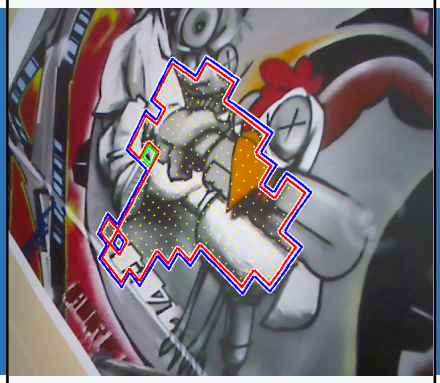
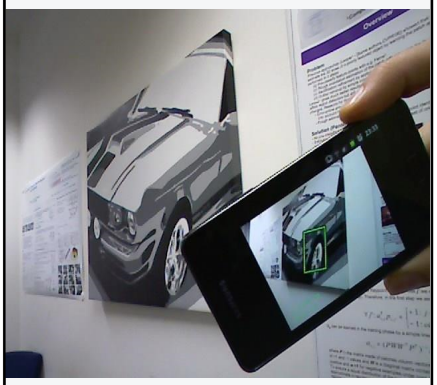
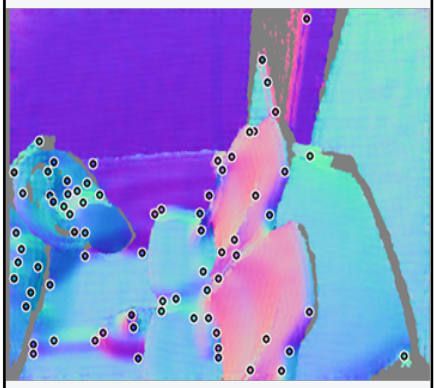
Computer Aided Medical Procedures
Prof. Dr. Nassir Navab







Dissertation

Learning-based Approaches for Template Tracking and Interest Point Detection

Stefan Johannes Josef Holzer

Fakultät für Informatik
Technische Universität München

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Computer Aided Medical Procedures & Augmented Reality / I16

Learning-based Approaches for Template Tracking and Interest Point Detection

Stefan Johannes Josef Holzer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. D. Cremers
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. N. Navab
2.	Prof. J. Matas, Ph.D., TU Prag / Tschechien
3.	Priv.-Doz. Dr. S. Ilic

Die Dissertation wurde am 16.06.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 16.12.2014 angenommen.

To Ruosi Li

Abstract

This thesis is concerned with problems in the field of learning-based template tracking and in the field of 3D data processing.

The goal of template tracking is to follow the position of an object or region within a sequence of images. It is an extensively studied field in computer vision with many possible applications in areas such as automobile, medicine, military or robotics. Since the seminal work of Lukas and Kanade [52], a lot of advances have been made in the field of template tracking, ranging from increased tracking and learning speed, over improved handling of motion, occlusions, noise, or lighting changes, to the ability to adjust to long-time changes of the real world object of interest. More importantly the works in learning-based template tracking approaches led to a significant increase in tracking speed and robustness. While these methods show superior characteristics in certain areas, they have problems in others. These problems include learning speed, flexibility, or occlusion handling. Therefore, the first part of this thesis focuses on improving these characteristics. Specifically, a new flexible method for learning and adapting linear predictors for template tracking is introduced. It demonstrates how it can be used to improve learning speed and handle occlusions. However, since learning time is still not optimal for online learning, two further approaches are presented which show how the learning time can be reduced by either reformulating the learning process or by applying a dimensionality reduction. These lead to a learning time that is two orders of magnitude faster than for the original approach. While both approaches show different characteristics with respect to robustness to object motion or low signal-to-noise ratio, a combined approach is also introduced which allows a trade-off between their characteristics.

Furthermore, the field of 3D data processing has recently obtained a boost in attention due to the introduction of low cost depth sensors such as the Microsoft Kinect. Therefore, the second part of the thesis focuses on fast methods for 3D data processing. First, a fast and robust normal estimation method is introduced which makes the computation of surface normals independent of the size of the neighborhood considered for its computation. Since normal estimation is often used as a pre-processing step for 3D processing algorithms, this helps to significantly improve many approaches in the field. Finally, a learning-based approach for interest point detection is presented. Current state-of-the-art methods are either robust but slow, or fast but significantly less robust. The introduced interest point detection method is based on decision trees, which not only mimics robust detection methods with significantly improved speed but also learns artificial interest point response maps which are optimized for specific characteristics.

Keywords:
Real-Time Tracking, Markerless Tracking, Optical Tracking, Depth Data, Surface Normal Estimation, Interest Point Detection

Zusammenfassung

Diese Doktorarbeit befasst sich mit Problemen im Bereich der Bildverfolgung mittels Lernverfahren sowie im Bereich der 3D Datenverarbeitung.

Das Ziel der Bildverfolgung ist es, die Position eines Objekt oder eine Region in einer Sequenz von Bildern zu verfolgen. Bildverfolgung ist ein eingehend erforschter Bereich der Bildverarbeitung und weißt eine Großzahl an verschiedenen Anwendungsmöglichkeiten in Bereichen wie zum Beispiel der Automobilindustrie, der Medizin, im militärischen Bereich oder in der Robotik auf. Seit der grundlegenden Arbeit von Lukas und Kanade [52] gab es eine Vielzahl an Fortschritten im Bereich der Bildverfolgung. Diese reichen von einer erhöhten Verfolgungs- und Erlernungsgeschwindigkeit, über verbesserten Umgang mit Bewegungen, Verdeckungen, Bildrauschen oder Beleuchtungsänderungen, bis zu der Fähigkeit sich and langsame Änderungen des zu verfolgenden Objektes anzupassen.

Besonders die Einführung von lern-basierten Verfahren zur Bildverfolgung führte zu einer drastischen Verbesserung der Verfolgungsgeschwindigkeit und -robustheit. Obwohl diese Verfahren überlegene Eigenschaften in bestimmten Bereichen aufweisen, zeigen sie allerdings auch Nachteile in anderen. Diese Nachteile beinhalten die Lerngeschwindigkeit, ihre Flexibilität oder das Behandeln von Verdeckungen. Deshalb konzentriert sich der erste Teilbereich dieser Arbeit auf die Verbesserung dieser Eigenschaften von lern-basierten Bildverfolgungsverfahren. Im Speziellen wird zunächst eine flexible Methode für das Lernen und Anpassen von Linearen Prädiktoren zur Bildverfolgung vorgestellt und gezeigt wie diese verwendet werden kann um die Lerngeschwindigkeit zu erhöhen und um sich an Verdeckungen anzupassen. Da die Lerngeschwindigkeit durch dieses Verfahren allerdings immer noch nicht ausreichend schnell ist werden zwei weitere Verfahren vorgestellt welche zeigen wie die benötigte Zeit zum Lernen entweder durch eine Umformulierung des Lernprozesses oder durch die Anwendung einer Dimensionalitätsverringerung weiter reduziert werden kann. Dies führt zu einer Lernzeit die um das hundertfache schneller ist als für das ursprüngliche Verfahren. Da beide Verfahren unterschiedliche Eigenschaften bezüglich Robustheit gegenüber Objektbewegungen oder einem geringen Signal-Rausch-Verhältnis aufweisen, wird zudem ein kombinierter Ansatz präsentiert, welcher eine Abstimmung zwischen einiger dieser Eigenschaften erlaubt.

Der Bereich der 3D Datenverarbeitung erfährt durch die Einführung günstiger Tiefensensoren, wie zum Beispiel dem Microsoft Kinect Sensor, seit kurzem einen Auftrieb an Aufmerksamkeit. Deshalb beschäftigt sich der zweite Teil dieser Arbeit mit schnellen Methoden zur 3D Datenverarbeitung. Zunächst wird ein schnelles und robustes Verfahren zum Berechnen von Oberflächennormalen vorgestellt, welches die Größe der für die Berechnung der Normalen verwendeten Nachbarschaftsregion automatisch bestimmt und deren Berechnungszeit unabhängig von der Größe dieser Nachbarschaftsregion ist. Da die Normalenberechnung häufig als Vorverarbeitungsschritt für 3D Verarbeitungsalgorithmen angewendet wird, hilft dies um viele Methoden in diesem Gebiet signifikant zu verbessern. Als letztes wird ein lern-basiertes Verfahren zur Erkennung von interessanten Punkten vorgestellt. Während die aktuellen Verfahren entweder robust aber langsam oder schnell aber deutlich weniger robust sind, erlaubt das vorgestellte, auf Entscheidungsbäumen

basierende Verfahren, robuste Erkennungsverfahren mit deutlich erhöhter Effizienz nachzuahmen. Zudem ermöglicht es das Erlernen von künstlichen Wahrscheinlichkeitsfelder für interessante Punkte, welche auf spezielle Eigenschaften optimiert sind.

Schlagwörter:
Echtzeit Tracking, Markerloses Tracking, Effizientes Lernen, Tiefendaten

Acknowledgements

I want to thank my supervisors Prof. Nassir Navab and PD. Slobodan Ilic for the fruitful discussions and their continuous support throughout my studies. Further, I want to thank Stefan Hinterstoißer, Jürgen Sotke, Pierre Georgel, and David Tan for the great time as well as the fruitful discussions while sharing an office. I also want to thank Ahmed Ahmadi, Selen Atasoy, Maximilian Baust, Vasileios Belagiannis, Ali Bigdelou, Tobias Blum, Richard Brosig, Cedric Cagniart, Victor Castaneda, Stefanie Demirci, Benoit Diotte, Danilo Djordjevic, Alexandru Duliu, Bertram Drost, Jose Gardiazabal, Ben Glocker, Martin Groher, Vladimir Haltakov, Hauke Heibel, Martina Hilla, Stuart Holdstock, Martin Horn, Chun-Hao Paul Huang, Athanasios Karamalis, Andreas Keil, Wadim Kehl, Tassilo Klein, Silvan Kraft, Oliver Kutter, Joe Lallemand, Tobias Lasser, Bastian Lieberknecht, Diana Mateus, Olivier Pauly, Kristof Ralovich, Tobias Reichl, Pierre Schroeder, Loren Schwarz, Jakob Vogel, Christian Wachinger, Mehmet Yigitsoy, Darko Zikic, as well as all my other collegues at the chair of Computer Aided Medical Procedures for the great time and discussions.

For the support and great time I had during my stays at Willow Garage I want to thank Radu Rusu, Steve Cousins, Gary Bradski, Kurt Konolige, Suat Gedikli, Ioan Sucan, Stefan Leutenegger, Jonathan Bohren, Lorenz Mösenlechner, Hauke Strasdat, Michael Dixon, Dirk Holz, Caroline Pantufaro, Vincent Rabaud, Matthew Robards, Mac Mason, Bastian Steder, Aaron Blasdel, Tobias Kunz, Alex Trevor, Daniel Hennes, and Troy Straszheim.

For my time at the Imperial College London I want to thank Prof. Andrew Davison, Hauke Strasdat, Richard Newcombe, Adrien Angeli, Ankur Handa, Steven Lovegrove, Gerardo Carrera, Margarita Chli, and Klaus Strobl.

I want to thank Jamie Shotton, Pushmeet Kohli, Indy, Nevena Lazic, Olga Nikolova, Ben Glocker, Ender Konukoglu, Olivier Pauly, Gerard de Melo, Johannes Feulner, Malte Weiss, Ali Eslami, Giuseppe Ottaviano, Min Sun, and Mat Cook for the great and fruitful time I spent in Cambridge during my stay at Microsoft Research Cambridge.

Further I want to thank Marc Pollefeys for the fruitful discussions which led to a publication. I also want to thank Juri Platonov, Radu Rusu, Suat Gedikli, Alex Ichim, Steven Miller, and Pantelis Kalegoris for the adventures that we had besides my PhD studies.

Last but not least, I want to thank my family, my parents Hannelore and Alfred, my sisters Christina and Sylvia as well as my brother Andreas for their support and believe in me during my whole life, and especially my love Rosie for her patience and support.

CONTENTS

THESIS OUTLINE

The following gives a brief outline of the single chapters of this thesis:

Chapter 1: Introduction. The first chapter gives an introduction to the fields of visual tracking and 3D point cloud processing. It points out possible applications as well as challenges that have to be faced. Further, related work is provided to give an overview of the current state of the art.

Chapter 2: Contributions. The second chapter discusses the publications that are part of this thesis. This includes work on adapting templates online for efficient learning and handling of occlusions as well as methods that significantly improve the learning speed for learning-based template tracking using linear predictors. Further, 3D point cloud processing methods are discussed. Those include a highly efficient normal estimation method as well as a learning-based approach for detecting interest points in depth data.

Chapter 3: Conclusions & Outlook. The final chapter concludes the presented work and gives an outlook to possible future research directions.

CHAPTER

ONE

INTRODUCTION

This thesis addresses the fields of visual tracking as well as processing of organized three-dimensional point clouds. The following gives an overview over both fields, including the definition of the problems considered in these fields, motivations and applications, challenges present in these fields, as well as related work.

1.1 Visual Tracking

1.1.1 Problem Definition

Given an image sequence, the goal of visual tracking is to estimate the motion of one or multiple objects or persons visible in the sequence of images (see Fig. 1.1), or to estimate the motion of the recording camera itself (see Fig. 1.2).

Initialization and Tracking. After initialization of the tracking, e. g. after a user manually selected the target region or object that should be tracked, visual tracking propagates information from frame to frame and uses this propagated information to search for the tracking target in a new image. This propagation of information is usually based on the assumption that the state of the considered system does not change significantly within a small time window.

Motion model. The propagation itself is based on a motion model which describes the expected change in the state of the tracking target over time. There is a wide range of possible motion models, e. g. assuming constant velocity or constant acceleration. However, often the explicit use of a motion model is omitted when a constant position model is assumed, i. e. when the search for the tracking target in the new frame is initialized at the same position as it was found in the previous frame.

Parameter space. The tracking target can depend on a large variety of parameters, including its position and orientation, its appearance, or possible deformations.While many methods focus on estimating the two-dimensional location of the tracking target within an

Figure 1.1: Tracking the keypad of a phone (©2010 IEEE).

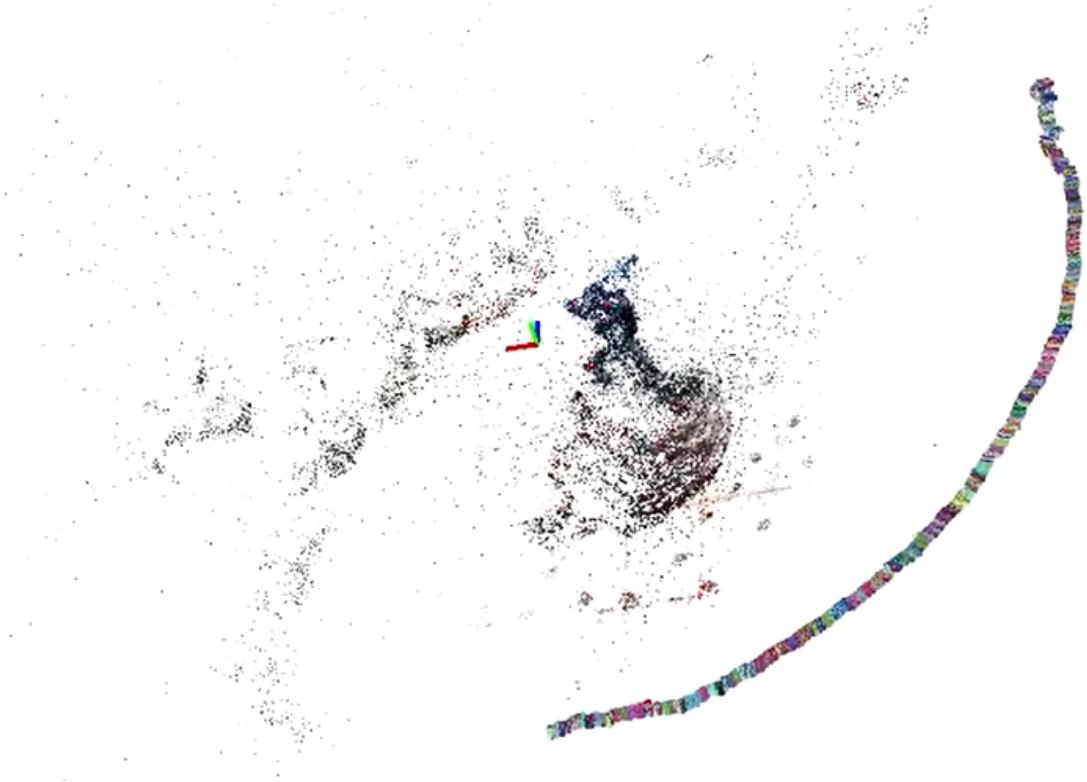


Figure 1.2: Tracking camera motion around an object (created using VisualSFM [88]).

image, others also estimate a bounding box which separates the target from background, or its three-dimensional orientation with respect to the recording camera. Yet others try to estimate the deformations a target undergoes or changes in appearance. Although not limited to it, the contributions within this thesis focus on estimating the eight-dimensional homography which describes the current location of the object of interest within the image space.

1.1.2 Motivation

The alternative to visual tracking are visual detection algorithms. These algorithms directly estimate the location of the target object or the camera pose from the current

input image without use of information from previous frames. The advantage of detection methods is their independence from previous frames. This way they can find the target at any position within the input image without having to rely on a motion model. However, this also comes with some drawbacks. Besides the significantly higher need in processing resources for purely detection-based tracking systems, visual tracking often shows an improved visual quality due to smoother pose transitions between frames. Here, using the pose from the previous frame for estimating the new one reduces jittering. Further, detection-based approaches are generally restricted in absolute pose while tracking-based approaches restricted only in relative pose and therefore, can often cover a larger pose space.

1.1.3 Applications

Visual tracking has a wide range of applications. In the following, several of these possible applications are discussed to present further motivation as well as future potentials for this field of research.

Augmented Reality. Augmented Reality (AR) becomes more and more present in people's lives. New devices, such as the newest generations of mobile phones as well as the Google Glasses, bring improved processing power and usability which makes AR interesting for daily use. In the consumer space, applications such as navigation, games, or ads are of particular interest. Navigation profits from AR by directly displaying navigation information over the real environment, e. g. showing which door to enter, which lane to use, or where exactly the destination is located. Similar holds for games. With AR technology they are able to integrate the user's surrounding into the game play, melting virtuality and reality together. In advertisements, having virtual data attached to a physical ad motivates a possible customer to interact with it and therefore, pay more attention to it. This is especially true at recent times where this kind of technology is just about to rise.

However, Augmented Reality is not only of interest for the end user but also for a variety of industrial and medical applications. In medical applications, for example, the doctor or surgeon can see additional information about his current patient using AR. An example could be the patients internal bones and organs, which helps to plan further steps during a surgery. Another application for AR at work is training. A mechanic can use an AR device to get used to a service or repair procedure of a new car, for example. This way, the mechanic can perform tasks on cars without prior training on this specific car

Car sensor systems. More and more cars, especially in higher class models, use cameras as input sensors for various tasks. Examples are to detect and follow the street lane or to monitor other traffic participants to detect possible threats, e. g. a person entering the street. Since cars are produced in high numbers and criteria like power consumption are extremely important, it is necessary to process the computer vision tasks as fast and with as few resources as possible. Visual tracking is extremely important here as it saves

processing resources by using information about the previous states of the car and its surrounding.

Personal robotics. Personal robotics has many applications for computer vision tasks, from detecting humans for interaction purposes, including their pose, mood, and so forth, to detecting objects in order to pick them up, to navigation which includes to build maps of the environment and to locate itself within these maps. All these tasks highly benefit from visual tracking as it helps to decrease the necessary processing power after the initial detection phase and therefore, the processor of the robot can focus more on other tasks, such as motion planning.

Surveillance. In surveillance, cameras are often mounted either at elevated positions, e. g. at a lantern looking at a parking lot, or mounted on airborne vehicles, e. g. a helicopter or a drone. The common goal there is to either find out whether an object is present in the observed area or to monitor the movements of objects, e. g. cars on a street or a suspect in a police chase.

Human-Machine Interaction. The field of human-machine interaction ranges from action and gesture recognition to things like detecting the mood of the user. User actions are usually built from a sequence of motions and therefore, it is important to follow the user and its state over time. Similar holds for gesture recognition. Although not all human-machine interaction tasks are based on a sequence of states, e. g. mood detection, they still benefit from tracking by reducing the processing costs.

1.1.4 Challenges

The following discusses the challenges present in the field of visual tracking.

Initialization. The first problem that arises in visual tracking is the problem on how to initialize the tracking. Depending on the application this is usually either done by user selection, e. g. if the object to track is not known a-priori, or by detecting the already known object of interest.

Parameter space. The next question that comes up when considering different tracking approaches is what type of transformation is important for the task, e. g. is it only important to know the x-y-location of the object within the image or is a full 3D pose of the object necessary. In general it holds that the more degrees of freedom the parameter space has the more challenging the tracking becomes.

Changing appearance. After initializing the visual tracking the appearance of the object of interest might change over time. In this case, the appearance model of the object, often directly encoded within the tracker, has to be updated accordingly. This is problematic as this might lead to drift in the tracking due to imprecisions in alignment when selecting new appearances.

Deformable objects. Another cause for a changing appearance is if the object of interest is deformable. Although a similar strategy as for simple appearance changes could be applied the knowledge of the possible deformations can help to improve tracking results. Furthermore, many applications explicitly need to know in which specific way the object deforms, e. g. if information has to be projected on the deforming surface.

Unpredictable motions. As mentioned in the definition of the visual tracking problem, visual tracking uses a motion model, implicitly or explicitly, to carry tracking information from one to the next frame. In certain applications however, the motion of the object to track is hard or even nearly impossible to predict, e. g. if either the camera or the object is hand-held. This leads to problems when the real motion is significantly different than the expected motion as then the tracking can be either lost or trapped in a local minima. Additionally, the search space has to be increased in such a scenario, which effects tracking speed.

Motion blur. Another problem of fast motion is that motion blur is likely to occur, especially under low light conditions. This can lead to a loss in accuracy or a total loss in tracking.

Occlusions. Occlusions are problematic in multiple ways. If only parts of an object are visible and the rest is occluded by some other object the tracker needs to be able to follow the object only with the available information while ignoring the information of the other object. However, especially in the case of tracking based on error minimization it is hard to distinguish between image differences due to a large motion and the image differences created by occlusions. Additionally, occlusion is usually only a temporary phenomenon and if the tracker adapts to changes of the appearance of the object it might actually get confused and drift towards tracking the occluding object. Finally, if the object of interest is completely occluded it is not possible to follow the object based on image information, at least if it is assumed that the motion or position of the object can not be inferred from the surrounding of the object. Possible solutions to a full occlusion are that either a motion model is applied until the object is visible again or that a detection algorithm is used to re-initialize the tracking after the occlusion vanishes.

Illumination changes. The appearance of an object is strongly dependent on the lighting present in its environment and on the reflection characteristics of the object. While indirect light provides a rather uniform illumination of the objects surface, a directed light can cause brighter spots and specularities. While uniform illumination changes usually can be handled by normalizing the image data of the object of interest, non-uniform illumination changes are harder to compensate. Extreme cases are shadows casted over the object or specularities which create strong bright areas on the object. In this case a global normalization of the image data is no longer sufficient.

Texture. There are many different types of objects with all kinds of textures. Tracking becomes especially problematic if the object of interest either has no or highly repetitive

texture. Depending on which type of texture is present, different kinds of tracking approaches are more or less suitable. For example, if no texture is present a tracking based on the shape of the object is better suitable than a tracking based on image value differences, if the object is single colored with a unique color in its environment then color-blob tracking is promising.

Available processing resources. A rather significant criteria for practical tracking applications is the problem of limited resources. This includes the necessary processing time as well as storage such as memory or disk space requirements.

1.1.5 Related Work

Methods for template tracking can be divided into two main categories, namely tracking-by-detection (TBD) and frame-to-frame tracking. Tracking-by-detection [25, 27, 28, 26, 29, 30, 61] methods do not rely on a motion model, i. e. they do not apply constraints on the possible location of the template based on its location in previous images, but search the whole image space for the object of interest. While TBD methods can find the object of interest anywhere in the image, this usually also makes them significantly slower than frame-to-frame tracking methods, which consider only a small region around the expected location of the object. Furthermore, tracking-by-detection methods often use a time consuming training phase to be able to detect the template under different poses. As the extent of this training phase also defines the space of poses under which the template can be detected TBD methods usually also have a stronger restriction on the possible pose space than frame-to-frame tracking methods. Frame-to-frame tracking methods can be further categorized into methods based on energy minimization [52, 75, 22, 10, 13, 2, 3, 53, 6, 12, 64] and methods based on learning [19, 45, 43, 44, 21, 63, 54, 56, 90, 31, 32]. While energy-minimization-based methods, in general, are faster and more flexible when creating and modifying templates, learning-based methods can achieve higher tracking speed and robustness. For example, in [43] and [31] it was demonstrated that a learning-based method, Linear Predictors, is superior to energy minimization using Jacobian approximation or using the Efficient Second-order Minimization [6] (ESM) approach.

Tracking-by-detection-based approaches. TBD methods can further be subcategorized into approaches based on interest point detection, where interest points are detected and matched against a reference set of interest points in order to find the location and pose of an object, and sliding-window-based approaches, where a set of template images is compared with the image of interest at every possible location.

The task of tracking an object or template using interest points consists of multiple stages: interest point detection, matching, and pose estimation. There exists a large set of different interest point detection approaches. Prominent examples are SIFT [51], SURF [5], the Harris interest point detector with it's variations [23, 73], or FAST and it's extensions [66, 68]. After detecting interest points for the reference object as well as for the current input image, the interest points have to be matched together. A common

approach is to do this by computing descriptors for each point and matching interest points by finding nearest neighbors in descriptor space. There exists a large set of different descriptors. Examples are the SIFT [51] and SURF [5] descriptors, Histogram of Oriented Gradients (HOG) [11], or BRIEF [8]. In [61], Özuysal et al. presented an interest point matching approach, called FERNs, based on classification where for each reference interest point a class is learned and then for each input interest point a probability is computed which states how likely it is that it belongs to that class. For pose estimation it is necessary to be able to deal with outliers in the matching process. Probably the most prominent approach for this is RANSAC [14], an approach that uses random sampling to iteratively compute different pose estimates, looking for the one with the most inliers. One of the main disadvantages of interest-point-based methods is that they usually require a significant number of interest points for a robust detection. To overcome this limitation, Hinterstoisser et al. [25, 27] introduced methods where single interest points are used to initialize a patch-based pose estimation and therefore, tracking-by-detection is possible with only a single interest point. However, similar to the other interest-point-based methods, those methods are limited by the detection rate of the underlying interest point detector. Instead of using interest points it is also possible to use other features, such as shapes or closed contours. An example for such a method are Distance Transform Templates (DTT), which were presented in [30]. There, closed contours are robustly extracted and matched with a reference set of contours using FERNs [61].

A common way to get past the dependency on interest points is to densly search for a patch that describes the object of interest from a specific view-point. An early approach on this was presented by Olsen et al. [58] where the target object was found by looking for the minimal Chamfer distance between the template and the input image contours. Since then, many advances have been proposed, including coarse-to-fine approaches [16, 7], use of different distance measures [7, 69, 41, 30], or using the Hough transform [4]. Since many of these methods are based on a contour extraction method, such as Canny [9], they tend to be sensitive to illumination changes, noise and blur. For this reason, other methods directly rely on image gradients [71, 11, 1, 28, 26, 29] where the dot product between template and input gradients is used as similarity measure. Learning-based approaches [19, 84, 39, 62] try to build classifiers that help to identify the object of interest.

While tracking-by-detection methods can lead to a robust tracking, e. g. if a sufficient number of interest points are available, in general they require a higher amount of processing than frame-to-frame tracking methods.

Keyframe-based approaches. If no prior model of the object or scene to track is available, keyframe-based methods are a common way to build a model of the object and to track it at the same time. These methods rely on creating a set of keyframes that represent the object or scene. If these keyframes are available through a offline process, the pose is estimated based on the iteratively computed model [82]. Otherwise, a map of the tracked environment is created at run-time using interest points [48] or other discriminative image features, e. g. edges [49]. Keyframe-based approaches show similar problems as some of the Tracking-by-Detection-based methods. If the number of available

interest points or features is limited they tend to become error-prone, which is especially a problem for small objects or scenes.

Energy-minimization-based approaches. In energy-minimization-based tracking, a set of pose parameters is optimized such that the image value differences between an input image and a reference image, which is warped according to the pose parameters, is minimized. Since the early work of Lucas and Kanade [52], a significant number of energy-minimization-based tracking approaches has been proposed with improvements in different areas. These areas include the use of different update rules for the warp function [52, 22, 10, 75, 13, 2], the explicit handling of occlusions and illumination changes [22], and the use of different orders of approximation of the error function [53, 6]. When optimizing pose parameters, this is usually done by computing a parameter difference which is then applied on the initial pose parameters. Over time, different update rules have been proposed on how these parameter differences are applied on the initial pose parameters. These update rules can be categorized into four different categories: the additive approach [52], the compositional approach [75], the inverse additive approach [22, 10] and the inverse compositional approach [13, 2]. The advantage of the inverse approaches is that they switch the roles of the current and the reference image. This modification makes it possible to pre-compute a certain amount of the necessary computation during the initialization phase and therefore, makes the tracking phase less computationally demanding. The problems of changing illumination as well as occlusions that partially hide the object of interest are addressed by Hager and Belhumeur [22]. In [53, 6], the first-order approximation originally used in the optimization process is replaced by a second-order approximation, leading to larger convergence areas and an improved speed in reaching convergence. Another common technique to increase the region of convergence is to apply smoothing on the energy term, which is generally done by smoothing the image data [79]. However, this smoothing can lead to loss of image information important for the optimization and therefore, cause the optimization to not lead to the desired minimum. Recently proposed methods reduce this problem by computing multi-dimensional descriptors for each image pixel on which they then apply smoothing [72, 57]. This way the smoothing is applied separately on each dimension, which helps to keep important information. A more detailed overview of energy-minimization-based tracking methods can be found in Baker et al. [3].

Learning-based approaches. In learning-based tracking it is distinguished between approaches that require an offline learning phase and approaches that can learn online at run-time. In online learning, it has to be distinguished between learning the necessary data for tracking completely online or updating the tracking information in order to adapt to changes in the object of interest. Online learning of the complete tracking information is essential for applications where formerly unknown objects have to be tracked. The approaches of Grabner et al. [19] and Kalal et al. [45] are prominent examples for state-of-the-art methods on learning-based tracking that are able to learn objects online. For both approaches, the tracking process is separated into a frame-to-frame tracking, a detection which localizes the previously seen appearances of the tracked object and corrects the

tracking information if necessary, and a learning procedure that updates the detector in order to minimize it's detection errors. The semi-supervised online boosting strategy of Grabner et al. [19] is replaced by a PN-learning strategy in Kalal et al. [45]. The PN-learning uses two sets of experts to estimate missed detections as well as false alarms. The combination of tracking, learning and detection is especially of advantage when an object changes it appearance online or if occlusions occur. A significant disadvantage of these methods is however, that they only give good results when tracking a bounding box but not when the full pose of the object of interest has to be estimated over time.

Jurie and Dhome [43] introduced a learning-based template tracking approach using hyperplane approximation which can be seen as a linear prediction process where image value differences are linearly mapped onto the corresponding parameter differences. In order to train this mapping, a set of randomly warped image samples of the initial template is generated and stored together with the parameter differences used to create these samples. From this a Linear Predictor is computed and used online to predict parameter updates. This Linear Predictors replace the role of Jacobians used in energy-minimization-based methods and have to be computed only once offline instead of newly at every iteration online. A more comprehensive introduction into this approach is given in Sec. 1.1.6. The work of Jurie and Dhome [43] was extended in [44] in order to handle occlusions. Obtaining invariance with respect to changes in illumination is achieved using the approach presented in [20]. An intelligent selection process for choosing sample points for sampling the image data to compute the image differences is described in [21]. While [43] uses a single large template, Zimmermann et al. [90] uses a large number of small templates which are tracked individually and their different motion estimates are combined back into a single one at the end. Instead of explicitly learning a predictor, Mayol and Murray [56] create and store a set of training samples which are then queried online in order to compute the parameter update using general regression.

1.1.6 Linear Predictors for Template Tracking

Before getting into more details on the contributions of this thesis an introduction on template tracking using Linear Predictors is given, as the presented visual tracking methods are based on them. The key idea behind Linear Predictors is to learn a relationship between an image value difference, between a template and the current image, and the template parameter changes that would align that template with the current image.

Template. A template is represented by a set of n_s sample points located within a reference image. For simplicity, these sample points are often arranged uniformly in a regular grid. The sample point locations are used to sample the image data which is stored in an $n_s \times 1$ vector $\mathbf{i} = [i_n]_{n=1}^{n_s}$.

Pose parametrization. Within this thesis, the pose of the template is represented by a homography which is defined by four control points. The pose parameters are stored in an 8×1 vector $\boldsymbol{\mu} = [\mathbf{p}_n]_{n=1}^{4}$, where $\mathbf{p}_n$ are the locations of the control points used to define the homography. In practice, these four control points are often chosen to be the four corner

points of the rectangular area in which the sample points are located. However, please note that Linear Predictors are neither limited to homographies nor to the representation using four corner points and that other transformations and representations are possible too.

Tracking. For estimating the pose parameters $\boldsymbol{\mu}_t$ at the current time step t, a parameter update $\delta\boldsymbol{\mu}$ is estimated which updates the pose parameters $\boldsymbol{\mu}_{t-1}$ of the previous time step $t-1$:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \delta\boldsymbol{\mu}. \tag{1.1}$$

As shown in [43], the parameter update $\delta\boldsymbol{\mu}$ can be directly estimated using the image difference vector $\delta\mathbf{i} = \mathbf{i}_t - \mathbf{i}_R$, where $\mathbf{i}_R$ are the image values sampled from the reference template and $\mathbf{i}_t$ are the image values sampled from the image given at the current time step. The position of the sample points is hereby defined by the pose parameters $\boldsymbol{\mu}_{t-1}$ obtained at the previous time step or iteration. The relation between the parameter update and the image difference vector is given by a Linear Predictor $\mathbf{A}$ as:

$$\delta\boldsymbol{\mu} = \mathbf{A}\delta\mathbf{i}. \tag{1.2}$$

Learning. The goal of the learning stage is to find a Linear Predictor $\mathbf{A}$ which obtains good results for Eq. (1.2). In the learning stage, tuples of corresponding $\delta\mathbf{i}_i$ and $\delta\boldsymbol{\mu}_i$ vectors are precomputed by perturbing the reference parameters $\delta\boldsymbol{\mu}$ and extracting the image value differences from the accordingly rendered template image. These tuples are then stacked into matrices $\mathbf{Y} = [\delta\boldsymbol{\mu}_1, \ldots, \delta\boldsymbol{\mu}_{n_t}]$ and $\mathbf{H} = [\delta\mathbf{i}_1, \ldots, \delta\mathbf{i}_{n_t}]$, where n_t is the number of training tuples. Along the lines of Eq. (1.2) these two matrices can be related as:

$$\mathbf{Y} = \mathbf{A}\mathbf{H}. \tag{1.3}$$

By applying the pseudo-inverse of $\mathbf{H}$ the Linear Predictor $\mathbf{A}$ can be computed as [43]:

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top \left(\mathbf{H}\mathbf{H}^\top\right)^{-1}. \tag{1.4}$$

Normalization. To gain robustness against uniform illumination changes a normalization is applied on the sampled intensity vectors such that they show zero mean and unit standard deviation. The resulting rank-deficiency of the matrix $\mathbf{H}\mathbf{H}^\top$, which needs to be inverted during the learning process, can be resolved by adding small amounts of random noise to the normalized intensity vectors.

Coarse-to-fine strategy. The goal of robust tracking is to find the correct pose parameters in the current image as precisely as possible and to do that even if a large motion is present. Because of this, a coarse-to-fine strategy or multi-level tracking is applied. For this, a cascade of Linear Predictors is learned where the first one is trained for large motions while the subsequent ones are trained for less and less motion. This way, the first Linear Predictor will roughly align the template in the current image while the following

Linear Predictors keep refining the results. Additionally to this multi-level tracking, multiple iterations are applied per Linear Predictor. Although this strategy is not explained in the original paper of Jurie and Dhome [43] it is a common way to improve tracking robustness.

1.2 3D Point Cloud Processing

1.2.1 Problem Definition and Motivation.

A 3D point cloud is a set of data points in a 3-dimensional coordinate system, usually acquired using a 3D scanner or manually constructed using a CAD[1] software. Nowadays, a large variety of different kinds of devices is available to create a 3D point cloud of a real object or scene. These include laser scanners, structured light cameras, stereo cameras and more generally multi-view camera setups, texture projectors, time-of-flight (TOF) cameras, and so on. Especially the recent development in cheap and broadly available 3D sensors (such as the Microsoft Kinect) boosted the interest and applicability of 3D point cloud processing techniques. However, to be able to use 3D point clouds in useful applications they often need to be further processed, e. g. by computing a corresponding mesh or finding a transformation between the given point cloud and a reference model.

Normal Estimation. One of the most common basic operations on 3D point clouds is to estimate the corresponding normal orientations. That is to find the unit length three-dimensional vector which is perpendicular to the tangential plane of the local surface defined by the 3D point cloud. Since 3D point clouds are only a sparse representation of a surface in space, this tangential criterion is only a weak constraint which is usually controlled by how smooth the surface is to be expected. Estimating surface normals builds the base for many other tasks and algorithms and it is therefore highly important that it is achieved as efficiently as possible and with high robustness, as errors in the normal estimation will propagate further in the processing pipeline. Failing to compute robust normals can lead to significant problems in higher-level approaches.

Interest Point Detection. The detection of interest points in 3D point clouds aims to select a subset of discriminative and uniquely identifiable points out of the available set of 3D points. This is important as common 3D sensor devices used for scanning objects or scenes provide 3D point clouds which contain a huge amount of points and are therefore computationally expensive to process if all points would be considered. What is usually looked for in this context are points that are easy to re-localize, e. g. points at corners or locations with maximal curvature, and for which meaningful descriptors can be computed such that when comparing the descriptors of several interest points together only those have a small difference which belong to the same physical location.

[1] Computer Aided Design (CAD)

1.2.2 Applications

3D point cloud processing has many fields of application and especially since the introduction of cheap and small sensor devices it starts to affect more and more application areas.

Robotics. Robotics is usually divided into industrial and personal robotics. 3D point cloud processing has applications in both. In industrial robotics it can, amongst other things, be used for motion planning, obstacle avoidance, or part inspection. In personal robotics it is used for things such as navigation, including motion planning and obstacle avoidance, object detection, place recognition, mapping, and many more.

Automotive. Car companies put more and more sensor technology into their cars in order to be able to better observe the surrounding of the car. This includes video cameras, ultra sonic senors, radar, as well as more advanced sensors, such as laser scanners. Although in the automotive field 3D laser scanners are currently mainly used in research projects, technologies like depth from stereo can create 3D point cloud data already without having to rely on expensive laser scanners. These research efforts reach into applications such as automated parking, autonomous driving, or threat detection.

Construction. When building large factories or renovating large buildings, a problem that often occurs is that the reality does not exactly fit to the plans of the buildings. 3D scanning can be used in order to incorporate such differences into the technical drawings of the building. This makes it easier to plan future changes and to make sure that e. g. a power plant can operate within its specifications using the given structure.

Gaming/Entertainment. The gaming and entertainment industry especially profited from the raise of cheap 3D sensing devices such as the Microsoft Kinect. Here, these sensors have multiple possible use cases. One is to capture the player and its motion such that the player can control the game using his body. But, 3D sensing can also be used to create a 3D map of the users environment in order to integrate it into the game dynamics.

Human-Machine Interaction. Besides controlling computer games using 3D sensors, 3D sensing technology also becomes interesting in other fields for human-computer interaction. In medicine, for example, 3D sensing can enable doctors to control electronic devices without having to touch them, which simplifies the process of keeping objects sterile. This also applies to all kinds of scenarios where people would have to touch input devices in public areas, where a touch-less approach will help to decrease spreading of diseases. Further, being able to control devices using hand motions makes interfaces feel more natural and more complex scenarios than it would be possible with restricted input devices, such as a computer mouse, can be handled.

1.2.3 Challenges

There is a wide variety of challenges that have to be faced when working with 3D point cloud data. The following tries to list the main challenges, starting with general challenges that are induced by the sensor devices and other hardware limitations, and going over to more specific challenges for normal estimation as well as interest point detection. Hereby, the general challenges tend to be more or less of a problem depending on the 3D sensing device used for capturing the object or scene of interest, while the later ones are more specific to the processing tasks. Often it is the case that choosing one or another sensor will lead to a compromise between these challenges. For example, while Kinect like sensors provide decent sensor resolution they show problems at occluding borders. On the other hand, Time-Of-Flight cameras have lower resolution but do not suffer from the occluding borders problem.

Sensor resolution. The resolution of the obtained 3D point cloud defines the amount of detail that can be extracted out of the scanned data. For example, navigation tasks often do not need a very high resolution while tasks like object detection and pose estimation profit from higher resolution.

Depth resolution. Another resolution that is of interest is the resolution in depth, i. e. the minimum possible distance in depth between two measured points. While a rough depth resolution can be sufficient to estimate the boundaries of an objects it would not be sufficient to get a detailed model of the complete 3D shape of it.

Noise. If the signal-to-noise ratio becomes too small a significant smoothing has to be applied to still get valuable information. However, this is time-intensive and can cause distortions.

Motion. If fast motion is present then certain sensors can show distortions in the 3D point cloud caused by the different locations of the sensor when measuring the 3D data. This, for example, can be a problem in automotive applications where lasers are mounted on cars.

Occluding borders. Occluding borders occur on the borders of an object and are caused by the structure of certain sensor devices. For example for the Kinect sensor the projector for the texture pattern is translated from the camera sensor. This leads to blind spots next to object borders.

Missing data. Besides the occluding borders, other factors can cause the sensor not to return 3D data for certain locations. For example if the surface of interest is too reflective, or not reflective enough, it can cause the sensor not to be able to capture 3D information there.

Data sparseness. Normal estimation depends on the local neighborhood of a point. If the obtained 3D data is too sparse this usually leads to a strong smoothing of the estimated normals.

Data organization. For finding local neighbors of a point, organized point cloud data, arranged similar to an image, helps to greatly improve processing speed.

Interest point sparseness. The goal of interest point detection is to find a sparse set of points to represent the current scene. However, if the selected points are too sparse then a robust alignment, for example, is no longer possible. On the other hand, if too many points are selected, follow-up algorithms will take too much time to handle them. This is a thin line that has to be considered when designing a detection algorithm.

Locality. Another important factor in interest point detection is the determination of how much of its neighborhood should be considered when searching for interest points. For example, if only a very small neighborhood is considered then it is more likely that noise can create interest points too, while when large neighborhoods are considered significant geometry changes have to be present to trigger an interest point to be detected and usually the accuracy of the detected interest points degrades.

Processing time. Since 3D sensors in general provide a large amount of data it is key to have efficient processing algorithms.

1.2.4 Related Work

Normal estimation. Methods for estimating 3D surface normals are often categorized into averaging and optimization-based methods [47]. Methods based on averaging compute the normal using the (weighted) average of the point of interest's neighborhood. Typical choices for a neighborhood are points within a certain distance or points connected to the point of interest, e. g. neighbors in a triangulated mesh. In case of a weighted average, the weights define the influence of the information taken from the local neighborhood. Typical methods for computing a weighted average include weighting all points equally [18], weighting by angle [80], weighting by sine and edge length reciprocals [55], weighting by areas of adjacent triangles [55], weighting by edge length reciprocals [55], and weighting by square root of edge length reciprocals [55]. A more detailed overview of methods based on averaging is given in [42].

Optimization-based methods are often based on fitting geometric primitives, such as a plane, to the point of interest using its local neighborhood or by penalizing other criteria, such as the angle between the desired normal and the tangential vectors. Tools like the singular value decomposition (SVD) or the principal component analysis (PCA) can be used to directly obtain the desired minimal solution if the optimization can be formulated as a linear problem [47]. Existing methods rely on fitting planes [40, 38, 87, 46], maximizing the angle between the tangential vectors and the normal vector [17], or trying

to estimate the orientation of the tangent plane as well as its curvature at the same time [89, 59, 83]. For a more detailed comparison the reader is referred to [47].

Interest point detection. The field of 3D interest point detection in depth data received surprisingly little attention in the past. While research in 2D interest point detection significantly advanced in the past decades, the number of available 3D interest point detection methods is still very limited. Besides the additional processing complexity of 3D data this can mainly be explained by the missing availability of cheap 3D sensors. In the following, existing 3D interest point detection methods as well as general learning-based interest point detection methods are reviewed.

3D interest point detection. In [77], Steder et al. introduced a method to compute interest points in range image data using the Laplacian-Of-Gaussian method. However, due to its computational complexity it is not suitable for real-time or near-real-time processing. Their later work, [78], presents an significantly more efficient method that locates interest points by finding and classifying different kinds of 3D object borders. However, due to its focus on range data characteristics the method only provides reduced robustness on depth maps, such as provided by the Kinect sensor. Unnikrishnan [81] presented a method that extracts interest points from unorganized 3D point cloud data and automatically estimates its scale. Due to its operation on unorganized 3D data it does not take any view-point related information into account.

Learning-based interest point detection. Using machine learning approaches is not completely new in the field of interest point detection. A number of approaches has been proposed for extracting interest points from color or gray value images. FAST, Features from Accelerated Segment Test, is an interest point detector introduced by Rosten et al. [65]. It uses the pixels on a circle around each point to determine whether the point is an interest point or not. The approach does not explicitly involve machine learning, however, it can be seen as a manually designed decision tree. Based on this, Rosten et al. proposed an extension in [67] where the manual design of the decision tree is replaced by a learning phase. Rosten et al. further improved their detector in [68] by considering more pixel locations for testing and using simulated annealing to optimize the decision tree with respect to repeatability and efficiency. This is done by randomly changing the initially learned decision tree and evaluating whether the applied change improves the repeatability and efficiency of the detector.

Sochman et al. [86] presented a learning-based approach where a WaldBoost [76] classifier is learned to emulate an existing binary-valued decision algorithm, which is acting as a black box. The binary-valued decision algorithm can hereby be a interest point detector that tells if there is a interest at a specific pixel location or not. The WaldBoost learning algorithm greedily emulates a given binary-valued decision problem by finding a quasi-optimal sequential strategy. The learning algorithm performs feature selection using the AdaBoost [70] algorithm and finds the thresholds used for making the decisions using Wald's sequential probability ratio test (SPRT). To demonstrate the usefulness of the method, they learned the Hessian-Laplace as well as the Kadir-Brady saliency detectors. While being able to emulate the detectors, their detection speed is too slow to process an image at a reasonable frame rate.

Taking a more broader look at detection algorithms, one can categorize high-level algorithms such as an object or a face detector as an interest point detector too. An example for such a high-level algorithm is the face detector presented by Viola et al. [85]. A further example is the method presented by Foresti et al. [15] which uses preceptron trees to classify surface types, such as planes or valleys, in range data. Although this identifies regions of interest it can also be considered as an interest point detector in a broader sense.

In image space, Lepetit et al. [50] estimated the probability that a considered point corresponds to a specific class by training classification trees. While the authors did not present this method as an interest point detector, but as a method for interest point matching, it can also be seen as a detector if densely applied on an input image. The same task was solved in [60] by making use of Ferns. In [74], body parts are estimated using decision trees, which were trained using artificially rendered humans under different poses.

The work on interest point detection performed as part of this thesis (see [37]) advances the above mentioned work by optimizing artificial interest point response maps such that they increase the detection performance and by using these artificial response maps to learn to detect interest points in depth data using regression trees.

CHAPTER

TWO

CONTRIBUTIONS

In the following the contributions of the publications contained within this thesis are discussed.

2.1 Visual Tracking

2.1.1 CVPR[1] 2010: Adaptive Linear Predictors for Real-Time Tracking

Although the standard learning method of Linear Predictors, as explained in Section 1.1.6, allows fast and robust tracking of a template, it is limited by the computational costs it needs for the learning itself. Computing the Linear Predictor matrix $\mathbf{A}$ using Eq. (1.4) is not sufficiently fast for learning it online on-the-fly, as it involves computing the inverse of a large matrix. Having such high demands on processing time, modifying the shape of a once learned template is not feasible using the standard approach.

Therefore, [31] presents methods for dynamically adapting Linear Predictors online (see Appendix A). This includes enlarging and shrinking of the template area, as well as evaluating the suitability of certain portions of the scene for tracking. Using this suitability criterion, a proper region can be selected for enlarging the template. These proposed methods show several advantages: since dynamically adapting a Linear Predictor can be done efficiently, a new learning strategy is possible where tracking starts with a small template and then new parts are added to the template on the fly. Furthermore, by being able to shrink the template online, without the need of re-learning the template in an computationally expensive way, the template can be adapted to potential changes in the scene as well as to visibility constraints caused by the sensor limitations, i. e. if parts of the template leave the visible image region.

The key idea for fast and efficient template adaption is to consider the template as a set of subsets (see Fig. 2.2). When enlarging or shrinking the template, one of those sets is either added or removed from it. This way, a new corresponding Linear Predictor can

[1] IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)

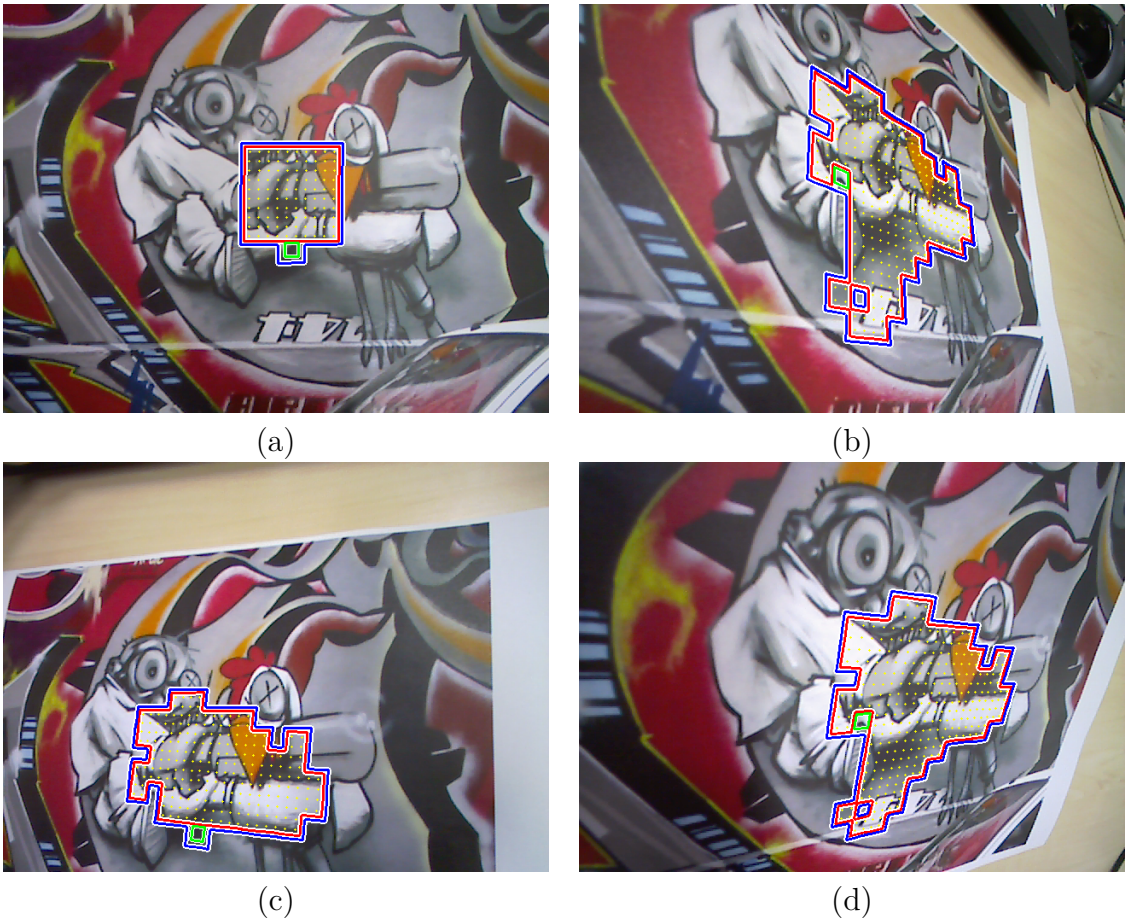


Figure 2.1: (a) A small initial template is (b) enlarged according to a tracking quality measure. The template is tracked over time and (c) reduced if parts of it go out of sight. The removed parts are reinserted (d) as soon as they become visible again. ©2010 IEEE

be computed by adapting the inverted matrix $(\mathbf{H}\mathbf{H}^\top)^{-1}$ of Eq. (1.4) instead of having to recompute it. If we consider $\mathbf{H}_I$ to be the image sample data of a large template area and $\mathbf{H}_E$ the image sample data of an extension or reduction subset then we can write:

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{H}_I\mathbf{H}_I^\top & \mathbf{H}_I\mathbf{H}_E^\top \\ \mathbf{H}_E\mathbf{H}_I^\top & \mathbf{H}_E\mathbf{H}_E^\top \end{bmatrix} \right)^{-1} = (\mathbf{H}\mathbf{H}^\top)^{-1}, \tag{2.1}$$

where the goal of template enlargement would be to compute $\mathbf{S}$ given $(\mathbf{H}_I\mathbf{H}_I^\top)^{-1}$ and the goal of template shrinking would be the vice versa. In case of template enlargement, the parts of $\mathbf{S}$ can be computed making use of formulas presented by Henderson and Searle [24]:

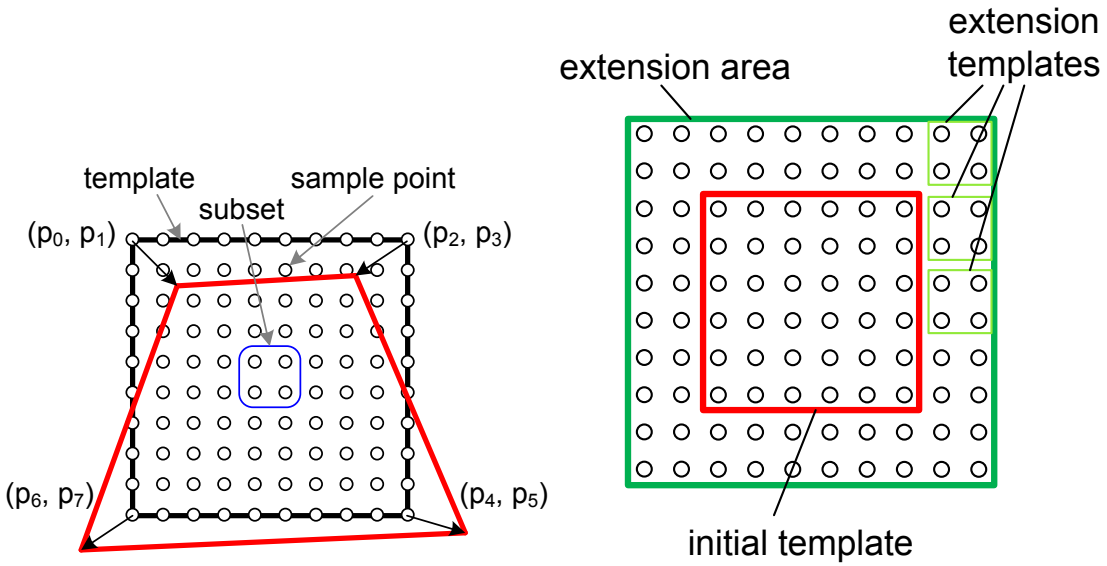


Figure 2.2: A template is represented by a set of regularly placed sample points, which are grouped into subsets of four points. The pose of a template is parametrized using four corner points. ©2010 IEEE

$$
\begin{aligned}
\mathbf{S}_{11} &= (\mathbf{H}_I\mathbf{H}_I^\top)^{-1} \\
&+ (\mathbf{H}_I\mathbf{H}_I^\top)^{-1}\mathbf{H}_I\mathbf{H}_E^\top\mathbf{S}_{22}\mathbf{H}_E\mathbf{H}_I^\top(\mathbf{H}_I\mathbf{H}_I^\top)^{-\top} && (2.2)\\
\mathbf{S}_{12} &= -(\mathbf{H}_I\mathbf{H}_I^\top)^{-1}\mathbf{H}_I\mathbf{H}_E^\top\mathbf{S}_{22}, && (2.3)\\
\mathbf{S}_{21} &= \mathbf{S}_{12}^\top, && (2.4)\\
\mathbf{S}_{22} &= \left(\mathbf{H}_E\mathbf{H}_E^\top - \mathbf{H}_E\mathbf{H}_I^\top(\mathbf{H}_I\mathbf{H}_I^\top)^{-1}\mathbf{H}_I\mathbf{H}_E^\top\right)^{-1}. && (2.5)
\end{aligned}
$$

For template reduction, further formula provided by Henderson and Searle [24] can be used to compute the desired result:

$$
\begin{aligned}
(\mathbf{H}_N\mathbf{H}_N^T)^{-1} &= \mathbf{S}_{11} - \mathbf{S}_{11}\mathbf{H}_N\mathbf{H}_R^T\mathbf{D}\mathbf{H}_R\mathbf{H}_N^T\mathbf{S}_{11}, && (2.6)\\
\mathbf{D} &= \left(\mathbf{H}_R\mathbf{H}_R^T + \mathbf{H}_R\mathbf{H}_N^T\mathbf{S}_{11}\mathbf{H}_N\mathbf{H}_R^T\right)^{-1}. && (2.7)
\end{aligned}
$$

This significantly decreases the necessary learning time (see Fig. 2.3) as the matrix inversion is the computationally most expensive part of the learning, especially for large templates. A more detailed derivation of the formulas for template extension and reduction, a method for computing the suitability of extension regions for tracking, notes on practical issues that have to be considered, and a more thorough experimental analysis can be found in Holzer et al. [31]. The corresponding publication can be found in Appendix A.

