, fatigue arises in the upper arms [Kyung 12]. For practical applications, a gesture based HCI system using camera input should therefore not require users to keep their hands inside the camera's field of view continuously. Instead, users should be allowed to act freely, which leads to the need for quick re-detection of hands as they appear in the field of view. This problem is often underestimated in real systems. The hand tracking method that was discussed in chapter 4 is capable of automatic re-initialization

Figure 6.1: A see-through display allows multiple users to interact with a virtual object while maintaining eye-contact.

and error recovery, and is therefore a perfect candidate for a gesture based HCI interface.

An increasingly popular subdomain of HCI research is related to multiple-user environments for performing collaborative physical tasks. Examples can be found in rapid prototyping or medical research, where multiple users collaborate to perform a single task. Kraut *et al.* [Kraut 03] showed that visual information, such as the possibility of maintaining eye-contact with other users, is of great importance when performing collaborative tasks. This information helps maintaining situational awareness and helps achieving mutual understanding. In a face-to-face setting, people normally use visual cues to interpret the intentions of others and to estimate the current progress in executing a particular task [Kraut 03]. These visual cues include facial expressions, body language and gestures, and views of the environment and of the object or task of interest. However, in a traditional HCI setting, where each user sits in front of his own computer display, most of these cues are lost. Even if multiple users share the same computer display, collaboration is often difficult because it is not possible to focus on the display, the environment and the other users at the same time.

In this chapter, we discuss the design of a transparent see-through display, illustrated by figure 6.1, that allows multiple users to cooperate while maintaining eye-contact. Furthermore, our display combines an optical illusion, known as the Pepper's ghost effect [Secord 02], with an optical lens system, to create the impression that a 3D virtual object shown on the display is floating in thin air. Although

the underlying idea of this display is simple and well known, several technical details have to be considered during its implementation. Moreover, we extend the idea of a 2D planar display surface to a 3D conic display surface which increases the illusion of observing a 3D holographic floating object. Finally, we discuss several difficulties that were encountered during the production of our prototype display devices.

This chapter is outlined as follows: In Section 2 we describe related work. Section 3.1 describes the development of a transparent, holographic display that produces an image that seems to float in thin air. In Section 3.2 this display is extended into a full 360-degrees display, and a theoretical analysis is provided to cope with optical distortion. Finally, Section 4 provides the technical details of our prototype displays, and compares different interpolation schemes.

2 Related Work

The ability to project floating 3D objects in thin air, combined with a gesture based HCI interface, would result in a truly immersive augmented reality experience, where the user could actually touch the non-existing object. Recently, Kimura *et al.* [Kimura 06] were able to generate a floating 3D hologram, with a maximum resolution of $37 \times 37 \times 37$ dots at 15 frames per second. To achieve this, they ionize air particles by focussing several lasers onto a single 3D point. The result is a monochromatic floating 3D object. Although their research is promising, the low resolution and framerate, the lack of full color support, and the physical danger of using high power lasers, limit its practical use.

Kakeya *et al.* [Kakeya 07] proposed the MOEVision system, which generates a 3D floating image using a complicated optical setup. A large array of carefully aligned projectors generates a multi-view image to achieve a true auto-stereoscopic 3D experience. A fly-eye lens array is used as an anisotropic diffuser to guide the light-field towards each viewing zone. Finally, a set of convex lenses is used to converge the light at a distance from the lens system, such that a real image is generated. This real image is then perceived by the viewers as a 3D image that seems to float in thin air.

The method proposed by Kakeya *et al.* effectively combines two well-known principles in 3D display design. The first principle is often referred to as 'integral imaging'. The second principle is often referred to as 'floating imaging'.

Integral imaging [Stern 06] is a technique to create an auto-stereoscopic effect, which means that depth can be perceived without the need for specialized glasses. Integral imaging uses an array of small fly-eye lenses in front of an LCD panel, each of which cover multiple pixels of the LCD display, to guide the light towards a specific direction. Each small lens of the lens array is called a lenslet, and every pixel underneath a single lenslet shows a different 3D perspective of the image.

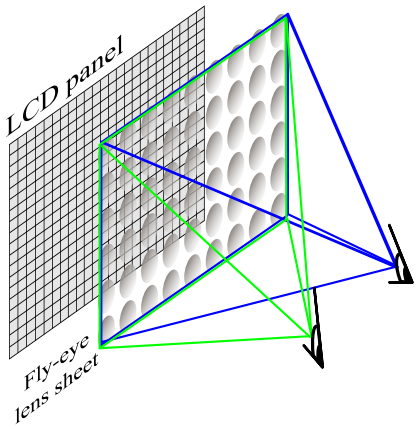


Figure 6.2: Integral imaging: Fly-eye lenses are used to steer the light of distinct, adjacent pixels to different observers.

Depending on the observer's physical position, a different 3D perspective is therefore perceived. As a result, the user experiences both a horizontal and a vertical parallax, which in turn results in an apparent depth, as illustrated by figure 6.2.

The main disadvantage of integral imaging is the loss of spatial resolution. If ten different perspective views are to be generated, both horizontally and vertically, then each lenslet covers $10 \times 10 = 100$ pixels. The horizontal and vertical resolutions of the original display are therefore divided by ten. To mitigate this effect, lenticular lenses are often used instead of fly-eye lenses. Lenticular lenses are cylindrical lens structures, such that only the horizontal resolution of the display is reduced while the vertical resolution is kept intact. In this case, only horizontal parallax can be observed, which means that the user only perceives depth if the head is not tilted.

Although integral imaging can yield an auto-stereoscopic display, the maximally perceived depth depends on the focal length of the lenslets and is therefore small. In order to create a *pop-out effect*, such that the projected image seems to float in front of the viewer, integral imaging can be combined with so called 'floating imaging' concept [Min 05]. Floating imaging is an old 3D display technique that uses a concave mirror or converging lens to produce a real image at a distance that is at least twice the focal length of the chosen optics.

Although integral imaging and floating imaging are often combined [Joohwan 09], each of these techniques can be used individually. Hoshi *et al.* [Hoshi 09] used a large concave mirror to produce a floating image of a 2D display. Although concave mirrors do not suffer from optical abberation effects as much as convex fresnel lenses, large concave mirrors are expensive and difficult to produce.

Whereas the above methods generate a floating image in front of the display, it is also possible to generate a floating image that appears behind the projection surface. A well known and obvious technique to accomplish this is called the 'Pepper's ghost effect': An LCD display is placed on top of a mirror, which is tilted at an angle of 45 degrees. An observer, looking at the tilted mirror, observes the reflection of the LCD display, which appears to be floating behind or 'inside' the mirror itself.

If the mirror is replaced by a transparent reflective material, a see-through display is obtained, which shows a virtual image, floating at a distance behind the projection material. This is used by Sidharta *et al.* [Sidharta 06] to develop an augmented reality display: Layered pieces of LCD material are positioned on top of a tilted mirror, and each LCD layer shows a different parts of an object. As a result, each reflected image appears at a different depth from the projection material, resulting in an auto-stereoscopic display [Sidharta 07]. However, the main disadvantage of this method, is the fact that the user can not touch the floating image, since it appears to float behind a solid, reflective surface. The main advantage compared to the previously discussed methods, is that the display can be made transparent, such that multiple users can keep eye-contact while using the system.

3 Materials and Methods

In this section, we discuss the methods used to develop a transparent, floating imaging display, by combining several of the earlier discussed concepts. We propose to combine floating imaging, based on a convex fresnel lens system, with the Pepper's ghost approach. The result is a transparent display that produces a floating image in front of its projection surface. Furthermore, we extend this idea by designing a conic reflection surface instead of a planar surface. The transparent, conic display shows a holographic image inside its volume, and allows the viewer to walk around the image. This increases the illusion of observing a volumetric object.

In the next paragraphs, both the materials needed to develop this display, and the software needed to cope with optical distortion of the reflected images, are discussed in more detail.

3.1 Floating Imaging: an Introduction

In optics, an important distinction exists between real images and virtual images. Convex mirrors (and planar mirrors) always generate a virtual image, which seems to exist inside or behind the mirror surface. Light rays that seem to originate from the reflected object inside the mirror do not actually exist, hence the term 'virtual

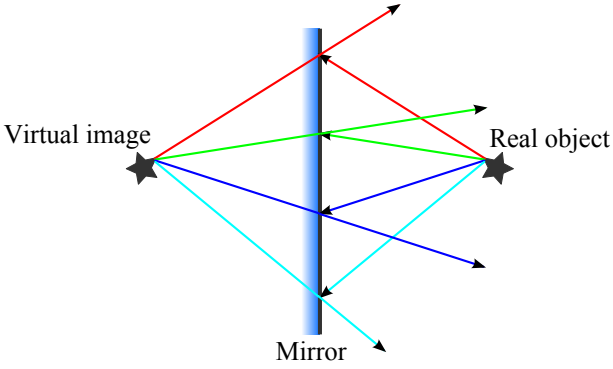


Figure 6.3: Reflection off a planar mirror always produces a virtual image. Light-rays seem to diverge from each point on this virtual image.

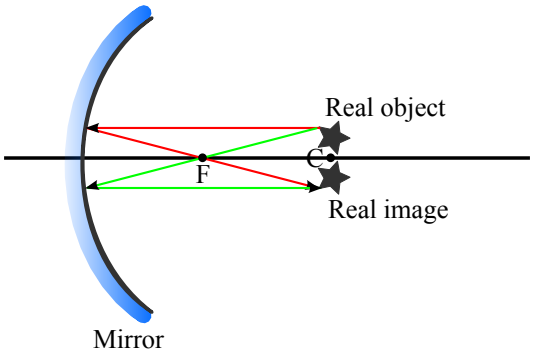


Figure 6.4: Reflection off a concave mirror can produce a real image, if the original object is positioned at a distance that is equal to or larger than the focal length of the mirror. The real image appears in front of the mirror.

image', and always diverge. This is illustrated by figure 6.3. Virtual images can also be created using a concave lens, instead of a convex mirror.

Concave mirrors, on the other hand, can produce a real image if the object is placed at a distance of at least one focal length from the mirror. A real image is an inverted version of the actual object, such that the reflection of the object appears upside-down. A real image is produced at the location where light rays, originating from an object, converge, and appears in front of the mirror. This is illustrated by figure 6.4. Real images can also be created using a convex lens, instead of a concave mirror. A well known example of real image formation, is the image that is produced on the eyeball retina, due to the convex lens in the human eye.

Floating imaging is based on this optical concept, and uses either a concave mirror, or a convex lens to produce a real image of the LCD display, that seems to

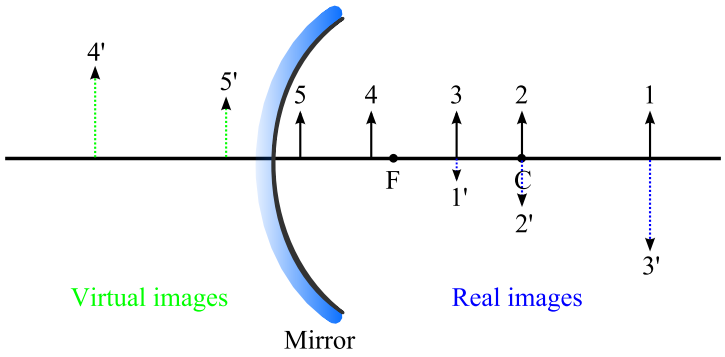


Figure 6.5: Real and virtual image creation by concave mirrors or convex lenses depending on the object's position, relative to F and C. Objects are represented by solid black lines, real images by dotted blue lines, and virtual images by dotted green lines.

be floating in front of the optical system. In the following, let P denote the pole of the mirror or the center of the lens, let F be the focal point of the optical system, and let C be the center of curvature ($C = 2F$ for spherical mirrors and lenses). We denote $f = \|P - F\|_2$ the focal length of the lens, and $c = \|P - C\|_2$ the Euclidean distance from the center of the optical system to the center of curvature. A convex lens or concave mirror only produces a real image if the object is placed at a distance $d \geq f$. If $d < f$, a virtual image is formed behind the optical system instead. The distance from the object to the optical system also defines the magnification factor, and the position of the produced image. This can be summarized as follows, and is illustrated by figure 6.5:

If $d > c$
: A real, inverted and diminished image is produced between C and F.

If $d = c$
: A real, inverted image of equal size to the object is produced at C.

If $f < d < c$
: A real, inverted and magnified image is produced beyond C.

If $d = f$
: A real, inverted and magnified image is produced at infinity. All light rays leaving the optical system are parallel.

If $d < f$
: A virtual, erected and magnified image is produced behind the mirror.

Optical systems can suffer from several types of aberration. The most important are chromatic aberration and spherical aberration. *Chromatic aberration* is

caused by the fact that the magnification factor of the lens depends on the wavelength, such that different colors of the image are focussed at different depths. On the other hand, *Spherical aberration* occurs because a spherical lens refracts light entering at its edges to a slightly different focus point, compared to light entering at its center. Moreover, these differences in focus become more apparent if the optical system is used to produce a magnified image. Thus, for floating imaging, magnification should be avoided, and the LCD display is usually placed at the center of curvature of the optical system.

Optical systems are often characterized by their f-number, which is defined as the ratio of the focal length of the system to the lens diameter. Systems with a lower f-number can collect more light, but are more difficult to produce, partially because a small focal length implies a highly curved optical surface. As a result, lenses with a higher f-number usually suffer less from aberration [Min 01,Ren 07]. For imaging purposes, one might therefore be inclined to opt for a lens system with a large f-number, and thus a focal length that is much larger than the width of the LCD display.

However, since the display needs to be placed at a distance $d = 2f$ from the lens, if no magnification is desired and if a spherical lens is used, a larger f-number implies a larger distance between lens and display. This distance in turn limits the maximal viewing angle under which the observer can still observe object completely. Indeed, let a be the aperture of the lens and let s be the width of the LCD display; If the display is placed at a distance $d = 2f$ from the lens, then the viewing angle in which the complete object can still be observed is given by:

$$\alpha = 2 \arctan\left(\frac{a-s}{4f}\right). \tag{3.1}$$

For example, if a standard 19-inch display is used with an aspect ratio of $16:9$, then the display width is $s = 42\ \mathrm{cm}$. If the lens has an aperture size $a = 84\ \mathrm{cm}$, and an f-factor of 2, then the focal length $f = 168\ \mathrm{cm}$. In this example, the real image would be produced at a distance of $336\ \mathrm{cm}$ in front of the lens, and the LCD display would have to be placed at a distance of $336\ \mathrm{cm}$ behind the lens. The viewing angle in which the complete floating object can be observed is then $\alpha = 7.15°$. If the f-factor would have been 1, then $\alpha = 14.25°$, and the floating image would appear at a distance of $168\ \mathrm{cm}$ in front of the lens. If the f-factor would have been 0.5, then $\alpha = 28.07°$, and the floating image would have been produced at a distance of $84\ \mathrm{cm}$ in front of the lens.

Finally, the intensity of light, coming from a point source (e.g. LCD pixel) follows the inverse-square law, which means that it is inversely proportional to the square of the distance from the source. For floating imaging, this means that the light collection capabilities of the lens varies inversely proportional to the square of its f-number. Therefore, smaller f-numbers result in brighter images. On the

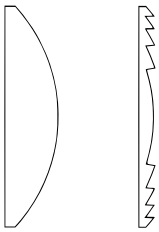


Figure 6.6: Cross-section of a fresnel lens. Fresnel lenses mimic the optical properties of traditional lenses but can be much thinner and cheaper to produce.

other hand, the f-number generally also determines the quality of the collimated output, and abberation effects increase with a decreasing f-number as discussed earlier.

Due to these many contradicting physical limitations, combined with production limitations, it is often difficult to design an optimal optical system. Because of production process limitations, very large convex lenses and concave mirrors, needed for floating imaging purposes, usually have an f-number close to 1.5. This means that the diameter of the LCD display directly determines the distance at which the real image will be perceived to be floating in the air, and the intensity and aberration artifacts of the perceived image.

Hoshi *et al.* [Hoshi 09] used a large concave mirror to achieve the floating image effect, whereas Min *et al.* [Min 05] used a convex lens. Although both methods are theoretically equivalent, mirrors usually suffer less from aberration than lenses. Moreover, a traditional convex lens, with a diameter equal to the diameter of a modern LCD display, would be extremely heavy and expensive, due to its required thickness.

Although this might seem to indicate that the use of a concave mirror is preferable over a convex lens, thin and inexpensive fresnel lenses can be used instead. A fresnel lens consists of a number of concentric rings, each of which have the same optical properties as the corresponding ring of a normal lens. The surface of a fresnel lens closely mimics the optical properties of the surface of a normal lens, while the vast body of material in a spherical lens, that would have no optical importance, is removed. This is illustrated by figure 6.6. Since fresnel lenses can easily be produced in much larger dimensions than concave mirrors, lenses should be preferred over mirrors in large-scale floating imaging systems. However, it should be noted that fresnel lenses introduce additional aberration artifacts, depending on their groove density. A high groove density results in better image quality by avoiding diffraction problems, while a low groove density yields better efficiency which results in a brighter image.

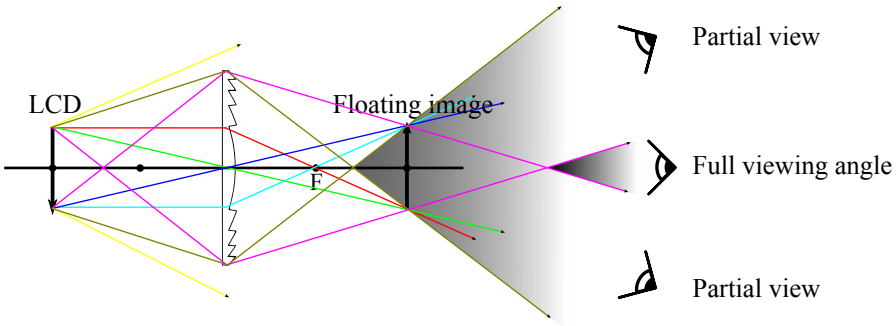


Figure 6.7: A real image of an LCD display is produced by a convex Fresnel lens, placed at a distance $2f$ from the display.

3.2 A Transparent Floating Image Display

Floating imaging produces a real image that is perceived to float at a specific depth in space. When combined with the robust hand detection and tracking methods discussed in previous chapters, this allows for the development of a holographic display, such that the observer can use his hands to touch and manipulate the floating object.

Holographic displays could solve one of the problems related to HCI interfaces, discussed in Section 1, namely the psychological barrier of peripheral hardware. In an interactive floating imaging setup, the user does not need external mechanical devices to manipulate the presented data. Additionally, the data is presented as being part of the real world in which the user operates, instead of being part of a virtual world that seems to exist inside or behind an LCD display. Figure 6.7 illustrates the floating imaging concept schematically. The lens diagram also shows the region in which the full floating object can be perceived, and the region in which a part of the floating object can still be seen.

However, this approach alone does not solve the second problem that was discussed in Section 1; namely the lack of visual information about the real environment, needed to maintain situational awareness and to achieve mutual understanding when executing collaborative tasks. If the lens is placed in front of an LCD display, the observer still sees an opaque medium behind the floating object. This medium, i.e. the lens itself and the display behind it, blocks the user's view such that he can not observe its surroundings.

To solve this problem, we propose to use the Pepper's ghost effect, as discussed in Section 1. A semi-transparent mirror is placed at an angle of 45 degrees w.r.t. an LCD display, such that a virtual image is produced behind the reflective medium. Due to this transparency, a holographic image seems to float in thin air. Since only part of the light is reflected, while the other part passes through the medium

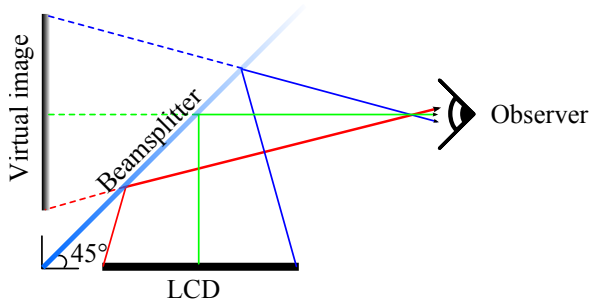


Figure 6.8: A virtual image of an LCD display is produced by a planar reflection medium, acting as a beam splitter.

and is lost, a transparent reflector is often referred to as a 'beam splitter' in optics. The beam splitter can be any kind of transparent material with a high reflection coefficient. The resulting Pepper's ghost display is illustrated conceptually by figure 6.8.

The major disadvantage of the Pepper's ghost display, however, is the fact that the produced image is virtual. Because the image seems to float behind the transparent medium, it can not be interacted with directly. We therefore propose to combine the concepts of floating imaging with the Pepper's ghost effect to develop a display that is transparent and that produces a real image, floating in front of the transparent material. As a result, the object seems to float in the real world, and the user can see through the object in order to observe its surroundings. This is shown schematically in figure 6.9.

The beam splitter should both be transparent and highly reflective. Obvious material choices that exhibit both properties are glass, polyacrylate (acrylic) and polycarbonate. We propose the use of tinted polycarbonate due to its interesting reflection properties, as will be explained in the following paragraphs. Since only a small fraction of the light is reflected by the beam splitter, the produced image will be much less bright compared to the original LCD display. To be viewable in daylight conditions, a high reflection coefficient is therefore extremely important. The percentage of light that is reflected by the beam splitter depends on several system parameters:

1. The wavelength of the light to be reflected;
2. The refraction index of the beam splitter;
3. The refraction index of the surround material (air);
4. The polarization of the light to be reflected;
5. The reflection angle, measured towards the normal of the beam splitter.

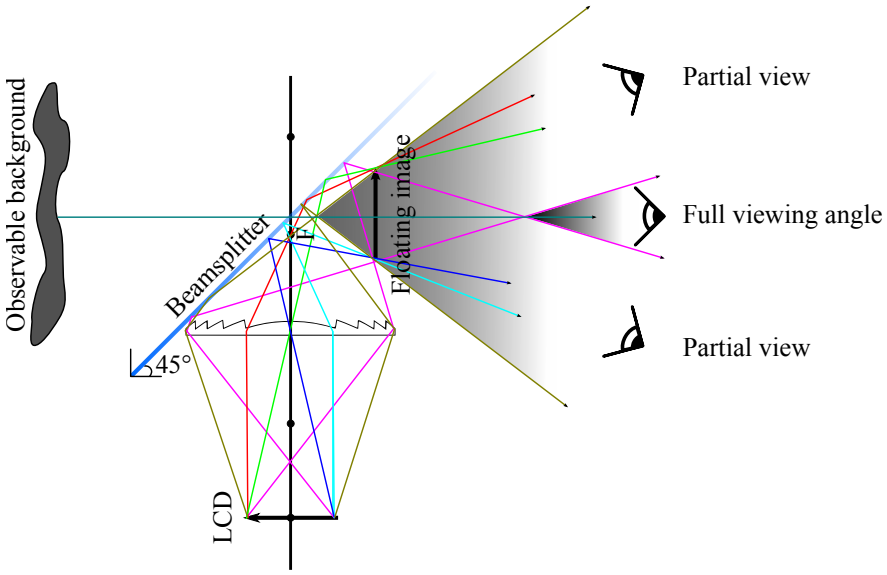


Figure 6.9: A virtual image of an LCD display is produced by a planar reflection medium, acting as a beam splitter.

Table 6.1: Refraction indices of different materials.

	Red (625 nm)	Green (550 nm)	Blue (440 nm)
Glass	1.541	1.544	1.554
Acrylic	1.493	1.496	1.505
Polycarbonate	1.581	1.590	1.610

The wavelength of the light to be reflected depends on the color of the LCD pixels and the type or brand of LCD display. To roughly estimate the reflection properties of the different types of material, we assume typical consumer model hardware, with a maximal spectral radiance for the red sub-pixels at 625 nm, for the green sub-pixels at 550 nm, and for the blue sub-pixels at 440 nm [Kwak 00, Sharma 02].

The refraction index of the beam splitter is a property of the material that also depends on the wavelength of the incoming light beam. Table 6.1 lists the refraction indices for three types of highly transparent material [Polyanskiy 14]. The surrounding medium is assumed to be air, with a refraction coefficient of 1.

The light to be reflected is assumed to be s-polarized (horizontal polarization). The polarization of the light and its effects on the reflection properties will be discussed in more detail on page 209. Finally, since the beam splitter is positioned at an angle of 45 degrees towards the LCD display, light beams always hit the

Table 6.2: Fraction of light that is reflected by each of the different materials.

	Red (625 nm)	Green (550 nm)	Blue (440 nm)
Glass	10.16%	10.24%	10.47%
Acrylic	9.03%	9.11%	9.32%
Polycarbonate	11.12%	11.32%	11.79%

beam splitter with an angle of 45 degrees or less, towards the normal vector on the beam splitting surface. To obtain an estimate of the reflection properties of each material, we assume the angle to be 45 degrees.