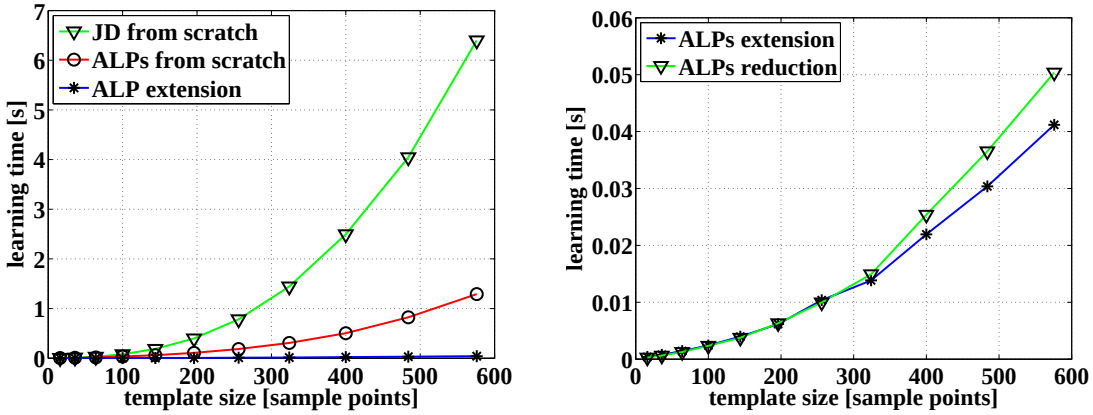


Figure 2.3: Comparison of the computation time necessary for learning a linear predictor using the Jurie-Dhome [43] (JD) approach (green) and using ALPs (red and blue). **(a)** For the latter case we distinguish between learning the predictor from scratch (red) and adding only one extension subset (blue) at a time. Learning from scratch means that we consider the entire time necessary to build up the template of the specified size. **(b)** Computation times for template extension and reduction, when one extension subset is added at a time. The blue curve corresponds to the blue curve at (a). ©2010 IEEE

2.1.2 TPAMI[2] 2012: Multi-Layer Adaptive Linear Predictors for Real-Time Tracking

Building on top of Holzer et al. [31] , [32] adds a multi-layer concept which detects and handles occlusions while tracking and also introduces an extension which robustifies the tracking with respect to large motions (see Appendix D). In this multi-layer approach for occlusion detection the first layer contains a large template which is then sub-subsequently divided in the following layers (see Fig. 2.4). When tracking, the first layer is used to align the template and the subsequent ones are used to further refine the pose and especially to detect occlusions. Since present occlusions would also influence the tracking of the original first layer template, it is important do adapt the shape of it when occlusions occur. For this, the methods presented in Holzer et al. [31] are used.

However, to adapt the template the occlusions have to be detected first. This is done using the introduction of *secure*, *insecure*, and *occluded* regions in the template area (see Fig. 2.5). Initially, when the first layer template is not occluded, the inner parts of the template are considered as *secure* and a certain area around it is considered as *insecure*. *Insecure* regions are monitored while tracking and continuously checked for whether an occlusion, i. e. a change in intensities, is present. If so, the insecure region is marked as occluded and all its neighboring regions are marked as *insecure*. If a region changes from *secure* to *insecure* then it is removed from the template and if it changes back to *secure* it is added again.

To achieve the earlier mentioned robustification of the tracking process the tracking is initialized at multiple locations for the first layer and only the best performing result is considered further in the tracking pipeline. This significantly increases the amount of motion that can be handled by the tracking (see Fig. 2.6). The corresponding publication

[2]IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)

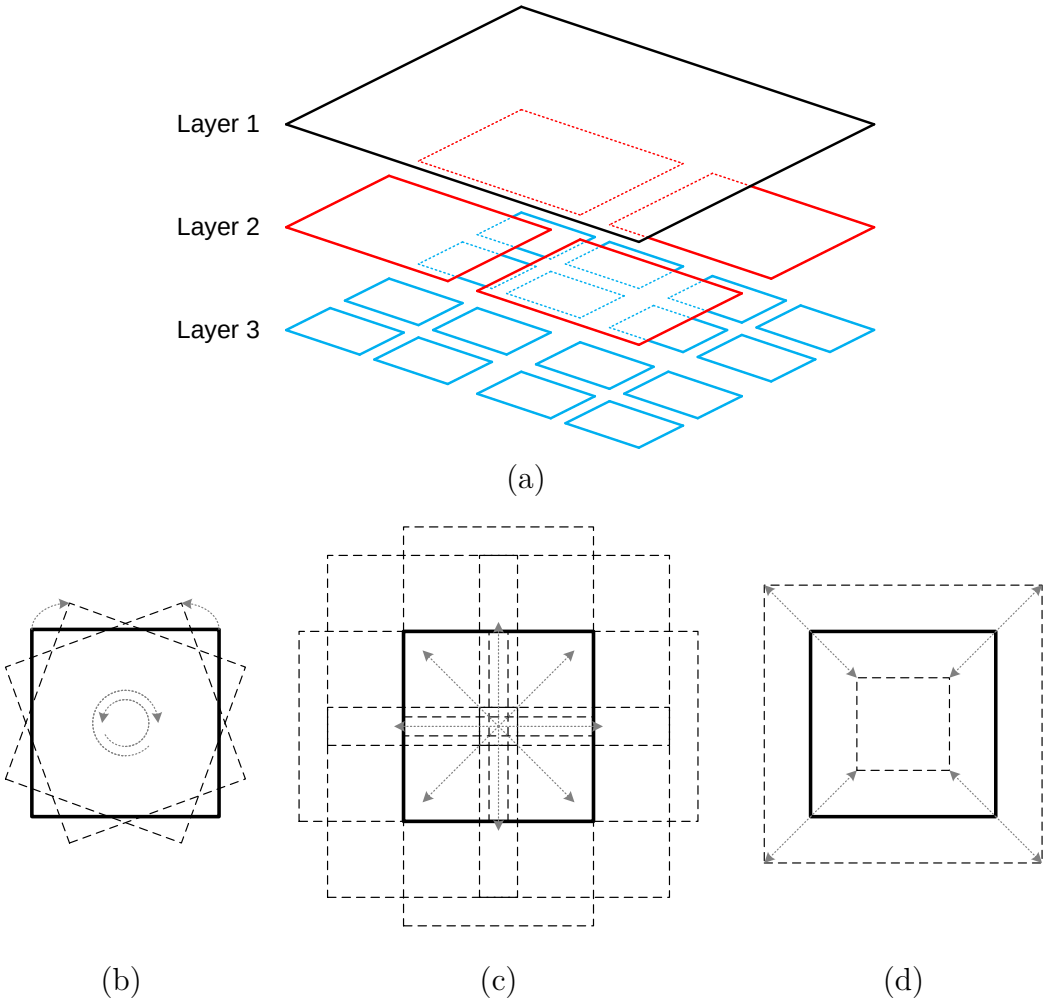


Figure 2.4: **(a)** The organization of the multiple layers used for tracking. The sub-figures **(b)**, **(c)** and **(d)** show different transformed templates of the top layer used to increase the robustness against large motion. **(b)** shows differently rotated, **(c)** differently translated and **(d)** differently scaled templates. ©2013 IEEE

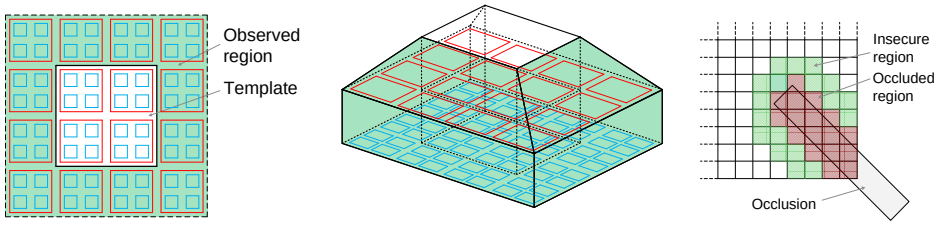


Figure 2.5: The left figure shows the multi-layered template with its observation region depicted as green area. The middle figure shows the different layers and their contribution to the observed region from a side-view. The right figure illustrates insecure regions, which are areas around the detected occlusions. ©2013 IEEE

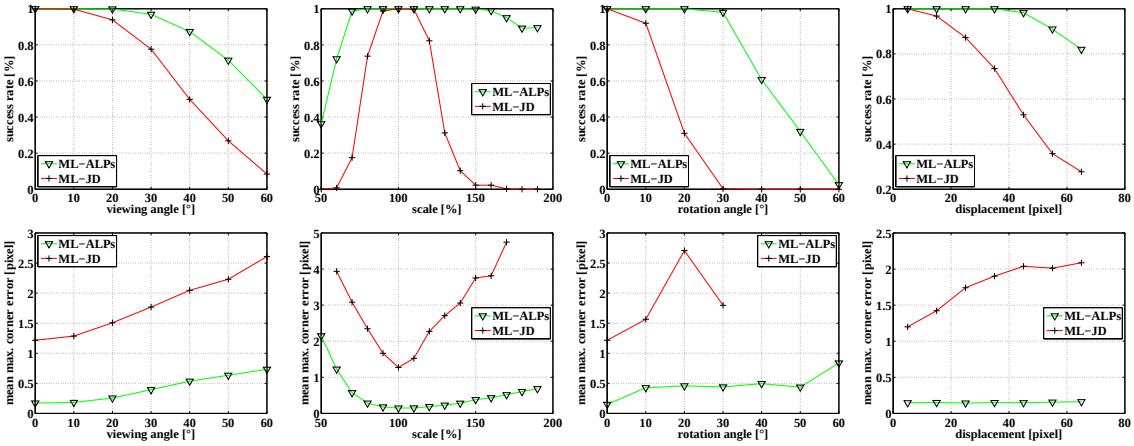


Figure 2.6: Results of the comparison between the ML ALPs approach and the ML approach of Jurie and Dhome [44] with respect to different types of motion without occlusion. The first row shows the tracking success rate and the second row the corresponding mean maximum corner errors. ©2013 IEEE

can be found in Appendix D.

2.1.3 ECCV[3] 2012: Online Learning of Linear Predictors for Real-Time Tracking

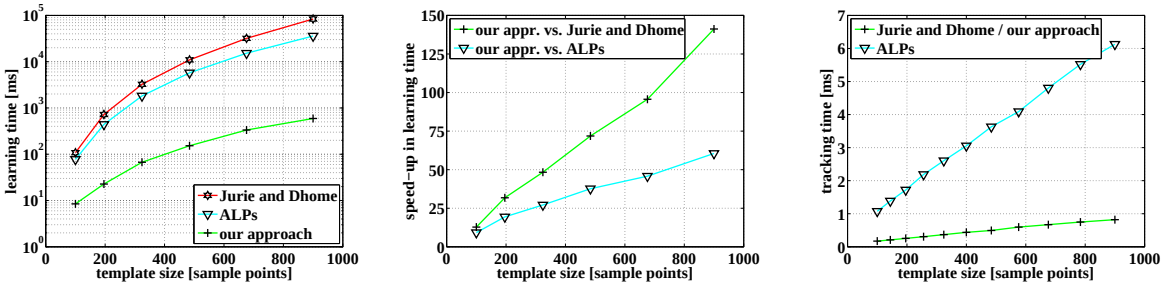
Although the work in [31] improved the speed of the learning process significantly it is still not capable of learning larger templates at speeds that are acceptable for user interaction. This holds especially for lower powered devices such as mobile phones. Therefore, [35] introduces a reformulation of the learning equations which allow efficient training that is up to two orders of magnitude faster than the original approach (see Appendix B).

The key idea behind the original learning strategy was to solve for the Linear Predictor matrix $\mathbf{A}$, which relates image differences values $\mathbf{H}$ and parameters updates $\mathbf{Y}$ as $\mathbf{Y} = \mathbf{AH}$, by using the right pseudo-inverse to bring $\mathbf{H}$ on the left side. Alternatively, it is possible to bring $\mathbf{Y}$ on the right side first using the left pseudo-inverse and then to use the right pseudo-inverse to bring it together with $\mathbf{H}$ back on the left side [35]. The difference between both approaches is in the size of the matrices that have to be inverted.

Assume that $\mathbf{Y}$ is of size $8 \times n_t$ and $\mathbf{H}$ of size $n_p \times n_t$, where n_t is the number of training samples and n_p the number of image sample points, i. e. the template size. Then, the original approach has to invert a matrix of size $n_p \times n_p$, where in practical cases n_p is usually significantly larger than 100 and therefore, the inversion takes a significant amount of processing time.

The alternative approach, as proposed in [35], first computes the left pseudo-inverse of $\mathbf{Y}$, which is $\mathbf{Y}^\top \left(\mathbf{YY}^\top\right)^{-1}$. This can be efficiently computed as it only involves the inversion of an 8×8 matrix. Next, the right pseudo-inverse of $\mathbf{B} = \mathbf{Y}^\top \left(\mathbf{YY}^\top\right)^{-1}$ has to be computed, where $\mathbf{B}$ is of the size of $n_p \times 8$. This way, the Linear Predictor matrix

[3] European Conference on Computer Vision (ECCV)



(a) (b) (c)

Figure 2.7: (a) Comparison of the necessary learning time with respect to the number of sample points used within the template for the approach proposed by Jurie and Dhome [43], by Holzer et al. [31] (referred as "ALPs") and our approach. **(b)** The corresponding speed-up in learning obtained by our approach. **(c)** The tracking time per frame with respect to the number of sample points used for the template. Copyright notice: Springer and the original publisher (Computer Vision - ECCV 2012, 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I, pp 470-483, Online Learning of Linear Predictors for Real-Time Tracking, Stefan Holzer, Marc Pollefeys, Slobodan Ilic, David Joseph Tan, Nassir Navab) is given to the publication in which the material was originally published, by adding: With kind permission from Springer Science and Business Media.

is computed as $\mathbf{A} = \left(\mathbf{B}^\top \mathbf{B}\right) \mathbf{B}^\top$. Again, the necessary inversion involves only an 8×8 matrix. Therefore, the Linear Predictor can be computed significantly more efficient than using the original approach (see Fig. 2.7).

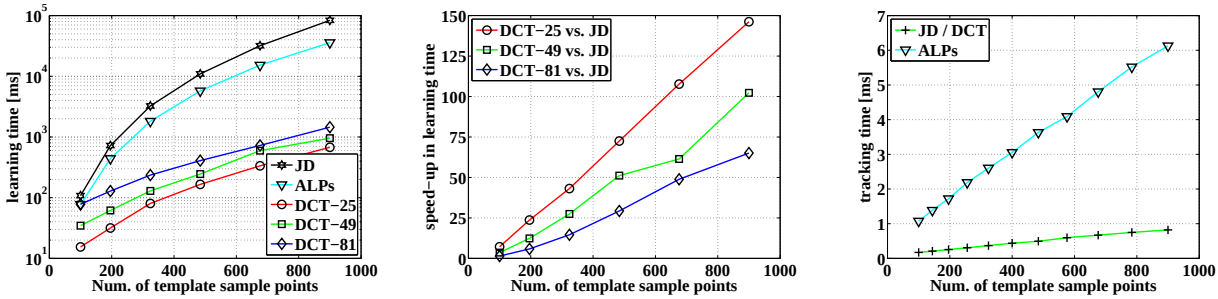
While this alternative formulation of the learning equations achieves similar robustness in tracking under regular conditions, it shows a significant improvement in conditions where a significant amount of noise is present, e. g. under low light conditions. This behavior can be explained by the projection of the training data on low-dimensional spaces. More details on this and further experimental evaluation is given in [35]. The corresponding publication can be found in Appendix B.

2.1.4 ACCV[4] 2012: Efficient Learning of Linear Predictors using Dimensionality Reduction

Another alternative for computing a Linear Predictor is given in [33] (see Appendix C). There, dimensionality reduction is used to reduce the dimensionality of the training matrix $\mathbf{H}$ such that the necessary inversion in the standard learning approach can be applied on a smaller matrix. Therefore, the goal is to reduce the size of $\mathbf{H}$ from $n_p \times n_t$ to $n_r \times n_t$, where $n_r \leq n_p$.

In order to achieve this, a matrix $\mathbf{W}$ of size $n_r \times n_p$ can be created which maps the original training data onto a lower-dimensional space. The original training data $\mathbf{H}$ is then mapped to $\hat{\mathbf{H}} = \mathbf{W}\mathbf{H}$, which shows the desired size. Computing a Linear Predictor from this results in $\hat{\mathbf{A}} = \mathbf{Y}\hat{\mathbf{H}}^\top \left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1}$. To be able to use the obtained Linear Predictor in the standard framework for tracking, i. e. where no dimensionality reduction is explicitly applied, it has to be multiplied on the right hand side with the mapping $\mathbf{W}$, such that

[4] Asian Conference on Computer Vision (ACCV)



(a) (b) (c)

Figure 2.8: Comparison of timings for the approach proposed by Jurie and Dhome [43] ('JD'), the approach of Holzer et al. [31] ('ALPs'), and our approach ('DCT-x'). **(a)** Comparison of learning time. **(b)** Obtained speed-up of our approach with respect to Jurie and Dhome [43]. **(c)** Comparison of tracking time. Copyright notice: Springer and the original publisher (Computer Vision - ACCV 2012, 11th Asian Conference on Computer Vision, Daejeon, Korea, November 5-9, 2012, Revised Selected Papers, Part III, pp 15-28, Efficient Learning of Linear Predictors using Dimensionality Reduction, Stefan Holzer, Slobodan Ilic, David Joseph Tan, Nassir Navab) is given to the publication in which the material was originally published, by adding: With kind permission from Springer Science and Business Media.

the dimensionality reduction is implicitly applied on the image difference data during tracking. Therefore, the final Linear Predictor is obtained as $\mathbf{A} = \hat{\mathbf{A}}\mathbf{W}$.

In [33], it is shown how the Discrete Cosine Transform can be used to construct the desired mapping $\mathbf{W}$. This way, the learning procedure can be tailored to the available amount of processing power by selecting an appropriate value for the reduced dimensionality n_r. Similar to the approach of [35], a speed-up of about two orders of magnitude can be reached, where the exact speed-up depends on n_r and the template size (see Fig. 2.8). Depending on the choice of n_r, a tracking robustness similar to the original approach can be achieved. The corresponding publication can be found in Appendix C.

2.1.5 IJCV[5] 2014: Efficient Learning of Linear Predictors for Template Tracking

In [34], a combination of both approaches for speeding up learning [35, 33] is presented and a thorough comparison between those three approaches is conducted (see Appendix E).

The combination of both approaches is achieved by plugging in $\hat{\mathbf{H}} = \mathbf{W}\mathbf{H}$ from [33] into the learning equation from [35]. This leads to a computation of the Linear Predictor matrix as $\mathbf{A} = \left(\mathbf{B}^\top\hat{\mathbf{H}}^\top\hat{\mathbf{H}}\mathbf{B}\right)^{-1}\mathbf{B}^\top\hat{\mathbf{H}}^\top$, where $\mathbf{B} = \mathbf{Y}^\top\left(\mathbf{Y}\mathbf{Y}^\top\right)^{-1}$. Again, all inversions are applied on small matrices.

As the experiments in [34] show, this ends in a compromise between the advantages and disadvantages of both approaches. The experiments further show that while the reformulation approach of [35] shows a slight drop in tracking robustness under regular conditions it leads to a significantly better robustness under noisy conditions. The approach based on dimensionality reduction [33] on the other hand shows the best results

[5] International Journal of Computer Vision

under regular conditions, even better than the original learning approach, while it can handle noisy conditions only at a level similar to the original approach. The combined approach shares the slightly reduced tracking robustness under regular conditions while it has improved robustness under noisy conditions, although it can not quite reach the robustness of [35]. More details on the evaluation as well as on the derivation of the combined approach are given in [34]. The corresponding publication can be found in Appendix E.

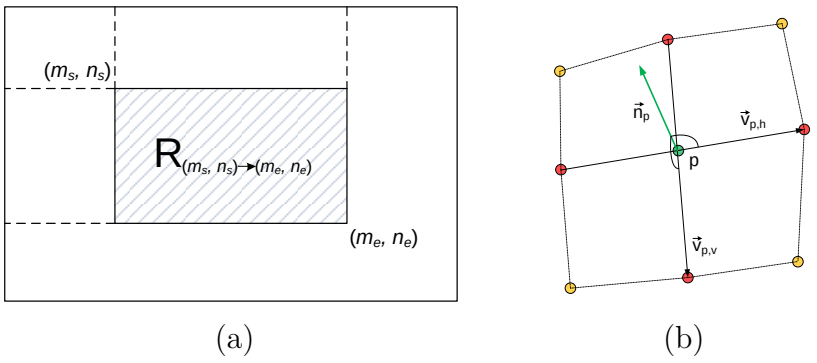


Figure 2.9: (a) The sum of a 2D region can be efficiently computed from an integral image by accessing only four data elements in memory which correspond to the for corners of the rectangular region. **(b)** Estimating a surface normal as cross-product of the vectors between the horizontal and vertical neighbors of the point of interest. ©2012 IEEE

2.2 3D Point Cloud Processing

The following discusses the contributions of this thesis in the field of 3D point cloud processing.

2.2.1 IROS[6] 2012: Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images

In [36], a highly efficient and robust method for estimating surface normals from organized point clouds is introduced, where a dynamic smoothing is applied that adapts based on the depth and neighborhood of the considered points (see Appendix F). An organized point cloud is a cloud where the points can be organized in a regular grid similar to an image. An example for a 3D sensor which provides organized point clouds is the Microsoft Kinect. It provides a 2-dimensional depth map which then can be transformed into a point cloud.

The presented method for surface normal estimation is based on integral images. Integral images are created from an source image by computing for each image pixel location the sum of all pixel values located in the rectangular area between the origin and the current image pixel location (see Fig. 2.9). The advantage is hereby that this computes the sum of pixel values within an area in constant time. Therefore, it is ideally suited for applying a dynamic smoothing.

For estimating the size of the smoothing area for every pixel, [36] uses two different indicators. The first is based on the observation that noise increases with increasing distance from the sensor. Therefore, the further away a point is the larger the smoothing area for the normal estimation will be. However, a problem that arises with large smoothing areas is that if smoothing is applied over objected borders, indicated by jumps in depth,

[6]Intelligent Robots and Systems (IROS)

the estimated normals will be inaccurate. Therefore, the distance to the closest object border or significant depth change is computed and used when determining the smoothing area size. The computation of the distance to the closest object border is done by first estimating all object borders and then computing a corresponding distance map. This can again be done in constant time.

Using these smoothing area sizes, two different ways for computing surface normals using integral images are shown in [36]. The first one is by computing the surface normals from smoothed changes in depth, by computing the perpendicular vector to the depth change in horizontal and vertical direction. The second approach uses covariance matrices, where the single entries of the covariance matrices are computed using integral images.

The experiments, see [36], show that the introduced methods outperform a state-of-the-art k-nearest-neighbors method by computing surface normals much faster while achieving smaller error rates. The corresponding publication can be found in Appendix F.

2.2.2 ECCV[7] 2012: Learning to Efficiently Detect Repeatable Interest Points in Depth Data

The work in the field of interest point detection from 3D point clouds is rather limited. The existing approaches mainly follow the rule of being good, in terms of repeatability and accuracy, but slow, or fast, but then lack robustness in repeatability and accuracy. In [37], a new interest point detector is introduced which is learned using regression trees in order to achieve high robustness as well as real-time performance (see Appendix G). Although this is not the first work on learning-based interest point detection it is the first that has the explicit goal of not only mimicking an existing detector but also to optimize the detection with respect to some optimality criteria.

The optimality criteria that are considered are common goals of interest point detectors: sparseness, repeatability, distinctiveness, and efficiency. The work hereby focuses on repeatability as sparseness is usually controlled using thresholds, distinctiveness is dependent on the used descriptors, and efficiency is implicitly obtained by using regression trees.

As a baseline, the learning approach uses an interest point detector based on high curvature, i. e. a detector that gives high responses for points with high curvature, e. g. corner points, and low responses for planar regions. Since this detector is not optimized for repeatability, a training data set is used to estimate a response which maximizes repeatability. For this, a set of pre-registered 3D point clouds is used. First, the base line detector is applied to obtain its responses for each point. Then, these responses are accumulated in a volume by summing up the response values of all points that fall into the same voxel of the volume. Using this and a visibility count for every voxel, a synthetic response volume is created and mapped back into the single point clouds of the training sets. This way a point receives a high response if it is likely to obtain high scores in many of the point clouds. The pipeline for computing these artificial responses is shown in Fig. 2.10. To further emphasize these characteristics, a non-max suppression is applied on the resulting responses (see Fig. 2.11).

[7] European Conference on Computer Vision (ECCV)

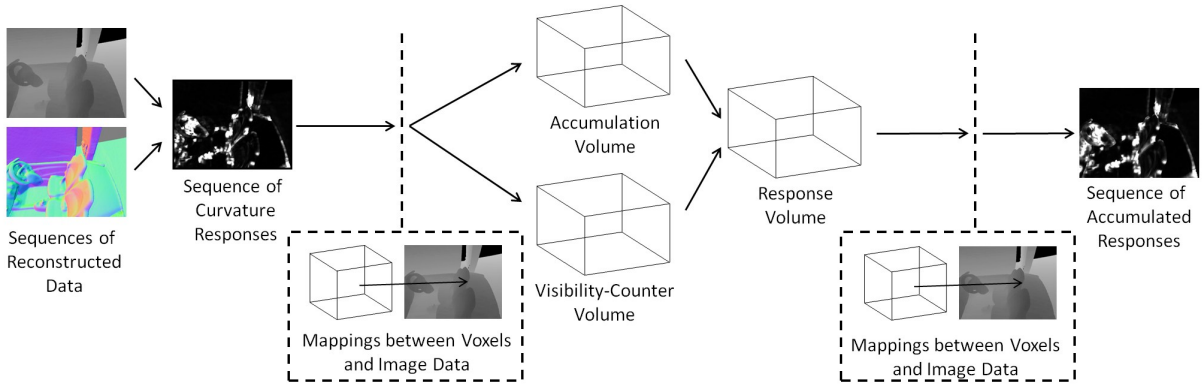


Figure 2.10: Illustration of the synthetic response computation. Copyright notice: Springer and the original publisher (Computer Vision - ECCV 2012, 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I, pp 200-213, Learning to Efficiently Detect Repeatable Interest Points in Depth Data, Stefan Holzer, Jamie Shotton, Pushmeet Kohli) is given to the publication in which the material was originally published, by adding: With kind permission from Springer Science and Business Media.

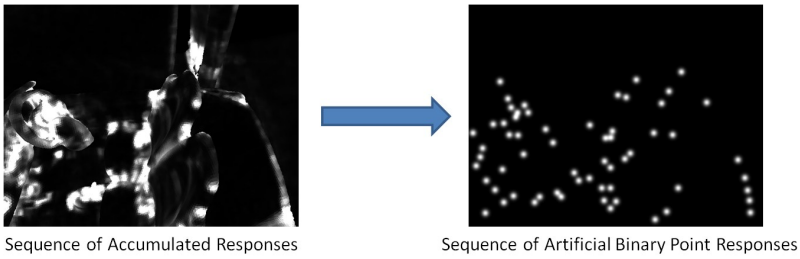


Figure 2.11: Computing artificial response maps from accumulated responses. Copyright notice: Springer and the original publisher (Computer Vision - ECCV 2012, 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I, pp 200-213, Learning to Efficiently Detect Repeatable Interest Points in Depth Data, Stefan Holzer, Jamie Shotton, Pushmeet Kohli) is given to the publication in which the material was originally published, by adding: With kind permission from Springer Science and Business Media.

Experiments show that the resulting interest point detector outperforms state-of-the-art approaches in robustness and speed [37]. The corresponding publication can be found in Appendix G.

CHAPTER

THREE

CONCLUSIONS & OUTLOOK

3.1 Visual Tracking

3.1.1 Conclusion

The contributions of this thesis in the field of visual tracking addressed several major challenges and limitations of learning-based template tracking. The introduction of Adaptive Linear Predictors (ALPs) [31] demonstrated that Linear Predictors can be adapted in a flexible and efficient way without the need to recompute them completely. This helps tracking in multiple ways: templates can be efficiently learned by starting with a comparably small template and growing it over time, such that learning unknown objects and scenes becomes feasible without interrupting the flow of a process or application. Further, the template can freely adapt on the shape of the object of interest by choosing the parts of the object that are best suited for robust tracking and it can adapt to occlusions, either caused by other objects, self-occlusion, or by moving the object partly out of the field of view.

In [32], it was demonstrated that occlusions can efficiently and robustly be identified using the concept of secure, insecure, and occluded regions. This way, occlusions can be handled using Adaptive Linear Predictors by tracking only secure regions and temporarily removing insecure and occluded regions from the template. Furthermore, a robustification approach was presented which improved the robustness with respect to large motions.

Although Adaptive Linear Predictors made it possible to learn template efficiently over time, an instant learning was still not possible at run-time. Therefore, a reformulation of the learning process was introduced in [35], which bypasses the computationally most expensive part of the learning, the inversion of very large matrices. This showed that the speed of learning Linear Predictors can be increased by a factor of two orders of magnitude while keeping a similar robustness in tracking. Even more, this introduced a significant improvement in robustness with respect to image noise. Therefore, it is of advantage when tracking in low-light conditions is necessary.

An alternative way of improving the learning speed was presented in [33] where it was demonstrated that a dimensionality reduction can be applied on the learning data

without loosing robustness in tracking. It was shown that the Discrete Cosine Transform (DCT) is a suitable technique for achieving the task of dimensionality reduction and that, depending on the template size and the number of used DCT-coefficients, a speed-up in learning in the range of two orders of magnitudes can be obtained. Furthermore, the obtained tracking robustness was superior to the original learning approach.

A combination of both the reformulation and the dimensionality reduction methods [34] allows a trade-off between robustness under noisy image conditions, e. g. in low-light, and tracking robustness in general cases, as the reformulation approach tends to be slightly less robust under these conditions.

3.1.2 Outlook

Despite the advances in learning-based tracking provided by this thesis, there is still need for improvements in certain areas. In the following, open problems and interesting future directions for research are identified.

One major limitation of Linear Predictors is that, up to now, they can only be applied on planar regions. Being able to track non-planar regions would allow tracking of complete scenes without having to divide the scene into single small planar templates. Having a large single template for this is likely to increase tracking robustness [31]. One could even think of having a SLAM system running on this where new parts are continuously added to the template. Parts that get out of the field of view can be removed from the template and stored in a database until they become visible again.

Another interesting direction of research would be the type of transformation used for tracking. So far, it has only been demonstrated that transformations up to homographies can be successfully modeled using Linear Predictors. However, it has not been shown yet whether or how they can cope with transformations in 3D space. Tracking a 3D pose would also be necessary for using Linear Predictors in SLAM approaches. Furthermore, deformable models could be considered for tracking objects. This would significantly increase the possible areas where Linear Predictors can be used for. Applications can be in the field of augmented reality where deformable surfaces are tracked and replaced by a different content, or in the field of human-computer interaction where a face can be tracked.

Since specific deformable objects, such as faces, are often treated using a special parametrization model, and not only a general deformable model, another direction for research is to investigate in how far the parameters of such models can be estimated and tracked using Linear Predictors.

Besides using image data for tracking, it would be interesting to figure out whether Linear Predictors can also efficiently be used for tracking in 3D data or depth maps, as they are provided by the Microsoft Kinect. Tracking 3D objects in 3D data is still a very computationally expensive task and having a fast and robust method for tracking those objects would benefit a large area of applications. Especially robotic applications, which often have access to 3D sensors, such as object detection and tracking, navigation, scene mapping would greatly benefit from this.

Finally, a new trend in template tracking is to use channel-based methods. These are

methods that, instead of using image intensity values for tracking, use multi-dimensional vectors which are comparable to histograms. These histograms contain only a single entry at the bin which belongs to the intensity value present at the pixel while all other entries are zero. For increased robustness, blurring is applied on the histogram. It would be interesting to see whether this concept can be used together with Linear Predictor and if, what the impact on the robustness of the tracking would be.

3.2 3D Point Cloud Processing

3.2.1 Conclusion

Surface normal estimation is a fundamental technique in 3D point cloud processing and builds the foundation for a large set of higher level approaches. In [36], it was demonstrated that the speed of the computation of surface normals from organized point cloud data can be significantly increased by using integral images. This allows flexible smoothing areas for every 3D point and therefore provides robust normal estimates.

In the field of 3D interest point detection, a new way of learning optimal interest points was shown to be superior to existing detectors [37]. Instead of trying to only mimic the response of an existing interest point detector, as done in previous work in the 2D image domain, it was shown that optimizing for specific intended characteristic of an interest point, such as repeatability, leads to a significant improvement. This way, a fast and reliable detection of interest points was made possible.

3.2.2 Outlook

While surface normal estimation can be done efficiently now, the presented method still shows problems at the borders of objects, where the smoothing area is reduced and therefore, noise is still an issue. This especially holds as a problem for object detection tasks where small objects have to be found. If an object only covers a small area in the resulting 3D data, almost no smoothing can be applied with the presented method. This becomes even more problematic if the object is built from small thin structures.

A problem that is present in the current work in designing interest point detectors and feature descriptors is that both approaches are usually done in different, independent steps. Only later it is checked which detectors fit well with which descriptors. For example, the presented approach for learning to detect interest points only optimizes the repeatability of a specific existing interest point detector. However, since it is highly important that the used interest point detector fits optimally to the used feature descriptor, an interesting future direction of research is to consider both concepts in the learning approach. A first step would be to learn a detector which is optimally suited for a specific descriptor, which could be done by training for points that are robust to local changes, in terms of stable descriptor values, but unique in a global perspective. If this is successful, a learning approach where both, the interest point detector and the corresponding descriptor are learned at the same time, could be thought of.

APPENDIX

A

ADAPTIVE LINEAR PREDICTORS FOR REAL-TIME TRACKING

Own contributions. My contributions to this work include the core idea, the design, and implementation of the methods for adapting Linear Predictors online as well as the evaluation of those. The core idea of the paper was refined in collaboration with the other co-authors. All co-authors were involved in the writing of the paper.

Adaptive Linear Predictors for Real-Time Tracking

Stefan Holzer, Slobodan Ilic, Nassir Navab

Department of Computer Science, Technical University of Munich (TUM)
Boltzmannstrasse 3, 85748 Garching, Germany
{holzers,slobodan.ilic,navab}@in.tum.de

Abstract

Enlarging or reducing the template size by adding new parts, or removing parts of the template, according to their suitability for tracking, requires the ability to deal with the variation of the template size. For instance, real-time template tracking using linear predictors, although fast and reliable, requires using templates of fixed size and does not allow on-line modification of the predictor. To solve this problem we propose the Adaptive Linear Predictors (ALPs) which enable fast online modifications of pre-learned linear predictors. Instead of applying a full matrix inversion for every modification of the template shape as standard approaches to learning linear predictors do, we just perform a fast update of this inverse. This allows us to learn the ALPs in a much shorter time than standard learning approaches while performing equally well.

We performed exhaustive evaluation of our approach and compared it to standard linear predictors and other state of the art approaches.

1. Introduction

Template tracking has been extensively studied and used in many computer vision applications such as vision-based control, human-computer interfaces, surveillance, medical imaging and visual reconstruction.

While there are many template tracking approaches based on the analytical derivation of the Jacobian [14, 19, 9, 5, 6, 1, 2, 15, 3, 4], learning-based methods [12, 13, 8, 18, 16, 17, 20] have proved to allow faster tracking and are generally more robust with respect to large perspective changes.

A very successful learning based template tracker was proposed by Jurie and Dhome [12]. It is based on learning linear predictors to efficiently compute template parameter updates. The costly off-line learning phase, however, prohibits this method from computing templates of varying size online.

Figure 1. An initial small template is enlarged according to a tracking quality measure. The template is tracked over time and reduced if parts of it go out of the image. The removed parts are reinserted as soon as they become visible.

Yet, the ability to dynamically change the template size is necessary in applications such as indoor SLAM. The fact that the 3D geometry of the scene is a priori unknown makes it necessary to initially rely on planar structures. In this case it is preferable to start from small-sized templates, in order to reduce the risk of loosing track due to non-planar structures, and to grow or shrink them online. Thus, the learning of large templates can be distributed over multiple frames while keeping the failure rate low. In combination with a planarity check this strategy enables online segmentation of planar structures and the reliable maintenance of large templates. As a result, the set of initially tracked templates evolves towards a relatively small number of comparably large, optimally shaped templates, yielding increased robustness.

Current learning based tracking approaches, like [12], use templates of fixed size, because the computation of the linear predictors requires the costly inversion of a large, template specific matrix. Since this is the computationally most expensive part of the learning process, the ef-

fort for changing the template size is nearly equivalent to that of learning a new template from scratch. Therefore, to overcome the limitations of fixed size template approaches, while maintaining their robustness to large perspective changes, we propose an extension to linear predictors which allows efficient online modification of the template size. Instead of computing the inversion of the whole matrix every time the template shape changes, we introduce a way to update the inverse computationally efficient which dramatically reduces the time needed for learning. We start with a small initial template and grow it by small extension templates as defined in Fig. 3 according to their suitability for tracking. As long as the object to track is planar, our approach can grow the template in any direction which can result in an arbitrarily shaped template, as shown in Fig. 1. This breaks the standard, rectangular shape assumption widely used in current template tracking approaches and can be seen as a first step towards a dense SLAM system.

We perform extensive quantitative testing and compare our approach to the standard approach of Jurie and Dhome [12] under different transformations and noise levels, and to other state-of-the-art approaches in template tracking. We demonstrate that our approach performs equally well while requiring much shorter learning time. In the remainder of the paper we will discuss related work on template tracking, give a detailed description of our approach, present our results and show examples on real world sequences.

2. Related Work

A lot of effort has been made in the field of template tracking and image alignment since the work of Lucas and Kanade [14]. Most of the presented approaches can be put into one of the two categories: template tracking based on the analytical derivation of the Jacobian [14, 19, 9, 5, 6, 1, 2, 15, 3, 4] or based on learning [12, 13, 8, 18, 16, 17, 20]. While the analytical approaches generally are more flexible with respect to the template shape modification at run-time, learning approaches enable higher tracking speed and are more robust with respect to large perspective changes.

Since the seminal work of Lucas and Kanade [14] a large variety of analytical tracking approaches has been presented. Amongst others, these variations include different update rules of the warp [14, 9, 5, 19, 6, 1], different orders of approximations of the error function [15, 3, 4], occlusion and illumination change handling [9]. Basically, there are four different types of update rules, the additive approach [14], the compositional approach [19], the inverse additive approach [9, 5], and the inverse compositional approach [6, 1]. In the latter two, the roles of the reference and current image are switched, which makes it possible to move some of the computations into an initialization phase, that makes the tracking computationally very efficient. Faster convergence rate for a larger convergence area can be additionally obtained by using a second-order instead of a first order approximation of the error function [15, 3, 4]. Furthermore, Hager and Belhumeur [9] show how illumination changes and occlusions can be efficiently handled. For a more detailed overview over analytical tracking methods refer to Baker and Matthews [2].

In contrast to analytical tracking methods, Jurie and Dhome [12] propose an approach that learns linear predictors using randomly warped samples of the initial template. The linear predictors are then used to predict the parameter updates during tracking. This allows a very fast tracking, since the "Jacobians" are initially computed once and for all and the update parameters can be obtained by simple matrix vector multiplications. In [13] the authors also extend the approach in order to handle occlusion. Gräßl *et al*. [7] additionally shows how the robustness of the linear predictor based approach can be further increased with regard to illumination changes. They [8] also present an intelligent way how to select the points for sampling the image data, such that the accuracy of the tracking is increased. Another linear predictor approach [20] describes a template using many small templates and tracks these small templates independently. Based on the local movements of these small templates they estimate the movement of the large template. Instead of using linear predictors, Mayol and Murray [17] present an approach that fits the sampling region to pretrained samples using general regression.

All the proposed learning approaches, however, do not deal with templates of variable size. To overcome this limitation we developed a method that extends the approach of Jurie and Dhome [12] to allow online template size adaptation.

3. Background and Terminology

In this section we introduce notation and, for the sake of completeness, review the original template tracking approach proposed by Jurie and Dhome [12].

3.1. Template and Parameter Description

A template consists of a set of n_p sample points, which are distributed within the template region and are used to sample image data. The template parameters $\boldsymbol{\mu}$ describe the current deformation of the template within an image. Within this paper we use a homography to represent the current perspective distortion of a planar template and parameterize it using four points as shown in Fig. 2. Note that our approach can also be easily adapted to any other parameterizable template deformation.

The sample points are arranged in a regular grid and grouped together into subsets of four points as shown in

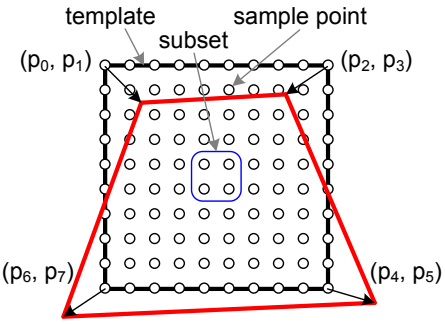


Figure 2. A template is represented by a set of regularly placed sample points, which are grouped into subsets of four points. The pose of a template is parameterized using four corner points.

Fig. 2. The usefulness of this grouping will be justified later in Section 4.1, when we describe our approach for template extension. However, neither the approach of Jurie and Dhome [12] nor our approach are restricted to this special kind of sample point arrangement. The image values obtained from the sample points, warped according to the current template parameters $\boldsymbol{\mu}$, are arranged in a vector $\mathbf{i} = \left(i_1, i_2, \ldots, i_{n_p}\right)^T$.

3.2. Template Tracking based on Linear Predictors

The goal of template tracking is to follow a reference template, defined by a vector $\mathbf{i}_R$ of reference image values and an initial parameter vector $\boldsymbol{\mu}_R$, over a sequence of images. The basic approach for this is to compute a vector $\delta\mathbf{i} = \mathbf{i}_C - \mathbf{i}_R$ of image differences, where the vector $\mathbf{i}_C$ stores the image values extracted from the current image. This vector is then used to estimate a vector of parameter differences $\delta\boldsymbol{\mu}$ used to update the current template parameters $\boldsymbol{\mu}$ such that the position of the template within the current image is optimized.

Instead of explicitly minimizing an error function, *e.g.* by iteratively solving a first- or second-order approximation of it, Jurie and Dhome [12] use a learned matrix $\mathbf{A}$ to compute $\delta\boldsymbol{\mu}$ based on the vector $\delta\mathbf{i}$ as:

$$\delta\boldsymbol{\mu} = \mathbf{A}\delta\mathbf{i}. \tag{1}$$

Here, the matrix $\mathbf{A}$ can be seen as a linear predictor. In order to learn $\mathbf{A}$ we apply a set of n_t random transformations to the initial template. This is done by applying small disturbances $\delta\boldsymbol{\mu}_i, i = 1, \ldots, n_t$, to the reference parameter vector $\boldsymbol{\mu}_R$. Then, each of these transformations is used to warp the sample points in order to obtain the corresponding vectors $\mathbf{i}_i$ of image values. The image value vector $\mathbf{i}_R$, obtained using the reference parameters $\boldsymbol{\mu}_R$, is used to compute the image difference vectors $\delta\mathbf{i}_i = \mathbf{i}_i - \mathbf{i}_R$ for each of the random transformations. These vectors of parameters and image differences are combined in the matrices $\mathbf{Y} = \left(\delta\boldsymbol{\mu}_1, \ldots, \delta\boldsymbol{\mu}_{n_t}\right)$ and $\mathbf{H} = \left(\delta\mathbf{i}_1, \ldots, \delta\mathbf{i}_{n_t}\right)$. In general, n_t is chosen such that it is much bigger than n_p. Using these matrices Equ. 1 can be written as $\mathbf{Y} = \mathbf{AH}$. Finally, the matrix $\mathbf{A}$ is learned using

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T\right)^{-1}. \tag{2}$$

In practice, we normalize the extracted image data with zero mean and unit standard deviation, which increases the robustness against illumination changes. In order to prevent $\mathbf{H}\mathbf{H}^T$ from being rank deficient we add random noise to the obtained image value difference vectors. Additionally, we apply a multi-predictor approach, where multiple linear predictors $\mathbf{A}_1, \ldots, \mathbf{A}_{n_l}$ are learned for one template, with n_l being the number of predictors per template. Thereby, the first linear predictor $\mathbf{A}_1$ is learned for large motions $\delta\boldsymbol{\mu}_i$ and the following predictors are learned for subsequently smaller motions. During tracking we iteratively apply the linear predictors. Additionally, every predictor is used multiple times. Within this paper we use five different predictors per template and three iterations for each of the predictors.

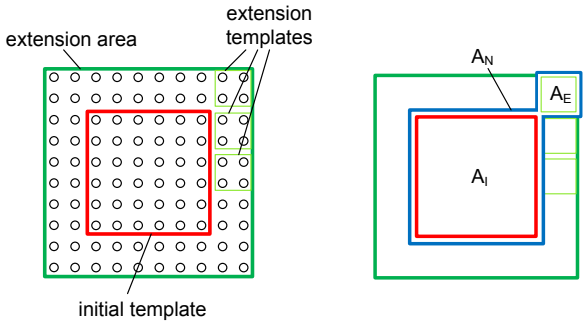


Figure 3. **(a)** The initial template together with possible extension templates defined by the corresponding extension area. **(b)** Different template areas and their corresponding linear predictors. The red border defines the initial template with its predictor $\mathbf{A}_I$, the light green border defines an extension template with its predictor $\mathbf{A}_E$ and the blue border defines the new extended template with its predictor $\mathbf{A}_N$.

4. Template Adaption

In this section we describe our approach for adapting the template by extending or reducing its size. This enables to start tracking with a small-sized template and grow or shrink it over time, automatically adapting its size and corresponding linear predictor according to the tracked scene.

4.1. Template Extension

In the following we denote the linear predictor of an initial template with $\mathbf{A}_I$, and the linear predictor of an extension template with $\mathbf{A}_E$ as depicted in Fig. 3. Using the standard approach of Sec. 3.2, the separate predictors would be

learned as:

$$\mathbf{A}_I = \mathbf{Y}\mathbf{H}_I^T \left(\mathbf{H}_I\mathbf{H}_I^T\right)^{-1} \text{ and} \quad (3)$$

$$\mathbf{A}_E = \mathbf{Y}\mathbf{H}_E^T \left(\mathbf{H}_E\mathbf{H}_E^T\right)^{-1}, \quad (4)$$

where $\mathbf{Y}$ stores the same random transformations for both linear predictors. The standard approach for learning a combined predictor $\mathbf{A}_N$ for the entire template leads to:

$$\mathbf{A}_N = \mathbf{Y}\mathbf{H}_N^T \left(\mathbf{H}_N\mathbf{H}_N^T\right)^{-1} \quad (5)$$

$$= \mathbf{Y}\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}^T \left(\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}^T\right)^{-1} \quad (6)$$

$$= \mathbf{Y}\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}^T \left(\begin{bmatrix} \mathbf{H}_I\mathbf{H}_I^T & \mathbf{H}_I\mathbf{H}_E^T \\ \mathbf{H}_E\mathbf{H}_I^T & \mathbf{H}_E\mathbf{H}_E^T \end{bmatrix}\right)^{-1}. \quad (7)$$

Now, instead of directly updating the old linear predictor $\mathbf{A}_I$ we will update the matrix $\mathbf{S}_I = \left(\mathbf{H}_I\mathbf{H}_I^T\right)^{-1}$ using the formulas presented by Henderson and Searle [10], such that we obtain the matrix $\mathbf{S}_N = \left(\mathbf{H}_N\mathbf{H}_N^T\right)^{-1}$. Let $\mathbf{S}_{11}$, $\mathbf{S}_{12}$, $\mathbf{S}_{21}$ and $\mathbf{S}_{22}$ be the four sub-matrices of $\mathbf{S}_N$:

$$\mathbf{S}_N = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{H}_I\mathbf{H}_I^T & \mathbf{H}_I\mathbf{H}_E^T \\ \mathbf{H}_E\mathbf{H}_I^T & \mathbf{H}_E\mathbf{H}_E^T \end{bmatrix}\right)^{-1}. \quad (8)$$

Then, we can update $\mathbf{S}_I$ to $\mathbf{S}_N$ using

$$\mathbf{S}_{11} = (\mathbf{H}_I\mathbf{H}_I^T)^{-1} + (\mathbf{H}_I\mathbf{H}_I^T)^{-1}\mathbf{H}_I\mathbf{H}_E^T\mathbf{S}_{22}\mathbf{H}_E\mathbf{H}_I^T(\mathbf{H}_I\mathbf{H}_I^T)^{-T} \quad (9)$$

$$\mathbf{S}_{12} = -(\mathbf{H}_I\mathbf{H}_I^T)^{-1}\mathbf{H}_I\mathbf{H}_E^T\mathbf{S}_{22}, \quad (10)$$

$$\mathbf{S}_{21} = \mathbf{S}_{12}^T, \quad (11)$$

$$\mathbf{S}_{22} = \left(\mathbf{H}_E\mathbf{H}_E^T - \mathbf{H}_E\mathbf{H}_I^T(\mathbf{H}_I\mathbf{H}_I^T)^{-1}\mathbf{H}_I\mathbf{H}_E^T\right)^{-1}, \quad (12)$$

where $(\mathbf{H}_I\mathbf{H}_I)^{-1}$ is known from the learning of the initial predictor. Therefore, the only inversion that has to be applied is for the computation of $\mathbf{S}_{22}$. However, this inversion is not a problem since the extension templates are always of smaller size than the entire extended template and therefore $\mathbf{S}_{22}$ is small, as well.

The approach as presented up to now is limited by the number of random transformations n_t, used for learning. Since n_t has to be the same for all extension templates as well as for the initial template, and since the number of random transformations has to be greater or at least equal to the number of used sample points, $n_t \geq n_p$, the maximum number of random transformations has to be known a priori. In order to remove this restriction we use the approach presented by Hinterstoisser *et al*. [11], which allows to update the matrix $\mathbf{S}_I$ in a way such that we can increase the number of random transformations n_t without the necessity to recompute the updated $\hat{\mathbf{S}}_I$ from scratch. This is done by using the Sherman-Morrison formula:

$$\hat{\mathbf{S}}_I = \left(\mathbf{S}_I^{-1} + \delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^T\right)^{-1} \quad (13)$$

$$= \mathbf{S}_I - \frac{\mathbf{S}_I\delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^T\mathbf{S}_I}{1 + \delta\mathbf{i}_{n_t+1}^T\mathbf{S}_I\delta\mathbf{i}_{n_t+1}}, \quad (14)$$

where $\delta\mathbf{i}_{n_t+1}$ is a vector of image value differences obtained from a new random transformation applied to the sample points. In practice, the number of random transformations is increased each time before a new extension template is added.

4.2. Template Reduction

In case that already learned templates have to be reduced, *e.g*. due to the presence of non-planarity or shortcoming for tracking, the corresponding linear predictors can be computed by updating the linear predictor of the larger template. For this, we denote the linear predictor of the large template with $\mathbf{A}_L$, the predictor of the new reduction template with $\mathbf{A}_R$ and the predictor of the reduced template with $\mathbf{A}_N$.

In order to reduce the matrix $\mathbf{S}_L$, it has to be rearranged first, such that the data corresponding to the reduction template is positioned in the last rows and columns of $\mathbf{S}_L$. After the rearrangement, the reduction template can be removed using the following approach. First, let us consider the sub-matrices of the matrix $\mathbf{S}_L$:

$$\mathbf{S}_L = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{H}_N\mathbf{H}_N^T & \mathbf{H}_N\mathbf{H}_R^T \\ \mathbf{H}_R\mathbf{H}_N^T & \mathbf{H}_R\mathbf{H}_R^T \end{bmatrix}\right)^{-1}, \quad (15)$$

where all the sub-matrices $\mathbf{S}_{11}$, $\mathbf{S}_{12}$, $\mathbf{S}_{21}$, $\mathbf{S}_{22}$, $\mathbf{H}_N\mathbf{H}_N^T$, $\mathbf{H}_N\mathbf{H}_R^T$, $\mathbf{H}_R\mathbf{H}_N^T$ and $\mathbf{H}_R\mathbf{H}_R^T$ are available from the large template. The goal is to compute

$$\mathbf{A}_N = \mathbf{Y}\mathbf{H}_N^T \left(\mathbf{H}_N\mathbf{H}_N^T\right)^{-1} \quad (16)$$

without the need of inverting $\mathbf{H}_N\mathbf{H}_N^T$, since this is a large matrix in general. Similar to the Equations 9-12 Henderson and Searle [10] also presents the formula

$$\mathbf{S}_{11} = \left(\mathbf{H}_N\mathbf{H}_N^T - \mathbf{H}_N\mathbf{H}_R^T(\mathbf{H}_R\mathbf{H}_R^T)^{-1}\mathbf{H}_R\mathbf{H}_N^T\right)^{-1}, \quad (17)$$

which can be reformulated as

$$\mathbf{H}_N\mathbf{H}_N^T = \mathbf{S}_{11}^{-1} + \mathbf{H}_N\mathbf{H}_R^T(\mathbf{H}_R\mathbf{H}_R^T)^{-1}\mathbf{H}_R\mathbf{H}_N^T. \quad (18)$$

Taking the inverse leads to the desired result:

$$(\mathbf{H}_N\mathbf{H}_N^T)^{-1} = \left(\mathbf{S}_{11}^{-1} + \mathbf{H}_N\mathbf{H}_R^T(\mathbf{H}_R\mathbf{H}_R^T)^{-1}\mathbf{H}_R\mathbf{H}_N^T\right)^{-1}. \quad (19)$$

Since we, however, have to invert a big matrix in this case, namely $\mathbf{S}_{11}$, this is not suitable for online computation. Therefore, we use the following formula presented in [10]:

$$(\mathbf{X} + \mathbf{U}\mathbf{Y}\mathbf{U}^T)^{-1} = \mathbf{X}^{-1} - \mathbf{X}^{-1}\mathbf{U}\mathbf{Z}\mathbf{U}^T\mathbf{X}^{-1}, \quad (20)$$

$$\mathbf{Z} = \left(\mathbf{Y}^{-1} + \mathbf{U}^T\mathbf{X}^{-1}\mathbf{U}\right)^{-1}. \quad (21)$$

By setting $\mathbf{X} = \mathbf{S}_{11}^{-1}$, $\mathbf{Y} = (\mathbf{H}_R\mathbf{H}_R^T)^{-1}$ and $\mathbf{U} = \mathbf{H}_N\mathbf{H}_R^T$ we obtain our desired result:

$$(\mathbf{H}_N\mathbf{H}_N^T)^{-1} = \mathbf{S}_{11} - \mathbf{S}_{11}\mathbf{H}_N\mathbf{H}_R^T\mathbf{D}\mathbf{H}_R\mathbf{H}_N^T\mathbf{S}_{11}, \quad (22)$$

$$\mathbf{D} = \left(\mathbf{H}_R\mathbf{H}_R^T + \mathbf{H}_R\mathbf{H}_N^T\mathbf{S}_{11}\mathbf{H}_N\mathbf{H}_R^T\right)^{-1} \quad (23)$$

Now, the necessary inversion is no longer a problem since the reduction template is chosen to be of small size and computing $\mathbf{D}$ is not expensive.

4.3. Practical Issues

In this section we discuss practical issues. These are the normalization of the image data and the estimation of the subset quality, which is used for the selection of the next extension template.

4.3.1 Normalization

As mentioned before, the image values are normalized to zero mean and unit standard deviation. However, instead of doing this globally by considering all image values of the template we apply a local normalization, where each subset is normalized by considering only its image values and the image values of its direct local neighboring subsets. This normalization is applied to the reference data, the learning data and the current image data during tracking. The local normalization is superior to the global normalization since in case of the global normalization the mean and standard deviation of the whole image data change if new parts are added to the template or some parts are removed.

4.3.2 Suitability Criterion for Subset Selection

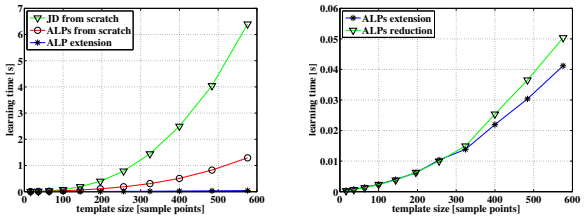
In order to decide which subset should be chosen for extending the current template we compute a quality measure for each of the potential extension templates in the local neighborhood of the current template. This is done by learning a local predictor $\mathbf{A}_\mathrm{S} = \mathbf{Y}_\mathrm{S}\mathbf{H}_\mathrm{S}^T(\mathbf{H}_\mathrm{S}\mathbf{H}_\mathrm{S}^T)^{-1}$ for this subset at first, where the image data $\mathbf{H}_S$ is collected using the set of random transformations represented by $\mathbf{Y}$. Then, using this predictor together with the collected image data we compute a prediction $\hat{\mathbf{Y}}_\mathrm{S}$ of $\mathbf{Y}$ as

$$\hat{\mathbf{Y}}_\mathrm{S} = \mathbf{A}_\mathrm{S}\mathbf{H}_\mathrm{S}. \quad (24)$$

Finally, we compute a similarity measurement, which defines the quality q_S of the corresponding subset as

$$q_\mathrm{s} = \frac{1}{n_t}\sum_{i=1}^{n_t}\frac{\hat{\mathbf{y}}_{\mathrm{s}i}\mathbf{y}_i^T}{|\hat{\mathbf{y}}_{\mathrm{s}i}||\mathbf{y}_i|}, \quad (25)$$

where $\mathbf{y}_i$ and $\hat{\mathbf{y}}_{\mathrm{s}i}$ are the i-th column vector of $\mathbf{Y}$ respectively $\hat{\mathbf{Y}}_\mathrm{S}$. The current template will then be extended using the subset with the highest quality measure.



(a) (b)

Figure 4. Comparison of the computation time necessary for learning a linear predictor using Jurie-Dhome approach (green) and using ALPs (red and blue). **(a)** For the later case we distinguish between learning the predictor from scratch (red) and adding only one extension subset (blue) at a time. Learning from scratch means that we consider the whole time necessary to build up the template of the specified size. **(b)** Computation times for template extension and reduction, when one extension subset is added at a time.

5. Experimental Results

In this section we perform extensive comparison of our approach with several state of the art approaches on template tracking. This includes comparisons with the standard learning approach of Jurie and Dhome [12], the analytical approach of Benhimane and Malis [4] and a recent approach called NoSLLip of Zimmermann *et al*. [20]. The comparisons are done in terms of tracking precision and computational efficiency. In the end we show several qualitative results from real video sequences showing tracking results with one and several templates. All experiments are performed on a 2.66 GHz Intel(R) Core(TM)2 Quad CPU with 8 GB of RAM, where only one core is used for the computations.

In all experiments the maximum random perturbation applied for learning the linear predictors is set to 21 pixels except for the comparison with NoSLLip, where we slightly increased the perturbation by 10% to make the tracking more robust against large motions.

5.1. Comparison with Jurie-Dhome Approach

Computational Complexity of Learning In Fig. 4 we show computation times for learning the linear predictors with respect to different template sizes. We compare our ALPs method, shown in red and blue, with the standard approach of Jurie and Dhome [12], depicted as green curve in Fig. 4(a). For our approach we distinguish between two cases. In the first case, shown as a red curve, the computation of the linear predictor is done iteratively from scratch. In that case we start with a small initial template, whose size is equal to the size of an extension template of Fig. 3. Such a small template is then grown until the specified size is reached. The obtained results reveal clearly that the adaptive learning of the linear predictor, which starts with the small sized template, is much more efficient than learning a linear predictor for the fixed size template. This proves

that our approach can also be used to efficiently learn linear predictors for templates of fixed size, starting from small templates and adapting their linear predictors until the desired template size is reached. In the second case, shown as a blue curve, we show the time necessary to add one extension template. This is a typical case during online tracking, where the template is grown step by step. As to be expected, adding the extension template does not significantly increase computation time, when changing the template size. In Fig. 4 (b) we show computation times for extension and reduction of templates. Note that the necessary time to grow or reduce the template by an extension template consisting of four sample points is around 0.05s for initial templates of sizes around 600 sample points, whereas the computation from scratch would need over 1s using ALPs and more than 6s when using the approach of Jurie and Dhome [12].

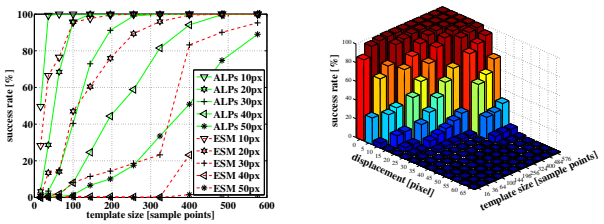
Robustness To evaluate the robustness of our approach we compare the tracking success rate of our approach with that of the standard approach proposed by Jurie and Dhome [12] for different template sizes and with respect to changes in translation (Fig. 7 (a)), in-plane rotation (Fig. 7 (b)), viewing angle (Fig. 7 (c)), and scale (Fig. 7 (d)). In addition we compare ALPs to Jurie and Dhome in respect to noise and different number of random transformations used for learning. The results are shown in Fig. 6.

For all experiments, we use synthetic images, corrupted by noise and warped according to the specific experiments. Noise is added according to $I_n(\mathbf{x}) = I(\mathbf{x}) + \epsilon$, with $\epsilon \in [-\alpha I_{\text{range}}/100, \alpha I_{\text{range}}/100]$, and $\alpha = 5$ for all experiments. An exception is the noise experiment, where different levels of noise were applied. I_{range} specifies the possible range of image values, *e.g.* $I_{\text{range}} = 255$ holds for image values between 0 and 255. In all experiments we also add a random displacement in the range of $[-5, 5]$ pixels, with the exception of the displacement experiments, and a random change in the view-point angle ranging between $[-5, 5]$ degree, again with the exception of the view-point angle experiments.

The results show that both approaches, the standard Jurie-Dhome approach as well as ALPs, yield similar success rates. The only exception is the sensitivity to noise, where the Jurie-Dhome approach performs better than ALPs. This performance difference, however, can be reduced by increasing the number of random warpings used for learning the linear predictors, as demonstrated in Fig. 6 (b).

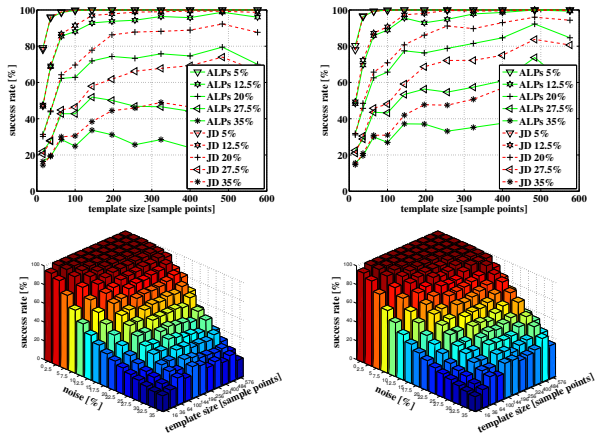
5.2. Comparison with ESM

To demonstrate the usefulness of learning-based approaches we compare our approach with the analytical method of Benhimane and Malis [4]. For this, we have



(a) (b)

Figure 5. Comparison of success rates with respect to different displacements and template sizes. **(a)** Performance of ESM vs. ALPs. **(b)** Performance of ESM for further displacements.



(a) (b)

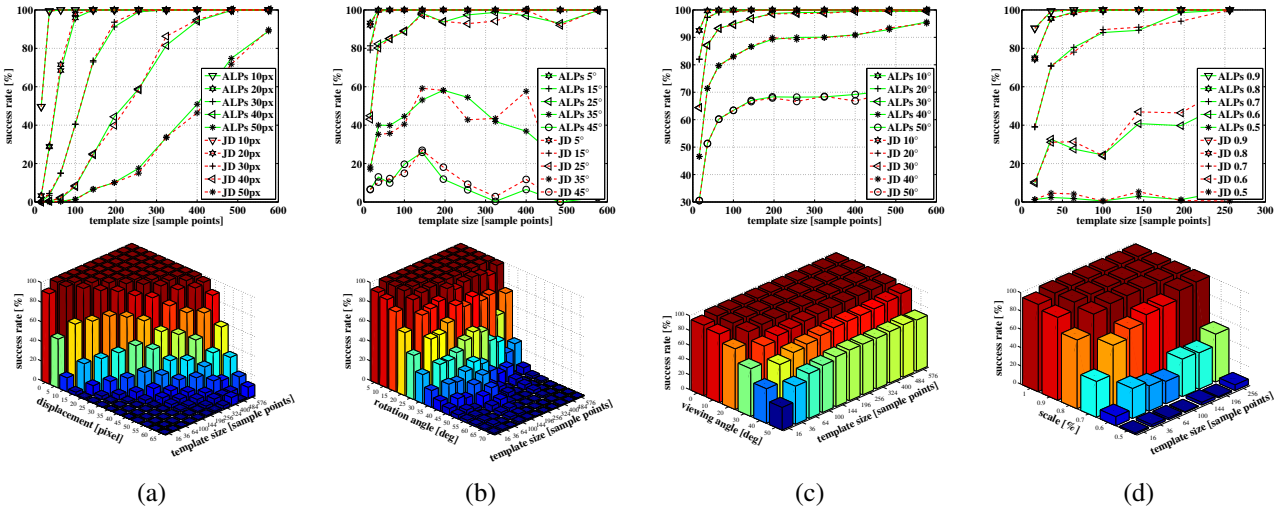
Figure 6. Comparison of success rates of ALPs and linear predictors of Jurie and Dhome(JD) with respect to different levels of noise and different template sizes. **(a)** Success rates of ALPs using 1000 random warpings and **(b)** using 2000 random warpings.

chosen ESM, a state of the art approach that minimizes the energy function using a second order approximation. In Fig. 5 we compare the success rate of the ESM tracking to that of ALPs regarding different magnitudes of displacements and different template sizes. Our learning-based approach clearly outperforms ESM, especially for larger template sizes.

5.3. Comparison with NoSLLiP

We also compare ALPs to the approach of Zimmermann *et al.* [20] using the phone sequence provided by the authors[1]. Example images of the tracking are shown in Fig. 8. The comparison between the tracking results of [20] and those obtained using ALPs are shown in Table 1. Although the number of provided images is larger than the number of images used by Zimmermann *et al.* [20], we still obtain a better loss-of-locks count. The given error values are rela-

[1]Zimmermann *et al.* [20] provide three different video sequences. One of them, however, includes occlusion, which can not yet be handled by our approach, and for another one the supplied ground truth data is erroneous. Therefore, we compare only to one of the provided sequences.



(a) (b) (c) (d)

Figure 7. Comparison of success rate of ALPs and JD linear predictors with respect to changes in translation **(a)**, in-plane rotation **(b)**, viewing angle **(c)**, and scale **(d)**. In all four cases the results of both approaches are approximately equal. The lower row shows more detailed results of ALPs.

Method	Frame-rate [fps]	Loss-of-locks [-/-]	Error [%]
NoSLLiP	16.8	20/1799	1.8
ALPs	96.7	10/2299	1.2

Table 1. Comparison between the tracking results of NoSLLiP (Matlab implementation) given in [20] and results obtained using ALPs (C++ implementation).

tive to the upper edge of the template. A frame is counted as loss-of-lock if one of the template corners has an error larger than 25%. Note that the template is reduced when it partially goes out of sight and enlarged again as it becomes visible again (see Fig. 8).

5.4. Usefulness of larger templates

As shown in Figures 5, 6 and 7, the success rates increase with increasing template sizes. The only exception are changes in the in-plane rotation angle, where the success rate reaches a maximum for templates with a size of approximately 100 to 200 sample points.

5.5. Qualitative Evaluation

In Fig. 1, 8 and 9 we show different image sequences, which demonstrate the processing of the proposed approach. In Fig. 1 and 9 we start with templates of size 10 by 10 sample points and iteratively grow them by adding the neighboring extension template with the highest quality. In Fig. 9 we demonstrate the use of multiple templates. In Figures 1 and 8 we track the templates, reduce them if they partially go out of sight and grow them back to the original size when their hidden parts become visible again.

6. Conclusion and Future Work

We introduced an efficient method for adapting linear predictors used in real-time template tracking to dynamically change the template shape. Our method allows both, enlargement and reduction of the template size. We demonstrated that our ALPs approach can also be used to efficiently learn linear predictors for templates of fixed size. In that case we start from small templates and adapt their linear predictors until the desired template size is reached. This resulted in much shorter learning time compared to the standard approach of Jurie and Dhome [12]. The efficiency of the presented approach derives from the special computation of the matrix inverse. In the standard approach the inverse has to be recomputed from scratch after each change of the template size. In contrast, our approach updates the matrix according to the change in the template shape.

We demonstrated that our ALPs yield tracking results comparable to those of the standard approaches, while learning is much faster. The current approach, however, lacks robustness against occlusion. Therefore, the next step will be to provide some means of occlusion handling for large templates.

Acknowledgments

We want to thank Stefan Hinterstoisser and Jürgen Sotke for prove-reading the paper. The project was partially funded by the Bayerische Forschungsstiftung.

References

[1] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *Conference on Computer*

Figure 8. Result images of the phone sequence, which is provided by Zimmermann *et al*. [20]. Note that the template is reduced if it goes out of the image and grown again of it gets visible again.

Figure 9. Iterative growing of two independent templates.

Vision and Pattern Recognition, volume 1, page 1090, Los Alamitos, CA, USA, 2001.

[2] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56:221–255, March 2004.

[3] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *Conference on Intelligent Robots and Systems*, volume 1, pages 943–948 vol.1, Sept.-2 Oct. 2004.

[4] S. Benhimane and E. Malis. Homography-based 2d visual tracking and servoing. *International Journal of Robotics Research*, 26(7):661–676, July 2007.

[5] M. Cascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4), April 2000.

[6] F. Dellaert and R. Collins. Fast image-based tracking by selective pixel integration. In *ICCV Workshop of Frame-Rate Vision*, pages 1–22, September 1999.

[7] C. Gräßl, T. Zinßer, and H. Niemann. Illumination insensitive template matching with hyperplanes. In *Proceedings of Pattern recognition: 25th DAGM Symposium*, pages 273–280, Magdeburg, Germany, September 2003.

[8] C. Gräßl, T. Zinßer, and H. Niemann. Efficient hyperplane tracking by intelligent region selection. In *Image Analysis and Interpretation*, pages 51–55, March 2004.

[9] G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.

[10] H. V. Henderson and S. R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23(1):53–60, 1981.

[11] S. Hinterstoisser, S. Benhimane, N. Navab, P. Fua, and V. Lepetit. Online learning of patch perspective rectification for efficient object detection. In *Conference on Computer Vision and Pattern Recognition*, pages 1–8, Anchorage, Alaska, 2008.

[12] F. Jurie and M. Dhome. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):996–100, 2002.

[13] F. Jurie and M. Dhome. Real time robust template matching. In *British Machine Vision Conference*, pages 123–131, 2002.

[14] B. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[15] E. Malis. Improving vision-based control using efficient second-order minimization techniques. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1843–1848 Vol.2, 26-May 1, 2004.

[16] J. Matas, K. Zimmermann, T. Svoboda, and A. Hilton. Learning efficient linear predictors for motion estimation. In *Computer Vision, Graphics and Image Processing*, pages 445–456, 2006.

[17] W. W. Mayol and D. W. Murray. Tracking with general regression. *Journal of Machine Vision and Applications*, 19(1):65–72, 2008.

[18] P. Parisot, B. Thiesse, and V. Charvillat. Selection of reliable features subsets for appearance-based tracking. *Signal-Image Technologies and Internet-Based System*, 0:891–898, 2007.

[19] H.-Y. Shum and R. Szeliski. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, pages 101–130, 2000.

[20] K. Zimmermann, J. Matas, and T. Svoboda. Tracking by an optimal sequence of linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):677–692, 2009.

APPENDIX

B

ONLINE LEARNING OF LINEAR PREDICTORS FOR REAL-TIME TRACKING

Own contributions. The core idea of the publication, an efficient and robust learning method for Linear Predictors, was created in a conversation between Marc Pollefeys and me. My further contributions to this work include the design and implementation of the method. The evaluation was performed together with David Tan and Slobodan Ilic. The core idea of the paper was refined in collaboration with all co-authors. All co-authors were involved in the writing of the paper.

Online Learning of Linear Predictors for Real-Time Tracking

Stefan Holzer[1], Marc Pollefeys[2], Slobodan Ilic[1], David Tan[1], and Nassir Navab[1]

[1]Department of Computer Science, Technische Universität München (TUM), Boltzmannstrasse 3, 85748 Garching, Germany
{holzers,slobodan.ilic,tanda,navab}@in.tum.de
[2]Department of Computer Science, ETH Zurich, CNB G105, Universitatstrasse 6, CH-8092 Zurich, Switzerland
marc.pollefeys@inf.ethz.ch

Abstract. Although fast and reliable, real-time template tracking using linear predictors requires a long training time. The lack of the ability to learn new templates online prevents their use in applications that require fast learning. This especially holds for applications where the scene is not known a priori and multiple templates have to be added online. So far, linear predictors had to be either learned offline [1] or in an iterative manner by starting with a small sized template and growing it over time [2]. In this paper, we propose a fast and simple reformulation of the learning procedure that allows learning new linear predictors online.

Key words: template tracking, template learning, linear predictors

1 Introduction

Template tracking is an extensively studied field in Computer Vision with a wide range of applications such as augmented reality, human-computer interfaces, medical imaging, surveillance, vision-based control and visual reconstruction. The main task of template tracking is to follow a template in an image sequence by estimating parameters of the template warping function that defines how the pixel locations occupied by the template are warped to the next frame of the image sequence. Examples for such warping functions are affine transformations or homographies.

Most approaches to template tracking are based on energy minimization [3–11], where the image intensity differences between template areas of two consecutive frames have to be minimized in terms of the template warping parameters. In many cases, analytical derivation of the Jacobian is used in order to provide real-time tracking capabilities. Alternative approaches to template tracking are based on learning [1, 2, 12–17], where the relation between image intensity differences and template warping parameters is learned. While energy minimization approaches are flexible at run-time, learning based methods have proven to allow much faster tracking.

A very successful learning based template tracker was proposed by Jurie and Dhome [1]. It is based on learning linear predictors to efficiently compute template warp parameter updates. Thanks to extensive training, this approach is very fast and tends to avoid local minima. The costly learning phase, however, prohibits this method from computing templates online.

In many applications, such as simultaneous localization and mapping (SLAM), the ability to learn new templates at run-time is crucial, since they have to deal with data, which is not available for prior offline learning. Contrary to the current development of methods for highly parallelized systems, which *e.g.* rely on modern graphics cards, most consumer-oriented applications, especially those placed on mobile devices, do not have such a huge processing power available.

We, therefore, propose a reformulation of the linear predictor learning step that drastically improves the learning speed. Although this way of training brings a small decrease in tracking robustness, it helps to improve robustness against image noise. However, we demonstrate how the tracking robustness can be increased online during tracking and how the tracking performance of the original approach of Jurie and Dhome [1] can be achieved.

2 Related Work

Since the seminal work of Lucas and Kanade [3], a large variety of template tracking approaches have been presented. They can be classified into three main categories: tracking-by-detection (TBD) [18–22], energy minimization [3–11] and learning [1, 2, 12–17]. In contrast to others, TBD-based approaches track a template over the whole image independent from the previous position. Nonetheless, they often require a time consuming training procedure and can hardly achieve the processing speed of frame-to-frame tracking. Furthermore, their possible pose space is limited.

On the other hand, the latter two categories use frame-to-frame tracking. Between them, energy minimization-based approaches are generally more flexible at run-time while learning-based approaches enable higher tracking speed. Additionally, Jurie *et al.* [12] demonstrated that Linear Predictors (LPs) are superior to Jacobian approximation and Holzer *et al.* [2] showed an experiment where LPs are superior to Efficient Second-order Minimization (ESM) [11].

Tracking-by-Detection-based approaches. Özuysal *et al.* [18] presented a TBD-based approach called FERNs where they extract keypoints from an image and match them using a classification-based approach by estimating the probability on which class the keypoints belong. However, it needs a time consuming learning stage and requires a sufficient number of visible keypoints which makes it less useful in tracking small regions. Another approach is DTTs from Holzer *et al.* [19] which builds on finding closed contours and matches them using a similar approach as the keypoint matching in [18]. Thus, this also needs a time consuming learning stage while detection speed was reported at only 10 fps. Furthermore, prominent advances in this field are reflected from the works of Hinterstoisser *et*

al. [20–22]. Their earlier works called Leopard [20] and Gepard [21] make use of the image patch that surrounds a keypoint. These patches are then used for matching and pose estimation. Hence, their methodology suggests that they benefit from any advancement in template tracking. Moreover, although they achieve a near real time performance, these approaches heavily rely on the repeatability of the underlying keypoint detector. In their recent work, DOT [22] aims to overcome the dependency on keypoint detection. It is a template matching based approach that learns templates for every pose. Due to this, it restricts the application space and is comparably slow in contrast to frame-to-frame tracking.

Energy minimization-based approaches. Numerous approaches have followed the work of Lucas and Kanade [3]. They consist of different update rules of the warp function [3–8], different orders of approximation of the error function [10, 11], and occlusion and illumination change handling [5]. The different update rules of the warp function can be classified into four types, namely, the additive approach [3], the compositional approach [4], the inverse additive approach [5, 6], and the inverse compositional approach [7, 8]. Among these types, the interesting component in the inverse additive and inverse compositional approach is that it switches the functions of the reference and current image. As a consequence, it is possible to transfer some of the computation to the initialization phase and to make the tracking computationally more efficient. Faster convergence rates for larger convergence areas can be additionally obtained by using a second-order instead of a first-order approximation of the error function [10, 11]. Lastly, Hager and Belhumeur [5] established a method to compensate for illumination changes and occlusions. A more detailed overview of energy-based tracking methods is given by Baker and Matthews [9].

Learning-based approaches. In contrast to energy minimization approaches, Jurie and Dhome [1] proposed a method that learns linear predictors using randomly warped samples of the initial template while using the learned linear predictors to predict the parameter updates in tracking. This simplifies the tracking process from the previous approach by using a matrix vector multiplication. Here, the "Jacobians" are computed once for the whole method. Furthermore, the same authors extended their approach to handle occlusions [12]. Other authors such as Gräßl *et al*. [23] demonstrated how linear predictors can be made invariant to illumination changes. In addition, to further increase accuracy in tracking, they [13] also formulated a method on how to select the points for sampling from the image data. Zimmermann *et al*. [17] use numerous small templates and track them individually. Based on the local movements of these small templates, they estimate the movement of a large template. Holzer *et al*. [2] start with a small template and grow it until a large template is constructed online. This idea showcased a way to adapt existing linear predictors to modify the shape of a template at run-time. Mayol and Murray [16] stepped back from linear predictors by presenting an approach that fits the sampling region to pre-trained samples using general regression.

All the proposed learning approaches, however, are not able to learn large templates online. To overcome this limitation, we introduce a learning scheme, which is different to the one proposed by Jurie and Dhome [1] and enables online learning of templates.

3 Background and Terminology

This section aims to introduce our notations and to summarize the fundamental aspects of the template tracking approach proposed by Jurie and Dhome [1], which is used in comparison to our approach as introduced in Sec. 4.

3.1 Template and Parameter Description

Without loss of generality, we define a template as a rectangular region in the first frame of a video sequence, which defines the region of interest that we want to track. Note that the method is not limited to rectangular regions and is capable of dealing with arbitrary shapes. The location of the region within the image is defined by the variable $\boldsymbol{\mu}$. In this paper, $\boldsymbol{\mu}$ is an 8×1 vector that stores the position of the four 2D corner points of the template region in the image. Thus, $\boldsymbol{\mu}$ has to be estimated in every new frame. Furthermore, to find the image intensities in the template, n_p sample points are positioned on a regular grid instead of using all the pixels in the template region. These intensities are stored in an $n_p \times 1$ vector $\mathbf{i}$, where $\mathbf{i} = \left(i_1, i_2, \ldots, i_{n_p}\right)^\top$.

3.2 Template Tracking based on Linear Predictors

Given a template region in a reference image, the corresponding initial parameter values and reference image intensities are stored in $\boldsymbol{\mu}_R$ and $\mathbf{i}_R$, respectively. The template parameter values $\boldsymbol{\mu}_C$ define the location of the template in the current image; henceforth, tracking is done by computing $\boldsymbol{\mu}_C$. The value of $\boldsymbol{\mu}_C$ depends on the previously computed parameter values $\boldsymbol{\mu}_{C-1}$, the image intensities in the reference image $\mathbf{i}_R$ and the image intensities in the current image $\mathbf{i}_C$. Jurie and Dhome [1] simplified this relation as:

$$\delta\boldsymbol{\mu} = \mathbf{A}\delta\mathbf{i}, \tag{1}$$

where $\delta\boldsymbol{\mu}$ are the template parameter updates, $\delta\mathbf{i} = \mathbf{i}_C - \mathbf{i}_R$ and $\mathbf{A}$ is a linear predictor matrix. It is important to mention that the image intensities $\mathbf{i}_C$ are extracted from the current image using the sample points from the previously computed parameter values $\boldsymbol{\mu}_{C-1}$. Therefore, in order to compute the update of the template parameters $\delta\boldsymbol{\mu}$, one needs to pre-compute the matrix $\mathbf{A}$. In this case, $\mathbf{A}$ is called a linear predictor since it establishes a linear relation between the image differences and the parameter updates.

$\mathbf{A}$ is a constant matrix of size $8 \times n_p$, which is computed during the learning phase. The learning process uses n_t random transformations on the reference

template, where n_t is much larger than n_p. These transformations are small disturbances $\delta\boldsymbol{\mu}_i$, $i = 1, \ldots, n_t$, to the reference parameters $\boldsymbol{\mu}_R$. As a consequence, this introduces a change in the image intensities $\delta\mathbf{i}_i = \mathbf{i}_i - \mathbf{i}_R$ for each random transformation. The vectors of those small disturbances to the template position parameters are concatenated into an $8 \times n_t$ matrix $\mathbf{Y}$, while the corresponding image intensity differences are stored in an $n_p \times n_t$ matrix $\mathbf{H}$. These can be written as $\mathbf{Y} = \left(\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \ldots, \delta\boldsymbol{\mu}_{n_t}\right)$ and $\mathbf{H} = (\delta\mathbf{i}_1, \delta\mathbf{i}_2, \ldots, \delta\mathbf{i}_{n_t})$. Using $\mathbf{Y}$ and $\mathbf{H}$, Eq. (1) is modified and becomes:

$$\mathbf{Y} = \mathbf{A}\mathbf{H}. \tag{2}$$

Finally, $\mathbf{A}$ is learned by minimizing:

$$\arg\min_{\mathbf{A}} \sum_{k=1}^{n_t} \left(\delta\boldsymbol{\mu}_k - \mathbf{A}\delta\mathbf{i}_k\right)^2 \tag{3}$$

which results in the closed-form solution:

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top \left(\mathbf{H}\mathbf{H}^\top\right)^{-1}. \tag{4}$$

This leads to an inverse-compositional tracking approach, where the parameter updates, obtained from Eq. (1), have to be applied to the reference parameters $\boldsymbol{\mu}_R$ and a corresponding transformation has to be estimated. The inverse of this transformation is then used to update the current template parameters. In our implementation, we compute a homography to represent the current perspective distortion.

To improve invariance to illumination changes, normalization is used on the extracted image data by imposing zero mean and unit standard deviation. As a consequence, zero mean makes $\mathbf{H}$ lose one rank and the resulting $\mathbf{H}\mathbf{H}^\top$ rank-deficient. In order to prevent this rank-deficiency, random noise is added to $\mathbf{H}$ after normalization.

3.3 Multi-Layered Tracking

In order to make tracking more robust, we use a multi-predictor approach where we compute n_l linear predictors: $\mathbf{A}_1, \ldots, \mathbf{A}_{n_l}$. Among the linear predictors, $\mathbf{A}_1$ has learned large template distortions, while $\mathbf{A}_{n_l}$ has learned smaller parameter changes. Intuitively, $\mathbf{A}_1$ accounts for large movements of the template and the subsequent linear predictors further refine the results of the previous predictor. In practice, each linear predictor is utilized several times before the next level is used. In this paper, we used $n_l = 5$ and three iterations for each predictor.

4 Fast Learning Strategy

Considering Eq. (4), it is evident that the computation of the linear predictor $\mathbf{A}$ using Jurie and Dhome [1] is time-consuming due to the pseudo-inverse of $\mathbf{H}$. This involves the inverse of an $n_p \times n_p$ matrix $\mathbf{H}\mathbf{H}^\top$.

To increase the speed, we propose to use the pseudo-inverse of $\mathbf{Y}$, instead of $\mathbf{H}$, in order to generate a much faster learning process. Using this approach, Eq. (2) leads to:

$$\mathbf{I} = \mathbf{A}\mathbf{H}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1} = \mathbf{A}\mathbf{B}, \tag{5}$$

where $\mathbf{B} = \mathbf{H}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1}$ is an $n_p \times 8$ matrix; henceforth, to learn $\mathbf{A}$, we compute:

$$\mathbf{A} = (\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top. \tag{6}$$

The pseudo-inverse is applied differently in Eqs. (5) and (6), since for matrix $\mathbf{Y}$, the rows are linearly independent while for matrix $\mathbf{B}$, the columns are linearly independent; and therefore, computing it the same way leads to a rank-deficient inversion in one of the two cases [24].

It is noteworthy to mention that the computation of the matrix $\mathbf{A}$ involves two matrix inverse, but both $\mathbf{Y}\mathbf{Y}^\top$ and $\mathbf{B}^\top\mathbf{B}$ are 8×8 matrices. However, computing the inverse of two 8×8 matrices is much faster in comparison to computing the inverse of an $n_p \times n_p$ matrix. In fact, $\mathbf{Y}\mathbf{Y}^\top$ can be precomputed. Therefore, only a single 8×8 matrix has to be inverted online.

Since the linear mapping denoted by the linear predictor should never encode fixed offsets, we normalize $\mathbf{Y}$ such that each parameter has zero mean and unit standard deviation; while de-normalizing $\delta\boldsymbol{\mu}$ when solving Eq. (1) in tracking. It is interesting to note that unlike the normalization used in Sec. 3.2 to obtain invariance on changes in lighting conditions, this normalization does not generate a rank-deficient matrix $\mathbf{Y}\mathbf{Y}^\top$ because the normalization is applied on the rows of $\mathbf{Y}$. The difference in performance using normalized and unnormalized $\mathbf{Y}$ is shown in Sec. 5.

Moreover, solving Eq. (4) in the approach of Jurie and Dhome [1] actually corresponds to approximating $\mathbf{Y}$ by orthogonally projecting it on $\mathbf{H}$. On the other hand, solving Eq. (6) in our approach approximates $\mathbf{H}$ by orthogonally projecting it on $\mathbf{Y}$. Given that we project $\mathbf{H}$ on $\mathbf{Y}$, all noise outside of the low-rank space represented by $\mathbf{Y}$ has no effect; while in case of Jurie and Dhome, the noise has more effect. This makes their approach more sensitive to noise. We also prove this in our experiments.

Updating Linear Predictors. Hinterstoisser *et al.* [20] showed that new training samples can be added to a linear predictor even after learning by making use of the Sherman-Morrison formula. It relies on the original way of computing linear predictors as $\mathbf{A} = \mathbf{Y}\mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top)^{-1}$ and efficiently updates the inverse $\mathbf{S} = (\mathbf{H}\mathbf{H}^\top)^{-1}$. In contrast to this, our method does not compute for $\mathbf{S}$ in the learning phase as Jurie and Dhome [1] do. We propose to derive $\mathbf{S}$ from an existing linear predictor $\mathbf{A}$ using Eq. (4):

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top)^{-1} = \mathbf{Y}\mathbf{H}^\top\mathbf{S} = \mathbf{D}\mathbf{S}, \tag{7}$$

where $\mathbf{D} = \mathbf{Y}\mathbf{H}^\top$ is an $8 \times n_t$ matrix. From this, $\mathbf{S}$ can be computed using the pseudo-inverse of $\mathbf{D}$:

$$\mathbf{S} = \mathbf{D}^\top(\mathbf{D}\mathbf{D}^\top)^{-1}\mathbf{A}. \tag{8}$$

In this computation, we are using the matrix inverse of $\mathbf{D}\mathbf{D}^\top$. Again, this is an 8×8 matrix and can be inverted very fast.

Sec. 5 shows that updating $\mathbf{S}$ by adding training samples using the Sherman-Morrison formula helps to further improve the tracking performance. The update is done by:

$$\hat{\mathbf{S}} = \left(\mathbf{S}^{-1} + \delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^\top\right)^{-1} = \mathbf{S} - \frac{\mathbf{S}\delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^\top\mathbf{S}_I}{1 + \delta\mathbf{i}_{n_t+1}^\top\mathbf{S}\delta\mathbf{i}_{n_t+1}}, \tag{9}$$

where $\delta\mathbf{i}_{n_t+1}$ is a vector of image value differences obtained from a new random transformation applied to the sample points. Note that before computing the updated linear predictor using Eq. (7), we also have to update the matrices $\mathbf{H}$ and $\mathbf{Y}$ by concatenating them with the new training samples. For the normalization of the parameter differences, we use the normalization as applied to the original learning.

5 Experiments

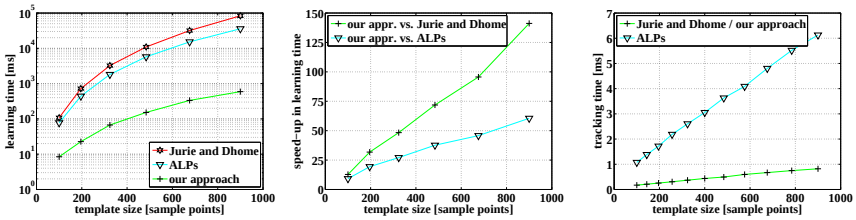
In this section, we evaluate our proposed approach for efficient learning of linear predictors by comparing it to the original learning approach proposed by Jurie and Dhome [1] and the iterative approach of Holzer *et al.* [2]. These comparisons are done using two kinds of evaluation – timing and accuracy. The former shows the difference in learning and tracking times; while the latter involves the computation of tracking robustness with respect to different types of motion as well as its sensitivity to noise. Additionally, we also compare the accuracy of our approach to the non-linear method of Benhimane *et al.* [11]. Moreover, we show several qualitative results from real video sequences. They show the algorithm used on a mobile phone for learning and tracking a single template as well as handling multiple templates. This demonstrates the need for fast learning in unknown environments.

All the algorithms are implemented in C++. For the implementation of Holzer *et al.* [2], we used the binaries provided by the authors; while the implementation of Benhimane *et al.* [11] is from the publicly available binaries[1]. The evaluation of these algorithms are conducted using a notebook with a 2.26 GHz Intel(R) Core(TM)2 Quad CPU and 4 GB of RAM, where only one core is used for the computation.

5.1 Computational Complexity

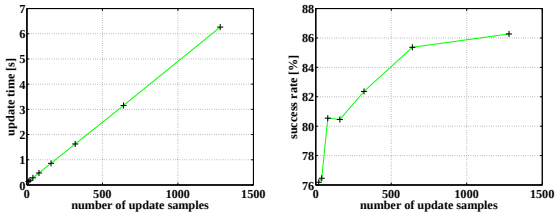
In the first evaluation we investigate the computational complexity of our approach in contrast to the approach of Jurie and Dhome [1] and Holzer *et al.* [2]. Our algorithm is divided into three parts – learning linear predictors, tracking using the learned linear predictors and updating the linear predictors while tracking. This section mainly focuses on the amount of time that each part requires to finish in relation to the number of sample points used.

[1] See version 0.4 available at `http://esm.gforge.inria.fr/ESM.html`.



(a) (b) (c)

Fig. 1. (a) Comparison of the necessary learning time with respect to the number of sample points used within the template for the approach proposed by Jurie and Dhome [1], by Holzer *et al.* [2] (referred as "ALPs") and our approach. **(b)** The corresponding speed-up in learning obtained by our approach. **(c)** The tracking time per frame with respect to the number of sample points used for the template.



(a) (b)

Fig. 2. (a) The time necessary to update an existing tracker with respect to number of update samples. This update can be performed in parallel with tracking. **(b)** The corresponding improvement in success rates with increasing number of updates.

Learning. Our main contribution is reflected on the learning time. We show in Fig. 1 (a) that, as the amount of sample points increases, the time required for learning using our approach increases much slower in comparison to the approach of both, Jurie and Dhome [1] and Holzer *et al.* [2]. This difference is emphasized in Fig. 1 (b) where it is evident that for templates with more than 800 sample points (*e.g.* 30×30), our approach is more than two orders of magnitude faster, *i.e.* almost 120 times faster, than Jurie and Dhome [1] and more than 50 times faster than the approach of Holzer *et al.* [2].

Tracking. Both the original approach [1] and our approach have similar tracking time because the time needed to de-normalize the parameter updates in our approach is negligible. Furthermore, the measure of tracking time per frame with respect to template size in Fig. 1 (c) demonstrates that our approach can easily reach frame rates higher than 1000 fps even with large templates. In contrast to Holzer *et al.* [2], their method is slightly slower and the necessary time for tracking increases faster as the template size increases. In comparison to this, the non-linear approach of Benhimane *et al.* [11] takes about 10 ms for tracking the same template.

Updating. The updating process is a way of adding new training samples to a learned linear predictor during tracking. Fig. 2 (a) shows the time necessary to update an existing tracker with respect to the number of update samples, where the number of update samples corresponds to the number of random transformations applied to the template. Note that this is the same template as used for the initial learning. This result illustrates that by adding a small number of training samples at each time, we can keep the computational cost low while improving the performance of the tracker over time. In Fig. 2 (b), we show an exemplary improvement of tracking robustness when updating the linear predictors with a specific number of update samples. Sec. 5.2 discusses more on the tracking robustness in updating.

5.2 Robustness

In this section, we analyze the influence of our learning approach on the robustness of tracking with respect to different movements and different levels of noise. We measure accuracy by finding the correct location of the template after inducing random transforms to several test images. The images used in the evaluation are taken from the Internet (see supplementary material[2]). Moreover, the random transforms include translation, rotation, scale and viewpoint change. Using the test images and random transforms, tracking is considered successful if the mean pixel distance between the reference template corner points and the tracked template corner points, that is back-projected into the reference view, is less than 5 pixels. Hence, robustness is measured as the percent of successfully tracked templates after applying several random transforms to each test image. For measuring the robustness in relation to noise we corrupted the image with noise sampled from a Gaussian distribution before applying the random transform.

At this point, it is important to mention that the goal of this type of evaluation is to generate more accurate comparison with the ground truth measurements. Indeed, there are other methods of testing such as using markers on real scenes to find the camera motion. However, this approach includes markers that generate its own error and limit the amount of motion available for testing. In addition, our evaluation also has the benefit of control which means that it is done by changing only variables that are being tested while keeping the others constant throughout the experiment. We can also specify the amount of change to fairly evaluate at which value the algorithm failed.

Here, we compare our approach to the methods of Jurie and Dhome [1], Holzer *et al.* [2], and Benhimane *et al.* [11]. For our approach, we considered three different cases:

- **Unnormalized:** we do not normalize the parameter differences;
- **Normalized:** we normalize the parameter differences before learning the linear predictors and de-normalize them during tracking; and,

[2] The supplementary material, which includes the images used for evaluation as well as videos, can be found at `http://campar.in.tum.de/Main/StefanHolzer`.

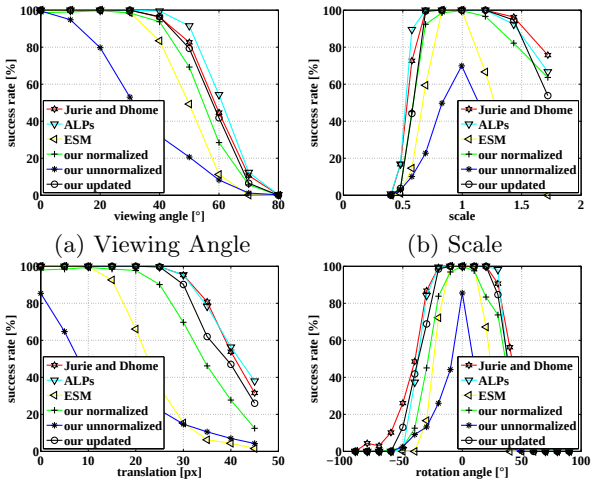
– **Updated:** we normalize the parameter differences and update the linear predictors with 1000 training samples before performing the experiments.

Given the learned linear predictor of a test image, the experiment starts by applying a random transform to the image and use the linear predictor to track this movement. This transform includes translation, rotation, scale and viewing angle. Therefore, after imposing several random transforms to a set of images, robustness is measured as the percent of successful estimation of the applied motions. For the evaluation with respect to noise, we corrupted the test image with Gaussian noise before applying the random transforms. Unless otherwise stated, the experiments are applied on templates of size 150×150 pixels with 18×18 sample points, and the initial learning of the linear predictors use $3 \cdot 18 \cdot 18 = 972$ training samples; while for the non-linear approach of Benhimane *et al.* [11], we use the complete template without subsampling.

Normalization of parameter differences. As we mentioned in Sec. 4, normalizing the parameter difference matrix $\mathbf{Y}$ before learning is important for our approach. To emphasize this, we included the results of the unnormalized approach in Fig. 3. It clearly shows that the unnormalized approach is not suitable for tracking. In contrast to that, the normalized approach gives results which are close to the original approach while the updated approach gets even closer to the results of Jurie and Dhome [1]. On the other hand, the outcome from Holzer *et al.* [2] also shows that it performs similarly well as Jurie and Dhome [1]. All the learning based approaches, except for the unnormalized version of our approach, give superior results compared to the non-linear approach of Benhimane *et al.* [11].

Number of samples points. In Fig. 4, we compare the tracking robustness in relation to the number of sample points. It is important to note that all the results show similar behavior across different transformations. Our normalized approach replicates the results of Jurie and Dhome [1] when the number of sample points per template is above 325. In all the results, the updated approach does not lose tracking robustness and performs consistently equal to the original approach.

Updating. Fig. 2 (b) depicts the change in robustness when we add new training samples to a learned linear predictor with normalization during tracking. In this experiment, we applied several random translations of approximately 30 pixels on the set of test images. After applying the updated linear predictors to the transformed images, we checked how often the translation was correctly estimated. This was done for linear predictors updated with different numbers of update samples, where the number of update samples is the number of random transformations applied to the template. The results show that tracking robustness increases as the number of update samples increases. We also show in Fig. 3 that updating brings the tracking performance closer to the original learning approach of Jurie and Dhome [1].



(a) Viewing Angle (b) Scale

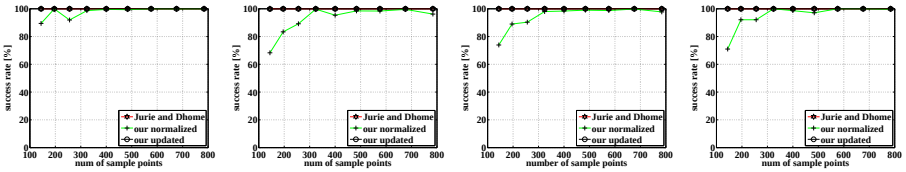
(c) Translation (d) In-plane Rotation

Fig. 3. Comparison of the approach of Jurie and Dhome [1], Holzer *et al.* [2] (referred as "ALPs"), Benhimane *et al.* [11] (referred as "ESM"), as well as our approach with and without normalization, and with updated predictors. We consider four different types of motions as specified. The success rate indicates the percent of successful estimation of the applied motions.

Sensitivity to Noise. A comparison among the different methods with respect to noise sensitivity is presented in Fig. 5. This experiment corrupts the input image by Gaussian noise with zero mean and varying standard deviation. After that, we impose a small translation to the corrupted image and measure the accuracy of the tracker for each algorithm. The noise parameter in Fig. 5 corresponds to the standard deviation of the Gaussian noise and the image intensity of the uncorrupted image ranges from 0 to 255.

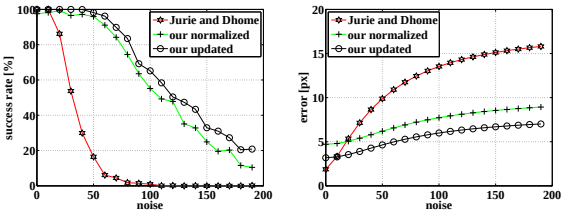
While our approach had a slightly worse tracking robustness for large motions as shown in Fig. 3, we illustrate in Fig. 5 (a) that the tracking robustness of our approach outperforms the original approach in terms of sensitivity to noise. An evidence for this is shown in Fig. 5 (b) where we analyze the average distance between the reference template corner points and the predicted template corner points that is back-projected into the reference view. Based on the figure, the prediction error of Jurie and Dhome [1] is smaller compared to our approach for small noise levels, but rapidly increases as the level of noise increases. Contrary to this, our approach has a higher error if no or only a small amount of noise is present, but the error increases slower when more noise is added. For a discussion on why our approach is less sensitive to noise in comparison to Jurie and Dhome [1], refer to Sec. 4.

It is noteworthy that being less sensitive to noise is an advantage in environments with bad lighting conditions, *e.g.* at night when the signal-to-noise ratio of cameras usually decreases. This is especially the case for cameras used in mobile devices.



(a) Viewing Angle (b) Scale (c) Translation (d) Rotation

Fig. 4. Comparison of the success rate in tracking with respect to the number of sample points for the approach of Jurie and Dhome [1], as well as our approach with normalization and with updated predictors.



(a) (b)

Fig. 5. Comparison of our approach to the approach of Jurie and Dhome [1] with respect to sensitivity to noise in tracking. **(a)** shows the success rate for different noise levels. **(b)** shows the average error in the predicted corner points of the template.

5.3 Application: Tracking on a Mobile Phone

Due to the high efficiency of learning and tracking using the proposed approach, it is optimally suited for applications running on mobile devices. In order to demonstrate this, we implemented it on a mobile phone with a 1.2 GHz dual core processor and 1 GB of RAM. Note that we only used a single core for learning and tracking, and that we directly used our implementation without optimizing it for the special processor technology used in mobile phones. Sample images of tracking using a mobile phone are shown in Fig. 6, and a video that demonstrates the learning and tracking can be found in the supplementary material.

Exemplary learning times for [1] are approximately 18000 ms for a template with 16×16 sample points, whereas our approach needs only approximately 350 ms. Therefore, our approach is more than 50 times faster than the approach of Jurie and Dhome [1], but more importantly, allows interactive applications to start tracking almost immediately. For tracking, both approaches need about 2.5 ms per frame for a template with 16×16 sample points.

5.4 Application: Tracking of Multiple Templates Simultaneously

In Fig. 7, we demonstrate the simultaneous tracking of multiple templates. The first row in this figure shows the tracking of three templates on a mobile phone while the second row demonstrates tracking of a large number of templates and shows that the use of multiple templates helps to handle occlusions. Because of

Fig. 6. Tracking on a mobile phone. The upper row shows tracking a non planar surface while the lower row shows tracking a planar scene.

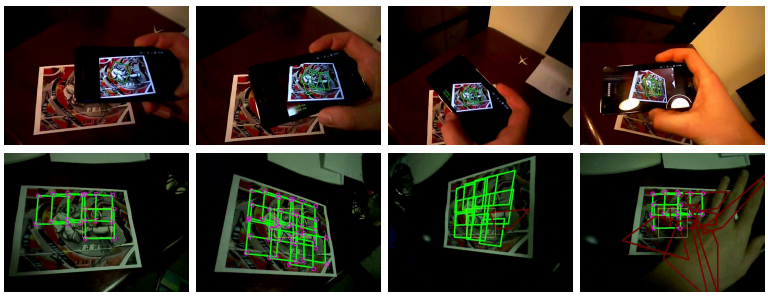


Fig. 7. Learning and tracking of multiple templates. The upper row shows tracking of multiple templates on a mobile phone while the lower row shows tracking of a large number of templates on a standard PC. This helps to handle occlusions.

the fast learning characteristic of our approach, we are able to learn such a large number of templates online. This can be useful for a SLAM and similar systems where a patch-based reconstruction of a scene is performed.

6 Conclusion

We introduced an efficient method for online learning of linear predictors for real-time template tracking by reformulating the original learning procedure presented by Jurie and Dhome [1]. This removes the time consuming inversion of large matrices and dramatically reduces the learning time. In addition, our approach yields tracking results comparable to those of the standard approach while sensitivity to image noise is reduced. Furthermore, the robustness in tracking can be increased by adding new training samples to an already learned tracker. Lastly, we demonstrated the usefulness of the proposed learning approach in a tracking application for mobile devices, where online learning is necessary.