Based on these parameters, the reflection properties of each of the materials listed in table 6.1 can be calculated using the well known Fresnel equations which describe the behavior of light that moves from a medium with one refractive index, to a medium with another refractive index. For simplicity, we assume the beam splitter to be infinitely thin, and thus only consider the top surface of the refractive material.

Let $n_1 = 1$ be the refraction index of air, n_2 be the refraction index of the beam splitter material, θ_i be the incident angle of the light, and θ_t be the angle of transmission of the light. The reflection coefficient R_s for horizontally polarized light is then given by the Fresnel equation:

$$R_s = \left| \frac{n_1 \cos\theta_i - n_2 \cos\theta_t}{n_1 \cos\theta_i + n_2 \cos\theta_t} \right|^2 = \left| \frac{n_1 \cos\theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin\theta_i\right)^2}}{n_1 \cos\theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin\theta_i\right)^2}} \right|^2 . \quad (3.2)$$

Plugging in the known parameters yields the percentage of horizontally polarized light that is reflected by each of the materials. These results are listed in table 6.2. The reflection coefficients show that polycarbonate is able to reflect more of the light, resulting in a brighter image, when compared to glass or acrylic. An additional advantage of polycarbonate is that it has an extremely high impact resistance and therefore does not break easily as opposed to acrylic or glass. For these reasons, polycarbonate was used as a beam splitter material for our final prototype. However, instead of using a completely transparent polycarbonate reflector, we propose to use a tinted version of the material. This helps reducing the apparent brightness of the ambient light that enters the display from the back-side and thus increases the contrast of the produced image.

The above analysis reveals two possible disadvantages of the Pepper's ghost display. Firstly, the reflection coefficient depends on the angle of incidence of the light wave. This angle of incidence differs for each pixel location of the LCD display. Therefore, the produced floating image will not exhibit uniform brightness; Pixels in the center of the display appear brighter than pixels at the edges of the

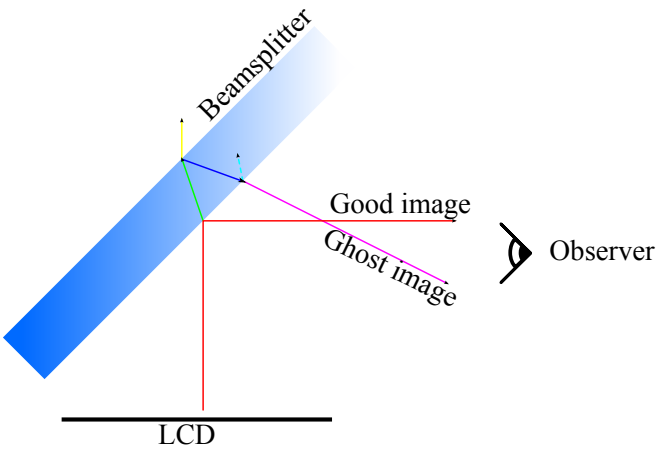


Figure 6.10: Ghosting effect due to the beam splitter thickness.

display. Secondly, the reflection coefficient depends on the wavelength of the incident light beam. Therefore, the produced image will not exhibit an exact color reproduction. Blue light is reflected more efficiently than red light, which in turn is reflected more than green light. However, the first problem can be largely solved by calibrating the brightness of the individual pixels of the LCD display, based on the expected position of the observer. The second problem can be counteracted by varying the pixel brightness in real-time, based on the pixel color.

Another unwanted effect that results from the proposed setup, is ghosting. In the above analysis, the thickness of the beam splitter was ignored. However, due to its thickness, light reflects and refracts both at the front side (air-polycarbonate boundary) and at the back size (polycarbonate-air boundary) of the material. For example, if 11% of the light is reflected by a polycarbonate beam splitter at an angle of 45 degrees, then 89% of the light is transmitted by the beam splitter material. However, due to the difference in refraction index between the polycarbonate material and air, the angle of transmission changes for the light that is not reflected. Using the Fresnel equations, it can be shown that the transmission angle becomes 26 degrees. A fraction of the light therefore hits the back side of the beam splitter at this angle of 26 degrees. Using the Fresnel equation again shows that 11% of the light that hits the back side is not transmitted outside the beam splitter, but is reflected back instead. This fraction of light hits the front side of the beam splitter at an angle of 26 degrees. At this angle, 93.4% of the light is transmitted outside the beam splitter, and thus reaches the observer's eye. This means that the observer sees at least two images. The first image has a brightness of 11% of the original LCD brightness. The second image is perceived at a slightly different angle and is seen as a shifted version of the first image. This image has a brightness

of $89\% \times 11\% \times 93.4\% = 9.14\%$ of the original LCD brightness, and is referred to as a 'Ghost image'. This is illustrated by figure 6.10.

Ghosting effects can be greatly reduced by applying an anti-reflective coating on the back side of the beam splitter. This is a transparent coating consisting of several thin layers, with diminishing refraction index. The refraction index of the first layer is close to the refraction index of the beam splitter material, whereas the refraction index of the outer layer is close to the refraction index of air. Since the reflection coefficient depends on the refraction index of the material, gradually lowering the refraction index allows the light to be bent slowly, until it transmits into the surrounding air.

Although we proposed to use a polycarbonate beam splitter, mainly due to its low price, impact resistance, and optical properties, several improvements can be considered to reduce some of the earlier described problems. If transparency is less important than the brightness of the final image, the beam splitter can be coated with a thin and highly reflective coating such as aluminum or titanium. This is often used in optics to produce beam splitters with different reflection / transmission ratios. A disadvantage of this technique is the fact that a metallic coating increases the amount of light absorption, resulting in a reduced transparency. A more expensive method that effectively solves this problem, is the use of a dielectric material. A dielectric reflector is composed of several thin layers of dielectric material. By varying the thickness of each layer, the resulting reflector can be produced such that it has low absorption. Furthermore, dielectric reflectors can be designed such that specific wavelengths of light are reflected more, whereas others are transmitted. This can be used to design a beam splitter that efficiently reflects the light produced by the LCD display, while transmitting ambient light. Such a beam splitter would reduce glare and ghosting, would be highly transparent, and would produce high brightness images.

Finally, it is important to consider the polarization of the LCD display when designing the Pepper's ghost display. Modern LCD displays contain a back-light that produces unpolarized light. The LCD panel itself is squeezed between a horizontal polarization filter and a vertical polarization filter. If a pixel is supposed to be black, the liquid crystal is set to its inactive state, such that the horizontal and vertical polarization filters effectively block most of the light from exiting the display. If a specific color needs to be produced the liquid crystal actively changes the polarization of the light, such that light of the expected wavelength exits the display. Since there is no official production standard, some manufacturers use a vertical-LCD-horizontal polarization configuration, whereas others employ a horizontal-LCD-vertical configuration. However, if the light that exits the display is vertically polarized (p-polarized), the brightness of the produced image will be extremely low. The reason for this is that vertically polarized light is reflected less efficiently than horizontally polarized light. This can be verified by calculating

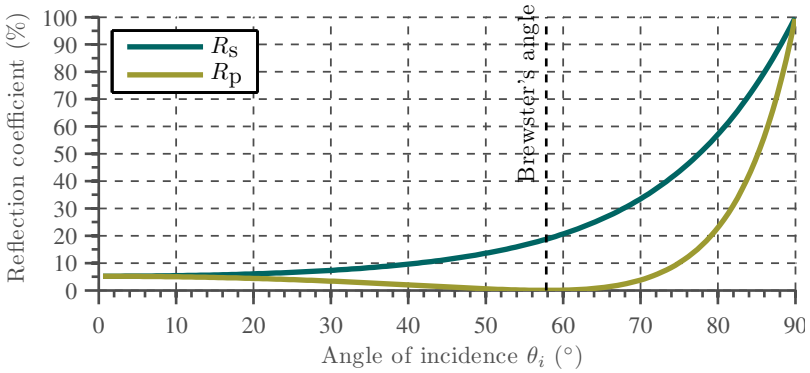


Figure 6.11: Reflection coefficient of polarized light, plotted against the incident angle for a polycarbonate reflector.

the reflection coefficient for different incident angles, using the Fresnel equations. Figure 6.11 shows a plot of the reflection coefficient R_s for horizontally polarized light, and the reflection coefficient R_p of vertically polarized light against the incident angle of light, for a polycarbonate beam splitter. This shows that the reflection angle for vertically polarized light approaches zero if the incident angle is close to 45 degrees. The reason for this is that 45 degrees is close to Brewster's angle for vertically polarized light in polycarbonate. Brewster's angle is the angle at which all light is transmitted instead of reflected.

Therefore, if an LCD display is known to produce vertically polarized light, simply positioning it in portrait mode, instead of landscape mode, increases the brightness of the produced image with a factor 9 due to the change in polarization.

Figure 6.12 shows our Pepper's ghost display, showcased at a tradeshow for the company Tomra NV.

3.3 A Conic 360° Display

The Pepper's ghost display, discussed in Section 3.2, can be made transparent such that users can see their natural environment through the floating object. However, due to the planar reflection surface, the viewing angle is limited. Moreover, although the produced image seems to float at a certain distance from the observer, the illusion of perceived depth is limited because of the planar character of the beam splitter.

To enhance the perceived depth experience, we propose a three dimensional beam splitter in the shape of a cone. The produced image appears to float inside the transparent cone. Due to the curved surface, and the fact that the observer can walk around the display, the illusion of observing a holographic 3D image is

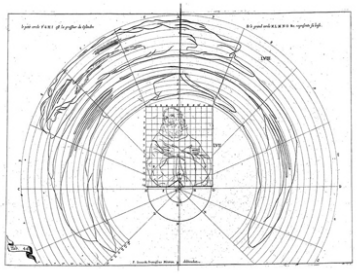
Figure 6.12: Our Pepper's ghost display prototype, showcased at an international tradeshow for Tomra NV.

Figure 6.13: Two conic 360 degrees displays prototype of size $80\ \mathrm{cm} \times 80\ \mathrm{cm} \times 50\ \mathrm{cm}$.

created, even though only a 2D projection is shown. Furthermore, by projecting an image at two opposing sides of the cone, multiple observers can interactively perform collaborative physical tasks while maintaining eye contact. Figure 6.13 shows two of our final prototype displays, both of which have a physical size of $80\ \mathrm{cm} \times 80\ \mathrm{cm} \times 50\ \mathrm{cm}$ and can be interactively controlled by using hand gestures. The cone was produced by means of vacuum forming the polycarbonate material onto a mold.

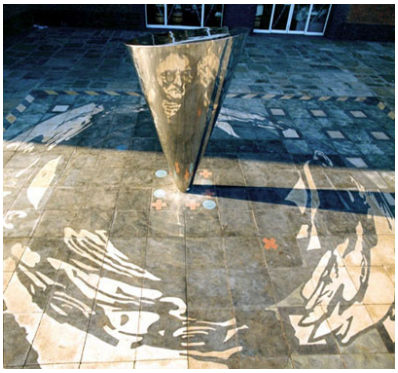
A major difference with the Pepper's ghost display presented in Section 3.2, is the fact that the beam splitting material is now a three dimensional curved surface instead of a 2D planar surface. Although a spherical surface could have been used instead of a conic surface, this would introduce both horizontal and vertical distortion of the image. A conical surface allows us to produce an image with full vertical resolution, limiting the optical distortion to the horizontal direction. Similarly, a cylindrical surface could have been used. However, this would require the LCD display to be tilted, limiting the user's viewing angle. By using a cone with apex angle $2\theta = 90\ \mathrm{degrees}$, the pixel column that is in front of the observer behaves as if it would have been reflected by an infinitely narrow planar mirror. Towards the edges of the image however, the optical distortion increases rapidly due to curvature of the reflective surface.

This distortion can be described by a non-linear transformation. By digitally applying the inverse of this transformation to the image displayed on the LCD, an



(a) Jean-Francois Niceron.

(b) István Orosz.



(c) John Dalton.

(d) Awtar Singh Virdi.

Figure 6.14: Anamorphic art from several well known artists.

undistorted floating image can be produced. While the context of this research is image formation, the same idea has been explored in literature in the context of image capturing [Burbridge 08]. In this case, a conical mirror is used to create an omnidirectional camera. These types of cameras are often referred to as 'catadioptric cameras' since they combine both refraction (lenses) and reflection (mirrors). Whereas catadioptric cameras are concerned with the forward transformation, i.e. finding the world coordinate of each distorted image coordinate, we are more interested in the backward transformation, i.e. calculating the image coordinate that would be observed at a specified world coordinate.