References

1. Jurie, F., Dhome, M.: Hyperplane approximation for template matching. PAMI (2002)

2. Holzer, S., Ilic, S., Navab, N.: Adaptive linear predictors for real-time tracking. In: CVPR, San Francisco, CA, USA (2010)
3. Lucas, B., Kanade, T.: An Iterative Image Registration Technique with an Application to Stereo Vision. In: International Joint Conference on Artificial Intelligence. (1981)
4. Shum, H.Y., Szeliski, R.: Construction of panoramic image mosaics with global and local alignment. IJCV (2000)
5. Hager, G., Belhumeur, P.: Efficient region tracking with parametric models of geometry and illumination. PAMI (1998)
6. Cascia, M., Sclaroff, S., Athitsos, V.: Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. PAMI (2000)
7. Dellaert, F., Collins, R.: Fast image-based tracking by selective pixel integration. In: ICCV Workshop of Frame-Rate Vision. (1999)
8. Baker, S., Matthews, I.: Equivalence and efficiency of image alignment algorithms. In: Conference on Computer Vision and Pattern Recognition, Los Alamitos, CA, USA (2001)
9. Baker, S., Matthews, I.: Lucas-kanade 20 years on: A unifying framework. IJCV (2004)
10. Malis, E.: Improving vision-based control using efficient second-order minimization techniques. In: ICRA. (2004)
11. Benhimane, S., Malis, E.: Homography-based 2d visual tracking and servoing. International Journal of Robotics Research (2007)
12. Jurie, F., Dhome, M.: Real time robust template matching. In: BMVC. (2002)
13. Gräßl, C., Zinßer, T., Niemann, H.: Efficient hyperplane tracking by intelligent region selection. In: Image Analysis and Interpretation. (2004)
14. Parisot, P., Thiesse, B., Charvillat, V.: Selection of reliable features subsets for appearance-based tracking. Signal-Image Technologies and Internet-Based System (2007)
15. Matas, J., Zimmermann, K., Svoboda, T., Hilton, A.: Learning efficient linear predictors for motion estimation. In: Computer Vision, Graphics and Image Processing. (2006)
16. Mayol, W.W., Murray, D.W.: Tracking with general regression. Journal of Machine Vision and Applications (2008)
17. Zimmermann, K., Matas, J., Svoboda, T.: Tracking by an optimal sequence of linear predictors. PAMI (2009)
18. Özuysal, M., Fua, P., Lepetit, V.: Fast Keypoint Recognition in Ten Lines of Code. In: CVPR, Minneapolis, MI, USA (2007)
19. Holzer, S., Hinterstoisser, S., Ilic, S., Navab, N.: Distance transform templates for object detection and pose estimation. In: CVPR. (2009)
20. Hinterstoisser, S., Benhimane, S., Navab, N., Fua, P., Lepetit, V.: Online learning of patch perspective rectification for efficient object detection. In: CVPR. (2008)
21. Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., Lepetit, V.: Real-time learning of accurate patch rectification. In: CVPR. (2009)
22. Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., Navab, N.: Dominant orientation templates for real-time detection of texture-less objects. In: CVPR. (2010)
23. Gräßl, C., Zinßer, T., Niemann, H.: Illumination insensitive template matching with hyperplanes. In: Proceedings of Pattern recognition: 25th DAGM Symposium, Magdeburg, Germany (2003)
24. Penrose, R.: A generalized inverse for matrices. In: Proceedings of the Cambridge Philosophical Society. (1955)

APPENDIX

C

EFFICIENT LEARNING OF LINEAR PREDICTORS USING DIMENSIONALITY REDUCTION

Own contributions. My contributions to this work include the core idea, the design, and the implementation of the method for efficient learning of Linear Predictors using dimensionality reduction. The evaluation was performed together with David Tan and Slobodan Ilic. The core idea of the paper was refined in collaboration with the other co-authors. All co-authors were involved in the writing of the paper.

Efficient Learning of Linear Predictors using Dimensionality Reduction

Stefan Holzer, Slobodan Ilic, David Tan, Nassir Navab

Department of Computer Science, Technische Universität München (TUM),
Boltzmannstrasse 3, 85748 Garching, Germany

Abstract. Using Linear Predictors for template tracking enables fast and reliable real-time processing. However, not being able to learn new templates online limits their use in applications where the scene is not known a priori and multiple templates have to be added online, such as SLAM or SfM. This especially holds for applications running on low-end hardware such as mobile devices. Previous approaches either had to learn Linear Predictors offline [1], or start with a small template and iteratively grow it over time [2]. We propose a fast and simple learning procedure which reduces the necessary training time by up to two orders of magnitude while also slightly improving the tracking robustness with respect to large motions and image noise. This is illustrated in an exhaustive evaluation where we compare our approach with state-of-the-art approaches. Additionally, we show the learning and tracking in mobile phone applications which demonstrates the efficiency of the proposed approach.

1 Introduction

Template tracking is an extensively studied field in Computer Vision with a wide range of applications such as augmented reality, human-computer interfaces, medical imaging, surveillance, vision-based control and visual reconstruction. The main task of template tracking is to follow a template in an image sequence. This is done by estimating the parameters of the template warping function that defines how the pixel locations, occupied by the template, are warped to the next frame of the image sequence. Examples for such warping functions are affine transformations or homographies.

Recently, tracking-by-detection methods became popular since they reached a state where they are able to track close to or at real-time performance. However, they show some limitations which we further address in Sec. 2. In frame-to-frame template tracking, image intensity differences between template areas of two consecutive frames have to be minimized in terms of the template warping parameters. Most of them are based on energy minimization [3–11] and in many cases, an analytical derivation of the Jacobian is used in order to provide real-time tracking capabilities. Alternative approaches are based on learning [1, 12–17, 2] where the relation between image intensity differences and template

warping parameters is learned. While energy minimization is flexible at run-time, learning based methods have proven to allow much faster tracking.

Jurie and Dhome [1] proposed a very successful learning based template tracker which learns Linear Predictors to efficiently compute template warp parameter updates. This is very fast in tracking and tends to avoid local minima. But, due to the computationally expensive learning phase, online-creation of templates is hardly possible. This, however, is a crucial ability for many applications that have to deal with data which is not available for prior offline learning. Some examples for such applications are Simultaneous Localization and Mapping (SLAM) and Structure from Motion (SfM). We address this limitation by introducing a more efficient learning procedure for creating Linear Predictors. This not only improves the learning speed drastically, but also brings a small improvement in robustness of tracking with respect to large motions and image noise.

The remainder of the paper is structured as follows: first, we discuss related work on template tracking (Sec. 2) and introduce the original approach of Jurie and Dhome (Sec. 3). This is followed by a detailed description of our approach (Sec. 4) and an extensive quantitative testing (Sec. 5.1 and Sec. 5.2). Finally, we demonstrate that the proposed approach can be used for efficient template learning and tracking on mobile phones, as well as applications similar to SLAM where multiple templates are being tracked simultaneously (Sec. 5.3).

2 Related Work

The existing template tracking approaches can be categorized in mainly three different sets of methods: tracking-by-detection (TBD) [18–22], template tracking based on energy minimization [3–11, 23, 24], and methods that utilize learning [1, 12–17]. While tracking-by-detection methods are able to track a template over the whole image independent of the previous position, they hardly achieve the processing speed of frame-to-frame tracking. Additionally, they often require a time consuming training procedure and are limited in their possible pose space. For frame-to-frame tracking, energy minimization-based approaches are generally more flexible at run-time by allowing fast creation and modification of templates, while learning-based approaches enable higher tracking speed. Looking at tracking performance, it has been shown in the past that learning-based approaches outperform methods based on energy minimization. Jurie *et al.* [12] demonstrated that Linear Predictors are superior to Jacobian approximation and Holzer *et al.* [2] showed an experiment where Linear Predictors are superior to Efficient Second-order Minimization (ESM) [11]. We further fortify the latter by showing additional comparisons in Sec. 5.2.

Tracking-by-Detection-based approaches. Some of the most prominent work on patch-based TBD was recently proposed by Hinterstoisser *et al.* [18–20]. Their former two methods, called Leopard [18] and Gepard [19], use the patch around detected keypoints for matching and pose estimation. While these methods enable near real-time performance, they heavily rely on the repeatability of the

underlying keypoint detector. Additionally, they apply template tracking approaches for pose refinement, which means that these approaches also benefit from advances in template tracking. To overcome the dependency on keypoint detectors, they proposed a template matching based approach (DOT) [20]. However, this requires to learn templates for every possible pose, which restricts the application space and makes it comparably slow in contrast to frame-to-frame tracking. Özuysal *et al.* (FERNs) [22] extract keypoints and match them using a classification-based approach by estimating the probability on which class the keypoints belong to. Although this gives real-time performance, it includes a time consuming learning stage and needs a sufficient number of keypoints visible. This makes it less useful to track small regions. Holzer *et al.* (DTTs) [21] proposed a detection based approach which builds on finding closed contours and matches them using a similar approach as [22] used for keypoint matching. However, this includes a time consuming learning stage and detection speed was reported at 10 fps only.

Energy minimization-based approaches. Numerous approaches have followed the work of Lucas and Kanade [3]. They consist of different update rules of the warp function [3, 5, 6, 4, 7, 8], handling of occlusions and illumination changes [5], as well as considering different orders of approximation of the error function [10, 11]. The different update rules of the warp function can be classified into four types, namely, the additive approach [3], the compositional approach [4], the inverse additive approach [5, 6] and the inverse compositional approach [7, 8], where the inverse approaches switch the roles of the reference and current image. As a consequence, it is possible to transfer some of the computation to the initialization phase, which makes the tracking computationally more efficient. Compensation of illumination changes and occlusions was addressed by Hager and Belhumeur [5]. Faster convergence rates as well as larger convergence areas can be additionally obtained by using a second-order instead of a first-order approximation of the error function [10, 11]. A more detailed overview of energy-based tracking methods is given by Baker and Matthews [9].

Learning-based approaches. Jurie and Dhome [1] proposed a method that learns Linear Predictors using randomly warped samples of the initial template, while using the learned Linear Predictors to predict the parameter updates in tracking. Here, the "Jacobians" are computed once for the whole method and a parameter update is computed using a simple matrix multiplication. More details on this are given in Sec. 3.2. The same authors extended their approach to handle occlusions [12]. Invariance to illumination changes was introduced by Gräßl *et al.* [25]. They [13] also formulated a method on how to select the points for sampling from image data to further increase accuracy in tracking. Zimmermann *et al.* [17] use numerous small templates and track them individually. Based on the local movements of these small templates, they estimate the movement of a large template. Holzer *et al.* [2] start with a small template and grow it until a large template is constructed online. This idea showcased a way to adapt existing Linear Predictors to modify the shape of a template at run-time. Mayol

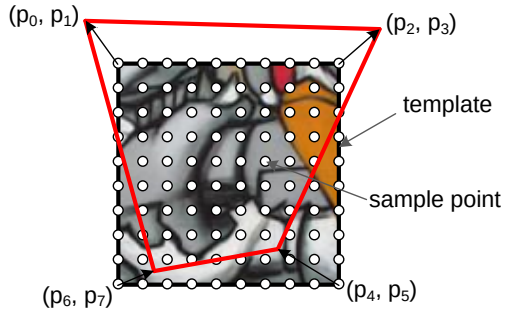


Fig. 1. A template is represented by a set of regularly placed sample points. Its pose is parameterized using its four corner points.

and Murray [16] stepped back from Linear Predictors by presenting an approach that fits the sampling region to pre-trained samples using general regression.

All the proposed learning approaches, however, are not able to learn large templates online. To overcome this limitation, we introduce a learning scheme, which is different to the one proposed by Jurie and Dhome [1] and enables online learning of templates.

3 Tracking Framework

Our proposed template tracking approach is based on the work of Jurie and Dhome [1]. While we introduce a new learning method in Sec. 4, the tracking stage itself stays the same as in [1]. Therefore, we first introduce our notations and review the method proposed by Jurie and Dhome [1].

3.1 Template and Parameter Description

Without loss of generality, we consider a $w \times h$ template with an area of $n_s = w \cdot h$ pixels within an image. Instead of using the full-resolution template, we apply a uniform subsampling as shown in Fig. 1 to obtain a grid of n_p sample points. However, neither the approach of Jurie and Dhome [1] nor our approach is restricted to this sample point arrangement or rectangular shapes.

The pose of the template is described using the parameter vector $\boldsymbol{\mu}$. Within this paper, we use a homography to represent the current perspective distortion of a planar template and parameterize it using the four corner points of the template. This leads to an 8-dimensional vector $\boldsymbol{\mu} = (p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7)^\top$ (see Fig. 1). Note that our approach is not limited to this type of transformations and can be easily adapted to any other parameterizable template deformation.

3.2 Template Tracking based on Linear Predictors

The goal of template tracking is to follow a reference template over a sequence of images. This reference template is defined by an initial parameter vector $\boldsymbol{\mu}_R$

that corresponds to the location of the template in the reference image, and a vector $\mathbf{i}_R = (i_{R,1}, i_{R,2}, \cdots, i_{R,n_p})^\top$ that corresponds to the image intensity at the sample points of the template.

Assuming that the reference template is located in the first frame of a video sequence, the location of the sample points is defined by the initial parameter vector $\boldsymbol{\mu}_R$ while the parameter vector $\boldsymbol{\mu}_C$ defines the location of the template in the current image. Henceforth, tracking is done by computing $\boldsymbol{\mu}_C$. The basic approach for this is to first compute a vector $\delta\mathbf{i} = \mathbf{i}_C - \mathbf{i}_R$ of image differences and then to use this to compute a parameter update $\delta\boldsymbol{\mu}$ which accounts for the present pose difference. Note that the vector $\mathbf{i}_C$ stores the image values extracted from the current image and is extracted by computing the sample point locations using the template pose $\boldsymbol{\mu}_{C-1}$ of the previous image frame.

Instead of explicitly minimizing an error function, *e.g.* by iteratively solving a first- or second-order approximation of the function, Jurie and Dhome [1] use a learned matrix $\mathbf{A}$ (also called as Linear Predictor) to compute $\delta\boldsymbol{\mu}$ based on the vector $\delta\mathbf{i}$ as:

$$\delta\boldsymbol{\mu} = \mathbf{A}\delta\mathbf{i}. \tag{1}$$

In order to compute $\delta\boldsymbol{\mu}$, one needs to precompute the matrix $\mathbf{A}$. This is done by collecting a set of n_t random transformations, where n_t is significantly larger than n_p, together with its corresponding image difference vectors. These random disturbances $\delta\boldsymbol{\mu}_i$ and image difference vectors $\delta\mathbf{i}_i$ are then combined in two matrices $\mathbf{Y} = \left(\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \ldots, \delta\boldsymbol{\mu}_{n_t}\right)$ and $\mathbf{H} = (\delta\mathbf{i}_1, \delta\mathbf{i}_2, \ldots, \delta\mathbf{i}_{n_t})$. Using these matrices, Eq. (1) can be written as:

$$\mathbf{Y} = \mathbf{A}\mathbf{H} \tag{2}$$

which solves for $\mathbf{A}$ using a closed-form solution:

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top \left(\mathbf{H}\mathbf{H}^\top\right)^{-1}. \tag{3}$$

In practice, we normalize the extracted image data with zero mean and unit standard deviation. This increases the robustness against illumination changes. Note that, to prevent $\mathbf{H}\mathbf{H}^\top$ from being rank-deficient due to the zero mean of the data, we have to add random noise to the obtained image value difference vectors.

3.3 Multi-Layered Tracking

For improved tracking performance, we use a multi-predictor approach where multiple Linear Predictors $\mathbf{A}_1, \ldots, \mathbf{A}_{n_l}$ are learned for one template. Among these, $\mathbf{A}_1$ is trained for large distortions and the subsequent ones for smaller distortions. Intuitively, $\mathbf{A}_1$ accounts for large motions but is less accurate, while $\mathbf{A}_{n_l}$ can handle only small template motions but has improved accuracy. During tracking, each Linear Predictor is utilized several times before the next level is used. Within this paper, we use $n_l = 5$ and three iterations for each predictor.

4 Efficient Predictor Learning using Dimensionality Reduction

As we show in Sec. 5.1, the original approach for learning Linear Predictors as proposed by Jurie and Dhome [1] is very time consuming and not applicable for learning on the fly. Therefore, we propose a simple yet powerful way of learning Linear Predictors that is much faster than [1]. The main idea behind our new learning approach is to compress the image difference vectors $\delta\mathbf{i}_i$ before using them to learn the Linear Predictor matrix. By reducing the dimensionality of $\delta\mathbf{i}_i$ from n_p to n_r, the size of $\mathbf{H}\mathbf{H}^\top$ gets reduced to $n_r \times n_r$ and therefore, the necessary matrix inversion $(\mathbf{H}\mathbf{H}^\top)^{-1}$ becomes less computational expensive.

We propose to reduce the dimensionality of $\delta\mathbf{i}_i$ by using Discrete Cosine Transform (DCT). This transform is known to give good results for compressing image data by removing DCT coefficients that correspond to high frequencies. Keeping only low-frequency information makes it well-suited for template tracking, since high-frequency information tends to de-stabilize tracking. In the following, we first introduce the 2-dimensional DCT, then show how we apply it on the 1-dimensional vectors $\delta\mathbf{i}_i$ which are sampled from the 2-dimensional templates. Mathematically, the 2-dimensional DCT $\mathbf{U}$ of a $k \times k$ matrix $\mathbf{V}$ is:

$$\mathbf{U} = \mathrm{DCT}(\mathbf{V}) = \mathbf{C}\mathbf{V}\mathbf{C}^\top \tag{4}$$

where the elements of the matrix $\mathbf{C}$ are defined as:

$$\mathbf{C}_{i,j} = \sqrt{\frac{\alpha_i}{k}} \cos\left[\frac{\pi(2j+1)i}{2k}\right] \tag{5}$$

with

$$\alpha_i = \begin{cases} 1 & \text{if } i = 0, \\ 2 & \text{otherwise.} \end{cases} \tag{6}$$

After transforming $\delta\mathbf{i}_i$ as $\delta\hat{\mathbf{i}}_i = \mathrm{DCT}(\delta\mathbf{i}_i)$, we form $\hat{\mathbf{H}} = (\delta\hat{\mathbf{i}}_1, \delta\hat{\mathbf{i}}_2, \ldots, \delta\hat{\mathbf{i}}_{n_t})$. However, since we reshaped the samples from a 2D template into a vector, Eq. (4) can not be directly applied to $\delta\mathbf{i}_i$. Therefore, we create an $n_p \times n_p$ matrix $\mathbf{W}_{DCT}$ which maps the difference $\delta\mathbf{i}_i$ of the sampled vectors directly to their DCT counterparts $\delta\hat{\mathbf{i}}_i$. Assuming that the vector $\delta\mathbf{i}_i$ is reshaped from the 2D matrix $\mathbf{V}_i$ written as $\delta\mathbf{i}_i = \mathrm{reshape}(\mathbf{V}_i)$, we compute $\mathbf{W}_{DCT}$ as:

$$\mathbf{W}_{DCT} = \left(\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_{n_p}\right) \tag{7}$$

where $\mathbf{b}_m = \mathrm{reshape}(\mathbf{C}\mathbf{B}_m\mathbf{C}^\top)$ and $\mathbf{B}_m$ is a matrix with all elements set to 0 except for the m-th element which is set to 1. By setting a single element to 1, the set of matrices $\{\mathbf{B}_1, \ldots, \mathbf{B}_{n_p}\}$ are a base of the image space of the template and the set of vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_{n_p}\}$ are the DCT projections of this base. This way, we can directly compute the 2-dimensional DCT of our image difference vectors as:

$$\delta\hat{\mathbf{i}}_i = \mathbf{W}_{DCT}\delta\mathbf{i}_i \quad \Rightarrow \quad \hat{\mathbf{H}} = \mathbf{W}_{DCT}\mathbf{H}. \tag{8}$$

In relation to the original learning formula in Eq. (3), we reformulate this by using the relation:

$$\mathbf{H} = (\mathbf{W}_{DCT})^{-1}\hat{\mathbf{H}}. \tag{9}$$

Thus, we subsitute $\mathbf{H}$ from Eq. (9) to Eq. (2) and solve for the Linear Predictor matrix $\mathbf{A}$ as follows:

$$\begin{aligned}
\mathbf{A}\mathbf{W}_{DCT}^{-1}\hat{\mathbf{H}} &= \mathbf{Y} \\
\mathbf{A}\mathbf{W}_{DCT}^{-1} &= \mathbf{Y}\hat{\mathbf{H}}^{\top}\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^{\top}\right)^{-1} \\
\mathbf{A}\mathbf{W}_{DCT}^{-1}\mathbf{W}_{DCT} &= \mathbf{Y}\hat{\mathbf{H}}^{\top}\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^{\top}\right)^{-1}\mathbf{W}_{DCT} \\
\mathbf{A} &= \mathbf{Y}\hat{\mathbf{H}}^{\top}\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^{\top}\right)^{-1}\mathbf{W}_{DCT}
\end{aligned} \tag{10}$$

To reduce the necessary computational load, we apply a dimensionality reduction by defining an $n_r \times n_p$ submatrix $\mathbf{W}_{DCT}^{(n_r)}$ with $n_r < n_p$, such that the necessary matrix inversion is no longer applied to an $n_p \times n_p$ matrix but rather to an $n_r \times n_r$ matrix. The final Linear Predictor is then computed as:

$$\mathbf{A}^{(n_r)} = \mathbf{Y}\hat{\mathbf{H}}^{(n_r)\top}\left(\hat{\mathbf{H}}^{(n_r)}\hat{\mathbf{H}}^{(n_r)\top}\right)^{-1}\mathbf{W}_{DCT}^{(n_r)}. \tag{11}$$

with $\hat{\mathbf{H}}^{(n_r)} = \mathbf{W}_{DCT}^{(n_r)}\mathbf{H}$. We show in Sec. 5.1 that by keeping n_r small, the learning time for large templates is significantly reduced. Moreover, depending on the size of n_r, the reduction in learning even increases tracking robustness.

5 Experiments

In this section, we evaluate our approach for efficient learning of Linear Predictors, as proposed in Sec. 4, by comparing it to the original learning approach proposed by Jurie and Dhome [1], the iterative approach of Holzer *et al.* [2] which is also referred to as Adaptive Linear Predictors (ALPs), as well as the approach of Benhimane *et al.* [11] known as Efficient Second-order Minimization (ESM). For the comparison, we use two kinds of evaluation – timing and tracking performance. The former shows the difference in learning and tracking times (see Sec. 5.1); while the latter involves the computation of tracking robustness with respect to different types of motion as well as its sensitivity to noise (see Sec. 5.2). Finally, we demonstrate the usefulness of fast template learning using tracking on mobile devices (see Sec. 5.3).

All algorithms used in this experiment are implemented in C++. For our approach, we consider three different instances where the difference is in the number of DCT coefficients used for learning. Specifically, we use varying DCT coefficients with values of 25, 49, and 81. The evaluation of Holzer *et al.* [2] is performed using binaries kindly provided by the authors while the approach of Benhimane *et al.* [11] is evaluated using publicly available binaries[1]. The

[1] See version 0.4 available at http://esm.gforge.inria.fr/ESM.html

Fig. 2. The data set used for synthetic experiments. These images are randomly taken from the Internet.

evaluations of these algorithms are conducted using a notebook with a 2.26GHz Intel(R) Core(TM)2 Quad CPU and 8 GB of RAM, where only one core is used for the computations. The images used for evaluation on synthetic data are taken from the Internet (see Fig. 2). For all synthetic experiments, the template size is 150×150 pixels. A template is located at the center of the image and tracking is applied on its warped versions.

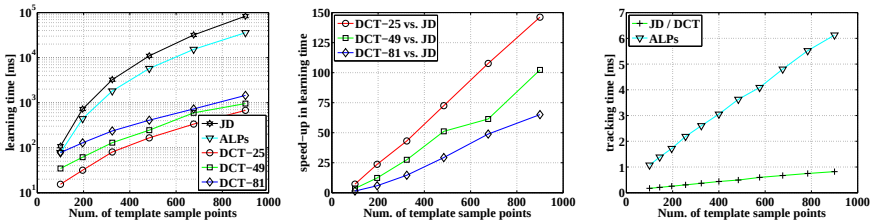
We want to emphasize that the reason for focusing on synthetic experiments in Sec. 5.1 and 5.2 is to generate a more accurate comparison using ground truth measurements. Using other methods of testing, such as using markers on real scenes to find the camera motion, generates its own error and limits the amount of motion available for testing. In addition, our evaluation also has the benefit of control which means that it is done by changing only variables that are being tested while keeping the others constant throughout the experiment. Furthermore, it allows to precisely specify the amount of change to fairly evaluate at which value the algorithm failed.

In addition to the synthetic evaluation, we also show several qualitative results from real video sequences in Sec. 5.3. These demonstrate the proposed approach used on a mobile phone for learning and tracking templates in unknown environments.

5.1 Computational Complexity

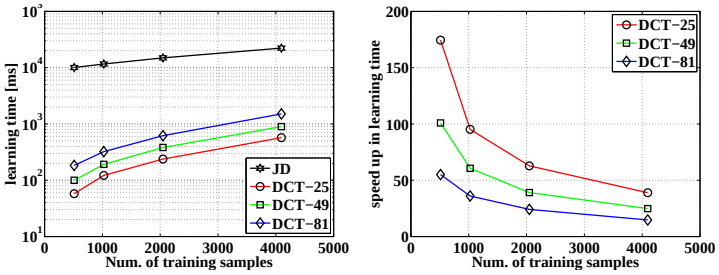
In the first evaluation, timing is measured by counting the amount of time to finish a specific part of the algorithm, *i.e.* learning and tracking. We compare the computational complexity of our approach with the approach of Jurie and Dhome [1] as well as Holzer *et al.* [2].

Learning. Our main contribution is reflected on the learning time. In Fig. 3 (a), we evaluate the amount of time necessary for learning with respect to the number of sample points. It shows that as the amount of sample points increases, the time required for learning using our approach increases much slower in comparison to the approach of both, Jurie and Dhome [1] and Holzer *et al.* [2]. As



(a) (b) (c)

Fig. 3. Comparison of timings for the approach proposed by Jurie and Dhome [1] ('JD'), the approach of Holzer *et al.* [2] ('ALPs'), and our approach ('DCT-x'). **(a)** Comparison of learning time. **(b)** Obtained speed-up of our approach with respect to Jurie and Dhome [1]. **(c)** Comparison of tracking time.



(a) (b)

Fig. 4. Evaluation of learning time depending on the number of training samples used for training. **(a)** Learning time of our approach ('DCT-x') in comparison to the approach of Jurie and Dhome [1] ('JD') and **(b)** the speed-up obtained by our approach with respect to the number of DCT coefficients used for training. The experiments were performed with a template-size of 150×150 pixels, where 22×22 sampling points were used.

expected, using less DCT coefficients for learning decreases the necessary time. This difference is emphasized in Fig. 3 (b) where it is evident that for templates with more than 800 sample points (*e.g.* 30×30), our approach is more than two orders of magnitude faster, *i.e.* almost 150 times faster, than Jurie and Dhome [1] if 25 DCT coefficients are used. Using 81 DCT coefficients, it is still approximately 70 times faster.

Fig. 4 compares the necessary learning time with the number of random samples used for training. Reducing the number of random samples drastically reduces the necessary time for learning Linear Predictors. Although this comes hand in hand with a decrease in tracking performance, we show in Sec. 5.2 that our approach is much more robust against this kind of reduction compared to the original approach of Jurie and Dhome [1].

Tracking. Both the original approach [1] and our approach share the same approach for tracking and therefore, have equal tracking times. Furthermore, the measure of tracking time per frame with respect to template size in Fig. 3(c) demonstrates that our approach can easily reach frame rates higher than 1000 fps even for templates with a high number of sample points. In contrast to Holzer *et al.* [2], their approach is slightly slower and the necessary time for tracking increases significantly faster as the template size increases. Considering a non-linear template tracking approach, the method of Benhimane *et al.* [11] takes about 10 ms for tracking the same templates.

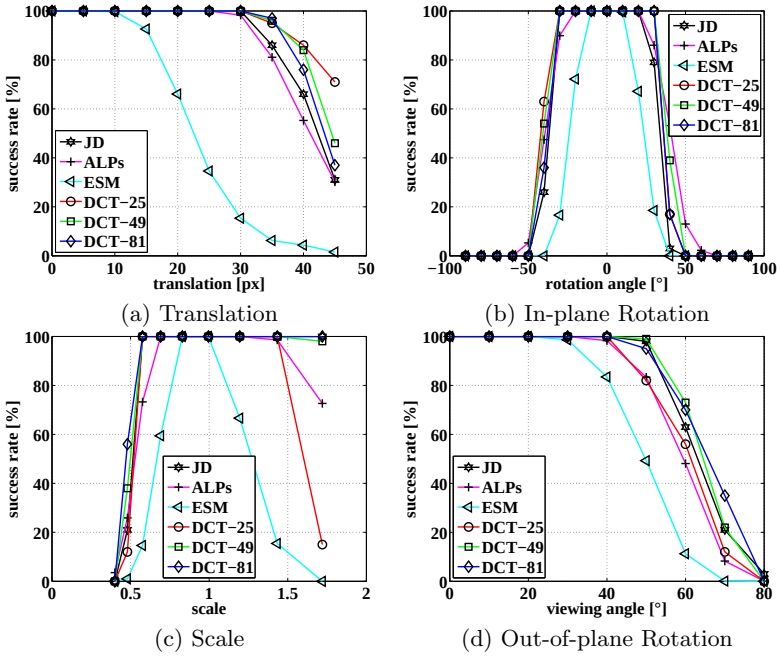
5.2 Robustness

In this section, we measure the tracking performance by finding the correct location of the template after inducing random transformations and noise to several test images. These random transformations include translation, rotation (in-plane rotation), scale and viewpoint changes (out-of-plane rotation). For the experiments on the influence of noise, we corrupted the images with noise sampled from a Gaussian distribution before applying the random transformation.

Applying these disturbances to the test images, tracking is considered successful if the mean pixel distance between the reference template corner points and the tracked template corner points, which are back-projected into the reference view, is less than 5 pixels. Hence, robustness is measured as the percentage of successfully tracked templates after applying several random disturbances to each test image.

Number of sample points. In Fig. 5, we compare the tracking robustness using different types of transformations in relation to the number of sample points. Here, we evaluate our approach with different numbers of DCT coefficients as well as the approach of Jurie and Dhome [1], Holzer *et al.* [2], and the non-linear approach of Benhimane *et al.* [11]. Hereby, the training stage as it is applied for the methods based on Linear Predictors leads to significantly better results than obtained using the non-linear approach of Benhimane *et al.* [11]. Comparing our approach with that of Jurie and Dhome [1] reveals that our approach is always better or comparable, except for the variant where we use only 25 DCT coefficients. This gives slightly worse results for large changes in viewing angle and scale. The approach of Holzer *et al.* [2] tends to give slightly worse results than that of Jurie and Dhome [1]. The improvement in tracking robustness using our approach can be explained by the fact that only low-frequency data is kept during the compression using the DCT and high-frequency data of the template is removed. As a result, noise and fine details, which tend to de-stabilize tracking, are removed.

Having a look at the tracking performance when varying the number of random transformations used for training (see Fig. 6), we see that the approach of Jurie and Dhome [1] lacks robustness when reducing the number of training samples while our approach still keeps high tracking performance even with re-



(a) Translation (b) In-plane Rotation

(c) Scale (d) Out-of-plane Rotation

Fig. 5. Comparison of tracking performance for the approaches proposed by Jurie and Dhome [1] ('JD'), Holzer *et al.* [2] ('ALPs'), Benhimane *et al.* [11] ('ESM'), and our approach ('DCT-x'). Four different types of motions are considered: **(a)** translation, **(b)** in-plane rotation, **(c)** scale and **(d)** out-of-plane rotation. The experiments were performed with a template size of 150×150 pixels, where 20×20 sampling points are used for JD, ALPs and our approach. ESM uses the complete template. For training we used 1200 training samples.

duced training examples. This property is useful to even further decrease the learning time if necessary, as we showed in Fig. 4.

Sensitivity to Noise. The results presented in Fig. 7 compare our proposed approach with Jurie and Dhome [1] with respect to sensitivity to noise, where the noise parameter specifies the standard deviation of the Gaussian noise and is with respect to an image value range from 0 to 255. Fig. 7 (a) shows that increasing the number of used DCT coefficients also increases the robustness against noise. Using 81 DCT coefficients, we obtain a template tracking approach which is more robust against noise than the one of Jurie and Dhome [1]. Looking at Fig. 7 (b), we see that our approach, in general, gives a smaller mean error in the tracking results.

It is noteworthy that being less sensitive to noise is an advantage in environments with bad lighting conditions, *e.g.* at night when the signal-to-noise ratio of cameras usually decreases. This is especially the case for cameras used in mobile devices.

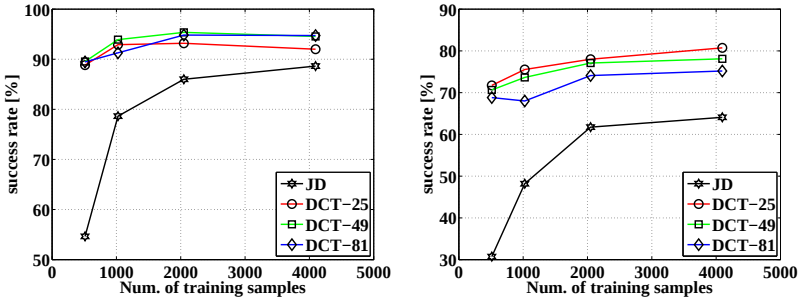


Fig. 6. Evaluation of tracking success rate with respect to the number of training samples. The left graph shows success rates for random translations in the range of 30 to 40 pixels while the right one shows them for random translations in the range of 35 to 45 pixels. For these experiments we used templates with 22×22 sample points.

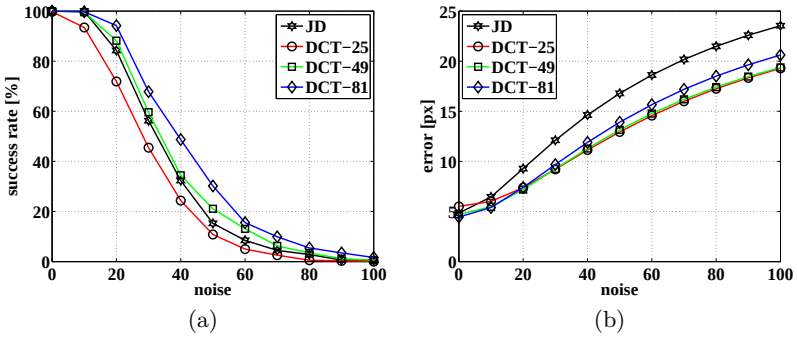


Fig. 7. Comparison of sensitivity to noise for the approach proposed by Jurie and Dhome [1] and our approach.

5.3 Exemplary Applications on Mobile Phones

To demonstrate the efficiency of the template learning and tracking, we implemented it on a standard mobile phone with a 1.2 GHz dual core processor with 1 GB of RAM. Note that we did not optimize the implementation for processor specific technology and used only a single core for the learning and tracking.

Tracking of a Single Template In Fig. 8, we show exemplary images demonstrating the tracking of a single template on a mobile phone. Learning times for a single template are approximately 18000 ms for the original approach of Jurie and Dhome [1] and about 600 ms for our proposed approach. To estimate the learning time, we trained a template with 16×16 sample points, $16 \cdot 16 \cdot 3 = 768$ training samples, and used 25 DCT coefficients in our approach. As a result, the tracking takes about 2.5 ms for both approaches.

Fig. 8. Tracking of a single template on a mobile phone.

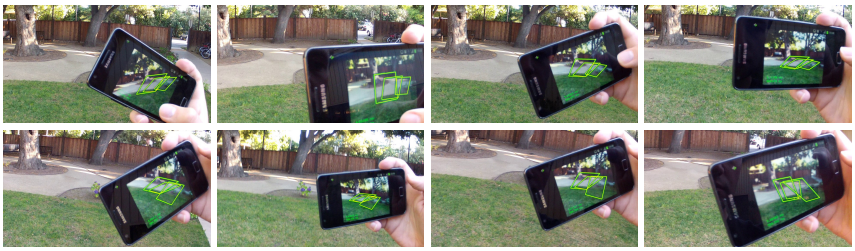


Fig. 9. Tracking of multiple templates on a mobile phone. The most right template shown in the first row failed during tracking and was replaced by a new template in the second row.

Tracking of Multiple Templates Fig. 9 shows the tracking of multiple templates. This can be useful for applications like SLAM or similar systems where a patch-based reconstruction of a scene is performed.

6 Conclusion

We proposed an efficient method for learning Linear Predictors for real-time template tracking by making use of the Discrete Cosine Transform for dimensionality reduction. This reduces the necessary computation dramatically and enables to learn Linear Predictors at run-time. We demonstrated that the introduced learning procedure leads to an improvement in handling of large motions and image noise, and showed its usefulness for mobile applications.

References

1. Jurie, F., Dhome, M.: Hyperplane approximation for template matching. IEEE Transactions on Pattern Analysis and Machine Intelligence (2002)
2. Holzer, S., Ilic, S., Navab, N.: Adaptive linear predictors for real-time tracking. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA (2010)
3. Lucas, B., Kanade, T.: An Iterative Image Registration Technique with an Application to Stereo Vision. In: International Joint Conference on Artificial Intelligence. (1981)
4. Shum, H.Y., Szeliski, R.: Construction of panoramic image mosaics with global and local alignment. International Journal of Computer Vision (2000)
5. Hager, G., Belhumeur, P.: Efficient region tracking with parametric models of geometry and illumination. IEEE Transactions on Pattern Analysis and Machine Intelligence (1998)

6. Cascia, M., Sclaroff, S., Athitsos, V.: Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. IEEE Transactions on Pattern Analysis and Machine Intelligence (2000)
7. Dellaert, F., Collins, R.: Fast image-based tracking by selective pixel integration. In: ICCV Workshop of Frame-Rate Vision. (1999)
8. Baker, S., Matthews, I.: Equivalence and efficiency of image alignment algorithms. In: Conference on Computer Vision and Pattern Recognition, Los Alamitos, CA, USA (2001)
9. Baker, S., Matthews, I.: Lucas-kanade 20 years on: A unifying framework. International Journal of Computer Vision (2004)
10. Malis, E.: Improving vision-based control using efficient second-order minimization techniques. In: IEEE International Conference on Robotics and Automation. (2004)
11. Benhimane, S., Malis, E.: Homography-based 2d visual tracking and servoing. International Journal of Robotics Research (2007)
12. Jurie, F., Dhome, M.: Real time robust template matching. In: British Machine Vision Conference. (2002)
13. Gräßl, C., Zinßer, T., Niemann, H.: Efficient hyperplane tracking by intelligent region selection. In: Image Analysis and Interpretation. (2004)
14. Parisot, P., Thiesse, B., Charvillat, V.: Selection of reliable features subsets for appearance-based tracking. Signal-Image Technologies and Internet-Based System (2007)
15. Matas, J., Zimmermann, K., Svoboda, T., Hilton, A.: Learning efficient linear predictors for motion estimation. In: Computer Vision, Graphics and Image Processing. (2006)
16. Mayol, W.W., Murray, D.W.: Tracking with general regression. Journal of Machine Vision and Applications (2008)
17. Zimmermann, K., Matas, J., Svoboda, T.: Tracking by an optimal sequence of linear predictors. IEEE Transactions on Pattern Analysis and Machine Intelligence (2009)
18. Hinterstoisser, S., Benhimane, S., Navab, N., Fua, P., Lepetit, V.: Online learning of patch perspective rectification for efficient object detection. In: Conference on Computer Vision and Pattern Recognition, Anchorage, Alaska (2008)
19. Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., Lepetit, V.: Real-time learning of accurate patch rectification. (2009)
20. Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., Navab, N.: Dominant orientation templates for real-time detection of texture-less objects. (2010)
21. Holzer, S., Hinterstoisser, S., Ilic, S., Navab, N.: Distance transform templates for object detection and pose estimation. (2009)
22. Özuysal, M., Fua, P., Lepetit, V.: Fast Keypoint Recognition in Ten Lines of Code. In: Conference on Computer Vision and Pattern Recognition, Minneapolis, MI, USA (2007)
23. Dame, A., Marchand, E.: Accurate real-time tracking using mutual information. In: Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on. (2010) 47 –56
24. Richa, R., Sznitman, R., Taylor, R., Hager, G.: Visual tracking using the sum of conditional variance. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. (2011) 2953 –2958
25. Gräßl, C., Zinßer, T., Niemann, H.: Illumination insensitive template matching with hyperplanes. In: Proceedings of Pattern recognition: 25th DAGM Symposium, Magdeburg, Germany (2003)

APPENDIX

D

MULTI-LAYER ADAPTIVE LINEAR PREDICTORS FOR REAL-TIME TRACKING

Own contributions. My contributions to this work include the core idea, the desig,n and the implementation of the methods for adapting Linear Predictors online, the handling of occlusions, as well as the evaluation of those. The core idea of the paper was refined in collaboration with the other co-authors. All co-authors were involved in the writing of the paper.

Multi-Layer Adaptive Linear Predictors for Real-Time Tracking

Stefan Holzer, *Student Member, IEEE,* Slobodan Ilic, *Member, IEEE,* and Nassir Navab, *Member, IEEE*

Abstract—Enlarging or reducing the template size by adding new parts, or removing parts of the template according to their suitability for tracking requires the ability to deal with the variation of the template size. For instance, real-time template tracking using linear predictors, although fast and reliable, requires using templates of a fixed size and does not allow on-line modification of the predictor. To solve this problem we propose the Adaptive Linear Predictors (ALPs), which enable fast online modifications of pre-learned linear predictors. Instead of applying a full matrix inversion for every modification of the template shape, as standard approaches to learning linear predictors do, we just perform a fast update of this inverse. This allows us to learn the ALPs in a much shorter time than standard learning approaches, while performing equally well. Additionally, we propose a multi-layer approach to detect occlusions and use ALPs to effectively handle them. This allows us to track large templates and modify them according to the present occlusions.
We performed exhaustive evaluation of our approach and compared it to standard linear predictors and other state of the art approaches.

Index Terms—Template tracking, linear predictors.

1 Introduction

Template tracking has been studied extensively and used in many computer vision applications such as vision-based control, human-computer interfaces, surveillance, medical imaging and reconstruction.

While there are many template tracking approaches based on the analytical derivation of the Jacobian [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], learning-based methods [11], [12], [13], [14], [15], [16], [17] have proved to be faster and generally more robust with respect to large perspective changes.

A very successful learning-based template tracker was proposed by Jurie and Dhome [11]. It is based on the learning of linear predictor to efficiently compute template parameter updates. The costly off-line learning phase, however, prohibits this method from computing templates of varying sizes online.

Yet, the ability to dynamically change the template size is necessary in many applications. In indoor SLAM, *e.g.*, the 3D geometry of the scene is a priori unknown making it necessary to initially rely on planar structures. In this case it is preferable to start from small-sized templates, in order to reduce the risk of loosing track due to non-planar structures, and to grow or shrink them online. Thus, the learning of large templates can be distributed over multiple frames, while keeping the failure rate low. In combination with a planarity check this strategy enables online segmentation of planar structures and the reliable maintenance of large templates. As a result, the set of initially tracked templates evolves towards a relatively small number of comparably large, optimally shaped templates, yielding increased robustness.

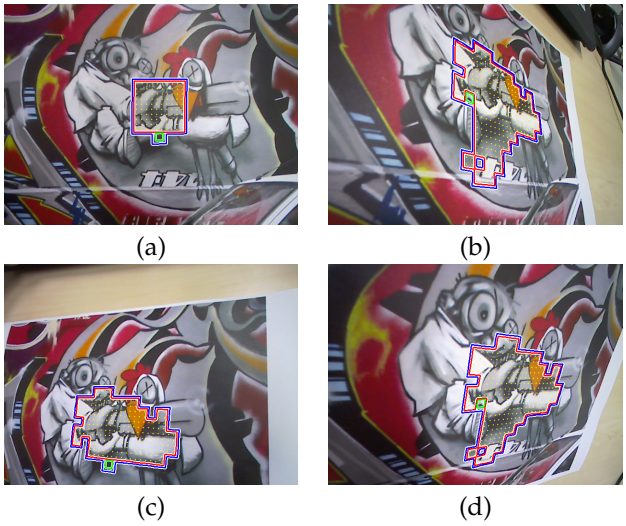


Fig. 1. (a) A small initial template is (b) enlarged according to a tracking quality measure. The template is tracked over time and (c) reduced if parts of it go out of sight. The removed parts are reinserted (d) as soon as they become visible again.

Current learning-based tracking approaches, like [11], use templates of a fixed size, because the computation of the linear predictors requires the costly inversion of a large, template-specific matrix. Since this is the computationally most expensive part of the learning process, the effort for changing the template size is nearly equivalent to that of learning a new template from scratch. Hence, to overcome the limitations of fixed size template approaches, while maintaining their robustness to large

• *S. Holzer, S. Ilic and N. Navab are with the Department of Computer Science, Technical University of Munich (TUM), Boltzmannstrasse 3, 85748 Garching, Germany.*
E-mail: {holzers,slobodan.ilic,navab}@in.tum.de

perspective changes, we propose an extension to linear predictors which allows efficient online modification of the template size. Instead of computing the inversion of the whole matrix every time the template shape changes, we present a computationally efficient way of updating the inverse which dramatically reduces the time needed for learning. We start with a small initial template and enlarge it by small extension templates, as shown in Fig. 4, according to their suitability for tracking. As long as the object to track is planar, our approach can expand the template in any direction, resulting in an arbitrarily shaped template, as shown in Fig. 1. This breaks the standard, rectangular shape assumption widely used in current template tracking approaches and can be seen as a first step towards a dense SLAM system.

Moreover, the ability to shrink and grow templates also enables us to handle occlusions. This, however, requires to detect occlusions in the first place, since it is usually barely possible to distuinguish between errors caused by occlusions and those caused by motions. We use an occlusion detection technique based on multiple layers of differently sized templates. In contrast to previous approaches [12], [17] which track all templates simultaneously, we track them sequentially. This means that the results of the tracking at higher layers are used to initialize the tracking of smaller templates at lower layers. If due to occlusion tracking at some layer fails, the smaller templates at the next lower layer are used for tracking. The tracking failure of some templates at the lowest layer indicates occlusion and corresponding regions are removed from the largest template of the top layer. This allows us to benefit from the stable tracking of a large template while using smaller templates for occlusion detection, which guide the change of the shape of the large template.

To further maximize the robustness of the template tracking with respect to large motion in case of occlusions, we introduce the concept of observed and insecure regions. These regions are used to detect incoming occlusions before they influence the tracking process. Finally, to achieve high tracking robustness we track a number of large, shifted templates at the top layer in parallel fashion.

We perform extensive quantitative evaluations and compare our approach to the standard approach of Jurie and Dhome [11] under different transformations and noise levels and with other state-of-the-art template tracking approaches [10], [17]. We demonstrate that our approach performs better or equally well with respect to the related approaches, while requiring much shorter learning times when templates are extended and shrinked. In case of occlusions, where we explicitly detect and handle them, it is superior to the other related works. In the remainder of the paper we will discuss related work on template tracking, give a detailed description of our approach on template extension and reduction, introduce our approach for occlusion detection, present our results and show examples on real world sequences.

2 Related Work

Since the seminal work of Lucas and Kanade [1], many efforts have been made in the field of template tracking and image alignment. Most of the presented approaches can be put into one of two categories: template tracking based on the analytical derivation of the Jacobian [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] or based on learning [11], [12], [13], [14], [15], [16], [17]. While analytical approaches are generally more flexible regarding template shape modifications at run-time, learning approaches enable higher tracking speed and are more robust in terms of large perspective changes.

A large variety of analytical tracking approaches have been presented since the work of Lucas and Kanade [1]. Amongst others, the variations in the presented analytical tracking approaches include different update rules of the warp [1], [3], [4], [2], [5], [6], different orders of approximations of the error function [8], [9], [10], occlusion [3], [12], [16], [17], [18], [19] and illumination change handling [3]. Basically, there are four different types of update rules, the additive approach [1], the compositional approach [2], the inverse additive approach [3], [4], and the inverse compositional approach [5], [6]. In the latter two, the roles of the reference and current image are switched, which allows to do some of the computations during an initialization phase, making the tracking computationally very efficient. Faster convergence rate can additionally be obtained by using a second-order instead of a first order approximation of the error function [8], [9], [10]. Furthermore, Hager and Belhumeur [3] showed how illumination changes and occlusions can be efficiently handled using iterativly reweighted least squares for occlusion handling. The weights are determined using a robust error function treating large image intensity differences as occlusions. Baker et. al [18] describe how such robust error functions can be incorporated into the inverse compositional template tracking framework. However, since large motion as well as occlusions generally result in large image intensity differences it is hardly possible to distinguish between them. For a more detailed overview of analytical tracking methods refer to Baker and Matthews [7].

In contrast to analytical tracking methods, Jurie and Dhome [11] proposed an approach for the learning of linear predictor using randomly warped samples of the initial template. The linear predictors are then used to predict the parameter updates during tracking. This allows very fast tracking, since the "Jacobians" are initially computed once and for all and the parameter updates can be obtained by simple matrix vector multiplications. In [12] the authors also extend the approach in order to handle occlusion. This extension uses multiple layers of differently sized trackers, which are applied in parallel fashion. All available trackers then vote for their resulting pose change and the final pose change is found by iteratively subdividing the space of possible pose changes

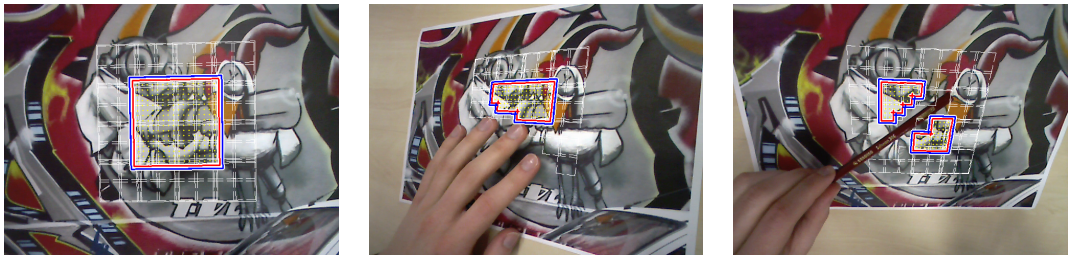


Fig. 2. Shows the tracking and adaption of a template according to present occlusions.

and selecting the N best sub-spaces. This is repeated until the resulting sub-spaces are sufficiently small so that they can be considered as a single pose change. However, as soon as occlusion occurs and tracking the largest template fails the robustness of the tracking is reduced. Gräßl *et al*. [20] show how the robustness of the linear-predictor based approach can be further increased with regard to illumination changes. In [13] they also present an intelligent way of selecting the points for sampling the image data, in order to increase the tracking accuracy. Another linear predictor approach [17] describes templates consisting of many small templates and tracks these small templates independently. The approach uses one layer of small-sized trackers, which allows for the computation of independent pose updates. A common pose is computed using RANSAC, which makes the approach robust against occlusions. However, due to the small size of the trackers this approach is less robust against large motion. Instead of using linear predictors, Mayol and Murray [16] present an approach that fits the sampling region to pre-trained samples using general regression. In order to increase the robustness against occlusions they use two filters. The first filter uses a weighted scheme in order to define whether the current scene fits the training samples or not. In case that it does not fit, it is considered as an occlusion. The second filter prohibits using the updated parameters from one iteration, if the obtained image value error increases. In both cases, the parameters from the previous frame iteration are kept. Therefore, this only works if the tracked object does not significantly move while occlusion is present. Patras and Hancock [19] use a particle filter approach that samples multiple observations and estimates the relevance of each of these observations. For each of the observations considered relevant they then use Bayesian Mixtures of Experts to compute a probabilistic prediction of the new pose. In that way, observations which include occlusions are avoided.

All of the proposed learning approaches, do not deal with templates of variable sizes. To overcome this limitation we developed a method that extends the approach of Jurie and Dhome [11] to allow online template size adaptation. We also propose a multi-layer approach for occlusion detection which, by contrast to other approaches based on linear predictors, is stable and precise. This is achieved by using relatively large templates for tracking, compared to other approaches which rely on tracking multiple small templates. Tracking small-sized templates is sensitive to large motion and such motion can be confused with occlusions. We benefit from the increased robustness against large motion and successfully handle detected occlusions by employing the proposed method for fast on-line adaption of the template size.

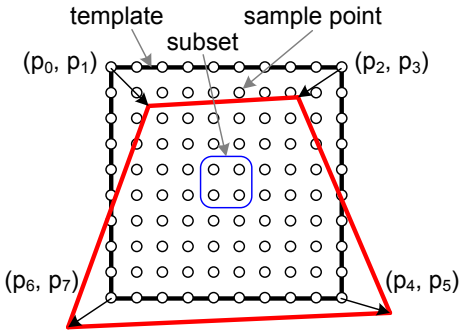


Fig. 3. A template is represented by a set of regularly placed sample points, which are grouped into subsets of four points. The pose of a template is parameterized using four corner points.

3 Background and Terminology

In this section we introduce our notation and, for the sake of completeness, review the original template tracking approach proposed by Jurie and Dhome [11].

3.1 Template and Parameter Description

A template consists of a set of n_p sample points, which are distributed over the template region and are used to sample image data. The template parameters $\boldsymbol{\mu}$ describe the current deformation of the template within an image. Within this paper we use a homography to represent the current perspective distortion of a planar template and parameterize it using four points as shown in Fig. 3. Note that our approach can also be easily adapted to any other parameterizable template deformation.

The sample points are arranged in a regular grid and grouped together into subsets of four points as shown in Fig. 3. The usefulness of this grouping will be justified later in Section 4.1, when we describe our approach for template extension. However, neither the approach of

Jurie and Dhome [11] nor our approach are restricted to this special kind of sample point arrangement. The image values obtained from the sample points, warped according to the current template parameters $\boldsymbol{\mu}$, are arranged in a vector $\mathbf{i} = \left(i_1, i_2, \ldots, i_{n_p}\right)^T$.

3.2 Template Tracking based on Linear Predictors

The goal of template tracking is to follow a reference template, defined by a vector $\mathbf{i}_R$ of reference image values and an initial parameter vector $\boldsymbol{\mu}_R$, over a sequence of images. The basic approach for this is to compute a vector $\delta\mathbf{i} = \mathbf{i}_C - \mathbf{i}_R$ of image differences, where the vector $\mathbf{i}_C$ stores the image values extracted from the current image. This vector is then used to estimate a vector of parameter differences $\delta\boldsymbol{\mu}$ used to update the current template parameters $\boldsymbol{\mu}$ so that the position of the template within the current image is optimized.

Instead of explicitly minimizing an error function, *e.g.* by iteratively solving a first- or second-order approximation of it, Jurie and Dhome [11] use a learned matrix $\mathbf{A}$ to compute $\delta\boldsymbol{\mu}$ based on the vector $\delta\mathbf{i}$ as:

$$\delta\boldsymbol{\mu} = \mathbf{A}\delta\mathbf{i}. \tag{1}$$

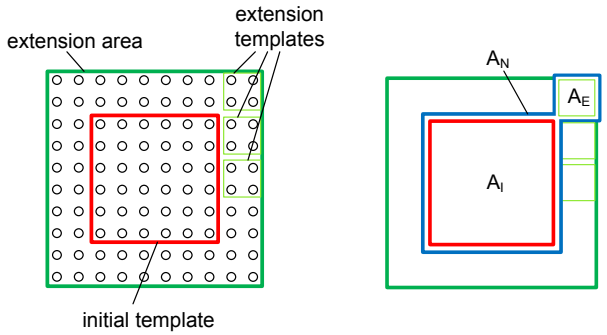
Here, the matrix $\mathbf{A}$ can be seen as a linear predictor. In order to learn $\mathbf{A}$, we apply a set of n_t random transformations to the initial template. This is done by applying small disturbances $\delta\boldsymbol{\mu}_i$, $i = 1, \ldots, n_t$, to the reference parameter vector $\boldsymbol{\mu}_R$. Then, each of these transformations is used to warp the sample points in order to obtain the corresponding vectors $\mathbf{i}_i$ of image values. The image value vector $\mathbf{i}_R$, obtained using the reference parameters $\boldsymbol{\mu}_R$, is used to compute the image difference vectors $\delta\mathbf{i}_i = \mathbf{i}_i - \mathbf{i}_R$ for each of the random transformations. These vectors of parameters and image differences are combined in the matrices $\mathbf{Y} = \left(\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \ldots, \delta\boldsymbol{\mu}_{n_t}\right)$ and $\mathbf{H} = (\delta\mathbf{i}_1, \delta\mathbf{i}_2, \ldots, \delta\mathbf{i}_{n_t})$. In general, n_t is chosen so that it is much bigger than n_p. Using these matrices Eq. 1 can be written as $\mathbf{Y} = \mathbf{AH}$. Finally, the matrix $\mathbf{A}$ is learned by minimizing:

$$\arg\min_{\mathbf{A}} \sum_{k=1}^{n_t} \left(\delta\boldsymbol{\mu}_k - \mathbf{A}\delta\mathbf{i}_k\right)^2 \tag{2}$$

which results in the closed-form solution:

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^T \left(\mathbf{H}\mathbf{H}^T\right)^{-1}. \tag{3}$$

In practice, we normalize the extracted image data with zero mean and unit standard deviation, which increases the robustness against illumination changes. In order to prevent $\mathbf{H}\mathbf{H}^T$ from being rank deficient, we add random noise to the obtained image value difference vectors. Additionally, we apply a multi-predictor approach, where multiple levels of linear predictors $\mathbf{A}_1, \ldots, \mathbf{A}_{n_l}$ are learned for one template, with n_l being the number of predictors per template. Thereby, the first linear predictor $\mathbf{A}_1$ is learned for large motions and the following predictors are learned for subsequently smaller motions.



(a) (b)

Fig. 4. **(a)** The initial template together with possible extension templates defined by the corresponding extension area. **(b)** Different template areas and their corresponding linear predictors. The red border defines the initial template with its predictor $\mathbf{A}_I$, the light green border defines an extension template with its predictor $\mathbf{A}_E$ and the blue border defines the new extended template with its predictor $\mathbf{A}_N$.

During tracking we sequentially apply the linear predictors. Additionally, every predictor is iteratively used n_i times. Within this paper we use five different levels of predictors per template and three iterations for each of the predictors. Alg. 1 formalizes the applied tracking approach.

Algorithm 1 Tracking without Occlusion Handling

```
function Track (in Image I,
    in/out TemplateParameters μ)
Compute homography T_μ from μ.
for level = 1 → n_l do
  for iteration = 1 → n_i do
    Extract image data from I at sample points
    warped with T_μ.
    Normalize image data.
    Compute image difference vector δi.
    Compute parameter update δμ = A_level δi.
    Compute homography T_δμ from δμ.
    T_μ ← T_μ T_δμ.
  end for
end for
Compute μ from T_μ.
```

4 Template Adaption

In this section we describe our approach for adapting the template by extending or reducing its size. This enables us to start tracking with a small-sized template and grow or shrink it over time, automatically adapting its size and corresponding linear predictor according to the tracked scene.

4.1 Template Extension

In the following we denote the linear predictor of an initial template with $\mathbf{A}_I$, and the linear predictor of an

extension template with $\mathbf{A}_E$ as depicted in Fig. 4. Using the standard approach of Sec. 3.2, the separate predictors would be learned as:

$$\mathbf{A}_I = \mathbf{Y}\mathbf{H}_I^T \left(\mathbf{H}_I\mathbf{H}_I^T\right)^{-1} \text{ and} \quad (4)$$

$$\mathbf{A}_E = \mathbf{Y}\mathbf{H}_E^T \left(\mathbf{H}_E\mathbf{H}_E^T\right)^{-1}, \quad (5)$$

where $\mathbf{Y}$ stores the same random transformations for both linear predictors. The standard approach for learning a combined predictor $\mathbf{A}_N$ for the entire template leads to:

$$\mathbf{A}_N = \mathbf{Y}\mathbf{H}_N^T \left(\mathbf{H}_N\mathbf{H}_N^T\right)^{-1} \quad (6)$$

$$= \mathbf{Y}\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}^T \left(\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}^T\right)^{-1} \quad (7)$$

$$= \mathbf{Y}\begin{bmatrix} \mathbf{H}_I \\ \mathbf{H}_E \end{bmatrix}^T \left(\begin{bmatrix} \mathbf{H}_I\mathbf{H}_I^T & \mathbf{H}_I\mathbf{H}_E^T \\ \mathbf{H}_E\mathbf{H}_I^T & \mathbf{H}_E\mathbf{H}_E^T \end{bmatrix}\right)^{-1}. \quad (8)$$

Now, instead of directly updating the old linear predictor $\mathbf{A}_I$ we will update the matrix $\mathbf{S}_I = \left(\mathbf{H}_I\mathbf{H}_I^T\right)^{-1}$ using the formulas presented by Henderson and Searle [21], so that we obtain the matrix $\mathbf{S}_N = \left(\mathbf{H}_N\mathbf{H}_N^T\right)^{-1}$. Let $\mathbf{S}_{11}$, $\mathbf{S}_{12}$, $\mathbf{S}_{21}$ and $\mathbf{S}_{22}$ be the four sub-matrices of $\mathbf{S}_N$:

$$\mathbf{S}_N = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{H}_I\mathbf{H}_I^T & \mathbf{H}_I\mathbf{H}_E^T \\ \mathbf{H}_E\mathbf{H}_I^T & \mathbf{H}_E\mathbf{H}_E^T \end{bmatrix}\right)^{-1}. \quad (9)$$

Then, we can update $\mathbf{S}_I$ to $\mathbf{S}_N$ using

$$\mathbf{S}_{11} = (\mathbf{H}_I\mathbf{H}_I^T)^{-1} + (\mathbf{H}_I\mathbf{H}_I^T)^{-1}\mathbf{H}_I\mathbf{H}_E^T\mathbf{S}_{22}\mathbf{H}_E\mathbf{H}_I^T(\mathbf{H}_I\mathbf{H}_I^T)^{-T} \quad (10)$$

$$\mathbf{S}_{12} = -(\mathbf{H}_I\mathbf{H}_I^T)^{-1}\mathbf{H}_I\mathbf{H}_E^T\mathbf{S}_{22}, \quad (11)$$

$$\mathbf{S}_{21} = \mathbf{S}_{12}^T, \quad (12)$$

$$\mathbf{S}_{22} = \left(\mathbf{H}_E\mathbf{H}_E^T - \mathbf{H}_E\mathbf{H}_I^T(\mathbf{H}_I\mathbf{H}_I^T)^{-1}\mathbf{H}_I\mathbf{H}_E^T\right)^{-1}, \quad (13)$$

where $(\mathbf{H}_I\mathbf{H}_I)^{-1}$ is known from the learning of the initial predictor. Therefore, the only inversion that has to be applied is for the computation of $\mathbf{S}_{22}$. However, this inversion is not a problem since the extension templates are always of a smaller size than the entire extended template and therefore $\mathbf{S}_{22}$ is small as well. In case the template is already known before run-time, *i.e.* if template extension and reduction is only used to handle occlusions, the necessary inversions can even be precomputed in order to further speed up processing. In practice, we use templates for extension and reduction which consist of 4 sample points.

Note that for the computation of the image value differences $\mathbf{H}_E$ we have to use the same random transformations as used for computing $\mathbf{H}_I$. The same holds for $\mathbf{H}_R$ in case of template reduction (see Section 4.2).

The approach as presented up to now is limited by the number of random transformations n_t, used for learning. Since n_t has to be the same for all extension templates as well as for the initial template, and since the number of random transformations has to be greater or at least equal to the number of used sample points, $n_t \geq n_p$, the maximum number of random transformations has to be known a priori. In order to remove this restriction we use the approach presented by Hinterstoisser *et al.* [22], which allows us to update the matrix $\mathbf{S}_I$ in such a way that we can increase the number of random transformations n_t without having to recompute the updated $\hat{\mathbf{S}}_I$ from scratch. This is done by using the Sherman-Morrison formula:

$$\hat{\mathbf{S}}_I = \left(\mathbf{S}_I^{-1} + \delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^T\right)^{-1} \quad (14)$$

$$= \mathbf{S}_I - \frac{\mathbf{S}_I\delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^T\mathbf{S}_I}{1 + \delta\mathbf{i}_{n_t+1}^T\mathbf{S}_I\delta\mathbf{i}_{n_t+1}}, \quad (15)$$

where $\delta\mathbf{i}_{n_t+1}$ is a vector of image value differences obtained from a new random transformation applied to the sample points. In practice, the number of random transformations is increased each time before a new extension template is added, such that $n_t = 3n_p$.

4.2 Template Reduction

In the case when already learned templates have to be reduced, *e.g.* due to the presence of non-planarity or tracking failure, the corresponding linear predictors can be computed by updating the linear predictor of the larger template. For this, we denote the linear predictor of the large template with $\mathbf{A}_L$, the predictor of the new reduction template with $\mathbf{A}_R$ and the predictor of the reduced template with $\mathbf{A}_N$.

In order to reduce the matrix $\mathbf{S}_L$, it has to be rearranged first, so that the data corresponding to the reduction template is positioned in the last rows and columns of $\mathbf{S}_L$. After the rearrangement, the reduction template can be removed using the following approach. First, let us consider the submatrices of the matrix $\mathbf{S}_L$:

$$\mathbf{S}_L = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{H}_N\mathbf{H}_N^T & \mathbf{H}_N\mathbf{H}_R^T \\ \mathbf{H}_R\mathbf{H}_N^T & \mathbf{H}_R\mathbf{H}_R^T \end{bmatrix}\right)^{-1}, \quad (16)$$

where all the sub-matrices $\mathbf{S}_{11}$, $\mathbf{S}_{12}$, $\mathbf{S}_{21}$, $\mathbf{S}_{22}$, $\mathbf{H}_N\mathbf{H}_N^T$, $\mathbf{H}_N\mathbf{H}_R^T$, $\mathbf{H}_R\mathbf{H}_N^T$ and $\mathbf{H}_R\mathbf{H}_R^T$ are available from the large template. The goal is to compute

$$\mathbf{A}_N = \mathbf{Y}\mathbf{H}_N^T \left(\mathbf{H}_N\mathbf{H}_N^T\right)^{-1} \quad (17)$$

without the need of inverting $\mathbf{H}_N\mathbf{H}_N^T$, since this is a large matrix in general. Similar to the Equations 10-13 Henderson and Searle [21] also present the formula

$$\mathbf{S}_{11} = \left(\mathbf{H}_N\mathbf{H}_N^T - \mathbf{H}_N\mathbf{H}_R^T(\mathbf{H}_R\mathbf{H}_R^T)^{-1}\mathbf{H}_R\mathbf{H}_N^T\right)^{-1}, \quad (18)$$

which can be reformulated as

$$\mathbf{H}_N\mathbf{H}_N^T = \mathbf{S}_{11}^{-1} + \mathbf{H}_N\mathbf{H}_R^T(\mathbf{H}_R\mathbf{H}_R^T)^{-1}\mathbf{H}_R\mathbf{H}_N^T. \quad (19)$$

Taking the inverse leads to the desired result:

$$(\mathbf{H}_N\mathbf{H}_N^T)^{-1} = \left(\mathbf{S}_{11}^{-1} + \mathbf{H}_N\mathbf{H}_R^T(\mathbf{H}_R\mathbf{H}_R^T)^{-1}\mathbf{H}_R\mathbf{H}_N^T\right)^{-1}. \quad (20)$$

Since we, however, have to invert a big matrix in this case, namely $\mathbf{S}_{11}$, this is not suitable for online computation. Therefore, we use the following formula presented in [21]:

$$(\mathbf{X}+\mathbf{UYU}^T)^{-1} = \mathbf{X}^{-1}-\mathbf{X}^{-1}\mathbf{UZU}^T\mathbf{X}^{-1}, \quad (21)$$

$$\mathbf{Z} = \left(\mathbf{Y}^{-1}+\mathbf{U}^T\mathbf{X}^{-1}\mathbf{U}\right)^{-1}. \quad (22)$$

By setting $\mathbf{X}=\mathbf{S}_{11}^{-1}$, $\mathbf{Y}=(\mathbf{H}_R\mathbf{H}_R^T)^{-1}$ and $\mathbf{U}=\mathbf{H}_N\mathbf{H}_R^T$ we obtain our desired result:

$$(\mathbf{H}_N\mathbf{H}_N^T)^{-1} = \mathbf{S}_{11}-\mathbf{S}_{11}\mathbf{H}_N\mathbf{H}_R^T\mathbf{D}\mathbf{H}_R\mathbf{H}_N^T\mathbf{S}_{11}, \quad (23)$$

$$\mathbf{D} = \left(\mathbf{H}_R\mathbf{H}_R^T+\mathbf{H}_R\mathbf{H}_N^T\mathbf{S}_{11}\mathbf{H}_N\mathbf{H}_R^T\right)^{-1} \quad (24)$$

Now the necessary inversion is no longer a problem since the reduction template is chosen to be of small size and computing $\mathbf{D}$ is not expensive.

4.3 Practical Issues

In this section we discuss practical issues. These are the normalization of the image data and the estimation of the subset quality, which is used for the selection of the next extension template.

4.3.1 Normalization

As mentioned before, the image values are normalized to zero mean and unit standard deviation. However, instead of doing this globally by considering all of the image values of the template we apply a local normalization, where each subset is normalized by considering only its image values and the image values of its direct local neighboring subsets. This normalization is applied to the reference data, the learning data and the current image data during tracking. The local normalization is superior to the global normalization since in the case of the global normalization the mean and standard deviation of the whole image data change, if new parts are added to the template or some parts are removed.

4.3.2 Suitability Criterion for Subset Selection

In order to decide which subset should be chosen for extending the current template, we compute a quality measure for each of the potential extension templates in the local neighborhood of the current template. This is done by learning a local predictor $\mathbf{A}_S=\mathbf{Y}_S\mathbf{H}_S^T(\mathbf{H}_S\mathbf{H}_S^T)^{-1}$ for this subset at first, where the image data $\mathbf{H}_S$ is collected using the set of random transformations represented by $\mathbf{Y}$. Then, using this predictor together with the collected image data we compute a prediction $\hat{\mathbf{Y}}_S$ of $\mathbf{Y}$ as

$$\hat{\mathbf{Y}}_S=\mathbf{A}_S\mathbf{H}_S. \quad (25)$$

Finally, we compute a similarity measurement, which defines the quality q_S of the corresponding subset as

$$q_\mathrm{s}=\frac{1}{n_t}\sum_{i=1}^{n_t}\frac{\hat{\mathbf{y}}_{\mathrm{s}i}\mathbf{y}_i^T}{|\hat{\mathbf{y}}_{\mathrm{s}i}||\mathbf{y}_i|}, \quad (26)$$

where $\mathbf{y}_i$ and $\hat{\mathbf{y}}_{\mathrm{s}i}$ are the i-th column vector of $\mathbf{Y}$ and $\hat{\mathbf{Y}}_\mathrm{S}$ respectively. This measures the similarity of the prediction and the data used for learning by computing the mean angle between the corresponding parameter vectors. This way an extension template is chosen which ensures best that each tracking iteration brings the template parameters closer to the desired result. The current template will then be extended using the subset with the highest quality measure. The suitability criterion is only computed on a subset, which consists of 4 sample points, and not on the whole resulting template, since this would be computationally to expensive. In practice, we use the suitability criterion to select the best subtemplates within a user-defined area such that we obtain a template with a certain number of sample points, which is also defined by the user.

5 Occlusion-Aware Tracking

In order to make template tracking robust against occlusions, we detect occlusions and consider them during the computation of the template pose parameters. Since this computation depends on the image differences between the current image and the template warped according to the pose parameters of the previous frame, the difficulty is to distinguish whether these differences come from the camera/template motion or from real occlusions. To distinguish this, we propose a multi-layer approach where on the top level we use a relatively large template for tracking and on the lower levels smaller sized templates are used to detect occlusions. Tracking a large template is more stable compared to multiple smaller sized templates, thus we always track a large template while failures in tracking of small sized templates indicate presence of occlusions. We use the technique proposed in Sec. 4 to remove the occluded parts from the large top level template and add them back as soon as they are visible again. This allows us to handle complex situations, such as occlusions passing over the entire template or moving templates with a hole as demonstrated in the results section. In the remainder of this section we discuss the proposed approach. An overview over the proposed approach is given in Alg. 2.

5.1 Multi-Layer Approach

Similar to the approach of Jurie and Dhome [12] we propose to use multiple layers of linear predictor grids, where the sizes of the templates, which correspond to the linear predictors, vary over the different layers. Fig. 5(a) shows such a layered organization. The size of the templates decreases with the depth of the layer. The template of the top layer (Layer 1 in Fig. 5(a)) corresponds to the actual template size. The template is subdivided in the next level (Layer 2 in Fig. 5(a)) to four equally sized templates. Every template is further subdivided into four new templates at the next layer. In practice, we use only three layers. This is sufficient to handle occlusions. In contrast to Jurie and Dhome [12] who applied linear

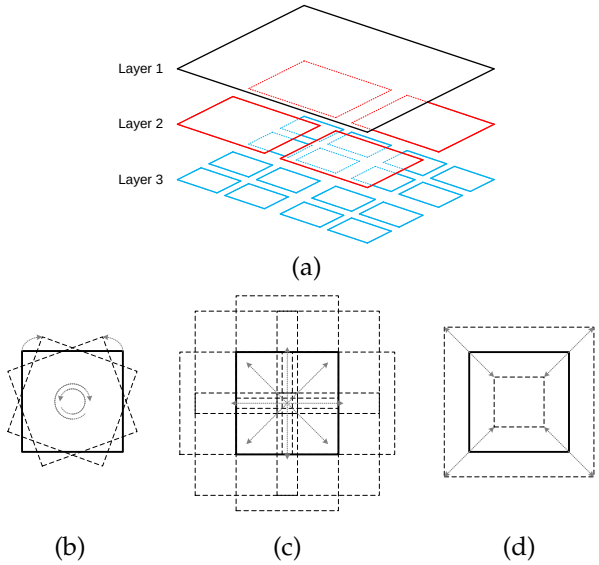


Fig. 5. **(a)** The organization of the multiple layers used for tracking. The sub-figures **(b)**, **(c)** and **(d)** show different transformed templates of the top layer used to increase the robustness against large motion. **(b)** shows differently rotated, **(c)** differently translated and **(d)** differently scaled templates.

predictors for each template at the same time, we do it sequentially. This means that in the first layer, we track a single large template. The resulting pose is then used as an initialization for the grid of trackers in the next layer. This is repeated for all available layers.

Occluded and non-occluded case. During the tracking, we can distinguish two possible scenarios. One is when there are no occlusions present and the other when occlusions intervene. If there are no occlusions, the successful tracking result at a higher level is forwarded to the next lower level. In general, the tracking of small templates can fail in the case of large motion. That is where the approach of Jurie and Dhome [12], in which all the templates at all the layers are tracked simultaneously, tends to fail. In our approach, the tracking of small templates at lower levels is preconditioned by the result of the preceding tracker layer. In the other scenario we consider that occlusions happen. In this case, depending on the size of the occluded area, the trackers at higher levels fail. Those on lower levels are then used to directly estimate the pose initialized by the resulting pose in the previous frame. The failure detection is done using a simple threshold applied on the mean image intensity differences of the normalized image data. In practice, we use different thresholds for each layer, for the first layer we use 0.03 as threshold on the mean image intensity differences of the normalized image data, 0.08 for the second layer and 0.15 for the third layer.

5.2 Combining multiple pose estimates.

Since the layers which consist of more than one template can lead to different pose parameters, an outlier rejection has to be applied, which removes erroneous results. For this, we consider the corner points of each template as single feature points and use RANSAC to robustly estimate a homography from them. Based on this, we only consider a template to be successfully tracked if its corner points are not rejected as outliers after the homography estimation. The final homography of each layer is then computed by considering the corner points of all the successfully tracked templates. In order not to replace a successful homography of a higher level by a failed tracking at a lower level, the homography is only updated if the change is not too big. The homography computed in the last layer is used for both pose estimation of the top layer template as well as for occlusion detection.

In order to increase robustness in the case of occlusions and large motion, we adapt the template size of the top layer using our approach discussed in the previous section. However, this has to be done before an occlusion occurs. Therefore, we introduce the concept of observed and insecure regions.

5.3 Observed and Insecure Regions

5.3.1 Observed Regions

The observed regions are placed within an area around the top layer template, as shown in Fig. 6, similar to the extension area in [23]. Here, these regions are used for early detection of incoming occlusions. In order to detect occlusions within the observed regions, we add corresponding trackers to the lower layers so that they cover these regions as depicted in Fig. 6. Then, these additional trackers are used together with the other trackers of the specific layer for multi-layer tracking. Successful trackers are considered in the RANSAC-process described in Sec. 5.1.

Detecting occlusions. After estimating the global pose using the multi-layer approach, we re-evaluate the image intensity differences of the lowest-layer templates as well as of the small subsets of the top layer template using the corresponding mean image intensity differences. These errors are then used to decide whether an occlusion is present or not. In practice, we use a value of 0.2 as threshold on the mean image intensity difference to decide whether a subset is occluded or not. In order to reduce the impact of noise we only consider subsets to be occluded if they are not covered by a successfully tracked template at one of the layers.

5.3.2 Insecure Regions

Knowing the positions of present occlusions, we define insecure regions around these occlusions as shown in Fig. 6. These insecure regions are considered as having a high chance of becoming occluded in the next frame. Therefore, the occlusions as well as the corresponding insecure regions are removed from the top layer template using techniques outlined in Sec. 4 as described below.

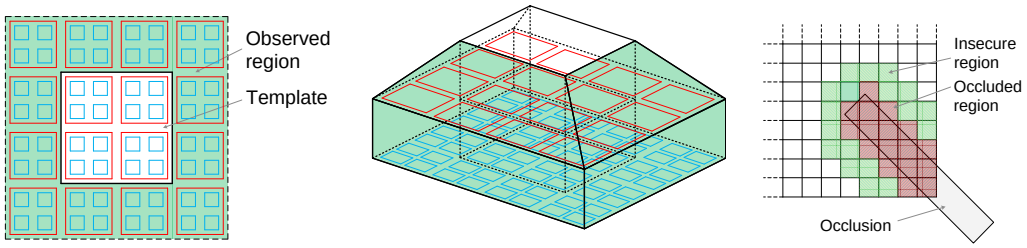


Fig. 6. The left figure shows the multi-layered template with its observation region depicted as green area. The middle figure shows the different layers and their contribution to the observed region from a side-view. The right figure illustrates insecure regions, which are areas around the detected occlusions.

5.4 Template Adaption

Once the occlusions and their corresponding insecure regions are detected, the shape of the top level template T_0 and its corresponding linear predictors are adapted. This means that as soon as a subset of T_0 is covered by an occluded part or a insecure region, it is removed from T_0. When the occlusion disappears and no longer covers the previously occluded parts of the template, they are added back to the template T_0. This enables us to continuously use a large template for tracking even in the case of occlusion and therefore, to maintain robustness against large motion.

5.5 Increased Robustness against Large Motions

To increase the tracking robustness in case of large motions we introduce multiple transformed versions of the template at the top layer (see Figures 5(b), (c) and (d)), which are tracked in parallel. For this, we use the same linear predictors for each of the transformed templates. However, since the multi-predictor approach as stated in Sec. 3.2 would be computationally expensive in this case we use only the first linear predictor, which is learned for the largest motions, for the transformed templates. Then, the parameters of the template that is best according to the resulting mean image intensity differences are selected and used to process the best template only. For this best template we continue with applying the remaining linear predictors of the multi-predictor approach.

Algorithm 2 Tracking with Occlusion Handling

function *TrackWithOcclusionHandling* (
  in Image **I**, ***in/out*** TemplateParameters $\boldsymbol{\mu}$)
Compute homography $\mathbf{T}_{\boldsymbol{\mu}}$ from $\boldsymbol{\mu}$.
for $layer = 1 \rightarrow 3$ **do**
  for each template i of this layer **do**
    Compute $\boldsymbol{\mu}_i$ from homography $\mathbf{T}_{\boldsymbol{\mu}}$.
    Track (I, $\boldsymbol{\mu}_i$) (see Alg.1).
    Compute homography $\mathbf{T}_i$ from $\boldsymbol{\mu}_i$.
  end for
  Combine pose estimates $\mathbf{T}_i$ to $\mathbf{T}_{layer}$ (Sec. 5.2)
  if $\mathbf{T}_{layer}$ is valid (Sec. 5.2) **then**
    $\mathbf{T}_{\boldsymbol{\mu}} \leftarrow \mathbf{T}_{layer}$.
  end if
end for
Detect occlusions (Sec. 5.3).
Adapt top-layer template (Sec. 5.4).

6 Experimental Results

6.1 Template Adaption

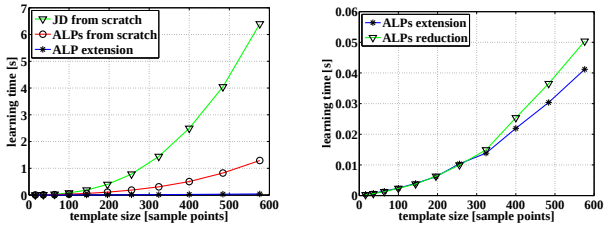
In this section we evaluate the template adaption proposed in Sec. 4 about template adaption. For this purpose, we perform an extensive comparison with several state-of-the-art approaches on template tracking. This includes comparisons with the standard learning approach of Jurie and Dhome [11], the analytical approach of Benhimane and Malis [10] and a recent approach called NoSLLip of Zimmermann *et al.* [17]. The comparisons are done in terms of tracking precision and computational efficiency. In the end we show several qualitative results from real video sequences showing tracking results with one and several templates. All of the experiments are performed on a 2.66 GHz Intel(R) Core(TM)2 Quad CPU with 8 GB of RAM, where only one core is used for the computations.

In all of the experiments the maximum random perturbation applied for learning the linear predictors is set to 21 pixels except for the comparison with NoSLLip, where we slightly increased the perturbation by 10% since this sequence contains very large motions.

The probably most important results for this section are shown in Fig. 10, where we demonstrate that our ALPs method gives similar tracking results as the approach proposed by Jurie and Dhome [11].

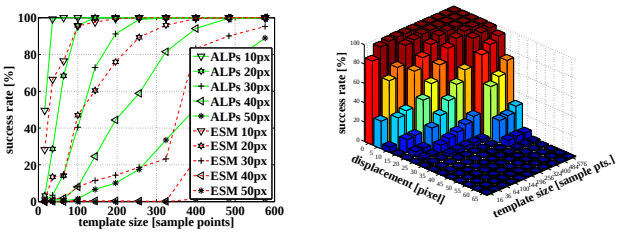
6.1.1 Comparison with Jurie-Dhome Approach

6.1.1.1 **Computational Complexity of Learning**: In Fig. 7 we show computation times for learning the linear predictors with respect to different template sizes. We compare our ALPs method, shown in red and blue, with the standard approach of Jurie and Dhome [11], depicted as a green curve in Fig. 7(a). For our approach we distinguish between two cases. In the first case, shown as a red curve, the computation of the linear predictor is done iteratively from scratch. In this case we start with a small initial template, which size is equal to the size of an extension template of Fig. 4. Such a small template is then grown until the specified size is reached. The



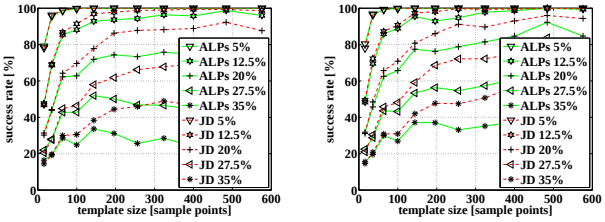
(a) (b)

Fig. 7. Comparison of the computation time necessary for learning a linear predictor using the Jurie-Dhome [11] (JD) approach (green) and using ALPs (red and blue). **(a)** For the latter case we distinguish between learning the predictor from scratch (red) and adding only one extension subset (blue) at a time. Learning from scratch means that we consider the entire time necessary to build up the template of the specified size. **(b)** Computation times for template extension and reduction, when one extension subset is added at a time. The blue curve corresponds to the blue curve at (a).



(a) (b)

Fig. 8. Comparison of the success rates with respect to different displacements and template sizes. **(a)** Performance of ESM vs. ALPs. **(b)** Performance of ESM for further displacements.



(a) (b)

Fig. 9. Comparison of the success rates of ALPs and linear predictors of Jurie and Dhome(JD) with respect to different levels of noise and different template sizes. **(a)** Success rates of ALPs using 1000 random warpings and **(b)** using 2000 random warpings.

obtained results clearly reveal that the adaptive learning of the linear predictor, which starts with the small sized template, is much more efficient than learning a linear predictor for the fixed size template. This proves that our approach can also be used for efficiently learning of a linear predictor for templates of a fixed size, starting from small templates and adapting their linear predictors until the desired template size is reached. In the following experiments for ALPs we always use this procedure for creating the linear predictors for a template. In the second case, shown as a blue curve and labeled as ALP extension, we show the time necessary to add one extension template. This is a typical case during online tracking, where the template is grown step by step. As to be expected, adding the extension template does not significantly increase computation time, when changing the template size. In Fig. 7 (b) we show computation times for the extension and reduction of templates. Note that the necessary time to grow or reduce the template by an extension template consisting of four sample points is around 0.05s for initial templates of sizes around 600 sample points, whereas computation from scratch would need over 1s using ALPs and more than 6s when using the approach of Jurie and Dhome [11].

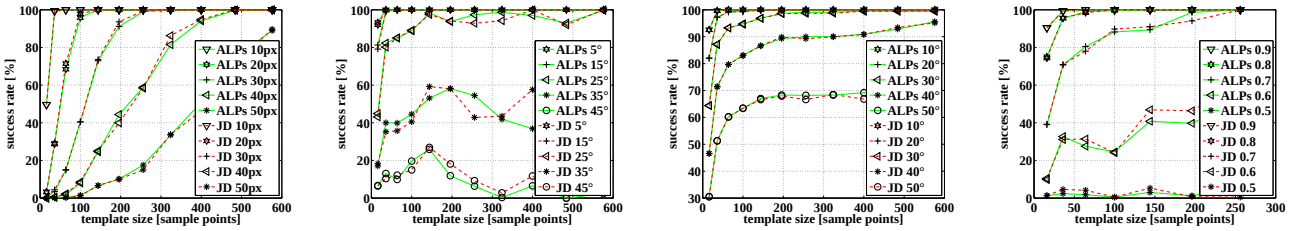
6.1.1.2 **Robustness**: To evaluate the robustness of our approach, we compare the tracking success rate of our approach with that of the standard approach proposed by Jurie and Dhome [11] for different template sizes and with respect to changes in translation (Fig. 10 (a)), in-plane rotation (Fig. 10 (b)), viewing angle (Fig. 10 (c)), and scale (Fig. 10 (d)). In addition we compare ALPs to Jurie and Dhome in respect to noise and different number of random transformations used for learning. The results are shown in Fig. 9. The robustness is measured using the tracking success rate. The accuracy is measured using the maximum mean template corner error. That is, after each tracking experiment the resulting template position is warped onto the original frame and the corner with the highest error is selected and used to compute the mean error. A template is considered as successfully tracked, if the maximum corner error is below 5 pixels.

For all of the experiments, we use synthetic images, corrupted by noise and warped according to the specific experiments. Noise is added according to $I_n(\mathbf{x}) = I(\mathbf{x}) + \epsilon$, with $\epsilon \in \left[-\alpha I_{\text{range}}/100, \alpha I_{\text{range}}/100\right]$, and $\alpha = 5$ for all experiments. An exception is the noise experiment, where different levels of noise were applied. I_{range} specifies the possible range of image values, *e.g.* $I_{\text{range}} = 255$ holds for image values between 0 and 255. In all of the experiments we also add a random displacement in the range of $[-5, 5]$ pixels, with the exception of displacement experiments, and a random change in the view-point angle ranging between $[-5°, 5°]$, again with the exception of the view-point angle experiments.

The results show that both approaches, the standard Jurie-Dhome approach as well as ALPs, yield similar success rates. The only exception is the sensitivity to noise, where the Jurie-Dhome approach performs better than ALPs. This performance difference, however, can be reduced by increasing the number of random warpings used for the learning of linear predictors, as demonstrated in Fig. 9 (b).

6.1.2 Comparison with ESM

To demonstrate the usefulness of learning-based approaches, we compare our approach with the analytical



(a) (b) (c) (d)

Fig. 10. Comparison of the success rate of ALPs and JD linear predictors with respect to changes in translation **(a)**, in-plane rotation **(b)**, viewing angle **(c)**, and scale **(d)**. In all four cases the results of both approaches are approximately equal.

Method	Frame-rate [fps]	Loss-of-locks [-/-]	Error [%]
NoSLLiP	16.8	20/1799	1.8
ALPs	96.7	10/2299	1.2

TABLE 1
Comparison between the tracking results of NoSLLiP (Matlab implementation) given in [17] and results obtained using ALPs (C++ implementation) using the PHONE sequence.

method called ESM of Benhimane and Malis [10]. This approach minimizes the energy function using a second order approximation. In Fig. 8 we compare the success rate of the ESM tracking to that of ALPs regarding different magnitudes of displacements and different template sizes. Our learning-based approach clearly outperforms ESM, especially for larger template sizes.

6.1.3 Comparison with NoSLLiP

We also compare ALPs to the approach of Zimmermann *et al*. [17] using the PHONE sequence provided by the authors. Example images of the tracking are shown in Fig. 11. The comparison between the tracking results of [17] and those obtained using ALPs are shown in Table 1. Although the number of provided images is larger than the number of images used by Zimmermann *et al*. [17], we still obtain a better loss-of-locks count. The given error values are relative to the upper edge of the template. A frame is counted as loss-of-lock if one of the template corners has an error larger than 25%. Note that the template is reduced, when it partially goes out of sight and enlarged again as it becomes visible again (see Fig. 11).

6.1.4 Usefulness of larger templates

As shown in Figures 8, 9 and 10, the success rates increase with increasing template sizes. The only exception are changes in the in-plane rotation angle, where the success rate reaches a maximum for templates with a size of approximately 100 to 200 sample points.

6.1.5 Qualitative Evaluation

In Fig. 1, 11 and 12 we show different image sequences, which demonstrate the processing of the proposed approach. In Fig. 1 and 12 we start with templates of size 10 by 10 sample points and iteratively grow them by adding the neighboring extension template with the highest quality. In Fig. 12 we demonstrate the use of multiple templates. In Figures 1 and 11 we track the templates, reduce them if they partially go out of sight and grow them back to the original size when their hidden parts become visible again.

6.2 Occlusion-Aware Tracking

In this section we compare our occlusion-aware tracking presented in Sec. 5 with different state-of-the-art linear predictor based tracking approaches in terms of robustness and accuracy with respect to different types of motion using ground truth data. This is done both in the case of the presence of occlusions and without them. Additionally, we show several qualitative results from real video sequences. All of the experiments are again performed on a 2.26GHz Intel(R) Core(TM)2 Quad CPU with 4 GB of RAM, where only one core is used for the computation.

The robustness is again measured using tracking success rate and accuracy is computed using the maximum mean template corner error as explained in Subsec. 6.1.1.2 on robustness. For all ground truth experiments, we again use synthetic images, corrupted by noise and warped according to the specific experiments as already described in Subsec. 6.1.1.2. In this case the size of the templates used for all ground truth experiments is 16×16 sample points. The tracking speed for such a template is approximately 45 frames per second.

The probably most important results for this section are shown in Fig. 15, where we demonstrate that our multi-layered approach gives better tracking results in presence of occlusion compared to the approach of Jurie and Dhome [12].

6.2.1 Experiments without Occlusion

In Fig. 13 we compare our multi-layered ALP approach of Sec. 5 with the ALP approach proposed in Sec. 4. For the ML-ALPs approach we distinguish between the one as described in Sec. 5 ('Robustified ML-ALPs') and the one without the use of the transformed versions of the top layer template as described in Sec. 5.1 ('ML-ALPs'). The results show that in most cases both versions perform better than ALPs. Only the accuracy of ALPs is sometimes better in case of large motion.

Fig. 11. Result images of the phone sequence, which is provided by Zimmermann *et al*. [17]. Note that the template is reduced if it goes out of the image and grown again if it once again becomes visible.

Fig. 12. Iterative growing of two independent templates.

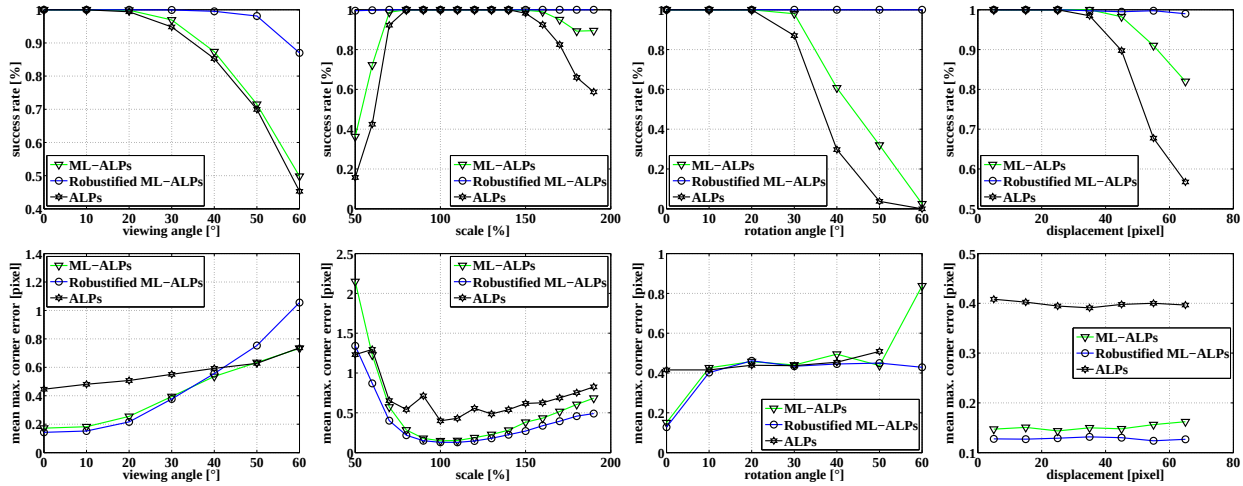


Fig. 13. Results of the comparison between the ML ALPs approach and the ALPs approach with respect to different types of motion without occlusion. The first row shows the tracking success rate and the second row the corresponding mean maximum corner errors.

In Fig. 14 we compare ML-ALPs ('ML-ALPs') with the approach of Jurie and Dhome ('ML-JD') [12]. To evaluate our method we do not use the transformed versions of the top layer template as described in Sec. 5.1 in order to present a fair comparison with the multi-layered approach of Jurie and Dhome [12]. The results show that our approach performs better than the one of Jurie and Dhome [12] with respect to robustness as well as accuracy.

6.2.2 *Experiments with Occlusion*

In Fig. 15 we compare our ML ALPs approach with the one of Jurie and Dhome ('ML-JD') [12] in the presence of occlusions. Again, we do not use multiple transformed versions of the top layer template for the sake of fair comparison. The results show that our approach performs better than the approach of Jurie and Dhome [12] with respect to robustness as well as accuracy. The only exception is the view-point angle experiment, where ML-JD gives better accuracy for large occlusion and better robustness for large occlusion and small motion. Note also that the results of the approach of Jurie and Dhome [12] with respect to changes in the view point angle are stable for varying amounts of occlusion.

6.2.3 *NoSLLiP with Occlusion*

In this section we compare ML-ALPs to the approach of Zimmermann *et al*. [17] using the MOUSEPAD sequence, which contains occlusion and is provided by the authors. Example images of the tracking are shown in Fig. 16. The comparison between the tracking results of [17] and those obtained using ML-ALPs are shown in Table 2. The given error values are relative to the upper edge of the template. A frame is counted as loss-of-lock if one of the template corners has an error larger than 25%.

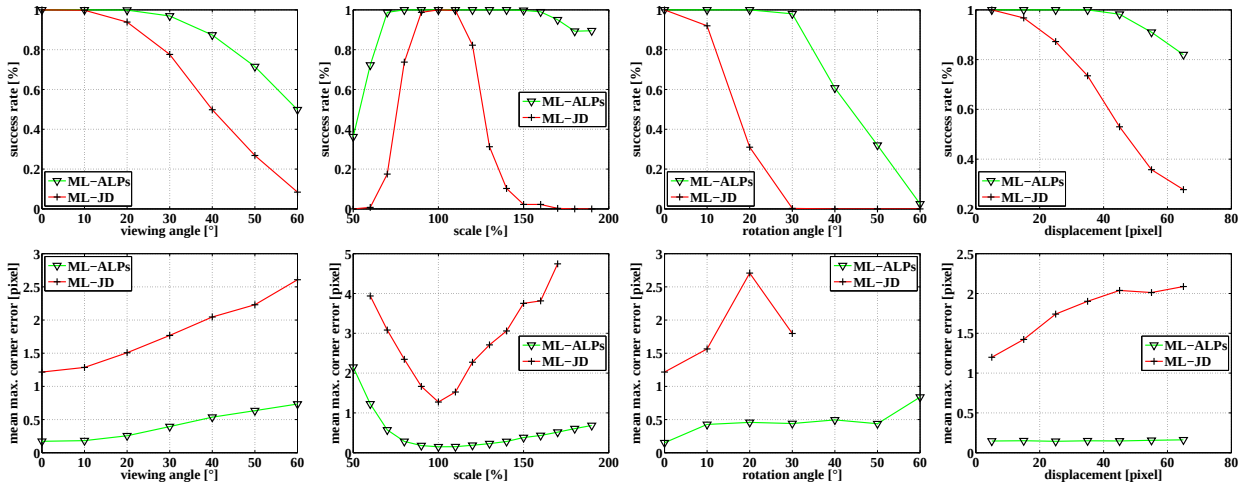


Fig. 14. Results of the comparison between the ML ALPs approach and the ML approach of Jurie and Dhome [12] with respect to different types of motion without occlusion. The first row shows the tracking success rate and the second row the corresponding mean maximum corner errors.

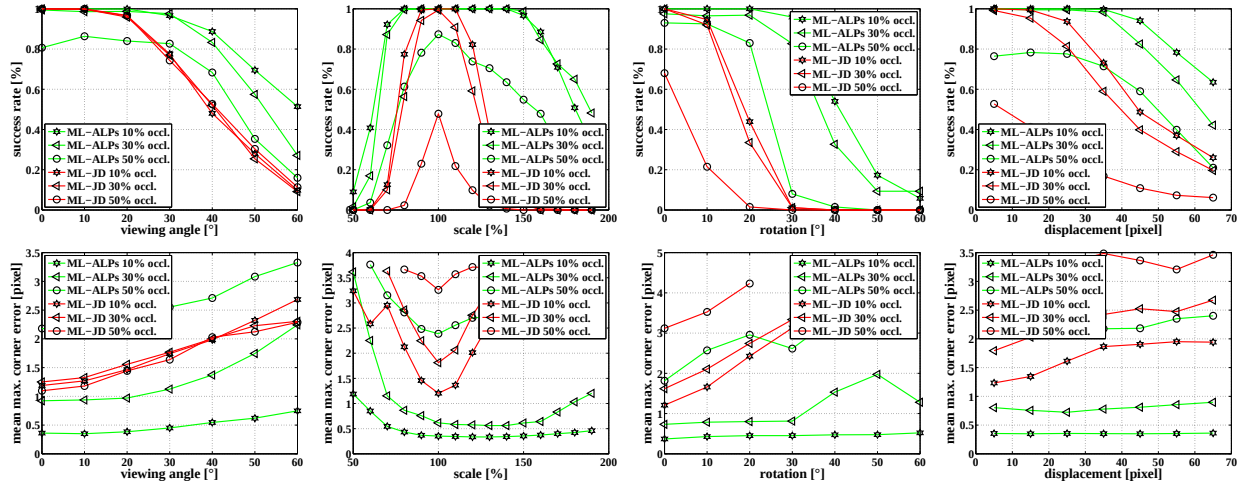


Fig. 15. Results of the comparison between the ML ALPs approach and the approach of Jurie and Dhome [12] with respect to different types of motion with occlusion. The first row shows the tracking success rate and the second row the corresponding mean maximum corner errors.

Method	Frame-rate [fps]	Loss-of-locks [-/-]	Error [%]
NoSLLiP	18.9	13/6935	1.5
ML-ALPs	17.2	1/6945	2.1

TABLE 2
Comparison between the tracking results of NoSLLiP (Matlab implementation) given in [17] and results obtained using ML-ALPs (C++ implementation) using the MOUSEPAD sequence.

6.2.4 *Qualitative Evaluation*

Fig. 16 and 17 demonstrate the performance of the proposed method on different image sequences. Fig. 16 shows several cases of partial occlusion, caused by a hand and a pen moving through the template from one side to the other. Fig. 17 shows a paper with a rectangular hole moving over a background surface. In this case the initially learned template, which includes the backround visible through the hole, is removed automatically when moving.

7 Conclusion

In order to support the dynamic change of the template shape, we introduce an efficient method for adapting linear predictors use for real-time template tracking . Our method allows both, the enlargement and reduction of the template size. We demonstrate that our ALPs approach can also be used to efficiently learn linear predictors for templates of a fixed size. In this case we start from small templates and adapt their linear predictors until the desired template size is reached. This results in much shorter learning time compared to the standard approach of Jurie and Dhome [11]. The efficiency of the presented approach derives from the special computation of the matrix inverse. In the standard approach the inverse has to be recomputed from scratch after each change of the template size. In

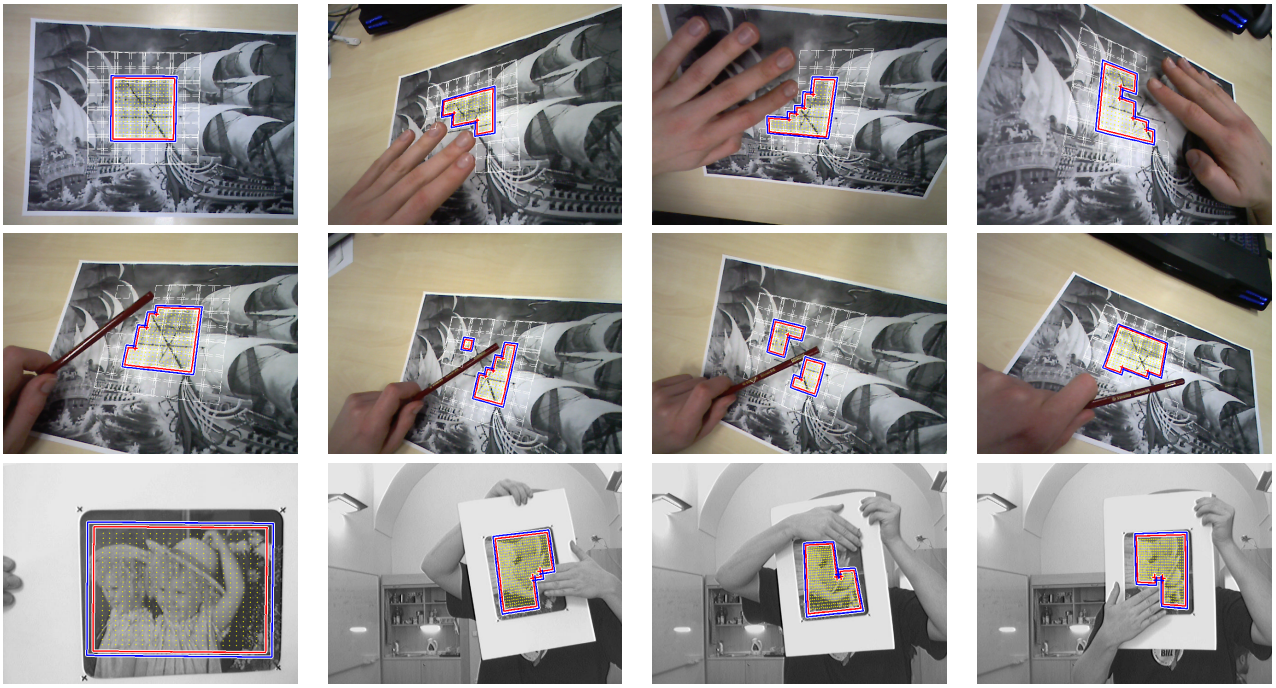


Fig. 16. Tracking with occlusions. **Upper two rows:** The upper left image shows the initially learned template. The other images show the template when occluded by a hand and when occluded by a pen which is moving through the template, both while the camera is moving. In both cases the shape of the template is automatically adapted according to the present occlusions. **Lower row:** The left image shows the initially learned template. The template is tracked over time, where occlusion occures at the end of the sequence.

Fig. 17. Tracking a template with a hole inside. The first image shows the template on a white background to show the hole in the middle. The second image shows the background. In the right image the template is learned with the background image. In the remaining images the template tracking results are shown. Note that the background behind the hole is changing, thus the hole is removed from the template and does not disturb the tracking.

contrast, our approach updates the matrix according to the change in the template shape.

In this context we also introduce a robust method for detecting and handling occlusions. The multi-layer approach enables tracking of a template in the case of the abrupt occurence of occlusions. Early detection of incoming occlusions is necessary to adapt the top layer template before it is occluded, so that occlusion is prevented. The use of multiple transformed versions of the top layer template significantly increases the robustness with respect to large motions.

We demonstrate that our ALPs yield tracking results comparable to those of the standard approaches, while learning is much faster, and that the occlusion-aware tracking yields superior tracking results with respect to robustness and accuracy.

ACKNOWLEDGMENTS

This project was partially funded by the Bayerische Forschungsstiftung.

REFERENCES

[1] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *International Joint Conference on Artificial Intelligence*, Vancouver, BC, Canada, August 1981, pp. 674–679.

[2] H.-Y. Shum and R. Szeliski, "Construction of panoramic image mosaics with global and local alignment," *International Journal of Computer Vision*, vol. 36, no. 2, pp. 101–130, February 2000.

[3] G. Hager and P. Belhumeur, "Efficient region tracking with parametric models of geometry and illumination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 10, pp. 1025–1039, October 1998.

[4] M. Cascia, S. Sclaroff, and V. Athitsos, "Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 4, April 2000.

[5] F. Dellaert and R. Collins, "Fast image-based tracking by selective pixel integration," in *ICCV Workshop of Frame-Rate Vision*, Kerkyra, Greece, September 1999, pp. 1–22.

[6] S. Baker and I. Matthews, "Equivalence and efficiency of image alignment algorithms," in *Conference on Computer Vision and Pattern Recognition*, vol. 1, Los Alamitos, CA, USA, December 2001, pp. 1090–1097.

[7] ——, "Lucas-kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, pp. 221–255, March 2004.

[8] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *IEEE International Conference on Robotics and Automation*, vol. 2, New Orleans, LA, USA, May 2004, pp. 1843–1848.

[9] S. Benhimane and E. Malis, "Real-time image-based tracking of planes using efficient second-order minimization," in *Conference on Intelligent Robots and Systems*, vol. 1, New Orleans, LA, USA, Sept 2004, pp. 943–948.

[10] ——, "Homography-based 2d visual tracking and servoing," *International Journal of Robotics Research*, vol. 26, no. 7, pp. 661–676, July 2007.

[11] F. Jurie and M. Dhome, "Hyperplane approximation for template matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 996–100, July 2002.

[12] ——, "Real time robust template matching," in *British Machine Vision Conference*, Cardiff, UK, September 2002, pp. 123–131.

[13] C. Gräßl, T. Zinßer, and H. Niemann, "Efficient hyperplane tracking by intelligent region selection," in *Image Analysis and Interpretation*, Lake Tahoe, NV, USA, March 2004, pp. 51–55.

[14] P. Parisot, B. Thiesse, and V. Charvillat, "Selection of reliable features subsets for appearance-based tracking," in *Signal-Image Technologies and Internet-Based System*, Shanghai, China, December 2007, pp. 891–898.

[15] J. Matas, K. Zimmermann, T. Svoboda, and A. Hilton, "Learning efficient linear predictors for motion estimation," in *Proceedings of 5th Indian Conference on Computer Vision, Graphics and Image Processing*, Madurai, India, December 2006, pp. 445–456.

[16] W. W. Mayol and D. W. Murray, "Tracking with general regression," *Journal of Machine Vision and Applications*, vol. 19, no. 1, pp. 65–72, January 2008.

[17] K. Zimmermann, J. Matas, and T. Svoboda, "Tracking by an optimal sequence of linear predictors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 99, no. 1, pp. 677–692, April 2009.

[18] S. Baker, R. Gross, I. Matthews, and T. Ishikawa, "Lucas-kanade 20 years on: A unifying framework: Part 2," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-01, February 2003.

[19] I. Patras and E. Hancock, "Regression-Based Template Tracking in Presence of Occlusions," in *Proceedings of the Eight International Workshop on Image Analysis for Multimedia Interactive Services*, Santorini, Greece, June 2007, p. 15.

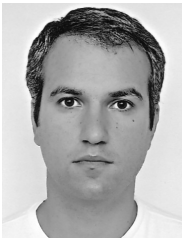
[20] C. Gräßl, T. Zinßer, and H. Niemann, "Illumination insensitive template matching with hyperplanes," in *Proceedings of Pattern recognition: 25th DAGM Symposium*, Magdeburg, Germany, September 2003, pp. 273–280.

[21] H. V. Henderson and S. R. Searle, "On deriving the inverse of a sum of matrices," *SIAM Review*, vol. 23, no. 1, pp. 53–60, January 1981.

[22] S. Hinterstoisser, S. Benhimane, N. Navab, P. Fua, and V. Lepetit, "Online learning of patch perspective rectification for efficient object detection," in *Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, USA, June 2008, pp. 1–8.

[23] S. Holzer, S. Ilic, and N. Navab, "Adaptive linear predictors for real-time tracking," in *Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, USA, June 2010, pp. 1807–1814.

Stefan Holzer is a PhD student at the institute for Computer Aided Medical Procedures (CAMP) at the Technische Universität München (TUM) where he is part of the Computer Vision group. He received his diploma in Computer Science from the University of Applied Sciences Landshut in 2006 and his M.Sc. in Computer Science from the TUM in 2009. He was affiliated with the group of Dr. Andrew Davison at the Department of Computing at Imperial College London, with Willow Garage where he worked on object detection in the context of robot based object grasping, and more recently with Microsoft Research Cambridge where he worked on learning-based approaches. Stefan Holzer's current research interests include reconstruction, object detection, tracking, and pose estimation for 2D/3D objects. You can find out more about his work by visiting http://campar.in.tum.de/Main/StefanHolzer.



Slobodan Ilic is senior research scientist working at TU Munich, Germany. Since February 2009 he is leading the Computer Vision Group of the CAMP Laboratory at TUM. Form June 2006 he was a senior researcher at Deutsche Telekom Laboratories in Berlin. Before that he was a postdoctoral fellow for one year at Computer Vision Laboratory, EPFL, Switzerland, where he received his PhD in 2005. His research interests include: deformable surface modeling and tracking, 3D reconstruction, real-time object detection and tracking, object detection and classification in 3D data. Slobodan Ilic serves as a regular program committee member for all major computer vision conferences, such as CVPR, ICCV and ECCV as well as journals, such as TPAMI and IJCV. Besides active academic involvement Slobodan has strong relations to industry and supervises a number of PhD students supported by industry.