The backward transformation used to map world coordinates to their corresponding image coordinates after distortion by reflection on a curved surface has been a subject of interest by many artists since the 16th century in the form of anamorphic art [Hunt 00]. One of the most notable artists in this field was Jean-Francois Niceron [Niceron 52], who produced anamorphic images for several

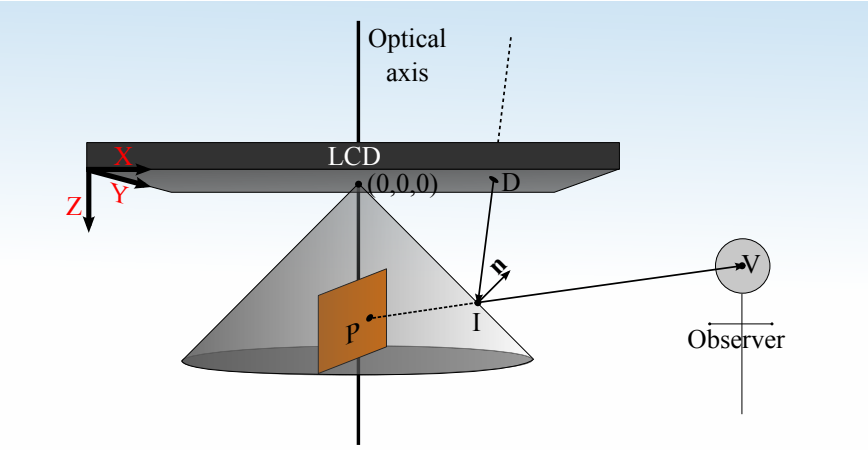


Figure 6.15: Anamorphic projection setup and definitions.

types of curved surfaces such as cylinders, spheres and cones. These images look distorted when viewed on canvas, but appear undistorted when observed after reflection off the curved surface, as illustrated by figure 6.14. Anamorphic art is created by manually mapping image regions from a rectangular grid to a non-rectangular grid, as illustrated by figure 6.14 (a). However, for our purpose, an analytical solution is needed for this transformation such that it can be implemented in software. To our knowledge, a solution to this problem for a conical reflector that is observed at a perpendicular angle to the cone's axis, has not been proposed in literature yet. The next paragraphs discuss the derivation of our proposed solution.

In the following analysis, we assume a left-handed coordinate system, with the LCD display positioned in the (x, y)-plane, and the optical axis of the cone passing through the origin. The cone segment is defined by its height h, the radius r of its base, and the aperture or apex angle 2θ. This is illustrated by figure 6.15. Although the apex of the cone coincides with the origin in figure 6.15, this is not a requirement in the following derivation. In practice, a truncated cone will be used, such that the virtual apex is shifted in the z-direction with respect to the origin. The reason is that the apex itself has no dimensions and can therefore not be used to reflect light.

The observer is assumed to be positioned at point V, looking at a virtual image pixel at point P:

$$P = (p_x, p_y, p_z),$$
$$V = (v_x, v_y, v_z).$$

The virtual image point P is produced by reflection of a light ray, exiting the LCD pixel at point D. The problem to be solved is then to find the coordinates of D, such that viewer at V perceives the image point to be floating inside the cone, at P.

To do so, we first find the coordinates of point I, which defines the intersection of the virtual light ray in the direction $V - P$ with the cone's surface. Due to the Pepper's ghost effect, the viewer observes image point D as if it was located at point P, and thus assumes that a light ray travels in the direction $V - P$. We define the line connecting V and P by its parametric form:

$$\begin{aligned} L(t) &= P + t(V - P) && (3.3)\\ &= (p_x, p_y, p_z) + t((v_x, v_y, v_z) - (p_x, p_y, p_z)) && (3.4)\\ &= ((p_x + t(v_x - p_x)), (p_y + t(v_y - p_y)), (p_z + t(v_z - p_z))), && (3.5) \end{aligned}$$

where $t \in [0, 1]$.

To determine the coordinates of I, equation (3.5) can simply be plugged into the equation of a cone. The equation of a truncated cone, such that the virtual apex is shifted in the z-direction with a distance $-z_0$ is defined as follows:

$$x^2 + y^2 = (z + z_0)^2 \tan^2 \theta. \tag{3.6}$$

The intersection point I can then be found as follows:

$$\begin{aligned} x^2 + y^2 &= (z + z_0)^2 \tan^2 \theta \\ \Leftrightarrow 0 &= x^2 + y^2 - (z + z_0)^2 \tan^2 \theta \\ \Leftrightarrow 0 &= (p_x^2 + 2p_x t(v_x - p_x) + t^2(v_x - p_x)^2) \\ &\quad + (p_y^2 + 2p_y t(v_y - p_y) + t^2(v_y - p_y)^2) \\ &\quad - \tan^2 \theta((p_z^2 + 2p_z t(v_z - p_z) + t^2(v_z - p_z)^2) + 2(p_z + t(v_z - p_z))z_0 + z_0^2) \\ \Leftrightarrow 0 &= t^2(v_x^2 - 2v_x p_x + p_x^2 + v_y^2 - 2v_y p_y + p_y^2 - \tan^2 \theta(v_z^2 - 2v_z p_z + p_z^2)) \\ &\quad + t(2p_x v_x - 2p_x^2 + 2p_y v_y - 2p_y^2 - \tan^2 \theta(2p_z v_z - 2p_z^2 + 2z_0 v_z - 2z_0 p_z)) \\ &\quad + (p_x^2 + p_y^2 - \tan^2 \theta(p_z^2 + 2p_z z_0 + z_0^2)) \end{aligned}$$

which is a quadratic equation in t:

$$at^2 + bt + c = 0$$

with

$$\begin{aligned} a &= v_x^2 - 2v_x p_x + p_x^2 + v_y^2 - 2v_y p_y + p_y^2 - \tan^2 \theta(v_z^2 - 2v_z p_z + p_z^2) \\ b &= 2p_x v_x - 2p_x^2 + 2p_y v_y - 2p_y^2 - \tan^2 \theta(2p_z v_z - 2p_z^2 + 2z_0 v_z - 2z_0 p_z) \\ c &= p_x^2 + p_y^2 - \tan^2 \theta(p_z^2 + 2p_z z_0 + z_0^2). \end{aligned}$$

Although this quadratic equation has two possible solutions ($L(t)$ intersects the cone both at the front side and at the back side), only the positive solution is of interest to us:

$$t_i = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

The intersection point I can now be obtained by replacing t in equation (3.5) by t_i:

$$\begin{aligned} L(t_i) &= ((p_x + t_i(v_x - p_x)), (p_y + t_i(v_y - p_y)), (p_z + t_i(v_z - p_z))) \\ &= (I_x, I_y, I_z) \\ &= I. \end{aligned}$$

Once the coordinates of I are known, the direction of the vector $\mathbf{r} = D - I$, towards the LCD display, can be found by reflecting vector $\mathbf{v} = I - V$ around the normal vector $\mathbf{n}$ at point I. This is the result of the law of reflection which states that angle of incidence equals the angle of reflection, where the angles are defined towards the normal of the reflective surface. The normal vector of the conic surface $f(x, y, z) = 0$ at $I = (x_0, y_0, z_0)$ is the unit vector in the direction of the gradient of f(x,y,z):

$$\nabla f(x, y, z) = (\frac{\partial f(x, y, z)}{\partial x}, \frac{\partial f(x, y, z)}{\partial y}, \frac{\partial f(x, y, z)}{\partial z})$$

where

$$\begin{cases} \frac{\partial f(x,y,z)}{\partial x} & = 2x \\ \frac{\partial f(x,y,z)}{\partial y} & = 2y \\ \frac{\partial f(x,y,z)}{\partial z} & = -2\tan^2\theta(z + z_0). \end{cases}$$

The normal vector of the conic surface at point I is therefore defined as:

$$\mathbf{n} = (I_x, I_y, -\tan^2\theta(I_z + z_0)).$$

In order to find point D of the LCD display, from which the light ray originates, we simply reflect light ray $L(t)$ around $\mathbf{n}$. Since $L(t)$ is defined by $\mathbf{v} = I - V$, this corresponds to reflecting vector $\mathbf{v}$ around the normal vector $\mathbf{n}$ at point I. The reflected vector is denoted $\mathbf{r} = D - I$, and can be obtained by considering the projection of $\mathbf{v}$ onto n. This is illustrated by figure 6.16.

From figure 6.16, it is clear that vector $\mathbf{r}$ can be written as:

$$\mathbf{r} = \mathbf{v} + 2\mathbf{a}. \tag{3.7}$$

Furthermore, since $\mathbf{p} = \mathbf{v} + \mathbf{a}$, vector $\mathbf{a}$ is obtained as:

$$\mathbf{a} = \mathbf{p} - \mathbf{v}. \tag{3.8}$$

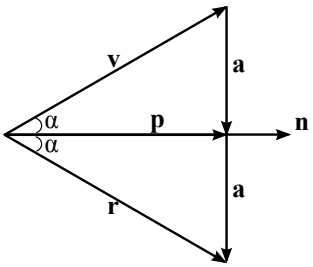


Figure 6.16: Illustration of the law of reflection.

Finally, since $\mathbf{p}$ is the projection of $\mathbf{v}$ on $\mathbf{n}$, we find $\mathbf{p}$ as:

$$\mathbf{p} = \frac{\langle \mathbf{n}, \mathbf{v} \rangle}{|\mathbf{n}|^2} \mathbf{n}. \tag{3.9}$$

Thus, from equations (3.8) and (3.9), it follows that:

$$\mathbf{a} = \frac{\langle \mathbf{n}, \mathbf{v} \rangle}{|\mathbf{n}|^2} \mathbf{n} - \mathbf{v}, \tag{3.10}$$

and from equations (3.7) and (3.10), it then follows that the reflection vector can be found as:

$$\begin{aligned} \mathbf{r} &= \mathbf{v} + 2(\frac{\langle \mathbf{n}, \mathbf{v} \rangle}{|\mathbf{n}|^2} \mathbf{n} - \mathbf{v}) \\ &= \frac{2\langle \mathbf{n}, \mathbf{v} \rangle}{|\mathbf{n}|^2} \mathbf{n} - \mathbf{v} \\ &= (r_x, r_y, r_z). \end{aligned}$$

Using the same parametric form as was used to define light ray $L(t)$ in equation (3.5), the reflected light ray along vector $\mathbf{r}$ is now defined as:

$$R(t) = (I_x, I_y, I_z) + t(r_x, r_y, r_z) \tag{3.11}$$

$$= (I_x + tr_x, I_y + tr_y, I_z + tr_z). \tag{3.12}$$

Finally, to determine the coordinates of D, the point of intersection between $R(t)$ and the LCD display needs to be calculated.

We assumed that the LCD display is positioned in the (x, y)-plane, such that by definition $z = 0$. Thus, setting the z-component in equation (3.12) to zero, allows us to find t:

$$\begin{aligned} I_z + t_j r_z &= 0 \\ \Leftrightarrow t_j &= \frac{-I_z}{r_z}, \end{aligned}$$

Table 6.3: Parameter specifications for the transformation results of figure 6.17.

Display resolution	1920 px × 1200 px
Display diagonal	24.0678 inches
Viewer coordinates (cm)	(0, 1000, -50)
Cone aperture	90°
Cone height	345 mm
Cone cutoff size(truncated)	15 mm
LCD display bezel size	21.05 mm
Air gap between display and cone	5 mm

such that finally the intersection is found as:

$$D = R(t_j) = (I_x - \frac{I_z r_x}{r_z}, I_y - \frac{I_z r_y}{r_z}, 0). \tag{3.13}$$

4 Results and Discussion

In order to transform an original image such that it is displayed correctly after reflection on the cone, each pixel P of the image has to be moved to its new coordinates defined by equation (3.13). This can be efficiently implemented in a fragment shader program (GLSL) such that the transformation is obtained in real-time. To further avoid redundant computations, we generated a lookup-table (LUT) which is transferred to the GPU as a texture map. This LUT allows the fragment shader to quickly obtain the destination coordinates for each of the original image coordinates.

An important side note is that the above analysis was carried out in world coordinates. Therefore, the coordinates of point D, i.e. distances in millimeters or centimeters, need to be translated to display coordinates, i.e. number of pixels. This transformation depends of the pixel size, and therefore the pixel density of the LCD display. The parameters of our prototype display, needed to apply the anamorphic transformation, are listed in table 6.3.

For illustration purposes, figure 6.17 shows the result of the transformation, applied to two different images, for a conic display with the specifications listed in table 6.3.

Note that the transformation, needed to undo the optical distortion introduced by the conical mirror, stretches the original image towards the apex of the cone. As a result, interpolation is needed to solve for unknown pixel values. Because the image is stretched more towards the apex of the cone, anisotropic interpolation along the iso-contours of the cone would yield smoother results than interpolation on a rectangular grid. However, in the context of our current research we use simple bilinear interpolation and consider more advanced techniques future work.