Nassir Navab is a full professor and director of the Computer Aided Medical Procedures & Augmented Reality institute at Technische Universität München. He also has a secondary faculty appointment at TU München's Medical School. Nassir has a PhD in computer science from INRIA and the University of Paris XI. In 2006 he was elected as a member of board of director of the MICCAI society. He also served on steering committee of IEEE Symposium on Mixed and Augmented Reality 2000-2008. He acts as associated editor of IEEE Transactions on Medical Imaging and serves on the editorial board of Medical Image Analysis. Nassir is the author of more the 50 US and international patents. He received the Siemens Inventor of the Year Award in 2001 and the SMIT Society Technology Award in 2010. His PhD students have received many prestigious awards including four MICCAI young scientist awards, ISMAR, ISBI, DAGM, VOEC-ICCV and AMDO best paper awards. His research interests include computer vision, augmented reality, computer-aided surgery and medical image registration.

APPENDIX

E

EFFICIENT LEARNING OF LINEAR PREDICTORS FOR TEMPLATE TRACKING

Own contributions. The core idea of the efficient and robust learning method based on a reformulation of the original learning process was created in a conversation between Marc Pollefeys and me. My further contributions to this work include the idea for the efficient learning method based on dimensionality reduction, the design and implementation of all included methods. The evaluation of those was performed together with David Tan and Slobodan Ilic. The core ideas of the paper were refined in collaboration with all co-authors. All co-authors were involved in the writing of the paper.

Int J Comput Vis
DOI 10.1007/s11263-014-0729-1

Efficient Learning of Linear Predictors for Template Tracking

Stefan Holzer · Slobodan Ilic · David Tan · Marc Pollefeys · Nassir Navab

Received: 3 June 2013 / Accepted: 6 May 2014

Abstract The research on tracking templates or image patches in a sequence of images has been largely dominated by energy-minimization-based methods. However, since its introduction in Jurie and Dhome (IEEE Trans Pattern Anal Mach Intell, 2002), the learning-based approach called linear predictors has proven to be an efficient and reliable alternative for template tracking, demonstrating superior tracking speed and robustness. But, their time intensive learning procedure prevented their use in applications where online learning is essential. Indeed, Holzer et al. (Adaptive linear predictors for real-time tracking, 2010) presented an iterative method to learn linear predictors; but it starts with a small template that makes it unstable at the beginning. Therefore, we propose three methods for highly efficient learning of full-sized linear predictors—where the first one is based on dimensionality reduction using the discrete cosine transform; the second is based on an efficient reformulation of the learning equations; and, the third is a combination of both. They show different characteristics with respect to learning time and tracking robustness, which makes them suitable for different scenarios.

Keywords Template tracking · Linear predictors · Learning · Dimensionality reduction · Discrete cosine transform (DCT)

Communicated by Cordelia Schmid.

S. Holzer (✉) · S. Ilic · D. Tan · N. Navab
Fakultät für Informatik, I-16, Technische Universität München, Boltzmannstr. 3, 85748 Garching b. München, Germany
e-mail: holzers@in.tum.de

S. Ilic
e-mail: Slobodan.Ilic@in.tum.de

D. Tan
e-mail: tanda@in.tum.de

N. Navab
e-mail: navab@in.tum.de

M. Pollefeys
Computer Vision and Geometry Lab (CVG), Department of Computer Science, Institute of Visual Computing, ETH Zurich, CNB G105 Universitatstrasse 6, 8092 Zurich, Switzerland
e-mail: marc.pollefeys@inf.ethz.ch

1 Introduction

Due to its large range of applications, template tracking has been extensively studied in the past. The main task of template tracking is to follow a specified template over time, i.e. over a sequence of consecutive images. This is done by estimating the parameters of the template warping function, defining how the image data occupied by the template is warped from one frame to another. Examples of such warping functions are affine transformations or homographies. Applications can be found in areas such as augmented reality, human-computer interfaces, medical imaging, surveillance, vision-based control, or visual reconstruction.

While energy minimization has become a common technique for estimating template parameters from frame to frame, tracking-by-detection methods became popular recently, as they reached a state where template parameter estimation is possible at or close to frame rate. A further alternative are learning based approaches, where the relation between image intensity differences and template parameters is learned using exemplary data. While energy minimization approaches are flexible at run-time and tracking-by-detection methods do not put constraints on inter-frame motions, learning based techniques have shown to allow much faster online tracking.

Published online: 27 May 2014

One of the most successful learning based approaches for template tracking are linear predictors, or template tracking using hyper-plane approximation, proposed by Jurie and Dhome (2002a). Although fast and robust, it contains a computationally expensive learning phase which prohibits it from being used in scenarios where new scenes need to be handled online. We address this limitation by presenting three learning approaches which lead to learning times that are up to two orders of magnitude faster than Jurie and Dhome (2002a).

In the remainder of this paper, we first discuss related work (Sect. 2) and introduce the original approach as proposed by Jurie and Dhome (2002a) (Sect. 3). We then present our learning approaches starting with a dimensionality reduction approach which makes use of the discrete cosine transform (Sect. 4), introduced in Holzer et al. (2012); followed by a reformulation of the learning process (Sect. 5), introduced in Holzer et al. (2012); and finally a combination of the two (Sect. 6). New training data is added by updating an existing linear predictor (Sect. 7) (Hinterstoisser et al. 2008). We then demonstrate the usefulness of our proposed approaches using mobile applications (Sect. 9). In the experiments (Sect. 8) we analyze the characteristics of the proposed approaches under different scenarios and with respect to the related work.

2 Related Work

Template tracking approaches can be categorized in three different sets of methods: tracking-by-detection (TBD) (Hinterstoisser et al. 2008, 2009, 2010, 2011, 2012; Holzer et al. 2009; Özuysal et al. 2007), template tracking based on energy minimization (Lucas and Kanade 1981; Shum and Szeliski 2000; Hager and Belhumeur 1998; Cascia et al. 2000; Dellaert and Collins 1999; Baker and Matthews 2001, 2004; Malis 2004; Benhimane and Malis 2007; Dame and Marchand 2010; Richa et al. 2011), as well as learning based methods (Grabner et al. 2008; Kalal et al. 2012; Jurie and Dhome 2002a,b; Gräßl et al. 2004; Parisot et al. 2007; Matas 2006; Mayol and Murray 2008; Zimmermann et al. 2009; Holzer et al. 2010, 2013). While tracking-by-detection methods do not put constraints on the possible location of the template to track, i.e. the template is searched in the whole image independent of its location in the previous frame, they are, in general, significantly slower than frame-to-frame tracking approaches. The time consuming training procedure and the restrictions in the possible pose space further limit these approaches. Energy minimization based approaches usually have the advantage in creating and modifying templates online, while learning based methods are significantly better in tracking speed. Additionally, Jurie and Dhome (2002a) showed that linear predictors are superior to Jacobian approximation in terms of tracking speed and robustness, and we showed in Holzer et al. (2010) that linear predictors outperform methods based on energy minimization such as Efficient Second-order Minimization (Benhimane and Malis 2007) (ESM).

Tracking-by-Detection-based approaches. We can further categorize TBD approaches into methods that are based on keypoint detection and on a sliding window which is also known as template matching. Özuysal et al. (2007) introduced FERNs where keypoints are extracted and then classified by estimating the probability of each keypoint falling under a specific class. Unfortunately, the corresponding learning process is time consuming and is error-prone if the number of available keypoints is limited, which is the case if small regions are considered. Holzer et al. (2009) presented an approach called Distance Transform Templates (DTTs), where the distance transform is used to extract closed contours, and to describe and match the extracted contours using FERNs (Özuysal et al. 2007). This approach needs a significant amount of time in learning, and detects objects only at about 10 frames per second. Hinterstoisser et al. (2008, 2009) proposed two patch-based TBD methods, where keypoints are used as starting point to match patches and estimate their pose. Although the use of keypoints enables processing at almost video frame-rate, it constrains its repeatability by the underlying keypoint detector. In their recent work, Hinterstoisser et al. (2010, 2011, 2012) overcome the limitations of keypoint detectors by using sliding window approaches. Indeed, this allows a robust object detection; however, it imposes restrictions on the possible pose space since the necessary processing increases with the amount of covered poses. Even with reasonable constraints on the pose space, it is still significantly slower than frame-to-frame tracking.

Keyframe-based approaches. This type of approach relies on creating a set of keyframes that represent the object or scene and are used in an optimization in order to obtain stable pose estimation (Vacchetti et al. 2004) and, if the model of the object or scene is not available, create a map of the tracked environment (Klein and Murray 2007). The keyframes are hereby either created in an offline process from a known model (Vacchetti et al. 2004) or selected online in case the model needs to be created while tracking (Klein and Murray 2007). These keyframe-based approaches usually rely on keypoints or other discriminative image features, e.g. edges Klein and Murray (2008), which are tracked from frame to frame and matched against keyframes. Similar to some of the Tracking-by-Detection-based methods, keyframe-based approaches tend to become error-prone if the number of available keypoints or features is limited. This especially holds if the object or region of interest is small. However, if sufficient image features are available, these approaches provide very stable and robust tracking.

Energy minimization-based approaches. These approaches build upon the early work of Lucas and Kanade (1981). Improvements since then include: different update rules of the warp function (Lucas and Kanade 1981; Hager and Belhumeur 1998; Cascia et al. 2000; Shum and Szeliski 2000; Dellaert and Collins 1999; Baker and Matthews 2001), handling of occlusions and illumination changes (Hager and Belhumeur 1998), as well as considering different orders of approximation of the error function (Malis 2004; Benhimane and Malis 2007). When looking at the different update rules that have been proposed over time, we can classify them into four categories, which are the additive approach (Lucas and Kanade 1981), the compositional approach (Shum and Szeliski 2000), the inverse additive approach (Hager and Belhumeur 1998; Cascia et al. 2000) and the inverse compositional approach (Dellaert and Collins 1999; Baker and Matthews 2001). The inverse approaches hereby switch the roles of the reference and the current image, which allows to move some of the computations into the initialization phase, making the tracking phase more efficient. Hager and Belhumeur (1998) addressed the problems in illumination changes and occlussions. The usage of second-order instead of first-order approximations led to a faster convergence speed and larger convergence areas (Benhimane and Malis 2007; Malis 2004). For a more detailed and complete overview of energy-based tracking methods, we refer the readers to Baker and Matthews (2004).

Learning-based approaches. Learning-based approaches can be divided into approaches that apply offline or online learning. Online learning is often used to track objects that are not known a-priori or change appearance over time. Prominent examples of an online learning approach is the tracking approach of Grabner et al. (2008), as well as the Tracking-Learning-Detection approach of Kalal et al. (2012). Both approaches are semi-supervised approaches where the tracking process is split into a frame-to-frame tracker, a detector that localizes all observed appearances of the tracked object and corrects the tracker if necessary, as well as a learning procedure which estimates the errors of the detector and updates it in order to avoid them in future. While Grabner et al. (2008) uses a semi-supervised online boosting strategie for learning, Kalal et al. (2012) uses so called PN learning where two sets of experts are used to estimate missed detections and false alarms. These combinations of tracking, detection and learning allow to handle changes in appearance as well as occlusions. However, while they give good results when tracking a bounding box around an object they are not suited to track the full pose of objects.

A prominent approach in tracking based on offline learning is tracking using linear predictors. Linear Predictors for template tracking or template tracking using hyperplane approximation was first introduced by Jurie and Dhome (2002a). Their proposed method uses randomly warped samples of the initial template to learn linear predictors and applies them to predict the parameter updates in template tracking. In contrast to previous methods, the "Jacobians" are here computed once during the learning phase and the parameter updates are computed using simple matrix multiplications. A more detailed description of this method is in Sect. 3.2. An extension of this method in order to handle occlusions has been presented in Jurie and Dhome (2002b). Gräßl et al. (2003) showed how invariance with respect to illumination changes can be achieved and how tracking accuracy can be increased by intelligently selecting the points for sampling from the image data (Gräßl et al. 2004). Zimmermann et al. (2009) moved away from using one big single template into numerous small templates, track them individually, and combine their seperate motion estimates into a single one. Holzer et al. (2010, 2013) presented a method for adapting existing linear predictors in order to modify online the covered area of template while tracking. This reduces the initial learning time by start tracking with a small template and grow it over time. Instead of creating a predictor, Mayol and Murray (2008) collect a set of training samples in order to fit the current sampling region to this pre-trained set using general regression.

Among the learning based approaches using linear predictors, none of them can learn large templates at run-time. Therefore, we present learning approaches with this criterion.

3 Tracking Framework

Our proposed template tracking approaches are built from the work of Jurie and Dhome (2002a). While we introduce new learning procedures that significantly reduce the necessary learning time, we keep the tracking stage the same. In this section, we introduce our notations and review the method proposed by Jurie and Dhome (2002a).

3.1 Template and Parameter Description

Without loss of generality, we use a rectangular template that covers an area of $n_s = w \cdot h$ square pixels and locate sample points that uniformly subsample the template into a grid of n_p points as shown in Fig. 1. The sample points are used to efficiently describe the image intensities in the template instead of using its full resolution. From the template in Fig. 1, we define the parameter vector $\boldsymbol{\mu} = (p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7)^\top$ that contains the image coordinates of the four corners which are used to parameterize a homography. Note that neither our proposed approaches nor the approach of Jurie and Dhome (2002a) are limited to this type of sample point arrangement, rectangular shape or transformation.

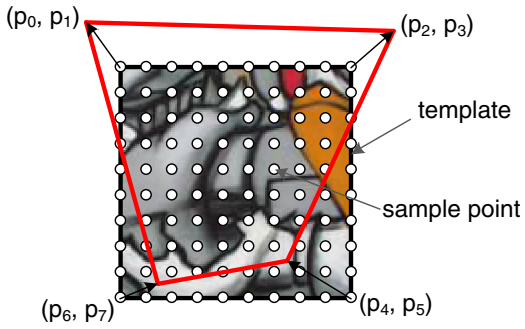


Fig. 1 A template is represented by a set of regularly placed sample points. Its pose is parameterized using its four corner points

3.2 Template Tracking based on Linear Predictors

Given a template in the reference image, the goal of template tracking is to follow the selected template from one frame to the next and to estimate its pose in each frame.

In the reference image, we define the initial location of the template using the parameter vector $\boldsymbol{\mu}_R$ that stores the eight parameters from the four corners of the template, and the image intensity vector $\mathbf{i}_R = (i_{R,1}, i_{R,2}, \ldots, i_{R,n_p})^\top$ that has the image intensity at the corresponding sample points; while we assign $\boldsymbol{\mu}_C$ as the current parameter vector and $\mathbf{i}_C$ as the image intensity vector in the current frame. However, unlike $\mathbf{i}_R$, the elements of $\mathbf{i}_C$ are the image intensities extracted from the current frame at the sample point locations of the template pose from the previous frame or previous iteration $\boldsymbol{\mu}_{C-1}$.

To track the location of the template and estimate the pose in the current frame, Jurie and Dhome (2002a) introduce the linear predictor matrix $\mathbf{A}$ that relates the parameter vectors and the intensity vectors as:

$$\delta\boldsymbol{\mu} = \mathbf{A}\delta\mathbf{i} \tag{1}$$

where $\delta\mathbf{i} = \mathbf{i}_C - \mathbf{i}_R$ is the image intensity difference and $\delta\boldsymbol{\mu}$ is the parameter update. This tracking algorithm is summarized in Alg. 1.

The $8 \times n_p$ matrix $\mathbf{A}$ is precomputed by imposing n_t, where $n_t \gg n_p$, random transformations $\delta\boldsymbol{\mu}_i$ for $i = 1, \ldots, n_t$ on $\boldsymbol{\mu}_R$ and computing the corresponding image difference vectors $\delta\mathbf{i}_i$. These vectors are concatenated into an $8 \times n_t$ matrix $\mathbf{Y} = \left(\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \ldots, \delta\boldsymbol{\mu}_{n_t}\right)$ and $n_p \times n_t$ matrix $\mathbf{H} = \left(\delta\mathbf{i}_1, \delta\mathbf{i}_2, \ldots, \delta\mathbf{i}_{n_t}\right)$, which are used to reformulate Eq. (1) as:

$$\mathbf{Y} = \mathbf{A}\mathbf{H}. \tag{2}$$

Using a closed-form solution, we solve for $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top \left(\mathbf{H}\mathbf{H}^\top\right)^{-1}. \tag{3}$$

Algorithm 1 Tracking using Linear Predictors

function *Track* (***in*** Image $\mathbf{I}$,
  in TemplateParameters $\boldsymbol{\mu}_{C-1}$,
  out TemplateParameters $\boldsymbol{\mu}_C$)
Compute homography $\mathbf{T}_{\boldsymbol{\mu}}$ from $\boldsymbol{\mu}_{C-1}$.
for $level = 1 \rightarrow n_l$ **do**
  for $iteration = 1 \rightarrow n_i$ **do**
    Extract image data from $\mathbf{I}$ at sample points warped with $\mathbf{T}_{\boldsymbol{\mu}}$.
    Normalize image data.
    Compute image difference vector $\delta\mathbf{i}$.
    Compute parameter update $\delta\boldsymbol{\mu} = \mathbf{A}_{level}\delta\mathbf{i}$.
    Compute homography $\mathbf{T}_{\delta\boldsymbol{\mu}}$ from $\delta\boldsymbol{\mu}$.
    $\mathbf{T}_{\boldsymbol{\mu}} \leftarrow \mathbf{T}_{\boldsymbol{\mu}}\mathbf{T}_{\delta\boldsymbol{\mu}}$.
  end for
end for
Compute $\boldsymbol{\mu}_C$ from $\mathbf{T}_{\boldsymbol{\mu}}$.

This formulation requires to invert an $n_p \times n_p$ matrix $\mathbf{H}\mathbf{H}^\top$ where a typical value of n_p is $20 \cdot 20 = 400$. Due to the large size of this matrix, learning linear predictors using Eq. (3) is computationally expensive, and limits their use in real-time applications where the environment is unknown and templates have to be learned online.Throughout the paper, we focus on reformulating the learning procedure to find $\mathbf{A}$ in Eq. (3) and to evaluate the robustness in tracking for each learning procedure.

In practice, the image data vector $\mathbf{i}$ is normalized such that the elements have zero mean and unit standard deviation, which increases the robustness against illumination changes. Furthermore, to prevent $\mathbf{H}\mathbf{H}^\top$ from being rank-deficient due to the zero mean of the vectors, we add random noise to the normalized image intensity difference vectors in $\mathbf{H}$.

3.3 Multi-layered Tracking

To improve the tracking performance, we apply a multi-layer approach where we compute n_l linear predictors $\mathbf{A}_1, \ldots, \mathbf{A}_{n_l}$ and use one linear predictor per layer. Each of these linear predictors is trained for different amounts of motion where the first one is trained for large motions and the latter ones for consecutively smaller motions. Additionally, each of the linear predictors is applied multiple times in tracking. Within this paper, we use $n_l = 5$ and three iterations for each predictor. The complete algorithm for learning linear predictors is given in Alg. 2.

4 Efficient Predictor Learning using Dimensionality Reduction

Our first approach reduces the size of $\delta\mathbf{i}_i$ from n_p to n_r by applying the discrete cosine transform (DCT). As a consequence, it reduces the number of rows in $\mathbf{H}$ from n_p to n_r and the size of $\mathbf{H}\mathbf{H}^\top$ from $n_p \times n_p$ to $n_r \times n_r$.

Algorithm 2 Learning Linear Predictors

function *Learn* (***in*** Image $\mathbf{I}$,
in TemplateParameters $\boldsymbol{\mu}$,
out Set of linear predictors $\mathcal{A} = \{\mathbf{A}_i\}$)
for $level = 1 \rightarrow n_l$ **do**
Create set of random transformations and put them into the matrix $\mathbf{Y} = (\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \ldots, \delta\boldsymbol{\mu}_{n_t})$.
Apply random transformations on $\boldsymbol{\mu}$ and extract image differences $\mathbf{H} = (\delta\mathbf{i}_1, \delta\mathbf{i}_2, \ldots, \delta\mathbf{i}_{n_t})$ from $\mathbf{I}$.
Normalize image data and add random noise.
Compute linear predictor $\mathbf{A}_{level}$
Add $\mathbf{A}_{level}$ to $\mathcal{A}$.
end for

In general, DCT is known to give good results in image compression. This process involves transforming the image into the frequency space and removing the DCT coefficients that correspond to high frequencies. In relation to our approach, it is advantageous to filter the high frequencies because they tend to include noise and de-stabilize tracking.

Mathematically, the 2-dimensional DCT $\mathbf{U}$ of a $k \times k$ matrix $\mathbf{V}$ is defined as:

$$\mathbf{U} = \text{DCT}(\mathbf{V}) = \mathbf{C}\mathbf{V}\mathbf{C}^\top \tag{4}$$

where the elements of the matrix $\mathbf{C}$ are defined as:

$$\mathbf{C}_{i,j} = \sqrt{\frac{\alpha_i}{k}} \cos\left[\frac{\pi(2j+1)i}{2k}\right] \tag{5}$$

with

$$\alpha_i = \begin{cases} 1 \text{ if } i = 0, \\ 2 \text{ otherwise.} \end{cases} \tag{6}$$

Contrary to Eq. (4) where the 2D DCT is applied on a 2D matrix, our approach needs to apply the DCT on a reshaped 1D vector $\delta\mathbf{i}_i$ as $\delta\hat{\mathbf{i}}_i = \text{DCT}(\delta\mathbf{i}_i)$ to form the matrix $\hat{\mathbf{H}} = (\delta\hat{\mathbf{i}}_1, \delta\hat{\mathbf{i}}_2, \ldots, \delta\hat{\mathbf{i}}_{n_t})$.

Therefore, we define an $n_p \times n_p$ matrix $\mathbf{W}_{DCT}$ that directly maps the image difference vectors $\delta\mathbf{i}_i$ to their DCT counterparts $\delta\hat{\mathbf{i}}_i$ (see Fig. 2). By using a function that converts the elements of a 2D matrix $\mathbf{V}_i$ into a 1D column vector $\delta\mathbf{i}_i$ as $\delta\mathbf{i}_i = \text{reshape}(\mathbf{V}_i)$, where $\mathbf{V}_i$ is the 2D template image data in frame i, we can compute $\mathbf{W}_{DCT}$ as:

$$\mathbf{W}_{DCT} = \left(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n_p}\right) \tag{7}$$

where $\mathbf{b}_m = \text{reshape}(\mathbf{C}\mathbf{B}_m\mathbf{C}^\top)$ and $\mathbf{B}_m$ is a matrix with all elements set to 0 except for the m-th element which is set to 1. This makes the set of matrices $\{\mathbf{B}_1, \ldots, \mathbf{B}_{n_p}\}$ as the base of the template in the image space and the set of vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_{n_p}\}$ as their DCT projection. As a result, the 2D DCT of the image difference vectors is computed as:

$$\delta\hat{\mathbf{i}}_i = \mathbf{W}_{DCT}\delta\mathbf{i}_i \;\Rightarrow\; \hat{\mathbf{H}} = \mathbf{W}_{DCT}\mathbf{H}. \tag{8}$$

In order to integrate this into the original learning formula in Eq. (3), we reformulate this by using the relation:

$$\mathbf{H} = (\mathbf{W}_{DCT})^{-1}\hat{\mathbf{H}}. \tag{9}$$

Thus, we subsitute $\mathbf{H}$ from Eq. (9) to Eq. (2) and solve for the linear predictor matrix $\mathbf{A}$ as follows:

$$\begin{aligned} \mathbf{A}\mathbf{W}_{DCT}^{-1}\hat{\mathbf{H}} &= \mathbf{Y} \\ \mathbf{A}\mathbf{W}_{DCT}^{-1} &= \mathbf{Y}\hat{\mathbf{H}}^\top\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1} \\ \mathbf{A}\mathbf{W}_{DCT}^{-1}\mathbf{W}_{DCT} &= \mathbf{Y}\hat{\mathbf{H}}^\top\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1}\mathbf{W}_{DCT} \\ \mathbf{A} &= \mathbf{Y}\hat{\mathbf{H}}^\top\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1}\mathbf{W}_{DCT}. \end{aligned} \tag{10}$$

Note that by integrating the DCT computation directly into the linear predictor matrix $\mathbf{A}$ we do not have to modify the tracking procedure.

We induce the dimensionality reduction by defining an $n_r \times n_p$ submatrix $\mathbf{W}_{DCT}^{(n_r)}$ with $n_r < n_p$. In this case, the necessary matrix inversion is no longer applied to an $n_p \times n_p$ matrix but rather to an $n_r \times n_r$ matrix. Hence, the final linear predictor is computed as:

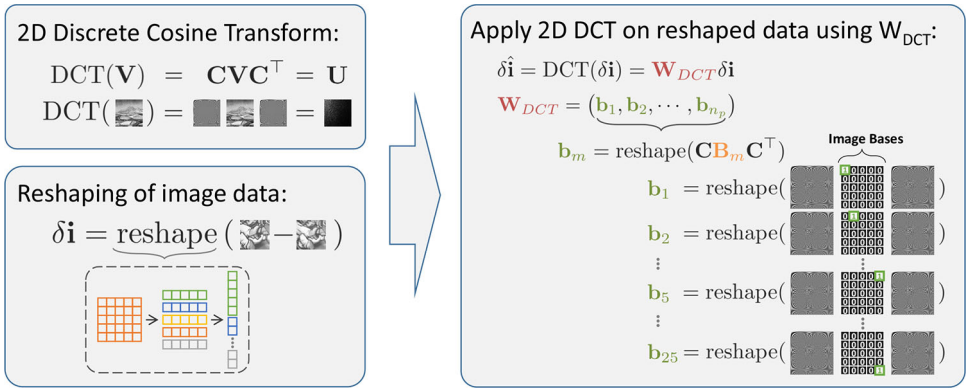


Fig. 2 Demostration of how the 2D-DCT is applied on reshaped image data

$$\mathbf{A}^{(n_r)} = \mathbf{Y}\hat{\mathbf{H}}^{(n_r)\top}\left(\hat{\mathbf{H}}^{(n_r)}\hat{\mathbf{H}}^{(n_r)\top}\right)^{-1}\mathbf{W}_{DCT}^{(n_r)}. \tag{11}$$

where $\hat{\mathbf{H}}^{(n_r)} = \mathbf{W}_{DCT}^{(n_r)}\mathbf{H}$.

The experiments in Sect. 8.1 show that keeping n_r significantly small reduces the learning time for large templates. Moreover, depending on the size of n_r, the reduction in learning even increases tracking robustness.

5 Reformulating the Learning Equations

Another way to increase the learning speed is to reformulate the learning equations such that it is no longer necessary to compute the pseudo-inverse of $\mathbf{H}$. Starting from Eq. (2), we first apply the pseudo-inverse of $\mathbf{Y}$ from the right which leads to:

$$\mathbf{I}_{8\text{x}8} = \mathbf{A}\mathbf{H}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1} = \mathbf{A}\mathbf{B}, \tag{12}$$

where

$$\mathbf{B} = \mathbf{H}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1} \tag{13}$$

is an $n_p \times 8$ matrix. Hence, the linear predictor $\mathbf{A}$ is computed as:

$$\mathbf{A} = (\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top. \tag{14}$$

Note that the pseudo-inverse is applied differently in Eqs. (12) and (14) because the rows are linearly independent in $\mathbf{Y}$ while the columns are linearly independent in $\mathbf{B}$ (Ben-Israel and Greville 2003).

Now, the learning process involves the inversion of the matrices $\mathbf{Y}\mathbf{Y}^\top$ and $\mathbf{B}^\top\mathbf{B}$ in Eqs. (12) and (14), respectively. However, both are 8×8 matrices and can be quickly calculated. Additionally, $\mathbf{Y}\mathbf{Y}^\top$ can be precomputed which requires us to invert a single 8×8 matrix online.

To avoid encoding of fixed offsets in the linear mapping, $\mathbf{Y}$ is normalized such that each parameter has zero mean and unit standard deviation. Accordingly, $\delta\boldsymbol{\mu}$ is de-normalized after using Eq. (1) in tracking. Unlike the normalization in Sect. 3.2 where we aim to obtain invariance on changes in lighting conditions, this normalization does not generate a rank-deficient matrix $\mathbf{Y}\mathbf{Y}^\top$ because the normalization is applied on the rows of $\mathbf{Y}$.

If we compare the formulation of $\mathbf{A}$, we can see that in Eq. (12) from our approach, $\mathbf{H}$ is approximated by orthogonally projecting it on $\mathbf{Y}$; while in Eq. (3) from the original approach of Jurie and Dhome (2002a), $\mathbf{Y}$ is approximated by orthogonally projecting it on $\mathbf{H}$. Given that we project $\mathbf{H}$ on $\mathbf{Y}$, all noise outside the low-rank space represented by $\mathbf{Y}$ has no effect; while in the case of Jurie and Dhome (2002a), the noise has more effect. Therefore, this learning process significantly improves tracking robustness with respect to noise as validated in Sect. 8.3.

6 Combining Dimensionality Reduction and Reformulation of Learning

To combine the dimensionality reduction in Sect. 4 with the reformulation in Sect. 5, we replace $\mathbf{H}$ in Eq. (13) by the dimensionality reduced version $\hat{\mathbf{B}} = \mathbf{W}_{DCT}^{-1}\hat{\mathbf{H}}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1}$; thus, similar to Eq. (14), we obtain:

$$\mathbf{A} = \left(\hat{\mathbf{B}}^\top\hat{\mathbf{B}}\right)^{-1}\hat{\mathbf{B}}^\top. \tag{15}$$

By assigning the $n_t \times 8$ matrix $\mathbf{Z}$ as:

$$\mathbf{Z} = \mathbf{Y}^\top\left(\mathbf{Y}\mathbf{Y}^\top\right)^{-1}, \tag{16}$$

we can write $\hat{\mathbf{B}}^\top\hat{\mathbf{B}}$ in the form of:

$$\hat{\mathbf{B}}^\top\hat{\mathbf{B}} = \mathbf{Z}^\top\hat{\mathbf{H}}^\top\mathbf{W}_{DCT}^{-\top}\mathbf{W}_{DCT}^{-1}\hat{\mathbf{H}}\mathbf{Z} \tag{17}$$

and since $\mathbf{W}_{DCT}$ is orthogonal (*i.e.* its inverse is equal to its transpose), this can be simplified to:

$$\hat{\mathbf{B}}^\top\hat{\mathbf{B}} = \mathbf{Z}^\top\hat{\mathbf{H}}^\top\hat{\mathbf{H}}\mathbf{Z}. \tag{18}$$

Therefore, the final learning equation then becomes:

$$\mathbf{A} = \left(\mathbf{Z}^\top\hat{\mathbf{H}}^\top\hat{\mathbf{H}}\mathbf{Z}\right)^{-1}\mathbf{Z}^\top\hat{\mathbf{H}}^\top. \tag{19}$$

Again, this formulation only requires us to invert an 8×8 matrix $\mathbf{Z}^\top\hat{\mathbf{H}}^\top\hat{\mathbf{H}}\mathbf{Z}$ which can be quickly calculated.

As we will show in the experiments, this leads to an compromise between the strengths and weaknesses of both approaches.

7 Online Updating

After learning a linear predictor, new training samples can be added using the Sherman-Morrison formula as demonstrated in Hinterstoisser et al. (2008). However, this relies on the original approach of computing linear predictors by efficiently updating $\mathbf{S} = (\mathbf{H}\mathbf{H}^\top)^{-1}$. Since in our reformulated learning approach the matrix $\mathbf{S}$ is not computed as an intermediate result, we find a way to derive it from an existing linear predictor $\mathbf{A}$. Thus, using Eq. (3),

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top)^{-1} = \mathbf{Y}\mathbf{H}^\top\mathbf{S} = \mathbf{D}\mathbf{S}, \tag{20}$$

Fig. 3 A set of images selected from the internet, which is used for synthetic experiments

where $\mathbf{D} = \mathbf{Y}\mathbf{H}^\top$ is a $8 \times n_p$ matrix. This implies that $\mathbf{S}$ can be directly computed using the pseudo-inverse of $\mathbf{D}$ as:

$$\mathbf{S} = \mathbf{D}^\top(\mathbf{D}\mathbf{D}^\top)^{-1}\mathbf{A}. \tag{21}$$

Here, $\mathbf{D}\mathbf{D}^\top$ is again an 8×8 matrix and therefore, can be efficiently inverted. Therefore, we update $\mathbf{S}$ as:

$$\hat{\mathbf{S}} = \left(\mathbf{S}^{-1} + \delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^\top\right)^{-1} \tag{22}$$

$$= \mathbf{S} - \frac{\mathbf{S}\delta\mathbf{i}_{n_t+1}\delta\mathbf{i}_{n_t+1}^\top\mathbf{S}_I}{1 + \delta\mathbf{i}_{n_t+1}^\top\mathbf{S}\delta\mathbf{i}_{n_t+1}}, \tag{23}$$

where $\delta\mathbf{i}_{n_t+1}$ is a vector of image intensity differences obtained from a new random transformation applied to the sample points.

Note that we also have to update the matrices $\mathbf{H}$ and $\mathbf{Y}$ before computing the updated linear predictor in Eq. (20). This is done by concatenating them with the new training samples where the parameter differences are normalized in the same way as in the initial learning.

8 Experiments

In this section, we evaluate our approaches for efficient learning of linear predictors as proposed in Sect. 4 (DCT), Sect. 5 (HP) and Sect. 6 (DCTHP), and compare them to the original approach of Jurie and Dhome (2002a) (JD) as well as to Efficient Second-order Minimization (ESM) introduced by Benhimane and Malis (2007). Our comparison is based on two kinds of evaluation—timing and tracking performance. The former is used to evaluate the differences in tracking and learning time while the latter reveals the impact gained from the learning performance on the tracking robustness with respect to different kinds of motions as well as its sensitivity to noise.

We used C++ implementations for all algorithms. The Efficient Second-order Minimization (Benhimane and Malis 2007) is from the publicly available binaries[1] and the original approach of Jurie and Dhome (2002a) is from our own implementation where it is similarly optimized as the proposed approaches. For the evaluations, we used a standard notebook with a 2.26 GHz Intel(R) Core(TM)2 Quad CPU and 8 GB of RAM with only a single core used for computations.

All synthetic experiments are done using a template size of 150×150 pixels, the template is taken from the center of an image and tracking is applied on its warped versions. Figure 3 shows the images used for the synthetic experiments which were selected randomly from the internet. The reason for focusing on synthetic experiments is that these allow an accurate comparison using ground truth. This also has the benefit that it is simple to estimate the influence of single parameters, compared to other methods of testing, like using markers on real scenes, which generate their own error and limit the amount of available motion. It further makes it possible to specify the exact amount of change. This allows a fair evaluation between the different approaches.

The homographies for the various tests are compute as follows:

- **Translations.** For a specified reference amount of a translation by t pixels we compute a set of random translations values in the range of $t \pm 5$ pixels. The translation is applied in a random direction.
- **Scale.** For a specified scale s we compute a set of random scale values in the range $[t, t \cdot 1.2]$. The scaling is applied relative to the center of the template.
- **Rotation.** For a specified angle α in degree we compute a set of random rotation values in the range of $\alpha \pm 5$ degree. The center of the rotation is equal to the center of the template.
- **In-plane rotation.** For the viewing angle. For the viewing angle tests we rotate the plane of the template around

[1] See version 0.4 available at http://esm.gforge.inria.fr/ESM.html.

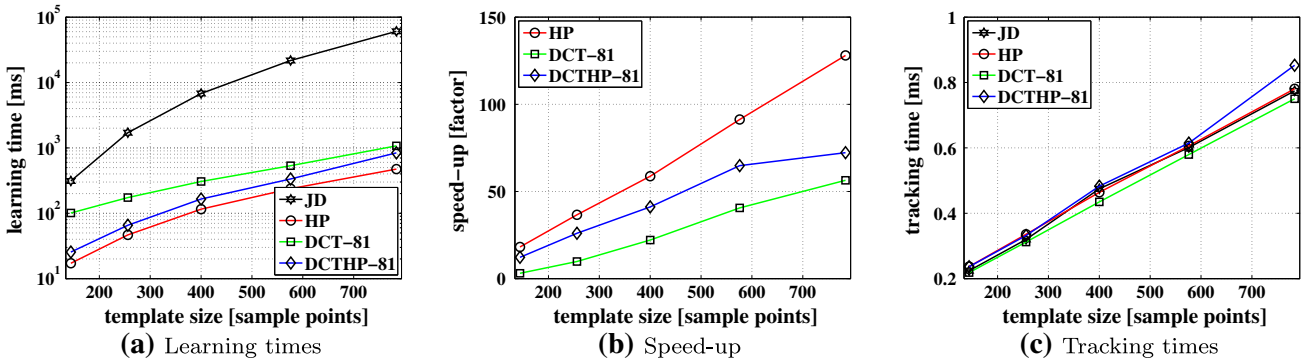


Fig. 4 Evaluation of learning and tracking times. **a** Learning times. **b** Speed-up obtained by the proposed methods, where for the dimensionality reduction based approaches 81 DCT-coefficients are used. **c** Tracking times

a random axis which is lying in the template plane. For a specified angle β in degree we compute a set of random rotation values in the range of $\beta \pm 5$ degree. The axis of the rotation goes through the center of the template.

In Sect. 8.6 we analyse our approaches on real data and compare them to several state-of-the-art approaches (Zimmermann et al. 2009; Holzer et al. 2010, 2013; Jurie and Dhome 2002a,b; Lucas and Kanade 1981; Lowe 2004; Kalal et al. 2012).

8.1 Computational Complexity

In order to analyze the computational complexity, we measure the time a specific phase of the approaches, *i.e*. learning and tracking, needs to complete.

Learning The necessary learning time reflects our main contribution. In Fig. 4a, we compare the learning times with respect to the template size in number of sample points for the approach of Jurie and Dhome (2002a) (JD), our dimensionality reduction approach with 81 DCT-coefficients (DCT-81), our approach with reformulated learning (HP), and our combined approach with 81 DCT-coefficients (DCTHP-81). All our proposed methods are significantly faster in learning than the original approach. Figure 4b compares the speed-up of the different proposed approaches with respect to Jurie and Dhome (2002a). This also shows that for larger templates the difference between the learning times gets even bigger. In detail, the reformulation approach (HP) is the fastest if considering 81 DCT-coefficients for the other approaches, while the combined approach (DCTHP) is the second fastest, followed by the pure dimensionality reduction. Note that by using less DCT-coefficients, the DCT-based approaches can achieve a similar speed-up as the reformulation in Fig. 5. Considering template sizes with more than 800 sample points, *e.g*. for a 30×30 template, our approaches reach learning times which are more than two orders of magnitude faster than the approach of Jurie and Dhome (2002a).

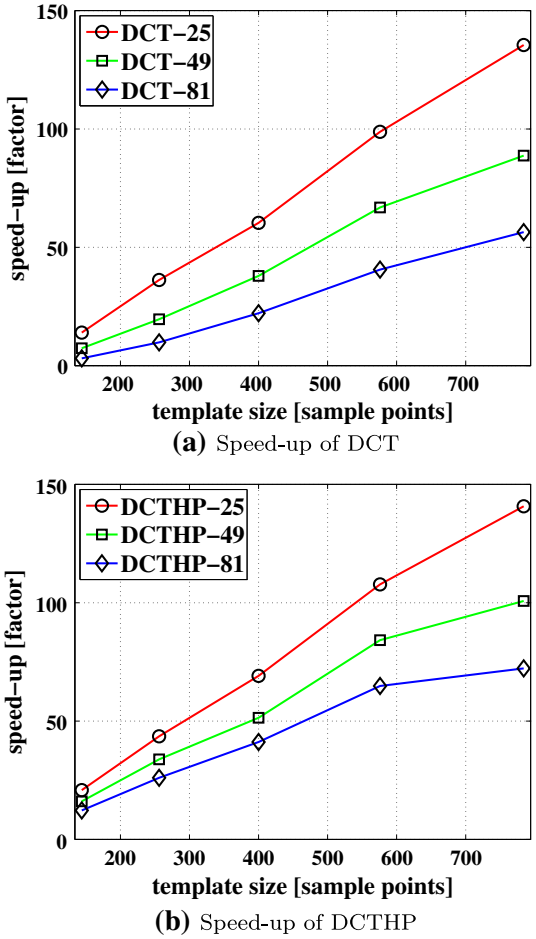


Fig. 5 Analysis of speed-ups for both DCT-based approaches with respect to the original approach of Jurie and Dhome (2002a). **a** Speed-up of the approach only based on dimensiontality reduction (DCT). **b** Speed-up of the combined approach (DCTHP)

Tracking When we compare the tracking times with respect to the template size in Fig. 4c, all proposed approaches need

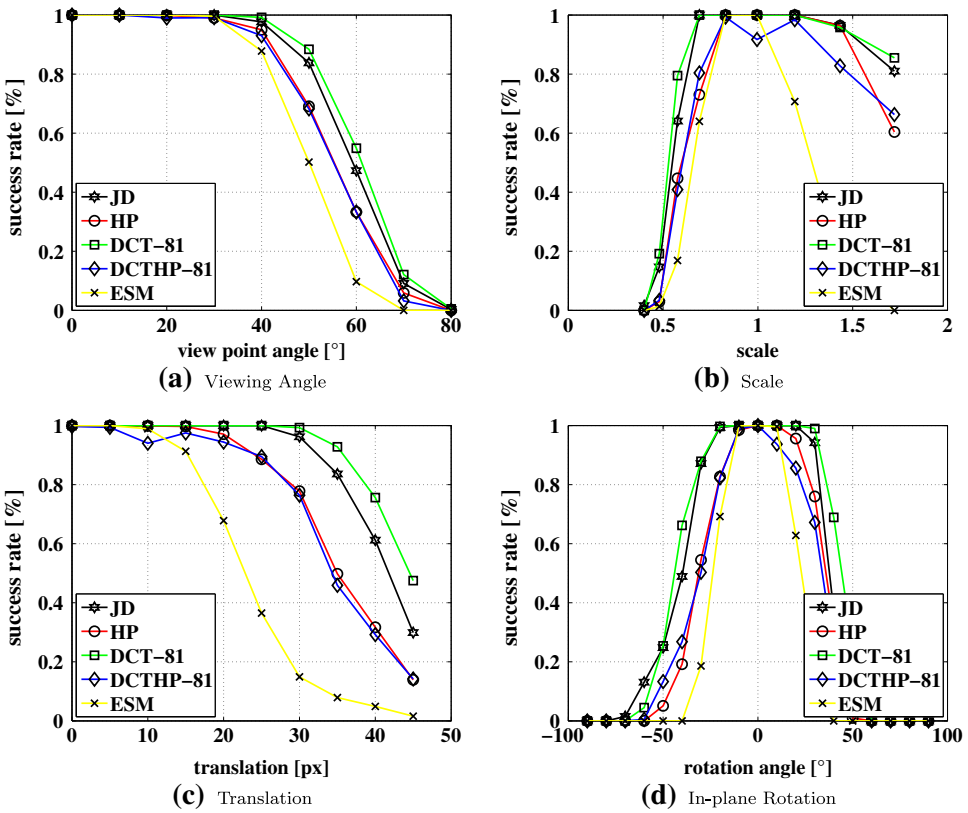


Fig. 6 Evaluation of tracking success rate with respect to different types of motion. **a** Viewing angle, **b** scale, **c** translation, **d** in-plane Rotation

approximately the same amount of time for tracking as the approach of Jurie and Dhome (2002a) and can even track large templates at more than 1, 000 fps. In contrast to this, the energy minimization based approach of Benhimane and Malis (2007) needs approximately 10 ms to track the same template.

8.2 Robustness

We analyze the success rates in tracking, where all approaches are considered in Fig. 6 and the effect of the number of DCT-coefficients n_r used in dimensionality reduction influences the tracking robustness of both DCT-based approaches is evaluated in Fig. 7.

The success rate in tracking is measured by finding the correct location of the template after the introduction of random transformations to several test images. The considered random transformations contain translation, rotation, scale, and viewpoint change. Tracking is considered successfull by computing the mean pixel distance between the reference template corner points and the tracked template corner points, which are back-projected into the reference view. If this mean difference is less than 5 pixels then the tracking is considered to be successful. The template size for the following experiments is 150×150 pixels with 20×20 sample points, if not otherwise mentioned. The initial learning of the linear predictors is done using $3 \cdot 18 \cdot 18 = 972$ training samples. For the non-linear approach of Benhimane and Malis (2007) the complete template without sub-sampling is used.

Our experiments show that the approach of Benhimane and Malis (2007) gives the worst results, our approach based on dimensionality reduction using the DCT gives the best results, followed by the original approach of Jurie and Dhome (2002a). The approaches using the reformulation of the learning process give slightly worse results than Jurie and Dhome (2002a). However, as we show in Sect. 8.5, this can be compensated by adding new training samples to the linear predictors.

When evaluating the influence of the number of used DCT-coefficients in Fig. 7, we see that the pure DCT-based approach gives similar results when using 81, 49, or 25 DCT-coefficients, while the combined approach shows a significant drop in success rate when using only 25 DCT-coefficients.

8.3 Noise

Figure 8 shows the influence of noise on the tracking process. In order to measure the robustness of the considered approaches with respect to noise we corrupt the test images with noise drawn from a Gaussian distribution. The evaluation is done by increasing the standard deviation of the Gaussian distribution. This is similar to decreasing the

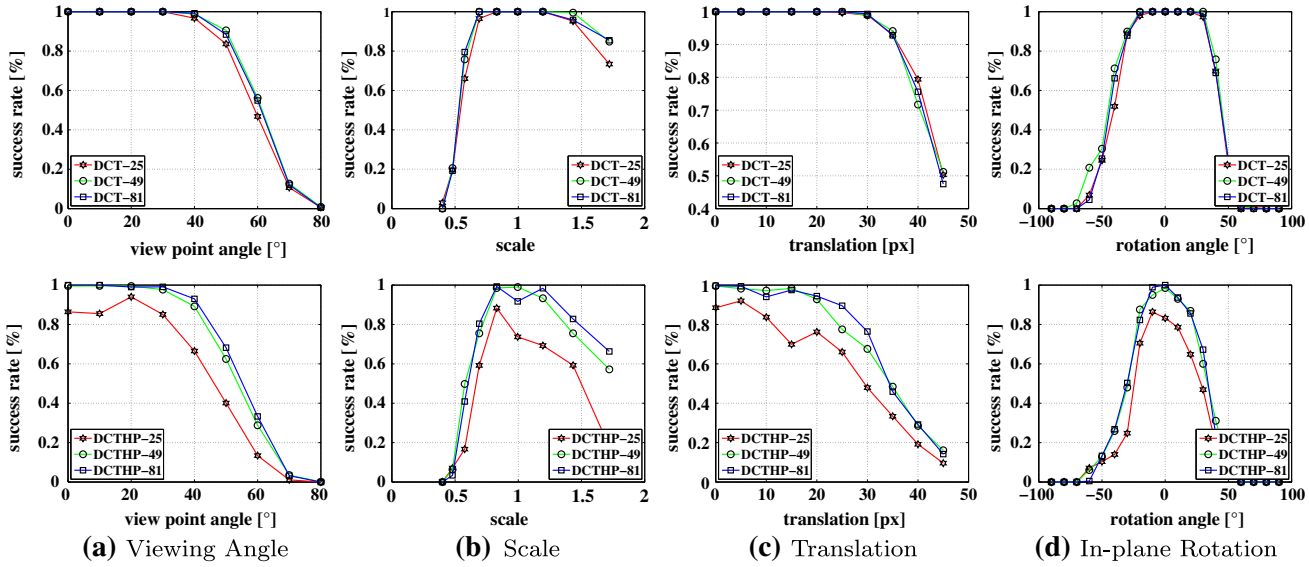


Fig. 7 Evaluation of tracking sucess rate for both DCT-based approaches in order to analyse the effect of the used number of DCT-coefficients. The first row shows the approach which is only based on dimensionality reduction and the second row the combined approach. **a** Viewing angle, **b** scale, **c** translation, **d** in-plane Rotation

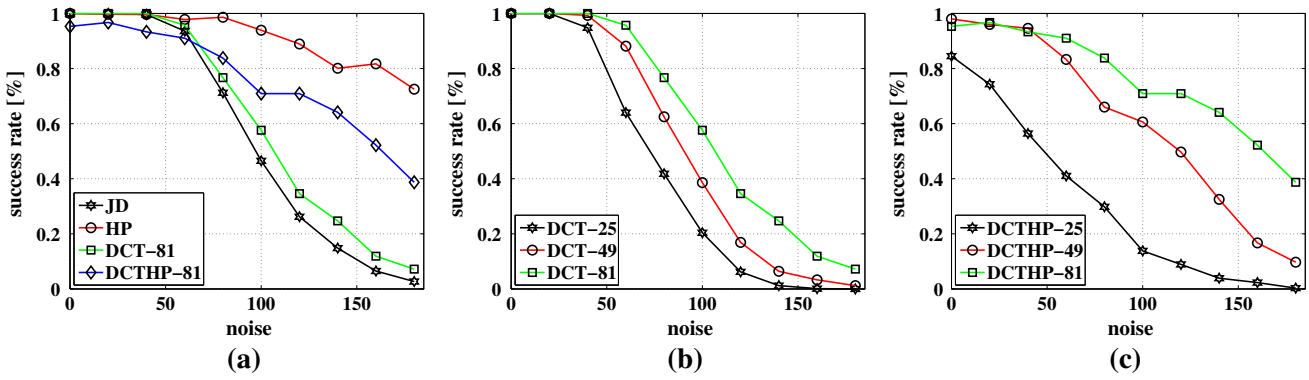


Fig. 8 Evaluation of the influence of noise on tracking success. **a** All approaches, where for the DCT-based approaches 81 DCT-coefficients are used. The evaluation of the influence of different numbers of DCT-coefficients for **b** the approach only based on dimensionality reduction and for **c** the combined approach. The shown results were compute from translation tests where a random translation of 10 ± 5 pixels is applied

signal-to-noise ratio. An example where this is important is in low-light conditions. This noise is added before we apply the random transformations. While Fig. 8a compares all considered learning approaches, Fig. 8b,c focus on DCT and DCTHP, respectively, in order to show how the number of DCT-coefficients influences the robustness with respect to noise. Here, JD gives the worst results and HP clearly the best. The pure dimensionality reduction approach gives results that are slightly better than the approach of Jurie and Dhome (2002a) and the combined approach gives results approximately in the middle between DCT and HP. In both DCT-based approaches, using more DCT-coefficients helps to improve the robustness with respect to noise.

8.4 Number of Training Samples

In Fig. 9 we evaluate the necessary learning time with respect to the number of random training samples used for training. By reducing the number of training samples used for learning we can significantly reduce the necessary learning time. However, this can produce linear predictors that result in less stable tracking. When looking at Fig. 10, we see that reducing the number of training samples has different effects for different approaches. While JD, DCT, and DCTHP show a certain variance in the resulting tracking success rate, the pure HP approach shows only a marginal difference between using 150 training samples and 1200 training sample. The

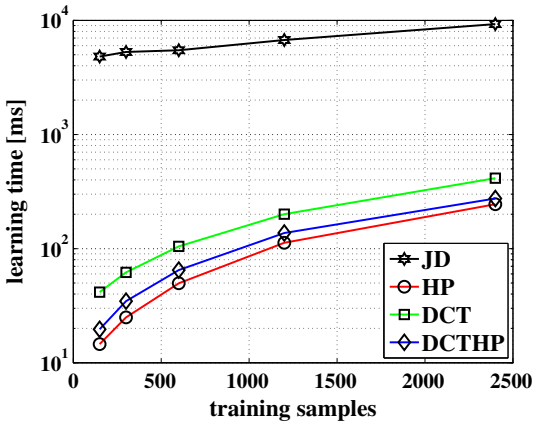


Fig. 9 Evaluation of the influence of the number of training samples on the necessary learning time

variance in the JD approach seems not to be related to the number of the training samples, while DCT and DCTHP show a clear relation between tracking robustness and the number of training samples.

8.5 Online Updating

In Fig. 11 we analyse the influence of updating linear predictors after learning, as described in Sect. 7. While the tracking robustness of JD and DCT is not influenced by updating, the HP-based approaches both significantly improve when being updated. However, this can be done online while tracking and therefore, does not influence the initial learning time. The reason for the improvement for the HP-based approaches can be seen in the fact that in the reformulated learning process of HP the training data is first projected onto a low-dimensional space, which makes the learning step less dependent on the number of training samples. However, the online updating is done in a different way where the training data is not projected onto a low-dimensional space and therefore, shows a more significant impact.

8.6 Real Data

In this section, we evaluate our proposed approaches on real data and analyze their results in relation to other state-of-the-art approaches. This includes quantitative evaluations on the datasets of Zimmermann et al. (2009) and Lieberknecht et al. (2009).