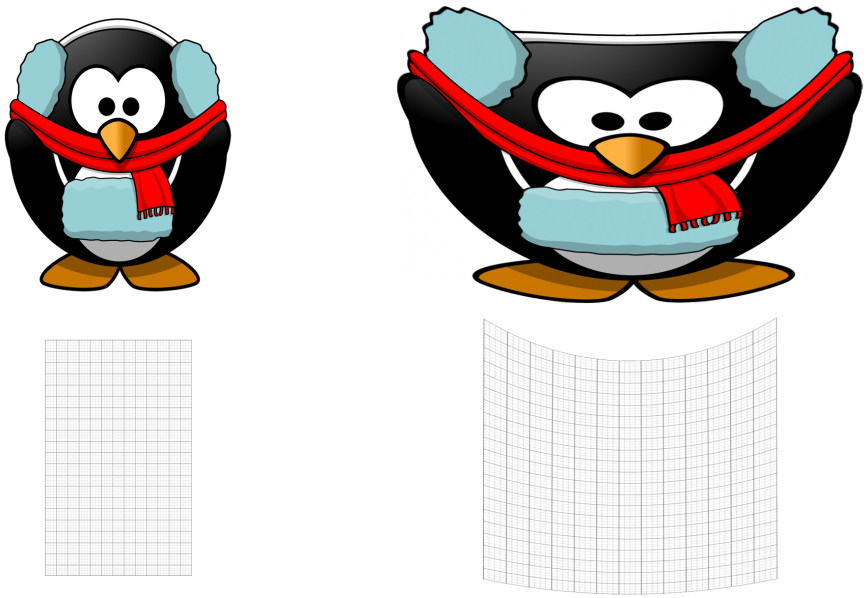


Figure 6.17: Anamorphic transformation applied to two different images. Bilinear interpolation was used to produce the results.

Figure 6.18 shows the final prototype display that was built based on the principles explained in this chapter.

The proof-of-concept display, illustrated by figure 6.18 was showcased at several national and international trade shows and conferences. Simple software demonstrator applications allowed visitors to rotate a virtual object by moving their hands in the air. Another software demonstrator presented the user with a racing game, in which the user controlled the car by means of hand gestures. The next few paragraphs present an overview of the shortcomings of the current gesture controlled interface, reported by these users.

Hand gestures used for controlling the objects were chosen to be simple swipe and pinch gestures as the learning curve for correctly executing more difficult gestures appeared to be too high for most users. An interesting point of future research would therefore focus on the human side of gesture controlled interfaces: Even if hands can be detected and tracked robustly, users might not want to use their hands to control an interface if it requires them to learn to execute certain gestures in a specific manner.

Moreover, although our algorithms allow for automatic error recovery and redetection when the hand leaves or enters the video frame, users still need to keep their hands within the camera's field of view as long as they are executing a task. For example, when playing the racing game, many users got tired and weren't

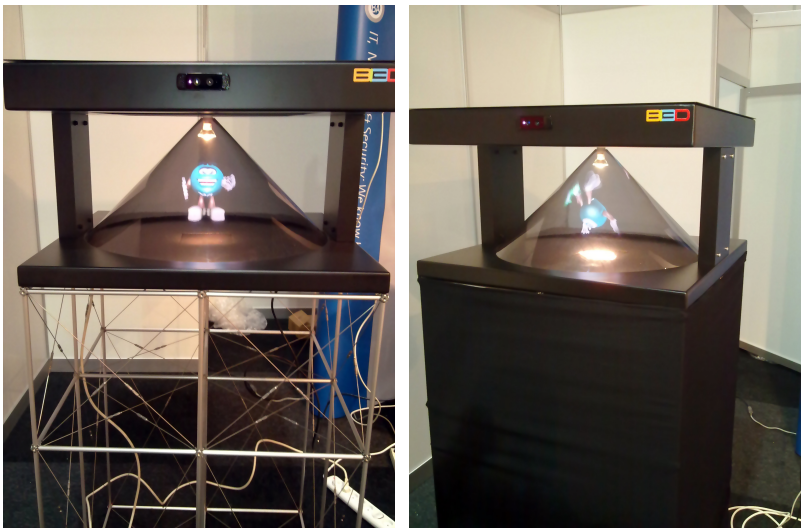


Figure 6.18: The final prototype display.

willing to keep their hands in an upwards position long enough to complete the game. This is often referred to as the 'gorilla' effect [Kyung 12], and should be taken into account when designing HCI software.

Finally, it is often difficult for users to estimate where their hands are positioned, relative to the boundaries of the camera's frustrum. It is thus important to notify the users when the hands tend to leave the video frame, or become invisible to the camera. Future research should therefore focus on the need for feedback towards the user.

5 Conclusion

In this chapter we discussed two related methods to create a transparent display that produces a 3D illusion by introducing a sense of depth. Each of the design steps involved were discussed in detail, and suggestions for future work were presented.

The first method combines the well known Pepper's Ghost effect with a technique known as 'floating imaging'. A large, convex fresnel lens is positioned in front of a beam-splitter, at a distance of twice the focal length of the lens. The beam-splitter is placed at an angle of 45 degrees towards an LCD panel, and reflects a virtual image towards the fresnel lens. The fresnel lens then produces a real image of this LCD panel in front of the lens such that the image seems to float in thin air.

The second method extends this idea to 3D by using a transparent, conic reflection surface instead of a planar beam-splitting medium. To undo the optical distortion, introduced by the conic reflection volume, we proposed an algorithmic solution that effectively inverts this distortion before the image is displayed.

The prototypes of our displays have been showcased at various national and international tradeshows and expos, and were received with much enthusiasm by the public. By integrating gesture recognition into the display, users can perform collaborative tasks while maintaining visual information. This results in a more immersive experience when compared to traditional computer displays and peripheral hardware.

7 Overall Conclusion

In the past chapters we discussed several important building blocks of the novel, robust object tracking method described in chapter 4. Furthermore, a new type of interactive human-computer interface was developed and discussed in chapter 6.

The main conclusions of this PhD research are summarized in Section 1 of this chapter. The scientific output of this research is summed up in Section 2. Finally, we discuss several possible areas for future research in Section 3.

1 Conclusions

In chapter 2, we proposed a novel object detector that is scale and rotation invariant, and operates in real-time. Scale and rotation invariance is directly incorporated in the training stage of the classifier, whereas traditional methods achieve invariance by repeatedly executing the detector on scaled and rotated versions of the image.

Our method can detect articulated and deformable objects, and can cope with occlusion by modeling the object as a collection of object parts, together with the spatial distribution of these parts. Experimental results showed that the proposed algorithm outperforms the state-of-the-art in hand detection, and is competitive with the state-of-the-art when applied to general object detection tasks such as people detection and animal detection. Nevertheless, our method provides a significant decrease in computational complexity, resulting in a real-time solution.

In chapter 3, a novel sparse optical flow estimation method was proposed. Due to its sparsity, this method is extremely efficient, consuming only a fraction of the available CPU power. We proposed a technique to incorporate a regularization scheme into the matching process, resulting in a smooth and noise-free optical flow field estimate. Moreover, a method is introduced to allow for local regularization such that the algorithm can cope with motion discontinuities due to object boundaries.

Experiments showed that our solution both outperforms traditional point-matching algorithms such as SURF and traditional sparse optical flow methods such as the Lucas-Kanade algorithm. Our method is also competitive with the state-of-the-art in dense optical flow estimation, while being faster to calculate than simple sparse flow estimation techniques. We showed that incorporation of our optical flow method into a particle filter based object tracker greatly improves its robustness and accuracy.

In chapter 4 we discussed a particle filter based hand tracking approach. Several techniques were introduced to reduce the computational complexity of the system. Several types of color, texture and motion related information are combined by a sensor fusion approach, resulting in a robust, illumination independent tracking method.

The hand tracker was combined with the earlier discussed hand detector to allow for automatic initialization and error recovery. Moreover, a segmentation algorithm was developed and used to gather information that is used by a feedback loop, such that the tracking algorithm can automatically adapt to its environment.

Experimental results showed that our solution largely outperforms the state-of-the-art in object tracking, when applied to the task of hand tracking. Additionally, our method is faster than most of the state-of-the-art algorithms, and is unique due to its automatic initialization and error recovery capabilities.

In chapter 5, we introduced a method to automatically learn which objects exhibit temporally correlated motion patterns. This allows our algorithm to quickly adapt in dynamic environments, and to cope with occlusions, even when occluding objects have similar appearance as the tracked object.

The contextual learner was used in a partitioned sampling based particle framework to track the arm, along with the user's hand. Moreover, we introduced a method to incorporate contextual information into the proposal distribution of the particle filter itself, to improve tracking robustness against occlusion.

Extensive evaluation showed that our solution outperforms state-of-the-art in object tracking. Moreover, we showed that the proposed approach yields comparable results as pictorial structure based tracking and detection methods, while showing a much lower computational complexity.

In chapter 6 we discussed a novel interactive human-computer display. Multiple users can use the display which is completely transparent, such that they can maintain eye-contact when performing collaborative tasks.

Moreover, the display produces an image that appears to float in thin air, such that the virtual object seems to be part of the real world. This results in a truly immersive experience, allowing the user to touch and manipulate the floating object with his hands. The display was showcased at several national and international events and tradeshows, and the first steps towards a spin-off company were taken, based on this research.

2 Contributions

The research during this PhD resulted in a number of scientific contributions that are summed up in this section. In order to discuss these contributions, it is useful to revisit the research questions that were posed in section 1, and to formulate an answer on each of these questions.

2D Hand Tracking: Given its initial location, a hand can be tracked throughout a video sequence in real-time by means of particle filtering. Due to the difficulty of predicting the hand's motion, it is important to choose a proposal distribution that closely resembles the posterior distribution of interest. We showed how such proposal distribution can be obtained, by combining an optical flow field with color information, motion information and context modeling. We showed that this approach greatly outperforms traditional bootstrap filters, and allows for robust hand tracking in real-time throughout a monocular, and possibly low-resolution video sequence. Moreover, we showed that this approach can cope with sudden illumination changes in the scene, quickly changing backgrounds, moving cameras, and unconstrained hand poses.

Hand segmentation: Although hand segmentation in video can be challenging due to motion blur in case of fast motion, we showed that it is possible to at least obtain a rough segmentation of the hand in real-time, even in cases where the edges of the hand are blurred. More importantly, we introduced a method that uses the resulting segmentation mask to automatically adapt the likelihood models of the particle filter, thereby increasing its robustness to illumination changes.

Color modeling: Chapter 2 showed that skin color alone does not suffice as a cue for robust hand detection. Instead, texture based cues and context modeling are needed to achieve robust hand detection. On the other hand, in chapter 4, we showed that skin color is an important feature for hand tracking, when

temporal information is available. We introduced a method to dynamically adapt the skin color model such that it is invariant to illumination changes and ethnicity of the human subject. Finally, we combined skin color with several other types of information such as motion detection and optical flow in order to overcome ambiguities when skin-like objects appear in the scene.

2D Hand Detection: In chapter 2, we discussed that robust hand detection is not feasible by means of traditional template matching based detection techniques. However, we introduced a parts-based method that is able to detect hand parts, such as fingers, and that models their spatial distribution in order to detect the hand's position. We showed that it is possible to model this distribution in a rotation and scale invariant manner, and applied the resulting detector in our hand tracking framework, where it takes care of error detection and recovery.

Closed-loop system: All components of our complete tracking and detection system share information and allow for the internal models to be adapted at run-time. Color models of the particle filter are updated based on the segmentation mask and the Hough voting map. Prior probability distributions of the random Hough forest are adapted automatically based on the hand tracking results, such that mistakes that were made in the past are unlikely to be made again in future video frames. Context, defined as objects whose motion patterns appear to be correlated with those of the hand, is learned automatically based on the hand bounding box, obtained from the particle filter. The context models are updated continuously and are able to adapt when the context changes. Optical flow is used in both particle filter tracking and context learning.

Unsupervised context learning: We showed that it is possible to automatically train a statistical model of the context, surrounding the tracked hand. Our solution was shown to be able to learn which objects exhibit correlated motion with the hand itself. These object can either be object such as a magazine or umbrella the hand interacts with, or can be objects that are attached to the hand, such as the arm. We showed that arm tracking becomes feasible even if the arm's appearance is unknown beforehand, for instance do to the color and texture of the sleeves. Finally, our context learning method was shown to increase the tracker's robustness to occlusion.

Real-time system: Each of the chapters in this dissertation focussed on real-time performance. We discussed and evaluated the computational complexity of all modules in detail, and introduced several design choices to achieve real-time performance. Some of these design choices are on the algorithmic level, e.g. by using integral histograms and integral images, whereas others

are on the implementation level, e.g. by using SSE2 instructions, parallelization, and GPU implementation by means of shaders.

Floating display: In chapter 6 we showed that it is possible to build a holographic display at relatively low cost. Although the quality of the resulting image is not perfect, due to the need of anisotropic interpolation, the proof-of-concept displays we built were much appreciated by the public at several demonstrations and trade shows. The displays can be controlled using hand gestures, based on the algorithms introduced in this dissertation.

The work on real-time object detection has led to three peer-reviewed publications at international conferences [Spruyt 12, Spruyt 13a, Bellekens 14].

The work on real-time sparse optical flow regularization resulted in three peer-reviewed international conference publications [Spruyt 13b, Van den Berghe 11b, Van den Berghe 11a].

The research on object segmentation has led to two peer-reviewed international conference publications [Spruyt 10a, Creemers 11].

The research work on object tracking resulted in one published international journal paper [Spruyt 14], one journal publication that is currently under review, one submitted journal paper, four peer-reviewed international conference publications [Spruyt 12, Spruyt 13a, Heyman 12, Heyman 11] and one peer-reviewed national conference publication [Spruyt 10b].

The work on the development of a transparent floating imaging display has led to a commercial license agreement between an interested party, i.e. spin-off company in formation BI3D, and both Ghent University and the University of Antwerp, concerning part of this PhD research. Furthermore, two types of hardware prototypes were developed and showcased on several national and international events.

3 Future Work

This section lists some important areas with potential of improvement in future research:

The region of interest detector used in chapter 2 is based on the entropy of the gray-scale histogram of each region of interest, and a measure of self-similarity at different scales. Therefore, this method does not incorporate any information about the temporal stability of the region if the detector is used for hand detection in video. Instead of simply detecting regions that are of interest from a spatial point of view, one could detect regions that are also of interest from a temporal point of view. Several spatio-temporal feature detection methods have been described in literature and could be used to improve the detection performance when applied to video.

The probabilistic Random Hough Forest voting mechanism of chapter 2 can be extended to a 3D Hough parameter space, such that it can be used to detect objects directly in a point cloud obtained by a depth camera. This would greatly increase the robustness of the algorithm, since background objects can often be discarded quickly based on their 3D shape.

Even human observers often have difficulties in detecting hands if no contextual information such as the wrist or arm is visible. An interesting extension of the detector proposed in chapter 2 would therefore allow the incorporation of contextual information. The detector could be tailored specifically to articulated objects, such that detection of the arm would improve detection of the hand, and vice versa.

In chapter 3, a simple and fast corner detector is used for point matching. As discussed above, spatio-temporal feature detection could be used in video sequences, and would potentially result in a higher repeatability of the feature point detector.

In chapter 4, a depth map can be combined easily with the probability images used by the particle filter tracker. By incorporating depth information, background objects can be discarded quickly, which increases the robustness of the tracking method. Furthermore, in the current implementation of the tracking algorithm, a custom proposal distribution is designed to estimate the position of the hands, whereas a standard bootstrap filtering approach is used to estimate the hand size. However, it would be possible to design a proposal distribution, which incorporates a segmentation step, such that the size of the hands can be estimated more accurately using less particles.

Based on the bounding box estimate, obtained by the particle filter tracker of chapter 4, a rough segmentation of the hand is obtained. This segmentation is then used to improve the detection and tracking models adaptively. However, the segmented contour could also be used directly in the particle filter framework to improve the obtained estimate instead of only using it in a post-processing step. Finally, The appearance of the hands is currently described using general color and texture based histograms. However, it would be interesting to use a P-N learning based approach to gather image patches of both the hands and the background over time. These patches then represent a codebook of the object of interest, and could be used to increase the robustness and adaptivity of both the tracking and the detection methods.

Chapter 5 introduced a method to improve hand tracking robustness by incorporation local context into the proposal distribution of the particle filter. We applied this framework to arm tracking. This concept could easily be extended to, or combined with, a fully articulated upper body tracker to further improve robustness of the arm location estimates in case of occlusion with other body parts.

In chapter 6 a novel type of transparent computer display was described. The main shortcoming of the proposed 360 degrees conic display is that the floating

3D appears flat to the human eye. An obvious extension would therefore be the use of a fly-eye lens sheet to introduce a true auto-stereoscopic effect.

Bibliography

[Alahi 12] A. Alahi, R. Ortiz & P. Vandergheynst. *FREAK: Fast Retina Keypoint*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 510–517, 2012.

[Alparone 99] L. Alparone, M. Barni, F. Bartolini & R. Caldelli. *Regularization of optic flow estimates by means of weighted vector median filtering*. IEEE Transactions on Image Processing, vol. 8, no. 10, pages 1462–1467, October 1999.

[Alvarez 99] L. Alvarez, J. Sanchez & J. Weickert. *A Scale-Space Approach to Nonlocal Optical Flow Calculations*. In Mads Nielsen, Peter Johansen, OleFogh Olsen & Joachim Weickert, editeurs, Scale-Space Theories in Computer Vision, volume 1682 of *Lecture Notes in Computer Science*, pages 235–246. Springer Berlin Heidelberg, 1999.

[Amit 06] A. Amit, E. Rivlin & I. Shimshoni. *Robust Fragments-based Tracking using the Integral Histogram*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 798–805. IEEE Computer Society, 2006.

[Asaari 10] M. Asaari & S. Suandi. *Hand gesture tracking system using Adaptive Kalman Filter*. In International Conference on Intelligent Systems Design and Applications, pages 166–171, 2010.

[Aubert 99] G. Aubert, R. Deriche & P. Kornprobst. *Computing Optical Flow via Variational Techniques*. SIAM Journal on Applied Mathematics, vol. 60, pages 156–182, 1999.

[Baker 11] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M. Black & R. Szeliski. *A Database and Evaluation Methodology for Optical Flow*. International Journal of Computer Vision, vol. 92, no. 1, pages 1–31, 2011.

[Bay 06] H. Bay, T. Tuytelaars & L. Van Gool. *Surf: Speeded up robust features*. In European Conference on Computer Vision, pages 404–417, 2006.

[Belgacem 12] S. Belgacem, C. Chatelain, A. Ben-Hamadou & T. Paquet. *Hand tracking using optical-flow embedded particle filter in sign language scenes*. In IEEE International Conference on Computer Vision, ICCVG'12, pages 288–295, Berlin, Heidelberg, 2012. Springer-Verlag.

[Belikov 97] V. V. Belikov, V. D. Ivanov, V. K. Kontorovich, S. A. Korytnik & A. Y. Semenov. *The non-Sibsonian interpolation: A new method of interpolation of the values of a function on an arbitrary set of points*. Computational mathematics and mathematical physics, vol. 37, pages 9–15, 1997.

[Bellekens 14] Ben Bellekens, Vincent Spruyt & Rafael Berkvens Maarten Weyn. *A survey of rigid 3D pointcloud registration algorithms*. In Fourth International Conference on Ambient Computing, Applications, Services and Technologies, Proceedings, pages 8–13. IARA, 2014.

[Bühlmann 02] Bühlmann & Y. Bin. *Analyzing Bagging*. Annals of Statistics, vol. 30, pages 927–961, 2002.

[Biau 12] G. Biau. *Analysis of a Random Forests Model*. Journal of machine learning research, vol. 13, no. 1, pages 1063–1095, April 2012.

[Bilal 12] S. Bilal, R. Akmeliawati, M.. Salami & A. Shafie. *Dynamic approach for real-time skin detection*. Journal of Real-Time Image Processing, pages 1–15, 2012.

[Borenstein 02] E. Borenstein & S. Ullman. *Class-Specific, Top-Down Segmentation*. In European Conference on Computer Vision, ECCV '02, pages 109–124, London, UK, UK, 2002. Springer-Verlag.

[Bosch 07] A. Bosch, A. Zisserman & X. Muoz. *Image Classification using Random Forests and Ferns*. In IEEE International Conference on Computer Vision, pages 1–8, 2007.

[Breiman 01] L. Breiman. *Random Forests*. Journal of Machine Learning, vol. 45, no. 1, pages 5–32, October 2001.

[Bretzner 02] L. Bretzner, I. Laptev & T. Lindeberg. *Hand Gesture Recognition using Multi-Scale Colour Features, Hierarchical Models and Particle Filtering*. In IEEE International Conference on Automatic Face and Gesture Recognition, FGR '02, page 423, Washington, DC, USA, 2002. IEEE Computer Society.

[Brox 04] T. Brox, A. Bruhn, N. Papenberg & J. Weickert. *High accuracy optical flow estimation based on a theory for warping*. In European Conference on Computer Vision, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer, May 2004.

[Brox 09] T. Brox, C. Bregler & J. Malik. *Large displacement optical flow*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 41–48, June 2009.

[Buehler 08] P. Buehler, M. Everingham, D. Huttenlocher & A. Zisserman. *Long Term Arm and Hand Tracking for Continuous Sign Language TV Broadcasts*. In British Machine Vision Conference, pages 110.1–110.10. BMVA Press, 2008.

[Buehler 11] P. Buehler, M. Everingham, D. Huttenlocher & A. Zisserman. *Upper Body Detection and Tracking in Extended Signing Sequences*. International Journal of Computer Vision, vol. 95, no. 2, pages 180–197, 2011.

[Burbridge 08] C. Burbridge, U. Nehmzow & J. Condell. *Omnidirectional Projections with a Cone Mirror and Single Mirror Stereo*. In Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras, Marseille, France, 2008. Rahul Swaminathan and Vincenzo Caglioti and Antonis Argyros.

[Caifeng 04] S. Caifeng, W. Yucheng, T. Tieniu & F. Ojardias. *Real time hand tracking by combining particle filtering and mean shift*. In IEEE International Conference on Automatic Face and Gesture Recognition, pages 669–674, 2004.

[Cerman 09] L. Cerman, J. Matas & V. Hlaváč. *Sputnik Tracker: Having a Companion Improves Robustness of the Tracker*. In Scandinavian Conference on Image Analysis, SCIA '09, pages 291–300, Berlin, Heidelberg, 2009. Springer-Verlag.

[Chollet 13] Mathieu Chollet, Magalie Ochs & Catherine Pelachaud. *A multimodal corpus for the study of non-verbal behavior expressing interpersonal stances*. In Intelligent Virtual Agents Workshop on Multimodal Corpora, 2013.

[Cooper 13] Todd Cooper, Akira Tsukada, Yoshinari Naruse & Akifumi Yamaguchi. *From Interviews to Total Communication: Virtualizing the English Foreign Language Classroom*. In Ron McBride & Michael Searson, editeurs, Proceedings of Society for Information Technology & Teacher Education International Conference 2013, pages 336–342, New Orleans, Louisiana, United States, March 2013. Association for the Advancement of Computing in Education (AACE).

[Cornelius 13] C.J. Cornelius, Mai Anh Nguyen, C.C. Hayes & R. Makena. *Supporting Virtual Collaboration in Spatial Design Tasks: Are Surrogate or Natural Gestures More Effective?* IEEE Transactions on Human-Machine Systems, vol. 43, no. 1, pages 92–101, Jan 2013.

[Creemers 11] C. Creemers, K. Guerti, S. Geerts, K. Van Cotthem, A. Ledda & V. Spruyt. *HEp-2 Cell pattern segmentation for the support of autoimmune disease diagnosis*. In International Symposium on Applied Sciences in Biomedical and Communication Technologies, 2011.

[Dalal 05] N. Dalal & B. Triggs. *Histograms of Oriented Gradients for Human Detection*. In IEEE Conference on Computer Vision and Pattern Recognition, volume 1, pages 886–893, 2005.

[Dawod 10] A.Y. Dawod, J. Abdullah & M.J. Alam. *Adaptive skin color model for hand segmentation*. In International Conference on Computer Applications and Industrial Electronics, pages 486–489, 2010.

[Deqing 12] S. Deqing, E.B. Sudderth & M.J. Black. *Layered segmentation and optical flow estimation over time*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1768–1775, June 2012.

[Donoser 08] M. Donoser & H. Bischof. *Real time appearance based hand tracking*. In International Conference on Pattern Recognition, pages 1–4, 2008.

[Durrant-Whyte 01] H. Durrant-Whyte. *Multi Sensor Data Fusion*. Technical report, Australian Centre for Field Robotics, The University of Sydney NSW, 2001.

[Ebner 07] M. Ebner. Color constancy. Wiley, 2007.