8.6.1 Zimmermann Dataset

In Table 1, we evaluate our approaches on the datasets provided by Zimmermann et al. (2009) and compare it to

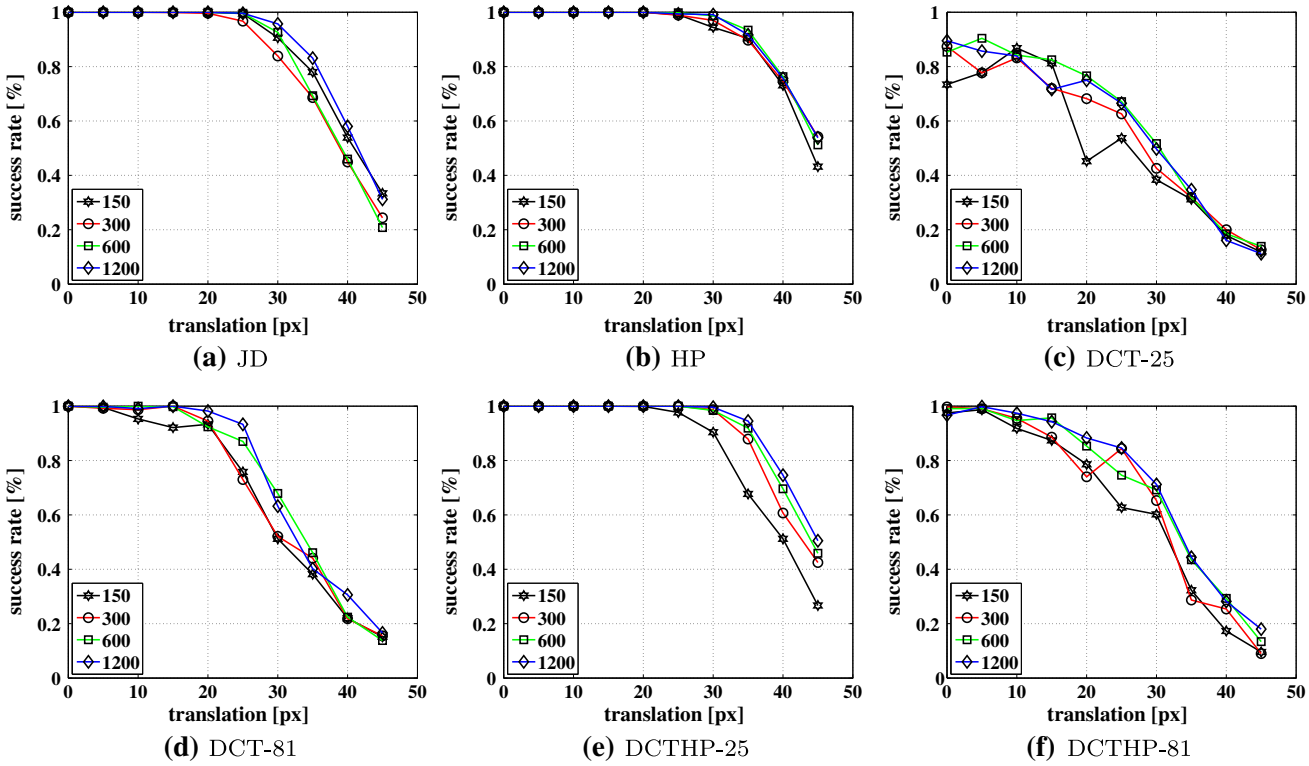


Fig. 10 Analysis of the influence of the number of training samples on the tracking success rate for **a** Jurie and Dhome (2002a), **b** the reformulated learning, the dimensionality reduction based approach for **c** 25 DCT-coefficients and **d** 81 DCT-ceofficients, and for the combined approach for **e** 25 DCT-coefficients and **f** 81 DCT-coefficients

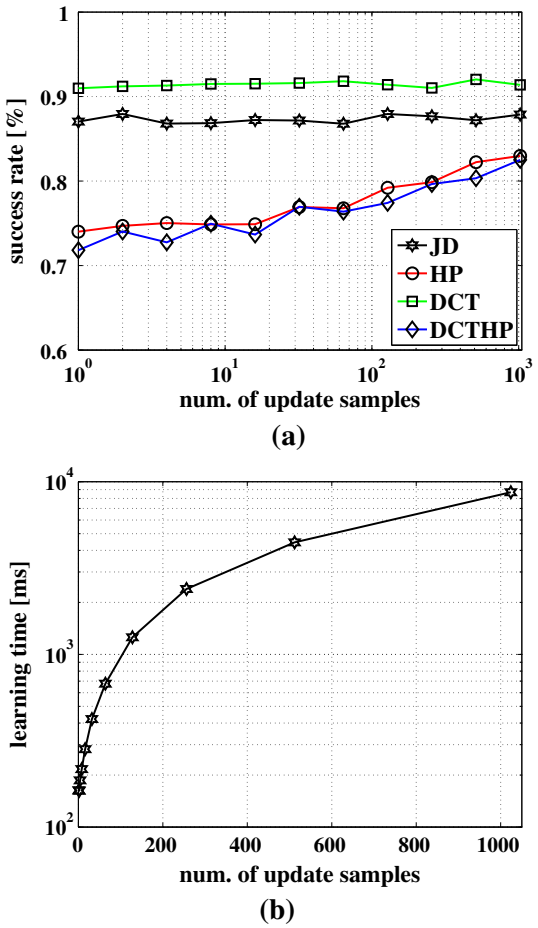


Fig. 11 **a** Evaluation of the influence of online updating on tracking robustness in case of random translations. The shown values are computed as average success rate of translation tests for translations ranging from 0 to 45 pixels. These average success rates are computed for different numbers of training samples. **b** Necessary time for updating

the results of other approaches as reported in Zimmermann et al. (2009), Holzer et al. (2010, 2013). Apart from our approaches, we include the results from Zimmermann et al. (2009) (NoSLLiP), Adaptive linear predictors (Holzer et al. 2010) (ALPs), Multi-Layered Adaptive linear predictors (Holzer et al. 2013) (ML-ALPs), an extension of the original linear predictor approach that is able to handle occlusions (Jurie and Dhome (2002b)) (LLiP LS, LLiP LS 1/2), Lukas-Kanade template tracking (Lucas and Kanade 1981) (LK), SIFT (Lowe 2004), as well as the tracking approach of Kalal et al. (2012) (TLD). Figure 12 shows a representative image for each of the (Zimmermann et al. 2009) datasets.

Speed is the most obvious difference between our approaches and the other approaches. Similar to Jurie and Dhome (2002a), we achieve processing frame rates of about 1,800 fps. While the ALPs approach still achieves almost 100 fps, all the others show frame rates below 25 fps and are therefore significantly slower. Note, that all three datasets (TOWEL, PHONE, MP) include frames where the template partly leaves the visible image region or is partly occluded.

Table 1 Comparison between the tracking results of NoSLLiP, SIFT, an extension of the original linear predictor approach that is able to handle occlusions (Jurie and Dhome 2002b) (LLiP LS, LLiP LS 1/2), Lukas-Kanade template tracking (Lucas and Kanade 1981) (LK), all as given in Zimmermann et al. (2009), as well as results for ALPs, Multi-layered ALPs (ML-ALPs), the approach of Jurie and Dhome (JD), TLD (Kalal et al. 2012), and our proposed approaches (HP, DCT, DCTHP)

Method	Object	Frame-rate (fps)	Loss-of-locks (-/-)	Error (%)
NoSLLiP	Towel	21.8	5/3,229	2.1
JD	Towel	1,887.8	74/3,230	1.8
HP	Towel	1,785.2	731/3,230	3.2
DCT-81	Towel	1,888.5	62/3,230	2.2
DCT-49	Towel	1,875.9	75/3,230	2.9
DCT-25	Towel	1,871.2	474/3,230	5.5
DCTHP-81	Towel	1,879.9	558/3,230	2.6
DCTHP-49	Towel	1,849.5	1628/3,230	4.7
DCTHP-25	Towel	1,839.8	2819/3,230	8.8
TLD	Towel	11.0	2,967/3,230	12.3
NoSLLiP	Phone	16.8	20/1,799	1.8
ALPs	Phone	96.7	10/2,299	1.2
JD	Phone	1,861.7	99/2,299	2.0
HP	Phone	1,758.8	1,765/2,299	4.0
DCT-81	Phone	1,862.4	214/2,299	3.8
DCT-49	Phone	1,883.7	2,018/2,299	5.1
DCT-25	Phone	1,886.1	1,951/2,299	14.5
DCTHP-81	Phone	1,827.7	2,271/2,299	2.0
DCTHP-49	Phone	1,869.8	2,203/2,299	11.7
DCTHP-25	Phone	1,871.6	2,299/2,299	16.0
TLD	Phone	7.7	1,623/2,299	11.7
NoSLLiP	MP	18.9	13/6,935	1.5
ML-ALPs	MP	17.2	10/6,945	2.1
SIFT	MP	0.5	281/6,935	1.4
LK (IC)	MP	2.6 (25)	398/6,935	2.4
LLiP LS	MP	24.4	1,083/6,935	6.3
LLiP LS 1/2	MP	24.2	93/6,935	3.0
JD	MP	1,773.2	249/6,945	2.0
HP	MP	1,666.9	524/6,945	2.2
DCT-81	MP	1,779	260/6,945	2.5
DCT-49	MP	1,799.6	254/6,945	2.9
DCT-25	MP	1,877.6	1,579/6,945	6.0
DCTHP-81	MP	1,776.3	619/6,945	2.6
DCTHP-49	MP	1,794	466/6,945	3.4
DCTHP-25	MP	1,868.9	1,378/6,945	4.3
TLD	MP	8.5	6,150/6,945	12.5

ALPs, ML-ALPs, JD, and our approaches are implemented in C++. For the evaluation of SIFT, Zimmermann et al. (2009) used a publicly available implementation. All other methods were implemented in Matlab

Fig. 12 Representative images of the three datasets provided by Zimmermann et al. (2009). From *left to right* the towel-, phone-, and mouse-pad-dataset

Therefore, approaches that are able to handle occlusions or out-of-image-scenarios (NoSLLiP, ALPs, ML-ALPs, LLiP LS 1/2) show the best loss-of-locks numbers.

For our algorithms, we see that the HP approach shows significant problems with the provided datasets. Among the datasets, it performs best in the MOUSEPAD sequence. The purely DCT-based approach performs well, compared to the original approach of Jurie and Dhome (2002a) (JD), in the TOWEL and MOUSEPAD sequences, but shows problems in the PHONE sequence.

As we confirm in Sect. 8.6.2, all presented approaches seem to have problems with repetitive textures. The DCTHP approach performs in a range between the HP and DCT approach, except for the phone sequence, where it fails in most cases. Since both, HP and DCT have problems with repetitive texture, this seems to add up in the combined approach. Again, similar observations can be made in Sect. 8.6.2.

In defense of TLD, we have to note that despite the high loss-of-locks numbers it is able to track the approximate location of the object of interest in the PHONE and MOUSEPAD datasets. But, it fails to do this most of the times in the TOWEL dataset. TLD is able to redected the tracked object when they lose tracking which poses a significant advantage. However, if other tracking approaches are combined with a detector a re-detection is possible for them too. While TLD is not able to track the full pose of the objects in the sequences, it also shows a tendency to drift apart from the initally selected template. Although it does not lose track of the template, the center of the bounding box can not be used to mark the center of the object in a stable way. The same holds for the extent of the bounding box which fluctuates over time. Its processing speed ranges within approximately $5 - 15$ fps and therefore, our proposed approaches are up to two orders of magnitude faster in tracking than TLD. In general, TLD might be well suited for tracking deformable or unconstrained objects that change their appearance online. However, if the change in appearance can be described via a model, *e.g.* a homography, other algorithms such as the ones shown in Table 1 seem more suitable.

8.6.2 Lieberknecht Dataset

For the evaluation in Table 2, we applied the original approach of Jurie and Dhome (2002a) as well as our proposed approaches on the datasets provided by Lieberknecht et al. (2009). For this evaluation, ground truth data is given every 250th frame and therefore, if tracking is lost, it can only be reinitialized at every 250th frame. The final evaluation results are obtained by providing the tracking results to the authors of the dataset. A frame is counted as successfully tracked if the average error of its tracked corners is less than 10 pixels. These corners are at an artificial position outside of the actual image data. Although this increases the effect of small misalignments, it prohibits from tracking data close to these control points in order to falsely improve results.

The (Lieberknecht et al. 2009) dataset considers four different texture scenarios (low, repetitive, normal, high) and five different motion and illumination scenarios (angle, range, fast far, fast close, illumination). For each texture scenario, two different scenes are provided (see Fig. 13). Exemplary changes for the motion and illumination scenarios are shown in Fig. 14. Before going into details of the evaluation, we want to note that a exact comparison between the success rates of the different approaches has to be done with caution, as the success percentage heavily depends on which of the 250 frame window the tracking failure occurs.

In general, the presented methods show good results on the low and normal texture scenarios. The success rate drops for DCT-based methods if the number of DCT-coefficients is reduced. In the 'Low-1' dataset, the original approach of Jurie and Dhome (2002a) (JD) is actually outperformed by the presented methods. In the case of 'Low-2', 'Normal-1', and 'Normal-2' similar results as achieved by JD are obtained. In the cases where the refumulated learning approach (HP) works well, it outperforms the other method in the illumination scenarios. This can be accounted to the increased robustness with respect to low signal-to-noise ratios, as shown in Sect. 8.3.

For the repetitive scenarios, we see that the reformulated learning (HP) shows significant problems. The purely

Table 2 Comparison of the approach of Jurie and Dhome (2002a) (JD) with our proposed approaches (HP, DCT, DCTHP) on the datasets of Lieberknecht et al. (2009)

Low-1	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)	Low-2	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)
JD	71.2	57.8	25.5	12.2	61.8	JD	100	73.6	33.5	31.6	79.2
HP	98.1	71.5	49.8	15.8	86.9	HP	98.6	75.3	21.2	27.6	79.6
DCT81	90.6	70.6	51.8	12.9	82.2	DCT81	96.0	73.7	28.4	27.8	76.2
DCT49	62.3	53.2	19.0	12.2	57.3	DCT49	92.8	72.2	26.9	17.4	74.2
DCT25	62.5	50.1	16.2	11.0	69.3	DCT25	28.0	8.2	2.7	4.3	19.2
DCTHP81	90.6	70.6	51.8	12.9	82.2	DCTHP81	99.6	73.7	21.1	23.4	75.7
DCTHP49	90.3	56.8	21.9	13.2	77.4	DCTHP49	89.3	78.8	20.3	16.6	73.8
DCTHP25	74.0	51.7	21.1	11.5	75.1	DCTHP25	6.8	6.2	4.8	2.4	7.5
Repetitive-1	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)	Repetitive-2	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)
JD	70.1	69.1	33.4	20.6	78.0	JD	84.1	47.8	9.4	34.8	92.5
HP	18.2	9.4	7.8	4.2	18.8	HP	6.1	1.5	2.3	0.6	10.2
DCT81	69.8	62.8	28.3	15.2	74.2	DCT81	76.9	35.6	10.1	19.9	79.6
DCT49	68.4	54.2	35.2	14.0	67.5	DCT49	76.6	21.8	7.8	13.2	71.8
DCT25	28.0	8.2	2.7	4.3	19.2	DCT25	18.7	6.2	1.2	2.2	22.5
DCTHP81	14.2	5.0	3.8	2.5	9.2	DCTHP81	2.4	0.3	0.2	0.9	0.4
DCTHP49	3.0	1.6	4.2	0.1	2.1	DCTHP49	0.4	0.2	0.2	0.3	0.8
DCTHP25	1.7	6.6	3.3	1.8	2.5	DCTHP25	0.3	1.0	0.6	0.6	0.3
Normal-1	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)	Normal-2	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)
JD	96.8	60.4	23.6	13.7	72.8	JD	97.8	55.2	17.2	12.3	82.2
HP	99.4	51.2	21.7	15.8	75.5	HP	75.3	55.2	15.5	11.3	96.9
DCT81	94.0	46.2	22.8	12.4	71.8	DCT81	96.5	54.6	20.4	12.0	77.7
DCT49	63.3	15.8	6.1	11.6	65.1	DCT49	54.1	36.6	12.7	11.2	72.2
DCT25	37.1	4.8	4.7	6.9	19.7	DCT25	32.2	4.9	4.0	6.3	18.8
DCTHP81	96.8	54.2	8.8	12.8	69.8	DCTHP81	88.5	54.9	15.0	10.9	74.0
DCTHP49	55.9	33.1	7.9	12.2	51.0	DCTHP49	75.9	51.2	14.5	10.8	82.0
DCTHP25	27.5	5.0	4.6	7.0	15.9	DCTHP25	9.8	0.8	0.4	2.8	2.5
High-1	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)	High-2	Angle (%)	Range (%)	Fast far (%)	Fast close (%)	Illum. (%)
JD	44.2	8.2	5.8	5.1	67.4	JD	29.6	14.8	8.1	19.2	50.8
HP	17.2	4.0	4.3	3.3	33.4	HP	6.8	6.2	4.8	2.4	7.5
DCT81	9.4	4.7	2.2	3.2	36.1	DCT81	26.8	12.2	7.8	13.5	50.9
DCT49	1.7	1.8	1.8	2.3	15.2	DCT49	29.5	12.2	7.2	9.6	49.1
DCT25	0.5	0.4	0.3	0.7	2.7	DCT25	10.4	4.0	6.0	4.8	22.7
DCTHP81	3.5	4.5	2.5	2.4	48.7	DCTHP81	20.3	6.3	6.0	9.8	21.1
DCTHP49	0.8	1.8	0.4	0.7	6.1	DCTHP49	13.4	3.7	4.3	3.6	0.5
DCTHP25	0.1	0.2	0.1	0.5	0.4	DCTHP25	1.8	0.4	0.9	2.4	1.8

The results show the tracking success rate under different texture, motion, and illumination conditions. Results for other approaches can be found in Lieberknecht et al. (2009)

DCT based methods still perform acceptable and in a similar range as JD. However, the combination of both approaches (DCTHP) also fails in these scenarios. The most challenging sequences are the high texture sequences. While DCT81 and DCT49 still achieve similar results as JD in 'High-2' all proposed methods show a significant drop in performance on 'High-1'. This can be explained by the fine details of the texture which are necessary to be considered for successfull tracking. Due to the compression applied in the DCT-based approaches, they tend to lose these details and therefore fail.

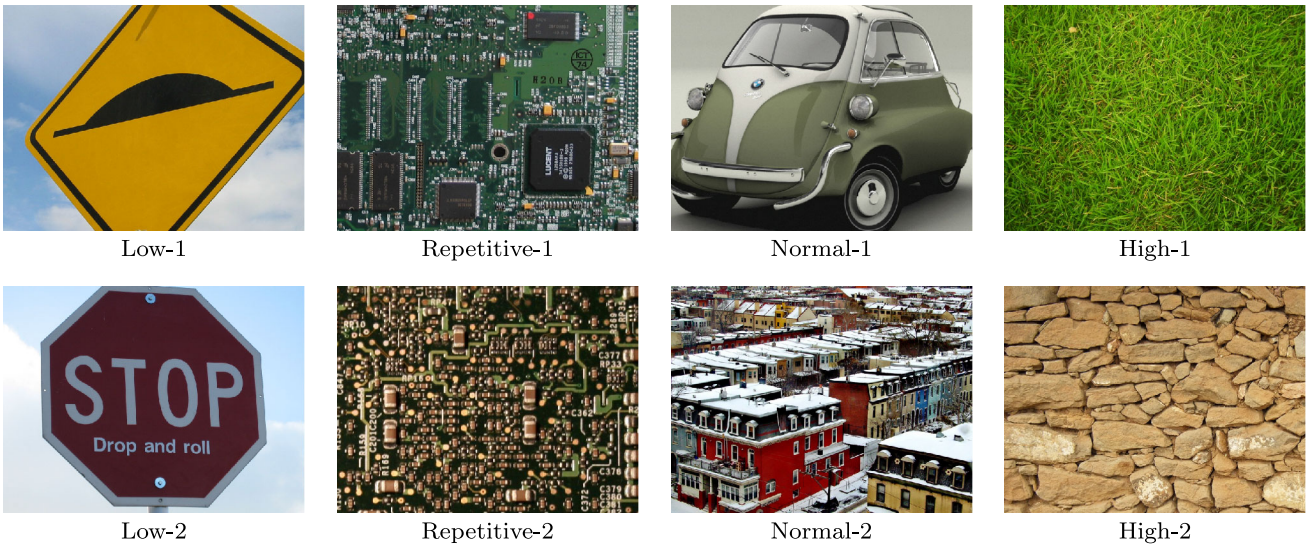


Fig. 13 Representative images of the datasets provided by Lieberknecht et al. (2009). From left to right column: low, repetitive, normal, and high texture

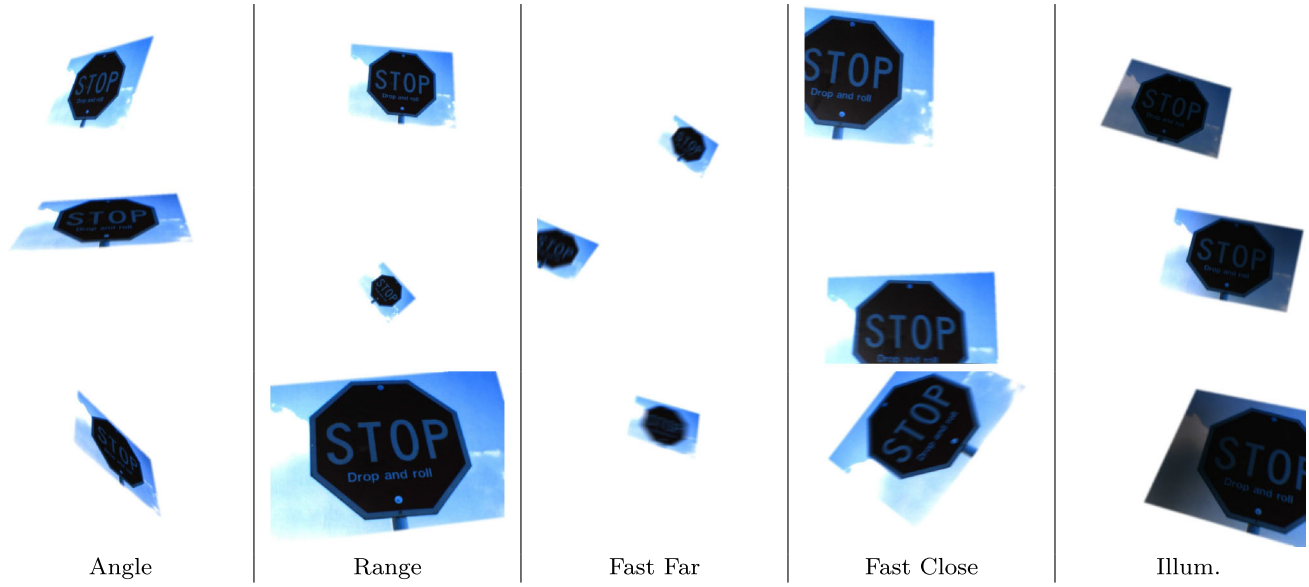


Fig. 14 Representative images of the Low-2 dataset provided by Lieberknecht et al. (2009) to illustrate the different motion and illumination scenarios used in Table 2

9 Applications

Due to their high efficiency, our proposed approaches are well-suited for applications using mobile devices. Therefore, we implemented them on a standard mobile phone with 1 GB of RAM and a 1.2 GHz dual core processor where no optimization for processor specific technology is implemented, and only a single core is used for learning and tracking.

The learning process for a template with 16×16 sample points and $16 \cdot 16 \cdot 3 = 768$ training samples took approximately 18000 ms for the original approach of Jurie and Dhome (2002a), about 600 ms for our proposed approach using dimensiontality reduction, and roughly 350 ms for the approach using the reformulation of the learning equation. Based on these results, the reformulated approach is more than 50 times faster than the approach of Jurie and Dhome (2002a) and allows interactive applications to start tracking almost immediately. On the other hand, tracking takes about 2.5 ms for all approaches.

10 Conclusions

Linear predictors were first introduced by Jurie and Dhome (2002a) and enable robust tracking at very high frame rates.

The main goal of this paper is to overcome the long learning time which was the main drawback of their approach. Thus, we proposed three different methods to speed-up this learning procedure for template tracking. While they all significantly reduced learning time in two orders of magnitude range, they individually have different properties that makes them suited for different application scheme.

Our dimensionality reduction based approach in Sect. 4 gives the highest tracking success rate. By choosing an appropriate number of DCT-coefficients in learning, it reaches similar learning speeds as the other approaches and even works with a very low number of training samples. However, if a significant amount of noise is present, the reformulation of the learning process in Sect. 5 gives the best results. But this approach leads to a slightly decreased success rate in tracking. Lastly, the combined approach in Sect. 6 is a compromise between the two other approaches and has the advantage of learning speed, robustness to noise and robustness to large motions.

Acknowledgments This work was partly funded by Willow Garage, Inc., California, USA.

References

Baker, S., & Matthews, I. (2001). Equivalence and efficiency of image alignment algorithms. In *Conference on Computer Vision and Pattern Recognition*.

Baker, S., & Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, *56*(3), 221–255.

Ben-Israel, A., & Greville, T. N. (2003). *Generalized inverses*. Berlin: Springer.

Benhimane, S., & Malis, E. (2007). Homography-based 2d visual tracking and servoing. *International Journal of Robotics Research*, *26*(7), 661–676.

Cascia, M., Sclaroff, S., & Athitsos, V. (2000). Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*, 322–336.

Dame, A., & Marchand, E. (2010). Accurate real-time tracking using mutual information. In *2010 9th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*.

Dellaert, F., & Collins, R. (1999). Fast image-based tracking by selective pixel integration. In *ICCV Workshop of Frame-Rate Vision*.

Grabner, H., Leistner, C., & Bischof, H. (2008). Semi-supervised online boosting for robust tracking. In: D. Forsyth, P. Torr & A. Zisserman (Eds.), *Computer vision ECCV 2008. Lecture notes in computer science* (Vol. 5302, pp. 234–247). Berlin Heidelberg: Springer.

Gräßl, C., Zinßer, T., & Niemann, H. (2003). Illumination insensitive template matching with hyperplanes. In *Proceedings of Pattern recognition: 25th DAGM Symposium*.

Gräßl, C., Zinßer, T., & Niemann, H. (2004). Efficient hyperplane tracking by intelligent region selection. In *Image Analysis and Interpretation*.

Hager, G., & Belhumeur, P. (1998). Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(10), 1025–1039.

Hinterstoisser, S., Benhimane, S., Navab, N., Fua, P., & Lepetit, V. (2008). Online learning of patch perspective rectification for efficient object detection. In *Conference on Computer Vision and Pattern Recognition*.

Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., & Lepetit, V. (2011). Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *IEEE International Conference on Computer Vision (ICCV)*.

Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., & Lepetit, V. (2009). Real-time learning of accurate patch rectification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., & Navab, N. (2010). Dominant orientation templates for real-time detection of texture-less objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., & Navab, N. (2012). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision*.

Holzer, S., Hinterstoisser, S., Ilic, S., & Navab, N. (2009). Distance transform templates for object detection and pose estimation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Holzer, S., Ilic, S., & Navab, N. (2010). Adaptive linear predictors for real-time tracking. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Holzer, S., Ilic, S., & Navab, N. (2013). Multi-layer adaptive linear predictors for real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(1), 105–117.

Holzer, S., Ilic, S., Tan, D., & Navab, N. (2012). Efficient learning of linear predictors using dimensionality reduction. In *Asian Conference on Computer Vision*.

Holzer, S., Pollefeys, M., Ilic, S., Tan, D.J., & Navab, N. (2012). Online learning of linear predictors for real-time tracking. In *12th European Conference on Computer Vision (ECCV)*.

Jurie, F., & Dhome, M. (2002). Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(7), 996–1000.

Jurie, F., & Dhome, M. (2002). Real time robust template matching. In *British Machine Vision Conference*.

Kalal, Z., Mikolajczyk, K., & Matas, J. (2012). Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(7), 1409–1422.

Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007. ISMAR 2007*, pp. 225–234.

Klein, G., & Murray, D. (2008). Improving the agility of keyframe-based slam. *Computer Vision ECCV 2008. Lecture Notes in Computer Science* (Vol. 5303, pp. 802–815). Berlin Heidelberg: Springer.

Lieberknecht, S., Benhimane, S., Meier, P., Navab, N. (2009). A dataset and evaluation methodology for template-based tracking algorithms. In *ISMAR*, pp. 145–151.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.

Lucas, B., & Kanade, T. (1981) An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conference on Artificial Intelligence*.

Malis, E. (2004). Improving vision-based control using efficient second-order minimization techniques. In *IEEE International Conference on Robotics and Automation*.

Matas, J., Zimmermann, K., Svoboda, T., Hilton, A. (2006). Learning efficient linear predictors for motion estimation. In *Computer Vision, Graphics and Image Processing*.

Mayol, W.W., & Murray, D.W. (2008). Tracking with general regression. *Journal of Machine Vision and Applications*, *19*(1), 65–72.

Özuysal, M., Fua, P., Lepetit, V. (2007). Fast Keypoint Recognition in Ten Lines of Code. In *Conference on Computer Vision and Pattern Recognition*.

Parisot, P., Thiesse, B., & Charvillat, V. (2007). Selection of reliable features subsets for appearance-based tracking. In *Signal-Image Technologies and Internet-Based System*, 16–18 Dec 2007, pp. 891–898.

Richa, R., Sznitman, R., Taylor, R., Hager, G. (2011). Visual tracking using the sum of conditional variance. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Shum, H.Y., & Szeliski, R. (2000). Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, *36*(2), 101–130.

Vacchetti, L., Lepetit, V., & Fua, P. (2004). Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(10), 1385–1391.

Zimmermann, K., Matas, J., & Svoboda, T. (2009). Tracking by an optimal sequence of linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(4), 677–692.

APPENDIX

F

ADAPTIVE NEIGHBORHOOD SELECTION FOR REAL-TIME SURFACE NORMAL ESTIMATION FROM ORGANIZED POINT CLOUD DATA USING INTEGRAL IMAGES

Own contributions. My contributions to this work include the core idea, the design, and the implementation of the methods for adaptive neighborhood selection and efficient computation of surface normals. The core idea of the paper was refined in collaboration with the other co-authors. All co-authors were involved in the writing of the paper.

Adaptive Neighborhood Selection for Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images

S. Holzer[1,2] and R. B. Rusu[2] and M. Dixon[2] and S. Gedikli[2] and N. Navab[1]

***Abstract*— In this paper we present two real-time methods for estimating surface normals from organized point cloud data. The proposed algorithms use integral images to perform highly efficient border- and depth-dependent smoothing and covariance estimation. We show that this approach makes it possible to obtain robust surface normals from large point clouds at high frame rates and therefore, can be used in real-time computer vision algorithms that make use of Kinect-like data.**

I. Introduction

The recent development of a new class of affordable depth sensors, like the Kinect, has been of great interest to the robotics community. These new sensors are able to simultaneously capture high-resolution color and depth images at high frame rates. When the camera's intrinsic calibration parameters are known, these depth images can be converted into *organized point clouds* (i.e., clouds of 3D points sampled from a regular 2D grid), which are useful in a wide array of important robotics applications, such as 3D registration and object recognition.

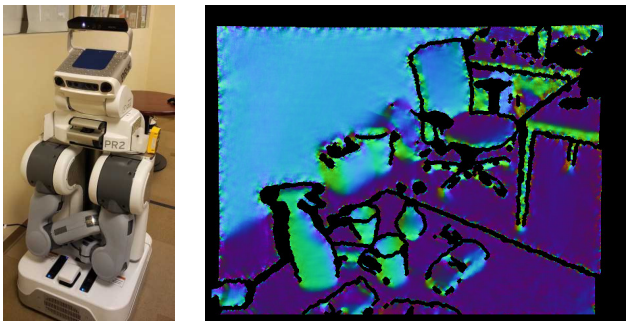
Besides depth, surface normal orientation is one of the most discrimative information that can be obtained from point clouds and is therefore optimally suited to be used in object detection approaches. However, state-of-the-art normal estimation algorithms are often slow when operating on large and/or noisy point clouds. Thus, for robotics applications that require real-time performance, fast normal estimation is essential.

In this work, we present two algorithms for estimating surface normals in organized point clouds. Both methods employ an adaptive window size to analyze local surfaces, which allows us to effectively handle depth-dependent sensor noise and to avoid common artifacts caused by depth discontinuities. Typically, multi-scale algorithms come with increased computational costs, especially when large window sizes are used; however, because of the point clouds' inherent grid structure, we are able to use integral images to perform the necessary computation in constant time, independent of the window size.

II. Related Work

Normal estimation methods can be divided into two different categories: *averaging* and *optimization-based* methods [1]. *Averaging methods* [1], [2], [3], [4], [5], [6] compute the normals at a certain point as a (weighted) average of point data within a certain neighborhood. Examples for these neighboring points can be the nearest neighbors, all points within a certain region of interest, or points that are in some other way connected to the point of interest, e.g. neighbors in a triangulated mesh. The weights, which define how strong the influence of particular information taken from the local neighborhood is, can be computed in different ways. Common ways of computing the weighted mean include weighting all points equally [2], weighting by angle [5], weighting by sine and edge length reciprocals [4], weighting by areas of adjacent triangles [4], weighting by edge length reciprocals [4], and weighting by square root of edge length reciprocals [4]. In [6], Holz *et al*. presented how integral images can be used in the efficient averaging for normal estimation process. For a more detailed overview of methods based on averaging please refer to [3].

Optimization-based methods usually try to fit geometric primitives, e.g. a plane, into the local neighborhood of the point of interest or penalize other criteria, e.g. the angle between the estimated normal vector and tangential vectors. If the optimization is formulated as a linear problem in matrix-vector notation, the desired minimization can be directly obtained from the result of a singular value decomposition (SVD) or a principal component analysis (PCA) [1]. Amongst others, existing methods try to fit planes [7], [8], [9], [10], maximize the angle between the tangential vectors and the normal vector [11], or try to estimate not only the orientation of the tangent plane but also the curvature [12], [13], [14]. A more detailed comparison of these methods can be found in [1].



(a) (b)

Fig. 1. **(a)** PR2 robot with a Kinect depth sensor mounted on its head. **(b)** Color-coded surface normals estimated using the proposed approaches.

*This work was supported by Willow Garage Inc.

[1]S. Holzer and N. Navab are with the Department of Computer Science, Technische Universität München, 85748 Garching bei München, Germany {holzers, navab}@in.tum.de

[2]S. Holzer, R. B. Rusu, M. Dixon and S. Gedikli are with Willow Garage Inc., Menlo Park, CA 94025, USA {holzers, rusu, mdixon, gedikli}@willowgarage.com

III. Normal Estimation

In the following we will describe the different steps we use to compute surface normals from organized point clouds based on integral images. First, we will describe the basic principle of integral images. Then we introduce the pre-processing step used to estimate the neighborhood size for each pixel. Finally, we introduce two different approaches for computing surface normals: a method based on averaging as well as an optimization-based method. Both techniques make use of integral images to improve processing speed. The first approach (see Sec. III-C) uses a single integral image for depth and border aware smoothing of the depth map, while the second approach (see Sec. III-D) computes covariance matrices using integral images and obtains the normals from the covariance matrices.

A. Integral Images

An integral image $\mathcal{I}_\mathcal{O}$ corresponding to an image $\mathcal{O}$ makes it possible to compute the sum of all values of $\mathcal{O}$ within a certain rectangular region $\mathcal{R}_{(m_s,n_s)\rightarrow(m_e,n_e)}$ (see Fig. 2) by accessing only four data elements in memory. This not only makes the computation very efficient but also makes the computational costs independent of the size of the rectangle. We will use this property later for smoothing since it allows us to use smoothing areas which differ in size for every point. To be able to compute the area sum of an image $\mathcal{O}$ each pixel element $(m,n)^\top$ in the integral image $\mathcal{I}_\mathcal{O}$ is defined as the sum of all elements which are inside of the rectangular area between $\mathcal{O}(0,0)$ and $\mathcal{O}(m,n)$:

$$\mathcal{I}_\mathcal{O}(m,n) = \sum_{i=0}^{m}\sum_{j=0}^{n}\mathcal{O}(i,j). \tag{1}$$

This can be efficiently computed iteratively as

$$\begin{aligned}\mathcal{I}_\mathcal{O}(m,n) \quad = \quad & \mathcal{O}(m,n) \\ & + \mathcal{I}_\mathcal{O}(m-1,n) \\ & + \mathcal{I}_\mathcal{O}(m,n-1) \\ & - \mathcal{I}_\mathcal{O}(m-1,n-1),\end{aligned} \tag{2}$$

where $\mathcal{I}_\mathcal{O}(u,v) = 0$ if (u,v) is not in the domain of $\mathcal{O}$. Therefore, a single pass over the input image is sufficient to compute the corresponding integral image. The average value within a region can then be computed as

$$\begin{aligned}S(\mathcal{I}_\mathcal{O},m,n,r) = \frac{1}{4r^2}\cdot \quad (\quad & \mathcal{I}_\mathcal{O}(m+r,n+r) \\ & - \mathcal{I}_\mathcal{O}(m-r,n+r) \\ & - \mathcal{I}_\mathcal{O}(m+r,n-r) \\ & + \mathcal{I}_\mathcal{O}(m-r,n-r) \quad),\end{aligned} \tag{3}$$

where $(m,n)^\top$ defines the center and r the inner radius of the rectangular region.

B. Smoothing

Data obtained from a 3D sensor typically contains noise. The standard approach for reducing the effect of noise is to use smoothing. A big advantage of using integral images for averaging is hereby that its processing speed is independent of the size of the smoothing area since we always need the same amount of memory accesses. Therefore, we can use varying smoothing sizes depending on the characteristics of the considered point and its neighborhood.

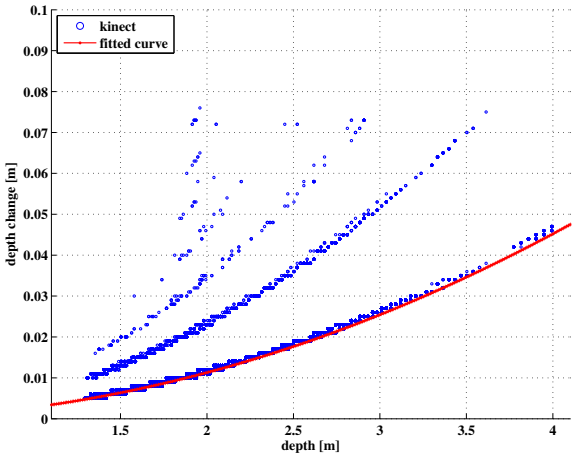


Fig. 3. Visualization of depth changes (blue circles) depending on the depth at which they occur. The minimal depth changes are fitted by a parabola (red line).

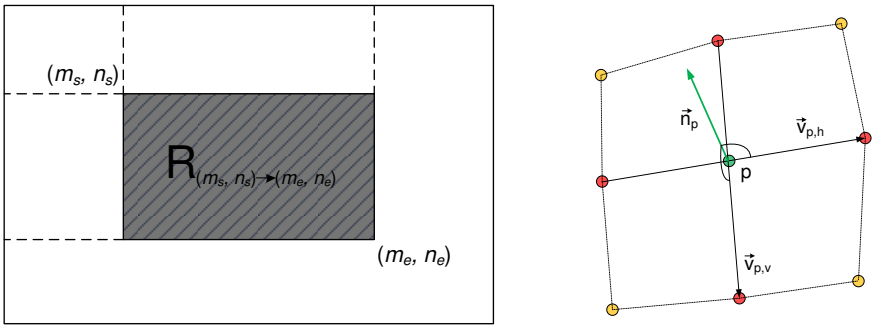
In the following we will use two different indicators for estimating the size of the smoothing area for a certain point of interest. The first indicator is the depth of the point of interest, since the noise usually depends on the depth of the perceived data, i.e. if data is acquired at a far distance then it has a worse signal-to-noise ratio than data that is acquired at close distances. However, if the size of the smoothing area is determined based only on the depth of the point of interest we would also smooth over object borders. This would merge information across two distinct surfaces and lead to incorrect normal estimates. In order to prevent this, we make the size of the smoothing area also dependent on large depth changes which are likely to be object borders. Both indicators are then combined into a single Smoothing Area Map which defines the smoothing area for each point of the organized point cloud.

1) Depth-Dependent Smoothing Area Map: Fig. 3 shows the minimal depth change (blue circles) that can occure at a specific depth. As shown, the value gets bigger with increasing depth. The red curve shows that the relationship between depth and minimum depth change can be described using the function

$$f_{DC}(d) = \alpha \cdot d^2, \tag{4}$$

where d is a depth value. For the device used in our experiments we estimated $\alpha = 0.0028$. Based on this observation it is clear that the smoothing area has to get bigger with increasing depth in a similar way. Therefore, we define the response of the depth-dependent *Smoothing Area Map* $\mathcal{B}(m,n)$:

$$\mathcal{B}(m,n) = \beta \cdot f_{DC}\left(\mathcal{D}\left(m,n\right)\right), \tag{5}$$



(a) (b)

Fig. 2. **(a)** The sum of a 2D region can be efficiently computed from an integral image by accessing only four data elements in memory which correspond to the for corners of the rectangular region. **(b)** Estimating a surface normal as cross-product of the vectors between the horizontal and vertical neighbors of the point of interest.

where β is a user specified value to control the increase of the smoothing area size and $\mathcal{D}(m,n)$ is the depth value at the image point $(m,n)^\top$.

2) Depth Change Indication Map: The simplest solution for computing the *Depth Change Indication Map* would be to apply a threshold on the first derivative of the depth map. However, as we just saw, the minimum possible depth change depends on the depth and therefore, a simple threshold would only be valid at a specific depth. Instead, we create a binary *Depth Change Indication Map* $\mathcal{C}$ by using a depth change detection threshold $t_{DC}(d)$ which is dependent on the actual distance:

$$t_{DC}(d) = \gamma \cdot f_{DC}\,(d)\,, \tag{6}$$

where γ is a scale factor which defines how sensitive the depth change detection will be. Assuming that $\delta\mathcal{D}_x(m,n) = \mathcal{D}(m+1,n) - \mathcal{D}(m,n)$ and $\delta\mathcal{D}_y(m,n) = \mathcal{D}(m,n+1) - \mathcal{D}(m,n)$, the depth change indication map $\mathcal{C}$ is then computed as

$$\mathcal{C}(m,n) = \begin{cases} 1 & \begin{cases} \text{if } \|\delta\mathcal{D}_x(m,n)\| \geq t_{DC}(\mathcal{D}(m,n)) \\ \text{or } \|\delta\mathcal{D}_y(m,n)\| \geq t_{DC}(\mathcal{D}(m,n)) \end{cases} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

3) Final Smoothing Area Map: Using the depth-dependent *Smoothing Area Map* $\mathcal{B}$ and the *Depth Change Indication Map* $\mathcal{C}$ we then compute the final *Smoothing Area Map* $\mathcal{R}$. For this, we first compute the distance transform map [15] corresponding to $\mathcal{C}$, which gives us for each point $(m,n)^\top$ the 2D distance to the next depth change as $\mathcal{T}(m,n)$. $\mathcal{R}$ is then computed as

$$\mathcal{R}(m,n) = \min(\mathcal{B}(m,n), \frac{\mathcal{T}(m,n)}{\sqrt{2}}), \tag{8}$$

where $\min(a,b)$ returns the minimum of a and b. The distance values $\mathcal{T}(m,n)$ are divided by the square root of 2 since we do not use circular but rectangular smoothing areas.

C. Normal Estimation based on Smoothed Depth Changes

A standard way of estimating the surface normal $\vec{n}_p$ at a point p at image location $(m,n)^\top$ is to compute the 3D vector $\vec{v}_{p,h}$ between the left and right neighbor as well as the vector $\vec{v}_{p,v}$ between the upper and lower neighbor of p and then computing the cross-product between these two vectors:

$$\vec{n}_p = \vec{v}_{p,h} \times \vec{v}_{p,v} \tag{9}$$

Due to the noise characteristics of depth sensors, *e.g.* the Kinect, this would lead to noisy normals. Smoothing the depth data before computing the normals is a common approach to reduce the influence of noise. However, smoothing with a fixed window size also smoothes over object boundaries, which leads to undesired artifacts. Therefore, we use the smoothing area map described in Sec. III-B.3 to prevent from smoothing over object boundaries and efficiently compute the vectors $\vec{v}_{p,h}$ and $\vec{v}_{p,v}$ as:

$$\vec{v}_{p,h,x} = \frac{\mathcal{P}_x(m+r,n) - \mathcal{P}_x(m-r,n)}{2}, \tag{10}$$

$$\vec{v}_{p,h,y} = \frac{\mathcal{P}_y(m+r,n) - \mathcal{P}_y(m-r,n)}{2}, \tag{11}$$

$$\vec{v}_{p,h,z} = \frac{S(\mathcal{I}_{\mathcal{P}_z}, m+1, n, r-1)}{2} - \frac{S(\mathcal{I}_{\mathcal{P}_z}, m-1, n, r-1)}{2} \tag{12}$$

$$\vec{v}_{p,v,x} = \frac{\mathcal{P}_x(m,n+r) - \mathcal{P}_x(m,n-r)}{2}, \tag{13}$$

$$\vec{v}_{p,v,y} = \frac{\mathcal{P}_y(m,n+r) - \mathcal{P}_y(m,n-r)}{2}, \tag{14}$$

$$\vec{v}_{p,v,z} = \frac{S(\mathcal{I}_{\mathcal{P}_z}, m, n+1, r-1)}{2} - \frac{S(\mathcal{I}_{\mathcal{P}_z}, m, n-1, r-1)}{2}, \tag{15}$$

where $\mathcal{P}_x$, $\mathcal{P}_y$, and $\mathcal{P}_z$ are two-dimensional maps storing the x-, y-, and z-coordinates of the organized point cloud, $\mathcal{I}_{\mathcal{P}_z}$ is the integral image of the z-components of the point cloud, and $r = \mathcal{R}(m,n)$. The normals are then computed using Eq. (9).

D. Normal Estimation based on Covariance Matrices

Our second method is an optimization-based method, where we estimate surface normals by trying to fit a plane

into the local neighborhood $\mathcal{N}_p$ of the point of interest p. This is done by computing the eigenvectors of the corresponding covariance matrix $\mathcal{C}_p$. The size of the neighborhood is estimated using the smoothing area map as described in Sec. III-B.3. Neighboring points are commonly found by performing a nearest neighbor search or selecting all points within a certain distance. This, however, is an expensive operation and therefore, we make use of the method described by Porikli and Tuzel [16] for efficient computation of covariance matrices using integral images. For this we have to compute nine integral images, where three of them, namely $\mathcal{I}_{\mathcal{P}_x}$, $\mathcal{I}_{\mathcal{P}_y}$, and $\mathcal{I}_{\mathcal{P}_z}$, are for the x-, y- and z-coordinates of the points of the point cloud and the remaining six are for all possible combinations of the point-coordinates: $\mathcal{I}_{\mathcal{P}_{xx}}$, $\mathcal{I}_{\mathcal{P}_{xy}}$, $\mathcal{I}_{\mathcal{P}_{xz}}$, $\mathcal{I}_{\mathcal{P}_{yy}}$, $\mathcal{I}_{\mathcal{P}_{yz}}$, $\mathcal{I}_{\mathcal{P}_{zz}}$, where $\mathcal{I}_{\mathcal{P}_{ab}}$ is the element-wise multiplication of $\mathcal{I}_{\mathcal{P}_a}$ and $\mathcal{I}_{\mathcal{P}_b}$. The covariance matrix $\mathcal{C}_p$ for a point p at $(m,n)^\top$ can then be computed as:

$$\mathcal{C}_p = \begin{pmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{pmatrix} - \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix}^\top, \quad (16)$$

with

$$\begin{aligned} c_{xx} &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_{xx}}, m, n, \mathcal{R}(m,n)), & (17)\\ c_{xy} = c_{yx} &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_{xy}}, m, n, \mathcal{R}(m,n)), & (18)\\ c_{xz} = c_{zx} &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_{xz}}, m, n, \mathcal{R}(m,n)), & (19)\\ c_{yy} &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_{yy}}, m, n, \mathcal{R}(m,n)), & (20)\\ c_{yz} = c_{zy} &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_{yz}}, m, n, \mathcal{R}(m,n)), & (21)\\ c_{zz} &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_{zz}}, m, n, \mathcal{R}(m,n)), & (22) \end{aligned}$$

and

$$\begin{aligned} c_x &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_x}, m, n, \mathcal{R}(m,n)), & (23)\\ c_y &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_y}, m, n, \mathcal{R}(m,n)), & (24)\\ c_z &= \mathcal{S}(\mathcal{I}_{\mathcal{P}_z}, m, n, \mathcal{R}(m,n)). & (25) \end{aligned}$$

Finally, we compute the normal $\vec{n}_p$ from the covariance matrix $\mathcal{C}_p$ as the eigenvector which corresponds to the smallest eigenvalue. Although this method is computationally more expensive than the method described in Sec. III-C it has the advantage that the eigenvalues of the covariance matrix can be directly used to get information about the planarity of the neighborhood around the point p at image location $(m,n)^\top$. This can be used for plane fitting as well as efficient edge or corner detection.

IV. Results

In this section we show how the presented surface normal estimation methods perform under various conditions and compare them to a state-of-the-art kNN-based implementation [17]. All experiments are performed on a standard laptop with a 2.26 GHz Intel(R) Core(TM)2 Quad CPU and 4 GB of RAM, where only one core is used for the computations. An open-source implementation of our approach is available in the Point Cloud Library (PCL)[1] [18].

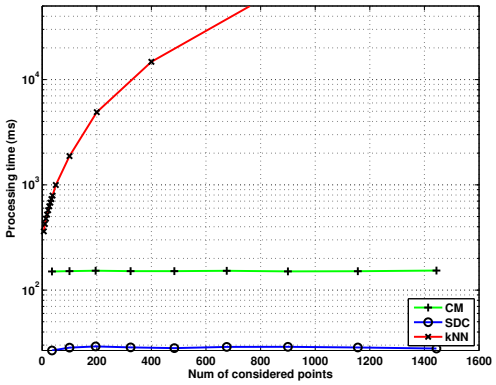


Fig. 4. Processing time of different normal estimation methods with respect to the size of the smoothing area. Note that the processing time is given in log-scale.

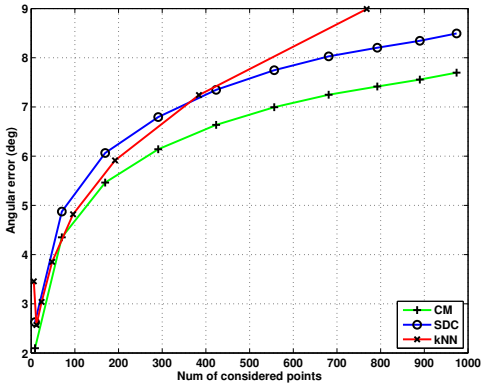


Fig. 5. Normal estimation error for different normal estimation methods with respect to the size of the smoothing area.

A. Processing Speed

Fig. 4 shows the processing time with respect to the number of points considered for smoothing for the approaches introduced in Sec. III-C (SDC), Sec. III-D (CM), and a state-of-the-art kNN-based approach (kNN) of Rusu *et al.* [17], which uses the k-nearest neighboring points for normal estimation. While the necessary processing time is constant for our approaches the processing time of *et al.* [17] increases with the number of considered neighboring points. The experiment is done using synthetic data by rendering a mesh model into a depth map of size 640×480 which is then converted into a point cloud of 307200 points. The average processing time for CM is approximately 151 ms and for SDC approximately 28 ms. Therefore, by using the SDC method we can obtain surface normals for full-resolution Kinect-data at around 35 Hz.

B. Normal Estimation Error

In Fig. 5 we compare different normal estimation methods with respect to the angular error between the ground truth

[1] http://www.pointclouds.org

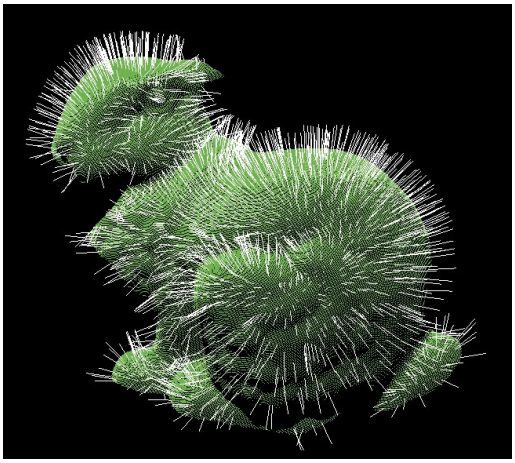


Fig. 6. Surface normals estimated from a partial view of the Stanford bunny.

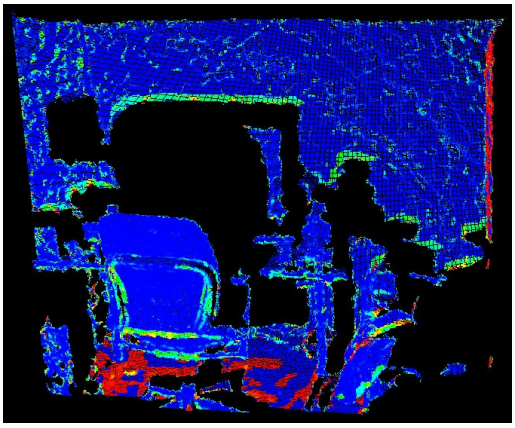


Fig. 7. Visualization of the difference between the normals estimated using our covariance matrix approach and the approach of [17]. For every point in the point cloud we visualize the dot-product between the normals estimated with the different methods, where blue color corresponds to a small and red corresponds to a large difference between the normals.

normal vectors and the estimated ones. As in Sec. IV-A this experiment is done using synthetic data without added noise. While the proposed methods (CM, SDC) behave similarly for different numbers of considered neighbors, the error obtained by the kNN-based approach increases faster when larger neighborhoods are used. An explanation for this is that the kNN-based approach uses a fixed number of neighbors independent of the presence of possible object borders. The proposed methods, on the other hand, adapt the number of considered points according to the specific depth as described in Sec. III-B.3.

Fig. 8 addresses the sensitivity to noise. As we see, our approach based on smoothed depth changes (SDC) has significantly better abilities to handle noise than our approach based on covariance matrices (CM) or the kNN-based approach, which both use the covariance matrix to estimate surface normals.

C. Qualitative Results

In Fig. 9 we show qualitative results of three different scenes. The first row shows the data which we get from a Kinect sensor, that is a color image and a depth map. The second row shows resulting normals that we get without depth-dependent smoothing (left) and with depth-dependent smoothing (right). The normals are color-coded, where each vector component is represented by a different color channel. It is easily visible that the depth-dependent smoothing helps to dramatically reduce the noise in the resulting normal vectors. Fig. 6 shows surface normals estimated from a point cloud of a partial view of the Stanford bunny. In Fig. 7 we visualize the difference between the normals obtained using the method of [17] and our covariance matrix based method.

V. Conclusion

In this paper we presented new methods for fast and robust estimation of surface normals from organized point cloud data. The use of integral images makes it possible to adapt the considered neighborhood size according to depth and object borders without any additional cost in terms of processing speed. We demonstrated that this way of normal estimation enables dense normal estimation at high frame rates and therefore, makes it possible to integrate surface normal information into vision based applications which need to run at a reasonable speed. However, the proposed approach shows two weaknesses: the lack of ability to compute normals for points very close to a depth change and the fact that it smoothes over edges where no depth change is present. Although not considered within this paper, the first problem can be easily addressed by using a different normal estimation method for close to a depth change. For the edge-smoothing problem, a two-pass approach can be applied where in the second pass areas with high variations in surface normal orientations are treated as object borders similar to high changes in depth.

References

[1] K. Klasing and D. Althoff and D. Wollherr and M. Buss, Comparison of surface normal estimation methods for range sensing applications, IEEE International Conference on Robotics and Automation (ICRA), 2009, Kobe, Japan.

[2] H. Gouraud, Continuous Shading of Curved Surfaces, IEEE Trans. Comput., June 1971.

[3] S. Jin and R. R. Lewis and D. West, A comparison of algorithms for vertex normal computation, The Visual Computer, 2005.

[4] N. Max, Weights for computing vertex normals from facet normals, J. Graph. Tools, March 1999.