[Eichner 12] M. Eichner, M. Marin-Jimenez, A. Zisserman & V. Ferrari. *2d articulated human pose estimation and retrieval in (almost) unconstrained still images*. International Journal of Computer Vision, vol. 99, no. 2, pages 190–214, 2012.

[ElKoura 03] G. ElKoura & K. Singh. *Handrix: animating the human hand*. In ACM SIGGRAPH/Eurographics symposium on Computer animation, SCA '03, pages 110–119, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Everingham 10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn & A. Zisserman. *The Pascal Visual Object Classes (VOC) Challenge*. International Journal of Computer Vision, vol. 88, no. 2, pages 303–338, June 2010.

[Exner 10] D. Exner, E. Bruns, D. Kurz, A. Grundhofer & O. Bimber. *Fast and robust CAMShift tracking*. In IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 9–16, 2010.

[Fisher 03] R. Fisher. *General-purpose SIMD within a register: Parallel processing on consumer microprocessors*. PhD thesis, Purdue University, 2003.

[Francis 03] J.J. Francis & G. de Jager. *The Bilateral Median Filter*. In Symposium of the Pattern Recognition Association of South Africa, 2003.

[Friedman 97] N. Friedman, D. Geiger & M. Goldszmidt. *Bayesian Network Classifiers*. Journal of Machine Learning, vol. 29, no. 2-3, pages 131–163, 1997.

[Gall 09] J. Gall & V. Lempitsky. *Class-specific Hough forests for object detection*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1022–1029, 2009.

[Giorgio 03] V. Giorgio. *Ensemble Methods Based on Bias-Variance Analysis*. PhD thesis, Dipartimento di Informatica e Scienze, 2003.

[Gomez 02] G. Gomez. *On selecting colour components for skin detection*. In International Conference on Pattern Recognition, volume 2, pages 961–964, 2002.

[Goncalves 95] L. Goncalves, E. Di Bernardo, E. Ursella & P. Perona. *Monocular tracking of the human arm in 3D*. In IEEE International Conference on Computer Vision, pages 764–770, Jun 1995.

[Grabner 10] H. Grabner, J. Matas, L. Van Gool & P. Cattin. *Tracking the invisible: Learning where the object might be*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1285–1292, June 2010.

[Grygorash 06] O. Grygorash, Y. Zhou & Z. Jorgensen. *Minimum Spanning Tree Based Clustering Algorithms*. IEEE International Conference on Tools with Artificial Intelligence, vol. 0, pages 73–81, 2006.

[Harris 88] C. Harris & M. Stephens. *A combined corner and edge detector*. In Alvey Vision Conference, pages 147–151, 1988.

[Heikkilä 09] M. Heikkilä, M. Pietikäinen & C. Schmid. *Description of interest regions with local binary patterns*. Journal of Pattern Recognition, vol. 42, no. 3, pages 425–436, March 2009.

[Heyman 11] T. Heyman, V. Spruyt & A. Ledda. *3D Face tracking and gaze estimation using a monocular camera*. In International Conference on Positioning and Context-Awareness, pages 23–28, 2011.

[Heyman 12] T. Heyman, V. Spruyt, S. Grünwedel, A. Ledda & W. Philips. *A canonical correlation analysis based motion model for probabilistic visual tracking*. In IEEE Visual Communications and Image Processing, pages HO2–25. Proceedings of Visual Communications and Image Processing, 2012.

[Horn 81] B. Horn & B. Schunck. *Determining Optical Flow*. Artificial Intelligence, vol. 17, pages 185–203, 1981.

[Hoshi 09] T. Hoshi, M. Takahashi, K. Nakatsuma & H. Shinoda. *Touchable Holography*. In ACM SIGGRAPH 2009 Emerging Technologies, SIGGRAPH '09, pages 23:1–23:1, New York, NY, USA, 2009. ACM.

[Huang 12] T.F. Huang, P. Chao & Y. Kao. *Tracking, recognition, and distance detection of hand gestures for a 3-D interactive display*. Journal of the Society for Information Display, vol. 20, no. 4, pages 180–196, 2012.

[Hunt 00] J. L. Hunt, B. G. Nickel & C. Gigault. *Anamorphic images*. American Journal of Physics, vol. 68, pages 232–237, March 2000.

[Isard 98] M. Isard & A. Blake. *ICONDENSATION: Unifying Low-Level and High-Level Tracking in a Stochastic Framework*. In European Conference on Computer Vision, ECCV '98, pages 893–908, London, UK, UK, 1998. Springer-Verlag.

[Jacob 13] Mithun George Jacob, Juan Pablo Wachs & Rebecca A Packer. *Hand-gesture-based sterile interface for the operating room using contextual cues for the navigation of radiological images*. Journal of the American Medical Informatics Association, vol. 20, no. e1, pages e183–e186, 2013.

[Jones 02] M. Jones & J. Rehg. *Statistical Color Models with Application to Skin Detection*. International Journal of Computer Vision, vol. 46, no. 1, pages 81–96, 2002.

[Joohwan 09] K. Joohwan, M. Sung-Wook & L. Byoungho. *Viewing window expansion of integral floating display*. Applied Optics, vol. 48, no. 5, pages 862–867, Feb 2009.

[Julier 02] S.J. Julier. *The scaled unscented transformation*. In American Control Conference, volume 6, pages 4555–4559, 2002.

[Jun 12] W. Jun, R. Qiuqi, A. Gaoyun & L. Wei. *Hand tracking and segmentation via graph cuts and dynamic model in sign language videos*. In International Conference on Signal Processing, volume 2, pages 1135–1138, 2012.

[Junseok 09] K. Junseok & Mu. Kyoung. *Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive Basin Hopping Monte Carlo sampling*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1208–1215, 2009.

[Just 06] A. Just, Y. Rodriguez & S. Marcel. *Hand Posture Classification and Recognition using the Modified Census Transform*. In International Conference on Automatic Face and Gesture Recognition, pages 351–356, 2006.

[Kadir 01] T. Kadir & M. Brady. *Saliency, Scale and Image Description*. International journal of computer vision, vol. 45, no. 2, pages 83–105, November 2001.

[Kakeya 07] H. Kakeya. *MOEVision: simple multiview display with clear floating image*. In Stereoscopic Displays and Virtual Reality Systems XIV, volume 6490, pages 64900J–64900J–8, 2007.

[Kalal 10] Z. Kalal, J. Matas & K. Mikolajczyk. *P-N learning: Bootstrapping binary classifiers by structural constraints*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 49–56, June 2010.

[Kalal 12] Z. Kalal, K. Mikolajczyk & J. Matas. *Tracking-Learning-Detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 7, pages 1409–1422, July 2012.

[Karlinsky 10] L. Karlinsky, M. Dinerstein, D. Harari & S. Ullman. *The chains model for detecting parts by their context*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 25–32, 2010.

[Kim 13] T. Kim, H. Lee & K. Lee. *Optical Flow via Locally Adaptive Fusion of Complementary Data Costs*. International Computer Vision Conference, 2013.

[Kimura 06] H. Kimura, T. Uchiyama & H. Yoshikawa. *Laser Produced 3D Display in the Air*. In ACM SIGGRAPH 2006 Emerging Technologies, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[Kodama 10] T. Kodama, T. Yamaguchi & H. Harada. *A method of object tracking based on particle filter and optical flow to avoid degeneration problem*. In SICE Annual Conference, pages 1529–1533, Aug 2010.

[Kölsch 04a] M. Kölsch & M. Turk. *Fast 2D Hand Tracking with Flocks of Features and Multi-Cue Integration*. In IEEE Conference on Computer Vision and Pattern Recognition Workshop, pages 158–158, 2004.

[Kölsch 04b] M. Kölsch & M. Turk. *Robust hand detection*. In IEEE International Conference on Automatic Face and Gesture Recognition, pages 614–619, 2004.

[Kölsch 05] M. Kölsch & M. Turk. *Flocks of Features for Tracking Articulated Objects*. In Real-Time Vision for Human-Computer Interaction, pages 67–83. Springer US, 2005.

[Kölsch 10] M. Kölsch. *An Appearance-Based Prior for Hand Tracking*. In Jacques Blanc-Talon, Don Bone, Wilfried Philips, Dan Popescu & Paul Scheunders, editeurs, Advanced Concepts for Intelligent Vision Systems, volume 6475 of *Lecture Notes in Computer Science*, pages 292–303. Springer Berlin Heidelberg, 2010.

[Kraut 03] R.. Kraut, S. Fussell & J. Siegel. *Visual Information As a Conversational Resource in Collaborative Physical Tasks*. Human-Computer Interaction, vol. 18, no. 1, pages 13–49, June 2003.

[Kumar 09] M.P. Kumar, A. Zisserman & P. H S Torr. *Efficient discriminative learning of parts-based models*. In IEEE International Conference on Computer Vision, pages 552–559, Sept 2009.

[Kwak 00] Y. Kwak & L. MacDonald. *Characterisation of a desktop LCD projector*. Displays, vol. 21, no. 5, pages 179–194, 2000.

[Kyung 12] S. Kyung, B. Gi & L. Sangwon. *Fatigue problems in remote pointing and the use of an upper-arm support*. International Journal of Industrial Ergonomics, vol. 42, no. 3, pages 293–303, 2012.

[LeapMotion 13] LeapMotion. *Leap Motion Controller*. San Francisco, USA, 2013.

[Leibe 08] B. Leibe, A. Leonardis & B. Schiele. *Robust Object Detection with Interleaved Categorization and Segmentation*. International journal of computer vision, vol. 77, no. 1–3, pages 259–289, May 2008.

[Lewis 95] J. Lewis. *Fast template matching*. In Vision interface, volume 95, pages 15–19, 1995.

[Liang 12] H. Liang, J. Yuan & D. Thalmann. *Hand pose estimation by combining fingertip tracking and articulated ICP*. In ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry, VRCAI '12, pages 87–90, New York, NY, USA, 2012. ACM.

[Liu 12] H. Liu, W. Cui & R. Ding. *Robust Hand Tracking with Hough Forest and Multi-cue Flocks of Features*. In Advances in Visual Computing, volume 7432 of *Lecture Notes in Computer Science*, pages 458–467. Springer Berlin Heidelberg, 2012.

[Lowe 04] D. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International journal of computer vision, vol. 60, no. 2, pages 91–110, November 2004.

[Lucas 81] B. D. Lucas & T. Kanade. *An Iterative Image Registration Technique with an Application to Stereo Vision*. In International Joint Conference on Artificial Intelligence, volume 2 of *IJCAI'81*, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[Lucena 03] M.J. Lucena, J.M. Fuertes, N. P. de la Blanca & A. Garrido. *Using optical flow as evidence for probabilistic tracking*. In Scandinavian conference on Image analysis, SCIA'03, pages 1044–1059, 2003.

[MacCormick 00] J. MacCormick & M. Isard. *Partitioned Sampling, Articulated Objects, and Interface-Quality Hand Tracking*. In European Conference on Computer Vision, ECCV '00, pages 3–19, London, UK, 2000. Springer-Verlag.

[Matas 02] J. Matas, O. Chum, M. Urban & T. Pajdla. *Robust Wide Baseline Stereo from Maximally Stable Extremal Regions*. In British Machine Vision Conference, pages 36.1–36.10. BMVA Press, 2002.

[Melax 13] S. Melax, L. Keselman & S. Orsten. *Dynamics based 3D skeletal hand tracking*. In ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '13, pages 184–184, New York, NY, USA, 2013. ACM.

[Metaxas 06] D. Metaxas, G. Tsechpenakis, Z. Li, Y. Huang & A. Kanaujia. *Dynamically Adaptive Tracking of Gestures and Facial Expressions*. In International Conference on Computational Science, ICCS'06, pages 554–561, Berlin, Heidelberg, 2006. Springer-Verlag.

[Microsoft 10] Microsoft. *Kinect for XBox 360*. Redmond WA, 2010.

[Min 01] S.-W. Min, S. Jung, J.-H. Park & B. Lee. *Three-dimensional display system based on computer-generated integral photography*. In A. J. Woods, M. T. Bolas, J. O. Merritt & S. A. Benton, editeurs, Stereoscopic Displays and Virtual Reality Systems VIII, volume 4297 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 187–195, June 2001.

[Min 05] S.-W. Min, M. Hahn, J. Kim & B. Lee. *Three-dimensional electro-floating display system using an integral imaging method*. Optics Express, vol. 13, no. 12, pages 4358–4369, June 2005.

[Mitra 07] S. Mitra & T. Acharya. *Gesture Recognition: A Survey*. Transactions on Systems, Man and Cybernetics, vol. 37, no. 3, pages 311–324, May 2007.

[Mittal 11] A. Mittal, A. Zisserman & P. Torr. *Hand detection using multiple proposals*. In British Machine Vision Conference, pages 75.1–75.11. BMVA Press, 2011.

[Mosabbeb 07] E. Mosabbeb, M. Sadeghi & M. Fathy. *A New Approach for Vehicle Detection in Congested Traffic Scenes Based on Strong Shadow Segmentation*. In International Conference on Advances in Visual Computing, ISVC'07, pages 427–436, Berlin, Heidelberg, 2007. Springer-Verlag.

[Neal 01] R. Neal. *Annealed importance sampling*. Statistics and Computing, vol. 11, no. 2, pages 125–139, 2001.

[Newcombe 11] R. Newcombe, S. Lovegrove & A. Davison. *DTAM: Dense tracking and mapping in real-time*. In IEEE International Conference on Computer Vision, pages 2320–2327. IEEE, 2011.

[Niceron 52] J.F. Niceron. La perspective curieuse. Chez la veufue F. Langlois, dit Chartres, 1652.

[Oikonomidis 11] L. Oikonomidis, N. Kyriazis & A. Argyros. *Efficient model-based 3D tracking of hand articulations using Kinect*. In British Machine Vision Conference, pages 101.1–101.11. BMVA Press, 2011.

[Ojala 96] T. Ojala. *A comparative study of texture measures with classification based on featured distributions*. Journal of Pattern Recognition, vol. 29, no. 1, pages 51–59, 1996.

[Ojala 02] T. Ojala, M. Pietikäinen & T. Mäenpää. *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pages 971–987, 2002.

[Ong 04] E.J. Ong & R. Bowden. *A boosted classifier tree for hand shape detection*. In IEEE International Conference on Automatic Face and Gesture Recognition, pages 889–894, 2004.

[Papenberg 06] N. Papenberg, A. Bruhn, T. Brox, S. Didas & J. Weickert. *Highly accurate optic flow computation with theoretically justified warping*. Internaltional Journal of Computer Vision, vol. 67, pages 141–158, 2006.

[Pfister 12] T. Pfister, J. Charles, M. Everingham & A. Zisserman. *Automatic and Efficient Long Term Arm and Hand Tracking for Continuous Sign Language TV Broadcasts*. In British Machine Vision Conference, pages 4.1–4.11. BMVA Press, 2012.

[Pham 09] T.B. Pham, T.B. Nguyen & D. Tu. *A New Approach to Hand Tracking and Gesture Recognition by a New Feature Type and HMM*. In International Conference on Fuzzy Systems and Knowledge Discovery, volume 4, pages 3–6, 2009.

[Phung 05] S.L. Phung, A. Bouzerdoum & D. Chai. *Skin segmentation using color pixel classification: analysis and comparison*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 1, pages 148–154, 2005.

[Polyanskiy 14] M.N. Polyanskiy. *Refractive index database*. http://refractiveindex.info, 2014.

[Price 72] G. Price. *Extension of covariance selection mathematics*. Annals of Human Genetics, vol. 35, no. 4, pages 485–490, 1972.

[Quinlan 86] J. R. Quinlan. *Induction of Decision Trees*. Journal of machine learning, vol. 1, no. 1, pages 81–106, March 1986.

[Quinlan 93] J. R. Quinlan. C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Ren 07] H. Ren & S.-T. Wu. *Variable-focus liquid lens*. Optics Express, vol. 15, pages 5931–5936, May 2007.

[Rizzi 02] A. Rizzi, C. Gatta & D. Marini. *Color correction between gray world and white patch*. In Electronic Imaging 2002, pages 367–375. International Society for Optics and Photonics, 2002.

[Robert 09] Y. W. Robert & P. Jovan. *Real-time hand-tracking with a color glove*. ACM Transactions on graphics, vol. 28, no. 3, pages 63:1–63:8, July 2009.

[Rosten 10] E. Rosten, R. Porter & T. Drummond. *FASTER and better: A machine learning approach to corner detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, pages 105–119, 2010.

[Rother 04] C. Rother, V. Kolmogorov & A. Blake. *"GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts*. ACM Trans. Graph., vol. 23, no. 3, pages 309–314, August 2004.

[Saffari 09] A. Saffari, C. Leistner, J. Santner, M. Godec & H. Bischof. *On-line Random Forests*. In IEEE International Conference on Computer Vision Workshops, pages 1393–1400, Sept 2009.

[Schikora 11] M. Schikora, W. Koch & D. Cremers. *Multi-object tracking via high accuracy optical flowand finite set statistics*. In IEEE International Conference on Acoustics, Speech and Signal Processing, pages 1409–1412, May 2011.

[Secord 02] J. A. Secord. *Quick and Magical Shaper of Science*. Science, vol. 297, no. 5587, pages 1648–1649, 2002.

[Shan 07] C. Shan, T. Tan & Y. Wei. *Real-time hand tracking using a mean shift embedded particle filter*. Pattern Recognition, vol. 40, no. 7, pages 1958–1970, July 2007.

[Sharma 02] G. Sharma. *LCDs versus CRTs-color-calibration and gamut considerations*. Proceedings of the IEEE, vol. 90, no. 4, pages 605–622, April 2002.

[Shi 94] J. Shi & C. Tomasi. *Good features to track*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 593–600, 1994.

[Shotton 08] J. Shotton, M. Johnson & R. Cipolla. *Semantic texton forests for image categorization and segmentation*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, 2008.

[Sidharta 06] R. Sidharta, A. Hiyama, T. Tanikawa & M. Hirose. *The Development of Multi-Depth Pepper's Ghost Display for Mixed Reality System*. In International Conference on Artificial Reality and Telexistence, pages 115–118, November 2006.

[Sidharta 07] R. Sidharta, A. Hiyama, T. Tanikawa & M. Hirose. *Volumetric Display for Augmented Reality*. In International Conference on Artificial Reality and Telexistence, pages 55–62, November 2007.

[Sinclair 10] D. A. Sinclair. *S-hull: a fast radial sweep-hull routine for Delaunay triangulation*, July 2010.

[SoftKinetic 14] SoftKinetic. *DepthSense 325*. Brussels, Belgium, 2014.

[Soriano 00] M. Soriano, B. Martinkauppi, S. Huovinen & M. Laaksonen. *Skin detection in video under changing illumination conditions*. In International Conference on Pattern Recognition, volume 1, pages 839–842, 2000.

[Spruyt 10a] V. Spruyt, A. Ledda & S. Geerts. *Real-time multi-colourspace hand segmentation*. In IEEE International Conference on Image Processing, pages 3117–3120, 2010.

[Spruyt 10b] V. Spruyt, A. Ledda & W. Philips. *Gesture based human-computer interface for 3D design*. In FEA PhD Symposium, Abstracts. Ghent University, Faculty of Engineering and Architecture, 2010.

[Spruyt 12] V. Spruyt, A. Ledda & W. Philips. *Real-time hand tracking by invariant hough forest detection*. In IEEE International Conference on Image Processing, pages 149–152, 2012.

[Spruyt 13a] V. Spruyt, A. Ledda & W. Philips. *Real-time, long-term hand tracking with unsupervised initialization*. In IEEE International Conference on Image Processing, pages 3730–3734, September 2013.

[Spruyt 13b] V. Spruyt, A. Ledda & W. Philips. *Sparse optical flow regularization for real-time visual tracking*. In IEEE International Conference on Multimedia and Expo, pages 1–6, July 2013.

[Spruyt 14] V. Spruyt, A. Ledda & W. Philips. *Robust Arm and Hand Tracking by Unsupervised Context Learning*. Sensors, vol. 14, no. 7, pages 12023–12058, 2014.

[Stefanov 05] N. Stefanov, A. Galata & R. Hubbold. *Real-time Hand Tracking With Variable-Length Markov Models of Behaviour*. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR '05, page 73, Washington, DC, USA, 2005. IEEE Computer Society.

[Stenger 06a] B. Stenger. *Template-Based hand pose recognition using multiple cues*. In Asian conference on Computer Vision, ACCV'06, pages 551–560, Berlin, Heidelberg, 2006. Springer-Verlag.

[Stenger 06b] B. Stenger, A. Thayananthan, P. H. S. Torr & R. Cipolla. *Model-based hand tracking using a hierarchical Bayesian filter*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 9, pages 1372–1384, 2006.

[Stern 06] A. Stern & B. Javidi. *Three-Dimensional Image Sensing, Visualization, and Processing Using Integral Imaging*. Proceedings of the IEEE, vol. 94, no. 3, pages 591–607, March 2006.

[Sundaram 10] N. Sundaram, T. Brox & K. Keutzer. *Dense point trajectories by GPU-accelerated large displacement optical flow*. In European Conference on Computer Vision, pages 438–451. Springer, 2010.

[Torralba 03] A. Torralba. *Contextual Priming for Object Detection*. International Journa on Computer Vision, vol. 53, no. 2, pages 169–191, July 2003.

[Van de Sande 10] K. Van de Sande, T. Gevers & C. Snoek. *Evaluating Color Descriptors for Object and Scene Recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pages 1582–1596, 2010.

[Van den Berghe 11a] S. Van den Berghe, M. Weyn, V. Spruyt & A. Ledda. *Combining wireless and visual tracking for an indoor environment*. In International Conference on Indoor Positioning and Indoor Navigation, page 4, 2011.

[Van den Berghe 11b] S. Van den Berghe, M. Weyn, V. Spruyt & A. Ledda. *Fusing camera and Wi-Fi sensors for opportunistic localization*. In International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, pages 169–174. 5th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2011.

[van der Merwe 01] R. van der Merwe, N. Freitas, A. Doucet & E. Wan. *The Unscented Particle Filter*. In Advances in Neural Information Processing Systems, November 2001.

[Vaudrey 10] T. Vaudrey, A. Wedel, C. Chen & R. Klette. *Improving Optical Flow Using Residual and Sobel Edge Images*. In Fay Huang & Reen-Cheng Wang, editeurs, Arts and Technology, volume 30 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 215–222. Springer Berlin Heidelberg, 2010.

[Viola 04] P. Viola & M. Jones. *Robust Real-time Object Detection*. International Journal of Computer Vision, vol. 57, pages 137–154, 2004.

[Wang 09] H. Wang, M. Muneeb Ullah, A. Klaser, I. Leptev & C. Schmid. *Evaluation of local spatio-temporal features for action recognition*. In British Machine Vision Conference, page 127, September 2009.

[Wedel 09] A. Wedel, T. Pock, C. Zach, H. Bischof & D. Cremers. *An Improved Algorithm for TV-L 1 Optical Flow*.

In Daniel Cremers, Bodo Rosenhahn, AlanL. Yuille & FrankR. Schmidt, editeurs, Statistical and Geometrical Approaches to Visual Motion Analysis, volume 5604 of *Lecture Notes in Computer Science*, pages 23–45. Springer Berlin Heidelberg, 2009.

[Wehenkel 91] L. Wehenkel & M. Pavella. *Decision trees and transient stability of electric power systems*. Journal of Automatica, vol. 27, no. 1, pages 115–134, January 1991.

[Weinzaepfel 13] P. Weinzaepfel, J. Revaud, Z. Harchaoui & C. Schmid. *DeepFlow: Large displacement optical flow with deep matching*. In International Conference on Computer Vision, Sydney, Australia, December 2013.

[Xu 08] L. Xu, J. Chen & J. Jia. *A Segmentation Based Variational Model for Accurate Optical Flow Estimation*. In European Conference on Computer Vision, pages 671–684. Springer, 2008.

[Xu 12] L. Xu, J. Jia & Y. Matsushita. *Motion Detail Preserving Optical Flow Estimation*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 9, pages 1744–1757, September 2012.

[Yao 10] A. Yao, D. Uebersax, J. Gall & L. Gool. *Tracking People in Broadcast Sports*. In Michael Goesele, Stefan Roth, Arjan Kuijper, Bernt Schiele & Konrad Schindler, editeurs, Journal of Pattern Recognition, volume 6376 of *Lecture Notes in Computer Science*, pages 151–161. Springer Berlin Heidelberg, 2010.

[Zhuoyuan 13] C. Zhuoyuan, J. Hailin, L. Zhe, S. Cohen & W. Ying. *Large Displacement Optical Flow from Nearest Neighbor Fields*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2443–2450, June 2013.

[Zitnick 05] C.L. Zitnick, N. Jojic & B.K. Sing. *Consistent segmentation for optical flow estimation*. In IEEE International Conference on Computer Vision, volume 2, pages 1308–1315, October 2005.

[Zivkovic 04] Z. Zivkovic. *Improved adaptive Gaussian mixture model for background subtraction*. In International Conference on Pattern Recognition, volume 2, pages 28–31, 2004.