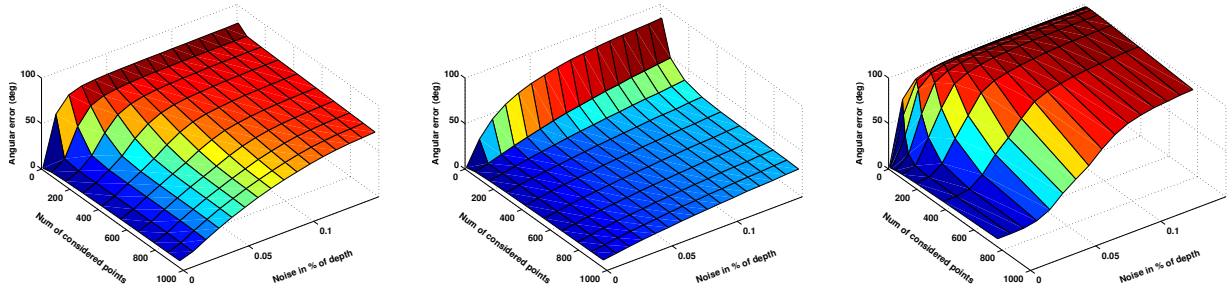
[5] G. Thürmer and C. A. Wüthrich, Computing vertex normals from polygonal facets, J. Graph. Tools, March 1998.

[6] D. Holz and S. Holzer and R. B. Rusu and S. Behnke, Real-Time Plane Segmentation using RGB-D Cameras, Proceedings of the 15th RoboCup International Symposium, July 2011, Istanbul, Turkey.

[7] J. Huang and C. H. Menq, Automatic data segmentation for geometric feature extraction from unorganized 3-D coordinate points, IEEE Transactions on Robotics and Automation, 2001.

[8] H. Hoppe and T. Derose and T. Duchamp and J. J. Mcdonald and W. Stuetzle, Surface reconstruction from unorganized points, Annual Conference on Computer Graphics, 1992.

[9] C. Wang and H. Tanahashi and H. Hirayu and Y. Niwa and K. Yamamoto, Comparison of Local Plane Fitting Methods for Range Data, Computer Vision and Pattern Recognition, 2001.



(a) CM (b) SDC (c) kNN

Fig. 8. Normal estimation error for different normal estimation methods with respect to the size of the smoothing area and the noise in the z-component.

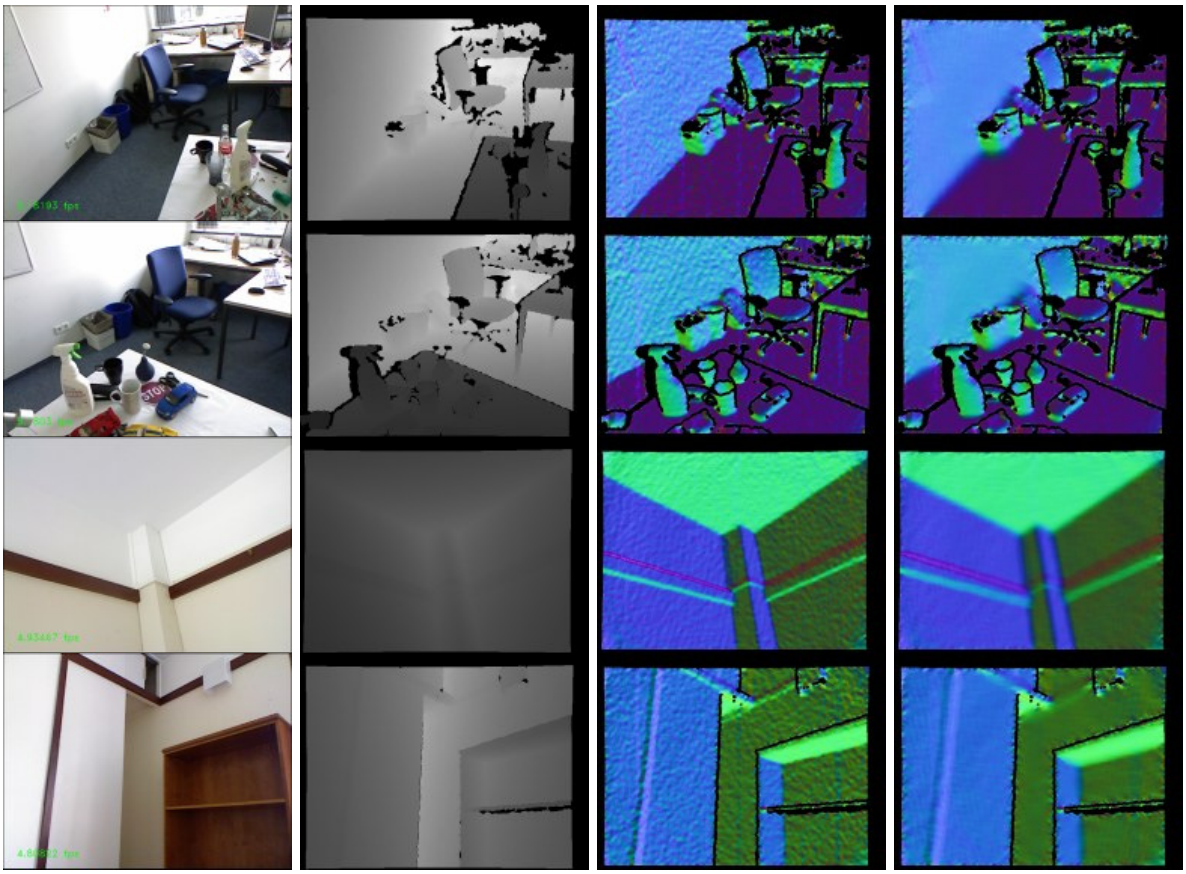


Fig. 9. Comparison of normal estimation with fixed and variable smoothing area size for real scenes. The first two columns show color images together with their corresponding depth maps. The third column shows normals estimated using a smoothing area of fixed size and the fourth column shows results using a variable smoothing size following our proposed approach. The normals are color-coded, where each vector component is represented by a different color channel.

[10] K. Kanatani, Statistical Optimization for Geometric Computation: Theory and Practice, 1996.

[11] M. Gopi and S. Krishnan and C. T. Silva, Surface Reconstruction Based on Lower Dimensional Localized Delaunay Triangulation, Computer Graphics Forum, 2000.

[12] M. Yang and E. Lee, Segmentation of measured point data using a parametric quadric surface approximation, Computer-aided Design, 1999.

[13] D. Ouyang and H. Feng, On the normal vector estimation for point cloud data from smooth surfaces, Computer-aided Design, 2005.

[14] M. Vanco, A Direct Approach for the Segmentation of Unorganized Points and Recognition of Simple Algebraic Surfaces, PhD thesis, University of Technology Chemnitz, 2003.

[15] G. Borgefors, Distance Transformations in Digital Images, Computer Vision, Graphics, and Image Processing, 1986.

[16] F. Porikli and O. Tuzel, Fast Construction of Covariance Matrices for Arbitrary Size Image Windows, IEEE International Conference on Image Processing, 2006.

[17] R. B. Rusu and Z. C. Marton and N. Blodow and M. Dolha and M. Beetz, Towards 3D Point cloud based object maps for household environments, Robot. Auton. Syst., 2008.

[18] R. B. Rusu and S. Cousins, 3D is here: Point Cloud Library (PCL), IEEE International Conference on Robotics and Automation (ICRA), 2011.

APPENDIX

G

LEARNING TO EFFICIENTLY DETECT REPEATABLE INTEREST POINTS IN DEPTH DATA

Own contributions. The idea of using decision trees to mimic the characteristics of an existing 3D interest point detector came from Jamie Shotton and Pushmeet Kohli. My contributions to this work include the ideas for improving the detector performance by creating and learning artificial interest point response maps as well as the design and implementation of the method. The core ideas of the paper were refined in collaboration with all co-authors. All co-authors were involved in the writing of the paper as well as in the evaluation.

Learning to Efficiently Detect Repeatable Interest Points in Depth Data

Stefan Holzer[1,2**], Jamie Shotton[2], and Pushmeet Kohli[2]

[1]Department of Computer Science, CAMP, Technische Universität München (TUM)
holzers@in.tum.de
[2]Microsoft Research Cambridge
Jamie.Shotton@microsoft.com,pkholi@microsoft.com

Abstract. Interest point (IP) detection is an important component of many computer vision methods. While there are a number of methods for detecting IPs in RGB images, modalities such as depth images and range scans have seen relatively little work. In this paper, we approach the IP detection problem from a machine learning viewpoint and formulate it as a regression problem. We learn a regression forest (RF) model that, given an image patch, tells us if there is an IP in the center of the patch. Our RF based method for IP detection allows an easy trade-off between speed and repeatability by adapting the depth and number of trees used for approximating the interest point response maps. The data used for training the RF model is obtained by running state-of-the-art IP detection methods on the depth images. We show further how the IP response map used for training the RF can be specifically designed to increase repeatability by employing 3D models of scenes generated by reconstruction systems such as KinectFusion [1]. Our experiments demonstrate that the use of such data leads to considerably improved IP detection.

1 Introduction

Recent developments in depth sensor technology have enabled the widespread use of inexpensive consumer devices such as the Kinect that is able to capture dense depth data at 30fps. This opens a wide range of opportunities for new applications based on depth information such as real time localization and object or scene reconstruction. Methods proposed for this and other such applications involve estimating the pose of the camera by matching the current depth map against a database of previously seen depth maps.

A common approach for solving the above-mentioned problem is to extract interest points in order to reduce the computational burden necessary to perform this matching task. A recent example where such a technique is of interest is the KinectFusion system [1], which demonstrated that depth sensors paired with efficiently parallelized programs running on high-end graphics cards allow

** This research was performed while Stefan Holzer was an intern at Microsoft Research.

to perform dense frame-to-frame depth tracking and enable accurate 3D reconstruction in real-time. However, memory restrictions force the reconstruction to be limited to small office-like environments. For reconstructing larger environments, *e.g.* complete buildings, the scene has to be split into several pieces, which by themselves, can be handled by the reconstruction system. A common way to connect such pieces is to extract interest points which can then be efficiently queried. However, estimating good interest points in noisy depth data is computationally expensive and therefore, only of limited use in online systems.

We reduce the computational load necessary to compute interest points by efficiently approximating response maps of interest point detectors using regression trees. We demonstrate the effectiveness of our approach by learning to predict the response of an interest point detector based on surface curvature. For training and evaluating the regression trees, we use a data set of depth and color image sequences, obtained using a Kinect sensor. Our data set also includes volumetric reconstructions of the recorded scenes as well as synthetic depth and surface normal maps. These have been created from the reconstructed data given by KinectFusion [1], and are therefore more accurate and include less noise than the raw images. Finally, we introduce a way of creating optimized response maps for interest point estimation and show that, by using these maps for training the regression forest model, we can improve the repeatability of online interest point detection.

2 Related Work

Although there has been a lot of research on interest point extraction in 2D color images, there is surprisingly little work on interest point extraction in dense depth data, as supplied for example by the Kinect depth sensor.

3D Interest Point Extraction. Steder *et al.* [2] used an approach based on the Laplacian-Of-Gaussian method to compute interest points in range images. However, this is computationally very expensive and not suitable for real-time or near-real-time operation. In [3], Steder *et al.* presented an interest point detector which first finds and classifies different kinds of 3D object borders and then locates interest points based on this information. Although efficient, this method is specifically designed for range images, which have different characteristics compared to the depth maps obtained from the Kinect sensor. In [4], Unnikrishnan presented a method for extracting interest points with automatic scale selection in unorganized 3D point clouds. However, this work does not take any view-point related information into account.

Learning-based Interest Point Extraction. A number of learning-based approaches have been proposed for efficient estimation of interest points in color or gray value images. Rosten *et al.* [5] introduced FAST (Features from Accelerated Segment Test), a interest point extraction method which considers all pixels on a circle around the current point to decide whether this point is a feature or not. Although no learning is involved in this approach, it can be seen as a manually

designed decision tree. In [6], Rosten *et al.* extend their work such that the output of FAST is learned using a decision tree. Again, pixels on a circle around the center pixel are used in the tree features. In [7], Rosten *et al.* try to improve the repeatability of the interest point detection. In contrast to the previous approaches they consider more test pixels and use simulated annealing to optimize the decision tree with respect to repeatability and efficiency. The optimization is done by randomly modifying an initially learned tree and by checking whether this modification improves repeatability and efficiency.

The core idea of [8] is to take an existing binary-valued decision algorithm as a black box performing some useful binary decision task and to train the WaldBoost [9] classifier as its emulator. WaldBoost is a greedy learning algorithm which finds a quasi-optimal sequential strategy for a given binary-valued decision problem. It combines the AdaBoost [10] algorithm for feature selection and Wald's sequential probability ratio test (SPRT) for finding the thresholds that are used for making the decision. As examples, they learned the Hessian-Laplace and Kadir-Brady saliency detector. However, the resulting emulators are slow and not able to process images at reasonable frame rates.

Considering more high-level interest points, the face detector of Viola *et al.* [11] can also be seen as a detector of interest points, where the interest points are actually faces. In [12], the authors try to classify surface types, *e.g.* planes or valleys, in range data using perceptron trees. This does not fit exactly into the category of interest point detection, but it identifies regions of interest which can be used for similar tasks as interest points. In [13], Lepetit *et al.* train trees such that they output the probability that the considered interest point corresponds to a specific class. Although not intended by the authors, this might be seen as kind of an interest point detector for every class if applied densely on an image. Similar things have been done in [14] using Ferns. Shotton *et al.* [15] use decision trees to estimate body parts and use artificially rendered humans to train their trees.

However, the above-mentioned approaches neither consider learning the detection of interest points in depth data using regression trees, as we do in Sec. 5, nor creating artificial interest point response maps (for training the regression model) that are optimized to increase the detection performance, as we propose in Sec. 5.

3 High Curvature as Baseline Interest Point Detectors

A common approach [16, 17] used for estimating interest points in 3D data is to consider surface curvature and select points where curvature reaches a maximum. In the following, we explain how curvature can be computed using the normal vectors of the surface surrounding the point of interest. For this, we first describe the process for estimating surface normals and then discuss how these estimates can be used to compute a curvature response.

3.1 Normal Estimation

For surface normal estimation from a depth map, we use a modified version of the approach proposed in [18]. They consider the first order Taylor expansion of the depth function $\mathcal{D}(x)$

$$\mathcal{D}(x + \mathrm{d}x) - \mathcal{D}(x) = \mathrm{d}x^\top \Delta\mathcal{D} + \mathcal{O}(\mathrm{d}x^2), \tag{1}$$

where x is the 2D coordinate in the depth map, $\mathrm{d}x$ is a 2D offset, $\Delta\mathcal{D}$ is a 2D depth gradient, and $\mathcal{O}(\mathrm{d}x^2)$ represents higher order terms. To estimate the value of the gradient $\hat{\Delta\mathcal{D}}$ they use 8 neighboring points around the point of interest to create a stack of equations. A neighbor is only considered if the difference in depth is below a certain threshold α. In our experiments we use $\alpha = 5$ cm. From $\hat{\Delta\mathcal{D}}$, one can then compute three 3D-points as

$$X = \boldsymbol{v}(x)\mathcal{D}(x), \tag{2}$$

$$X_1 = \boldsymbol{v}(x + [1,0]^\top)(\mathcal{D}(x) + [1,0]\hat{\Delta\mathcal{D}}), \tag{3}$$

$$X_2 = \boldsymbol{v}(x + [0,1]^\top)(\mathcal{D}(x) + [0,1]\hat{\Delta\mathcal{D}}), \tag{4}$$

which form two vectors $\mathbf{v}_{X\to X_1}$ and $\mathbf{v}_{X\to X_2}$ between X and X_1 as well as X and X_2. The desired normal can be computed from these two vectors by computing the cross-product $\mathbf{n} = \mathbf{v}_{X\to X_1} \times \mathbf{v}_{X\to X_2}$.

In contrast to [18], we compute the position of the 8 neighboring points based on the inverse of the depth of the point of interest instead of using a fixed position. This means, that we take a bigger image region into account for points further away. We do this, since the depth of 3D points located at further distance to the depth sensor is disturbed by stronger noise and discretization effects than that of 3D points close to the sensor.

3.2 Curvature Response

Having computed the normals, we select all neighboring points within a 15×15 pixel image window and project all their normals onto the plane defined by the normal of the point of interest. From these projected normals, we then compute the covariance matrix C and use its second eigenvalue as curvature response.

4 Learning Interest Point Detectors

In this Section we introduce the learning procedure for our proposed interest point detectors based on decision trees. For this, we first present the dataset we used for training (see Sec. 4.1), then introduce the learning procedure for obtaining regression trees which approximate the interest point response estimation process (see Sec. 4.2), and finally, explain the post-processing steps we use in order to obtain the interest points from the interest point response approximation provided by the regression trees (see Sec. 4.3).

Fig. 1. The shown color images present representative frames of each sequence used for training and testing. The first 23 images represent the training sequences while the last 10 images represent the test sequences.

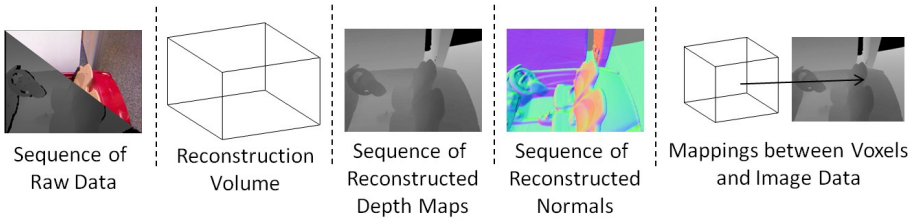


Fig. 2. The different types of data each test and training sequence contains.

4.1 Data Set

In total, we use 33 video sequences captured from the Kinect sensor: 23 sequences for training, and 10 sequences for testing (see Fig. 1). Each video sequence consists of approximately 900 continuously recorded color images and corresponding depth images. Both, the training as well as the test set of sequences depict typical real world scenes expected to be encountered in an office-like scenario, including desks, specialized work spaces, recreational areas as well as meeting rooms. To ensure a fair evaluation, we ensured that the test sequences did not record any of the same volume of 3D world space.

Each of the available sequences contains not only the raw data obtained from the Kinect sensor but also a volumetric reconstruction obtained using the KinectFusion [1] reconstruction system. Fig. 2 shows the different types of data we have available in the sequences. These include the raw depth and image data, the reconstructed volume, synthetic depth and normal maps obtained from the reconstructed volume, and for each frame, a mapping between the volume voxels and the pixels in the images. This volumetric mapping will let us create optimized synthetic interest point response maps as described in Sec. 5.

4.2 Learning using Regression Trees

In the following we describe the process for learning the structure of a binary regression tree that approximates interest point responses from depth maps. Every non-leaf node of the tree uses a depth comparison between two sample positions relative to the point under consideration to decide whether to follow the left or right tree branch emanating down from the node. Every leaf node of the tree stores an interest point response value which is the mean of the responses of all the training pixels that reached that leaf node. This leads to the following parameterization of a node:

$$n = \begin{cases} (f, n_l, n_r) & \text{if } n \text{ is node} \\ (m_c) & \text{if } n \text{ is leaf} \end{cases}, \tag{5}$$

where f is a feature, n_l and n_r are the left and right child node respectively, and m_c is the mean interest point response of the training examples that fall into the corresponding node. The feature f implements the depth comparison between two sample positions and is defined as

$$f = (x_1, y_1, x_2, y_2, t), \tag{6}$$

where $p_1 = (x_1, y_1)^\top$ and $p_2 = (x_2, y_2)^\top$ are the sample positions for the depth values and t is a threshold which is applied on the depth difference $\mathcal{D}(p_1) - \mathcal{D}(p_2)$. The sample points are placed within a $w_D \times h_D$ window. The first sample point p_1 is either placed in the center or randomly within this window around the center point, where both of these possibilities have equal chance. The second sample point p_2 is placed randomly within the window. In our experiments, we use $w_D = h_D = 41$ at a depth of 1 meter and we scale the window according to the depth. The thresholds as well as the offsets of the feature sample positions are selected automatically during the training.

During the learning, we select for every node n_i a feature $f_j \in \mathcal{F}$ and a threshold t_k which best separate the set $\mathcal{E}$ of examples. We consider an example as a quintuple e with

$$e = (q, r, x, y, c), \tag{7}$$

where q is the index of training sequence, r the index of the frame within this sequence, $p = (x, y)^\top$ is the location of the example within the image r of sequence q, and c is the corresponding interest point response value. A feature f_j separates the set $\mathcal{E}$ best if it minimizes

$$\epsilon = v_l \frac{N_l}{N_l + N_r} + v_r \frac{N_r}{N_l + N_r}, \tag{8}$$

where v_r and v_l are the variances of the examples that fall into the left respectively right child node, and N_l and N_r are the corresponding numbers of examples. This variance reduction objective follows the standard entropy minimization strategy used for regression tree learning [19]. In our experiments we sample 1000 feature tests *ie.* $|\mathcal{F}| = 1000$ and select 10 thresholds which are

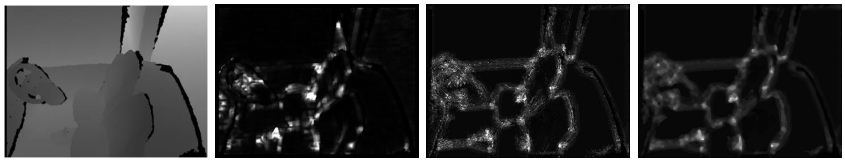


Fig. 3. Left to right: exemplary depth map with corresponding surface curvature map, the unfiltered response obtained from regression trees as well as its median filtered counterpart.

distributed over the range of the possible feature results. The example set $\mathcal{E}$ is created by selecting every second pixel in x and y from approximately 1000 images taken from our training sequences. However, we use an example only if curvature information is available and the depth is not larger than 4 meters.

At test time, for every image position, we follow the path down from the root to the leaf, and return as the result, the response m_c corresponding to the reached leaf. In case of multiple trees, the results of the individual trees are combined together by averaging them.

4.3 Post-processing

Fig. 3 shows the interest point response map approximation obtained from a depth map using a regression tree. As one can easily see it contains a significant amount of salt-and-pepper-like noise which has to be filtered out in order to get a stable response over multiple frames. Therefore, we apply a median filter of size 5×5 and then a Gaussian filter with sigma 3 in order to get distinctive peaks. Finally, the interest points are extracted as maxima of the resulting map.

5 Designing Optimal Interest Point Detectors

In Section , we have shown how random forests can be trained to predict the response of any existing curvature-based interest point detector. This process however, does not in itself lead to better interest points. In this section we show how to compute desirable interest points for training the random forest that may not be obtained from existing methods. For this, we first discuss the properties desired from a good interest point and then show how to compute such response maps from 3D reconstructions of the scenes.

5.1 Optimality Criteria

In the following we define interest points as a set of points in the image coordinate system which fulfil certain specific properties. We will use these properties to select an evaluation criteria. The following list of properties are desirable for a set of scene elements in order to be useful as interest points:

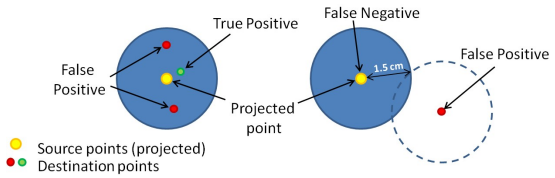


Fig. 4. Illustration of IP repeatability. The evaluation always considers a source frame and a destination frame. All the extracted interest points in the source frame are projected into the destination frame using the provided reconstruction. For every projected source point we search for corresponding destination points within a radius of 1.5 cm. Only the closest match within this radius is considered as a true positive and the others are considered as false positives. If no point is present, we count it as a false negative. If a destination point is not assigned to any projected source point then it is considered a false positive.

- **Sparseness**: there should be only a small number of points in the scene.
- **Repeatability**: the points should be detected in all views of the scene.
- **Distinctiveness**: the area around an interest point should be unique.
- **Efficiency**: points could be estimated efficiently.

While analyzing the criteria, one sees that *sparseness* is hard to evaluate since it is usually defined using a threshold. *Distinctiveness* is highly dependent on the matching method, especially on the construction of the descriptor and thus is difficult to evaluate objectively. *Repeatability* of an interest point, however, is easy to measure since we have access to the reconstruction of the scene depicted in every training/test sequence. This enables us to propagate interest points from one frame to any other frame of the sequence and check the consistency of results.

We also evaluate *repeatability* with respect to the number of extracted interest points (see Sec. 6.1 and 6.2), which provides us with information about the detection performance with respect to *sparseness*. Furthermore, we evaluate *repeatability* with respect to the number and depth of trees in Sec. 6.3 and 6.4. Since the *efficiency* of the presented approach depends on the number and depth of trees used for detecting interest points, this allows us to quantify the trade-off between *efficiency* and *repeatability*. The influence of the tree parameters on the interest point detection performance is discussed in detail in Sec. 6.5.

As a measure for repeatability we use the number of true-positives, false-positives and false-negatives. The estimation of these numbers is described in Fig. 4. Repeatability is computed for a 5 frame difference as well as a 40 frame difference between compared images. This was done to compare repeatability for both, small- as well as wide-baseline matching applications.

5.2 Creating a Response Tailored for High Repeatability

In order to create a response which leads to highly repeatable interest points we make use of the mapping between pixels of the input depth maps and the vox-

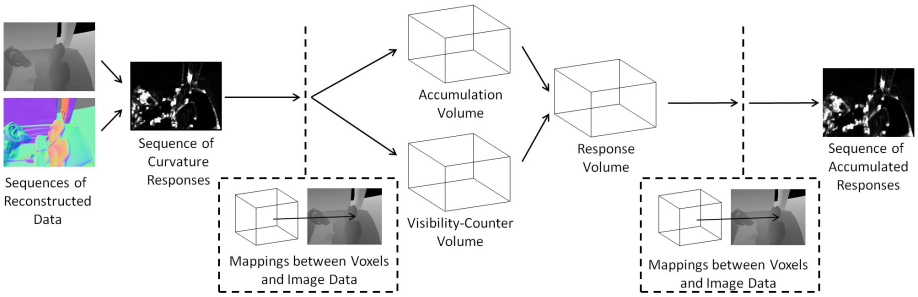


Fig. 5. Illustration of the synthetic response computation.

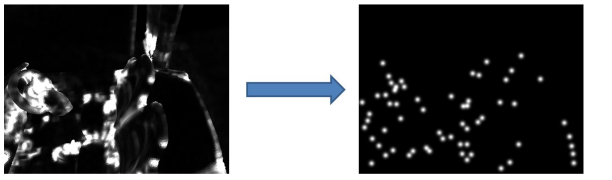


Fig. 6. Computing artificial response maps from accumulated responses.

els of the reconstruction volume, as shown in Fig. 5. For this, we first compute the curvature response for every image of each sequence based on the synthetic surface normals, which are obtained from the reconstruction. These curvature responses are then accumulated in an accumulation volume. A second accumulator volume is used to count how often a voxel was visible in one of the images of the sequence. Finally, we project the resulting accumulation volume into each frame of the sequence and select the best N interest points from the rendered image. Note that this procedure also correctly handles occlusions. The final response map is then created by creating a Gaussian response for each of the selected interest points, as shown in Fig. 6. This artificial response gives high responses for repeatable points and is then approximated using a regression tree. Results for this are given in Sec. 6.2.

6 Results

In the following we evaluate our proposed approach against the baseline method using the test set introduced in Sec. 4.1. As discussed in Sec. 5.1, the evaluation is based on the basis of repeatability of the obtained interest points. If not otherwise mentioned, the evaluations are done with respect to the number of interest points obtained from the detectors. The number of IPs is controlled by changing the threshold which is applied on the response map created by the specific IP detector. Selecting a high threshold results in only a few, but very stable IPs, while choosing a lower threshold increases the number of points. A higher number of points, on the other hand, generally leads to a worse true-positive rate.

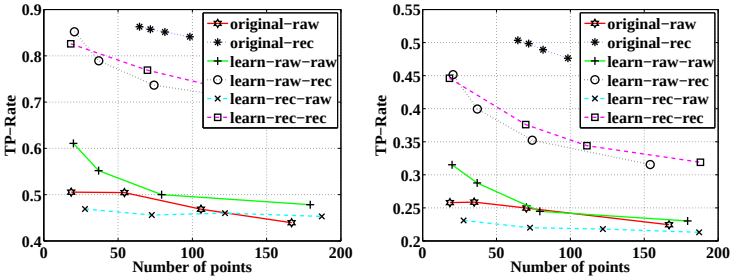


Fig. 7. Comparison of repeatability of detected interest points with respect to different training and test data characteristics. True-positive rate with image pairs with 5 frames difference **(left)** and 40 frames difference **(right)**.

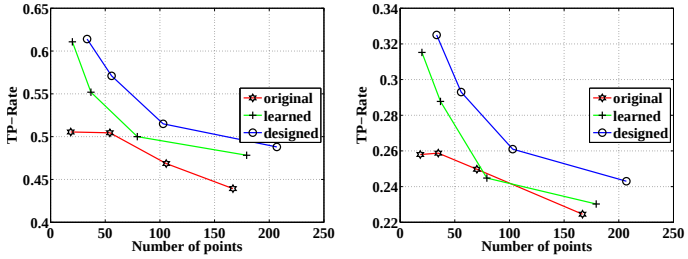


Fig. 8. Comparison of the interest point detector trained using designed response maps (described in Sec. 5 and Fig. 6) with the original curvature-based approach for a 5 frames difference **(left)** and a 40 frames difference **(right)**.

The drop in the true-positive rate with increasing number of obtained interest points is explained by the fact that a lower threshold applied on the response map results in more but less reliable points. The evaluation is conducted using a notebook with a 2.26GHz Intel(R) Core(TM)2 Quad CPU and 4 GB of RAM, where only one core is used for the computation.

6.1 Learning Interest Point Detector Responses

Fig. 7 compares the repeatability of results obtained from learned and hand-coded detectors on both, raw and reconstructed data[1]. The results of the curvature-based interest point detector applied on the raw Kinect depth map are annotated by *original-raw*, while those obtained by applying it on the rendered depth map (obtained from the 3D model KinectFusion system) are annotated by *original-rec*. As expected, *original-rec* results are better than *original-raw* because the rendered depth maps have less noise, are smoother, and do not suffer from missing data.

[1] This section deals only with the input data. The use of designed response maps (the training labels) as described in Sec. 5 will be analyzed below in Sec. 6.2.

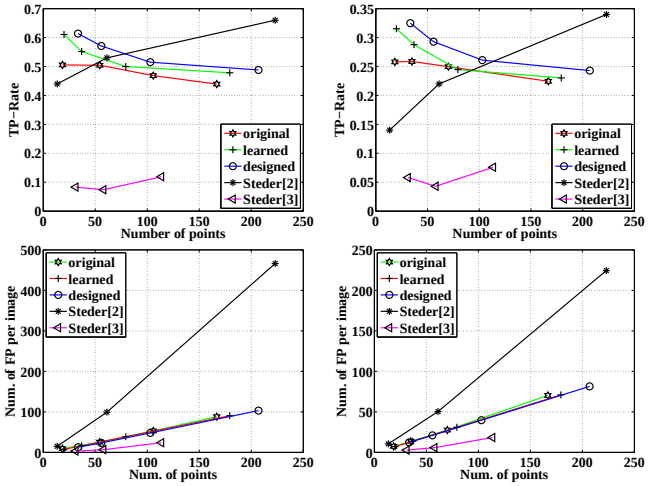


Fig. 9. Comparison of the interest point detector trained using designed response maps (described in Sec. 5 and Fig. 6) with the original curvature-based approach, its learned counterpart, as well as the detectors of Steder *et al.* [2, 3]. Results are for a 5 frames difference **(left)** and a 40 frames difference **(right)**.

We now analyze the effect of using raw depth (*learn-raw-...*), as well as rendered depth (*learn-rec-...*) for training the regression forest. These two different cases are then evaluated on raw depth data (*learn-...-raw*) as well as on reconstructed depth data (*learn-...-rec*). This leads to a total of four different possibilities. Note that the curvature response used for training the regression trees is always computed from the reconstructed data given by the KinectFusion system. We are changing here only the data on which the tree features are evaluated in order to decide on the split. As one would expect, training the regression tree using the data type it is later applied to gives superior results compared to training it from a different type of data.

Comparing the IP repeatability of the original surface curvature estimation approach (*original-raw*) with our proposed approximation using regression trees (*learn-raw-raw*) in Fig. 7, one can see that our approach is not only able to approximate the behaviour of the curvature-based IP detector with respect to repeatability, but it even gives better results. The improvement can be explained by the fact that we use the curvature estimated from the reconstructed data instead of the raw data for training our regression trees.

6.2 Learning Designed Response Maps

In Fig. 8 we compare the results obtained by a regression forest trained using the artificial IP response maps (which we introduced in Sec. 5 and Fig. 6) with the results of the original interest point detector based on curvature as well as the regression forest trained using its IP responses. Our results show that

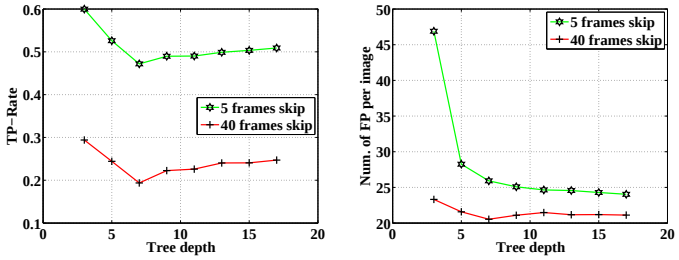


Fig. 10. Evaluation of the influence of the depth of trees on the true-positive rate and the number of false positives per image.

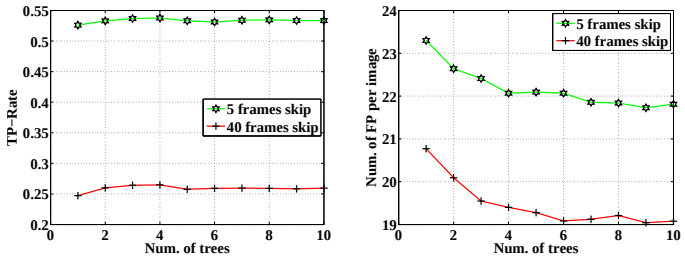


Fig. 11. Evaluation of the influence of the number of trees on the true-positive rate and the number of false positives per image.

it is possible to train a regression forest to output such desired IPs resulting in better repeatability performance compared to the curvature-based interest point detector.

In Fig. 9 we additionally compare to [2] and [3]. Although [2] tends to have a higher true-positive rate for large numbers of points, it also shows a higher number of false-positives. The approach presented in [3] on the other hand has a very low number of false positives, but also a far worse true-positive rate.

6.3 Depth of Trees

Fig. 10 evaluates the influence of the depth of a regression tree on the performance of the interest point detector. While the true-positive rate drops until a depth of 7 and then increases slowly with increasing depth, the number of false positives per image is high for trees with a depth less than 7.

6.4 Number of Trees

As Fig. 11 indicates, the number of trees has only a small effect on the resulting detection performance. While the true-positive-rate stays almost constant, the number of false-positives per image drops only by a value of less than 2 when going from a single tree up to ten trees. This can be explained by the applied

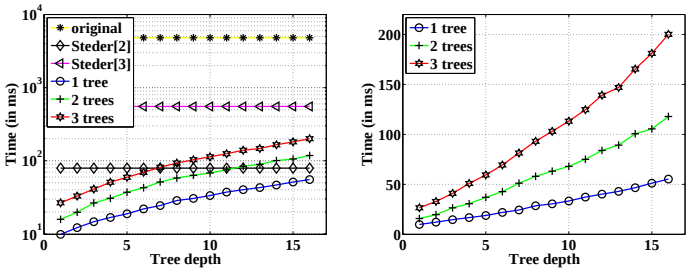


Fig. 12. Timings for interest point detection. **Left:** comparison of the original curvature-based approach with learned interest point detectors using different numbers of trees and tree depths as well as the detectors of Steder *et al.* [2, 3]. **Right:** zoomed-in version where we only the learned approaches are compared. For our approach we used the regression trees learned from the designed response maps (see Sec. 5), including the post-processing step (see Sec. 4.3).

filters (see Sec. 4.3), which are used to remove different types of noise from the obtained response maps, since adding more trees to the evaluation process has a similar effect on the response. Note, however, applying the described filters is computationally more efficient than using a large number of trees (see Sec. 6.5).

6.5 Processing Time

In Fig. 12 we compare the computation time of the original curvature-based IP detector with our random forest based detector using different number of regression trees with different depths and with the detectors of Steder *et al.* [2, 3]. As one can see, the processing time for our proposed approach using regression trees is much less than for the original approach based on surface normals. It is also faster than the approach of Steder *et al.* [3]. The single tree or low depth forest variants of our approach are faster than [2] also. Note that for the processing time of the original approach as well as for the approaches of Steder *et al.* [2, 3] we estimated only a single value since it does not depend on the depth of a tree. We visualize it as a line for better comparison. For the evaluation of [3], we used an open-source implementation which is available in the Point Cloud Library[2].

7 Conclusion

In this paper, we presented a novel regression forest based approach to efficiently detect interest points in depth maps. Our experimental results show that a curvature-based interest point detector can be approximated using the regression forest model. Furthermore, we show that by using a reconstruction of the available scenes we can create an improved interest point detector which gives interest points with higher repeatability.

[2] `http://www.pointclouds.org`.

Acknowledgements. The authors would like to thank Rasmus Kyng, Shahram Izadi, David Kim, Dave Molyneaux, and Otmar Hilliges for the inspiring discussions and for help in obtaining the training and test data.

References

1. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohli, P., Shotton, J., Hodges, S., Fitzgibbon, A.: Kinectfusion: Real-time dense surface mapping and tracking. In: ISMAR. (2011)
2. Steder, B., Grisetti, G., Burgard, W.: Robust place recognition for 3D range data based on point features. In: ICRA. (2010)
3. Steder, B., Rusu, R.B., Konolige, K., Burgard, W.: Point feature extraction on 3D range scans taking into account object boundaries. In: ICRA. (2011)
4. Unnikrishnan, R.: Statistical approaches to multi-scale point cloud processing. (2008)
5. Rosten, E., Drummond, T.: Fusing points and lines for high performance tracking. In: ICCV. (2005)
6. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: ECCV. (2006)
7. Rosten, E., Porter, R., Drummond, T.: Faster and better: A machine learning approach to corner detection. PAMI (2010)
8. Šochman, J., Matas, J.: Learning a fast emulator of a binary decision process. In: Proceedings of the 8th Asian Conference on Computer vision - Volume Part II. (2007) 236–245
9. Sochman, J., Matas, J.: Waldboost - learning for time constrained sequential detection. In: CVPR. (2005)
10. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. In: Machine Learning. (1999) 80–91
11. Viola, P., Jones, M.: Fast and robust classification using asymmetric adaboost and a detector cascade. In: Advances in Neural Information Processing System 14, MIT Press (2001) 1311–1318
12. Foresti, G.: Invariant feature extraction and neural trees for range surface classification. IEEE Transactions on Systems, Man, and Cybernetics (2002)
13. Lepetit, V., Fua, P.: Keypoint recognition using randomized trees. PAMI (2006)
14. Özuysal, M., Calonder, M., Lepetit, V., Fua, P.: Fast keypoint recognition using random ferns. PAMI (2010)
15. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from a single depth image. In: CVPR. (2011)
16. Stückler, J., Behnke, S.: Interest point detection in depth images through scale-space surface analysis. In: ICRA. (2011)
17. Gelfand, N., Mitra, N.J., Guibas, L.J., Pottmann, H.: Robust global registration. In: Eurographics Symposium on Geometry Processing. (2005)
18. Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., Lepetit, V.: Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In: ICCV. (2011)
19. Criminisi, A., Shotton, J., Konukoglu, E.: Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. In: Foundations and Trends in Computer Graphics and Vision. (2012)

LIST OF FIGURES

AUTHORED AND CO-AUTHORED PUBLICATIONS

[Bohren et al., 2011] Bohren, J., Rusu, R. B., Jones, E. G., Marder-Eppstein, E., Pantofaru, C., Wise, M., Moesenlechner, L., Meeussen, W., and Holzer, S. (2011). Towards autonomous robotic butlers: Lessons learned with the pr2. In *IEEE International Conference on Robotics and Automation*, Shanghai, China.

[Hinterstoisser et al., 2011] Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V. (2011). Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *IEEE International Conference on Computer Vision*, Barcelona, Spain.

[Hinterstoisser et al., 2012] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., , and Navab, N. (2012). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision*, Daejeon, Korea.

[Holz et al., 2011] Holz, D., Holzer, S., Rusu, R. B., and Behnke, S. (2011). Real-Time Plane Segmentation using RGB-D Cameras. In *Proceedings of the 15th RoboCup International Symposium*, Istanbul, Turkey.

[Holzer et al., 2009] Holzer, S., Hinterstoisser, S., Ilic, S., and Navab, N. (2009). Distance transform templates for object detection and pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, Miami Beach (Florida), USA.

[Holzer et al., 2010] Holzer, S., Ilic, S., and Navab, N. (2010). Adaptive linear predictors for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco (California), USA.

[Holzer et al., 2013] Holzer, S., Ilic, S., and Navab, N. (2013). Multi-layer adaptive linear predictors for real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Holzer et al., 2012a] Holzer, S., Ilic, S., Tan, D., and Navab, N. (2012a). Efficient learning of linear predictors using dimensionality reduction. In *Asian Conference on Computer Vision*, Daejeon, Korea.

[Holzer et al., 2014] Holzer, S., Ilic, S., Tan, D., Pollefeys, M., and Navab, N. (2014). Efficient learning of linear predictors for template tracking. *International Journal of Computer Vision*.

[Holzer et al., 2012b] Holzer, S., Pollefeys, M., Ilic, S., Tan, D. J., and Navab, N. (2012b). Online Learning of Linear Predictors for Real-Time Tracking. In *European Conference on Computer Vision*, Firenze, Italy.

[Holzer et al., 2012c] Holzer, S., Rusu, R. B., Dixon, M., Gedikli, S., and Navab, N. (2012c). Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vila Moura, Algarve, Portugal.

[Holzer et al., 2012d] Holzer, S., Shotton, J., and Kohli, P. (2012d). Learning to Efficiently Detect Repeatable Interest Points in Depth Data. In *European Conference on Computer Vision*, Firenze, Italy.

REFERENCES

[1] Amit, Y., Geman, D., and Fan, X. A coarse-to-fine strategy for multiclass shape detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 12 (2004), 1606–1621.

[2] Baker, S., and Matthews, I. Equivalence and efficiency of image alignment algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition* (Kauai, HI, USA, December 2001).

[3] Baker, S., and Matthews, I. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision 56*, 3 (2004), 221–255.

[4] Ballard, D. H. Generalizing the hough transform to detect arbitrary shapes. In *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, M. A. Fischler and O. Firschein, Eds. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987, pp. 714–725.

[5] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. Speeded-up robust features (surf). *Computer Vision and Image Understanding 110*, 3 (June 2008), 346–359.

[6] Benhimane, S., and Malis, E. Homography-based 2d visual tracking and servoing. *International Journal of Robotics Research 26*, 7 (July 2007), 661–676.

[7] Borgefors, G. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 10*, 6 (Nov. 1988), 849–865.

[8] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. Brief: Binary robust independent elementary features. In *European Conference on Computer Vision* (Heraklion, Crete, Greece, September 2010).

[9] Canny, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*, 6 (June 1986), 679–698.

[10] Cascia, M., Sclaroff, S., and Athitsos, V. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 4 (2000), 322–336.

[11] Dalal, N., and Triggs, B. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (San Diego, CA, USA, 2005).

[12] Dame, A., and Marchand, E. Accurate real-time tracking using mutual information. In *IEEE International Symposium on Mixed and Augmented Reality* (2010).

[13] Dellaert, F., and Collins, R. Fast image-based tracking by selective pixel integration. In *ICCV Workshop on Frame-Rate Vision* (1999).

[14] Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (June 1981), 381–395.

[15] Foresti, G. Invariant feature extraction and neural trees for range surface classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3 (June 2002).

[16] Gavrila, D., and Philomin, V. Real-time object detection for "smart" vehicles. In *International Conference on Computer Vision* (1999), pp. 87–93.

[17] Gopi, M., Krishnan, S., and Silva, C. T. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum 19* (2000), 467–478.

[18] Gouraud, H. Continuous shading of curved surfaces. *IEEE Transactions on Computers 20* (June 1971), 623–629.

[19] Grabner, H., Leistner, C., and Bischof, H. Semi-supervised on-line boosting for robust tracking. In *European Conference on Computer Vision* (Marseille, France, October 2008).

[20] Grässl, C., Zinsser, T., and Niemann, H. Illumination insensitive template matching with hyperplanes. In *Proceedings of Pattern Recognition: 25th DAGM Symposium* (Magdeburg, Germany, September 2003).

[21] Grässl, C., Zinsser, T., and Niemann, H. Efficient hyperplane tracking by intelligent region selection. In *Image Analysis and Interpretation* (Lake Tahoe, Nevada, USA, March 2004).

[22] Hager, G. D., and Belhumeur, P. N. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 10 (1998), 1025–1039.

[23] Harris, C., and Stephens, M. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference* (1988), pp. 147–151.

[24] Henderson, H. V., and Searle, S. R. On deriving the inverse of a sum of matrices. *SIAM Review 23*, 1 (January 1981), 53–60.

[25] Hinterstoisser, S., Benhimane, S., Navab, N., Fua, P., and Lepetit, V. Online learning of patch perspective rectification for efficient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (Anchorage, Alaska, USA, June 2008).

[26] Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *International Conference on Computer Vision* (Barcelona, Spain, November 2011).

[27] Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., and Lepetit, V. Real-time learning of accurate patch rectification. In *IEEE Conference on Computer Vision and Pattern Recognition* (Miami Beach (Florida), USA, June 2009).

[28] Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., and Navab, N. Dominant orientation templates for real-time detection of texture-less objects. In *IEEE Conference on Computer Vision and Pattern Recognition* (San Francisco (California), USA, June 2010).

[29] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., , and Navab, N. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision* (Daejeon, Korea, November 2012).

[30] Holzer, S., Hinterstoisser, S., Ilic, S., and Navab, N. Distance transform templates for object detection and pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition* (Miami Beach (Florida), USA, June 2009).

[31] Holzer, S., Ilic, S., and Navab, N. Adaptive linear predictors for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition* (San Francisco (California), USA, June 2010).

[32] Holzer, S., Ilic, S., and Navab, N. Multilayer adaptive linear predictors for real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 1 (2013), 105–117.

[33] Holzer, S., Ilic, S., Tan, D., and Navab, N. Efficient learning of linear predictors using dimensionality reduction. In *Asian Conference on Computer Vision* (Daejeon, Korea, November 2012).

[34] Holzer, S., Ilic, S., Tan, D., Pollefeys, M., and Navab, N. Efficient learning of linear predictors for template tracking. *International Journal of Computer Vision* (2014).

[35] Holzer, S., Pollefeys, M., Ilic, S., Tan, D. J., and Navab, N. Online Learning of Linear Predictors for Real-Time Tracking. In *12th European Conference on Computer Vision (ECCV)* (Firenze, Italy, October 2012).

[36] Holzer, S., Rusu, R. B., Dixon, M., Gedikli, S., and Navab, N. Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images. In *International Conference on Intelligent Robots and Systems* (Vila Moura, Algarve, Portugal, October 2012).

[37] Holzer, S., Shotton, J., and Kohli, P. Learning to Efficiently Detect Repeatable Interest Points in Depth Data. In *European Conference on Computer Vision* (Firenze, Italy, October 2012).

[38] Hoppe, H., Derose, T., Duchamp, T., Mcdonald, J. J., and Stuetzle, W. Surface reconstruction from unorganized points. In *ACM SIGGRAPH* (Chicago, USA, July 1992).

[39] Huang, C., Ai, H., Li, Y., and Lao, S. Vector boosting for rotation invariant multi-view face detection. In *International Conference on Computer Vision* (Bejing, China, October 2005).

[40] Huang, J., and Menq, C.-H. Automatic data segmentation for geometric feature extraction from unorganized 3-d coordinate points. *IEEE Transactions on Robotics and Automation 17* (2001), 268–279.

[41] Huttenlocher, D. P., Klanderman, G. A., and Rucklidge, W. A. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 9 (Sept. 1993), 850–863.

[42] Jin, S., Lewis, R. R., and West, D. A comparison of algorithms for vertex normal computation. *The Visual Computer 21* (2005), 71–82.

[43] Jurie, F., and Dhome, M. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 7 (2002), 996–1000.

[44] Jurie, F., and Dhome, M. Real time robust template matching. In *British Machine Vision Conference* (Cardiff, UK, September 2002).

[45] Kalal, Z., Mikolajczyk, K., and Matas, J. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 7 (2012), 1409–1422.

[46] Kanatani, K. Statistical optimization for geometric computation: Theory and practice.

[47] Klasing, K., Althoff, D., Wollherr, D., and Buss, M. Comparison of surface normal estimation methods for range sensing applications. In *International Conference on Robotics and Automation* (Kobe, Japan, May 2009).

[48] Klein, G., and Murray, D. Parallel tracking and mapping for small ar workspaces. In *International Symposium on Mixed and Augmented Reality* (Nara, Japan, November 2007).

[49] Klein, G., and Murray, D. Improving the agility of keyframe-based slam. In *European Conference on Computer Vision*. Marseille, France, October 2008.

[50] Lepetit, V., and Fua, P. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 9 (2006), 1465–1479.

[51] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (Nov. 2004), 91–110.

[52] Lucas, B., and Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conference on Artificial Intelligence* (1981), vol. 2, pp. 674–679.

[53] Malis, E. Improving vision-based control using efficient second-order minimization techniques. In *International Conference on Robotics and Automation* (Barcelona, Spain, April 2004).

[54] Matas, J., Zimmermann, K., Svoboda, T., and Hilton, A. Learning efficient linear predictors for motion estimation. In *Computer Vision, Graphics and Image Processing* (2006), vol. 4338, pp. 445–456.

[55] Max, N. Weights for computing vertex normals from facet normals. *Journal of Graphics Tools 4* (1999), 1–6.

[56] Mayol, W. W., and Murray, D. W. Tracking with general regression. *Journal of Machine Vision and Applications 19* (2008), 65–72.

[57] Mears, B., Sevilla-Lara, L., and Learned Miller, E. Distribution fields with adaptive kernels for large displacement image alignment. In *British Machine Vision Conference* (Bristol, UK, September 2013).

[58] Olson, C., and Huttenlocher, D. Automatic target recognition by matching oriented edge pixels. *IEEE Transactions on Image Processing 6*, 1 (Jan 1997), 103–113.

[59] Ouyang, D., and yung Feng, H. On the normal vector estimation for point cloud data from smooth surfaces. *Computer-aided Design 37* (2005), 1071–1079.

[60] Özuysal, M., Calonder, M., Lepetit, V., and Fua, P. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 3 (2010), 448–461.

[61] Özuysal, M., Fua, P., and Lepetit, V. Fast Keypoint Recognition in Ten Lines of Code. In *IEEE Conference on Computer Vision and Pattern Recognition* (Minneapolis, Minnesota, USA, June 2007).

[62] Özuysal, M., Lepetit, V., and Fua, P. Pose estimation for category specific multiview object localization. In *IEEE Conference on Computer Vision and Pattern Recognition* (Miami Beach (Florida), USA, June 2009).

[63] Parisot, P., Thiesse, B., and Charvillat, V. Selection of reliable features subsets for appearance-based tracking. *Signal-Image Technologies and Internet-Based System* (December 2007), 891–898.

[64] Richa, R., Sznitman, R., Taylor, R., and Hager, G. Visual tracking using the sum of conditional variance. In *International Conference on Intelligent Robots and Systems* (Seoul, Korea, October 2011).

[65] Rosten, E., and Drummond, T. Fusing points and lines for high performance tracking. In *International Conference on Computer Vision* (Bejing, China, October 2005).

[66] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *European Conference on Computer Vision* (Graz, Austria, May 2006).

[67] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *European Conference on Computer Vision* (Graz, Austria, May 2006).

[68] Rosten, E., Porter, R., and Drummond, T. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32* (2010), 105–119.

[69] Rucklidge, W. Efficiently locating objects using the hausdorff distance. *International Journal of Computer Vision 24*, 3 (1997), 251–270.

[70] Schapire, R. E., and Singer, Y. Improved boosting algorithms using confidence-rated predictions. In *Machine Learning* (1999), pp. 80–91.

[71] Serre, T., and Riesenhuber, M. Realistic modeling of simple and complex cell tuning in the hmax model, and implications for invariant object recognition in cortex, 2004.

[72] Sevilla-Lara, L. Distribution fields for tracking. In *IEEE Conference on Computer Vision and Pattern Recognition* (Providence, RI, USA, June 2012).

[73] Shi, J., and Tomasi, C. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition* (Seattle, Washington, USA, June 1994).

[74] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from a single depth image. In *IEEE Conference on Computer Vision and Pattern Recognition* (Colorado Springs, CO, USA, June 2011).

[75] Shum, H.-Y., and Szeliski, R. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision 36*, 2 (2000), 101–130.

[76] Sochman, J., and Matas, J. Waldboost - learning for time constrained sequential detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (San Diego, CA, USA, june 2005).

[77] Steder, B., Grisetti, G., and Burgard, W. Robust place recognition for 3D range data based on point features. In *International Conference on Robotics and Automation* (Anchorage, Alaska, USA, May 2010).

[78] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W. Point feature extraction on 3D range scans taking into account object boundaries. In *International Conference on Robotics and Automation* (Shanghai, China, May 2011).

[79] Szeliski, R. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision 2*, 1 (Jan. 2006), 1–104.

[80] Thürmer, G., and Wüthrich, C. A. Computing vertex normals from polygonal facets. *Journal of Graphics Tools 3* (1998), 43–46.

[81] Unnikrishnan, R. Statistical approaches to multi-scale point cloud processing.

[82] Vacchetti, L., Lepetit, V., and Fua, P. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 10 (2004), 1385–1391.

[83] Vanco, M. A direct approach for the segmentation of unorganized points and recognition of simple algebraic surfaces. *PhD thesis, University of Technology Chemnitz* (2003).

[84] Viola, M., Jones, M. J., and Viola, P. Fast multi-view face detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (Madison, Wisconsin, USA, June 2003).

[85] Viola, P., and Jones, M. Fast and robust classification using asymmetric adaboost and a detector cascade. In *Neural Information Processing Systems* (Vancouver, British Columbia, Canada, December 2001).

[86] Šochman, J., and Matas, J. Learning a fast emulator of a binary decision process. In *Asian Conference on Computer Vision* (Tokyo, Japan, November 2007).

[87] Wang, C., Tanahashi, H., Hirayu, H., Niwa, Y., and Yamamoto, K. Comparison of local plane fitting methods for range data. In *IEEE Conference on Computer Vision and Pattern Recognition* (Kauai, HI, USA, December 2001).

[88] Wu, C., Agarwal, S., Curless, B., and Seitz, S. M. Multicore bundle adjustment. In *IEEE Conference on Computer Vision and Pattern Recognition* (Colorado Springs, CO, USA, June 2011).

[89] Yang, M., and Lee, E. Segmentation of measured point data using a parametric quadric surface approximation. *Computer-aided Design 31* (1999), 449–457.

[90] Zimmermann, K., Matas, J., and Svoboda, T. Tracking by an optimal sequence of linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 4 (2009), 677–692.