TECHNISCHE UNIVERSITÄT MÜNCHEN

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

Learn to Track: From Images to 3D Data

David Joseph Tan

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:	Prof. Dr.-Ing. Darius Burschka
Prüfer der Dissertation:	
1.	Prof. Dr. Nassir Navab
2.	Prof. Dr. Andrew Davison

Die Dissertation wurde am 29.09.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.03.2017 angenommen.

Abstract

Object pose estimation aims at determining the relation between the camera and the object of interest. For rigid objects, the pose is defined through three degrees of freedom for rotation and three for translation. Across different applications in robotic perception, augmented reality and human-computer interaction, achieving a real-time robust object pose estimation is an integral component to reach their goals. This thesis focuses on object pose estimation using a temporal tracker from 3D data. The objective is not only to achieve robustness through public benchmarks, but also to integrate into a wide-range of real-world applications.

A temporal tracker is a frame-to-frame algorithm that relays the object's pose from the previous frame and updates it for the current frame. When we investigate the use case of temporal trackers, they are frequently applied on reconstruction frameworks to track a single object or a static scene. However, when tracking several independently moving objects present in the scene, most frameworks rely on tracking-by-detection. Conversely, tracking-by-detection methods look at each frame independently. For each frame, they localize the object on the image and estimate the pose.

On one hand, the limitation of a temporal tracker is the requirement to initialize on the first frame either manually or by a detector. On the other, the benefit of a temporal tracker is its efficiency because it simplifies the problem to updating the pose between two consecutive frames. Intuitively, the combination of both approaches, where the temporal tracker is initialized by the detector and then temporally track the objects, is a natural procedure. But, to be robust against clutter, occlusions and fast movements, most temporal trackers require at least 100 ms per frame with a GPU optimization. Since independent temporal trackers are triggered for each object in the scene, it also scales linearly to the number of objects. In effect, tracking-by-detection approaches overcome the efficiency of the temporal trackers with about five objects in the scene. As a consequence, the temporal tracker is not commonly used because it cannot uphold to its promised efficiency.

Keeping this in mind, we formulate a learning-based 3D temporal tracker that learns random forests solely from a 3D CAD model to predict the transformation update from the input depth images. On average, it runs at less than 2 ms per frame for each object on a single CPU core. Hence, even if the computational complexity of the tracker is linear to the number of objects, the extremely low magnitude of its tracking time allows the algorithm to track over a hundred moving objects in a scene at 30 fps with only 8 CPU cores. This is significantly faster than any tracking-by-detection approach. Due to the tracker's efficiency, we have successfully demonstrated the combination of detection and temporal tracking in a single framework to achieve a seamless performance, *i.e.* with low latency.

Although the first version of the tracker works on specific object instances, meaning that the shape of the object of interest that appears in the scene has to match the geometry of the learned CAD model, we further developed an extension that generalizes the tracker to perform pose estimation for any object within a class. Through the 3D head pose estimation, this allows the generalized tracker to directly estimate the pose of an arbitrary user without any a priori information about them.

Nevertheless, tracking with an object instance still attains better accuracy than a generalized tracker. Due to this, we introduce a fast and reliable calibration procedure that optimizes the shape model of a specific object from a given class. In this thesis, to

improve the accuracy in hand tracking, we apply the calibration to optimize a detailed personalized hand shape model for a specific user through a small set of depth images.

Zusammenfassung

Posenbestimmung von Objekten ist die Aufgabe die Relation zwischen der Kamera und eines Objektes zu bestimmen. Für rigide Objekte ist die Pose durch drei Freiheitsgrade für Translation und drei Freiheitsgrade für Rotation eindeutig bestimmt. Posenbestimmung in Echtzeit ist für viele Anwendungen wie Wahrnehmung in der Robotik, Erweiterte Realität und Mensch-Maschine Interaktion von großer Bedeutung. Diese Arbeit befasst sich mit der Posenbestimmung anhand der Entwicklung eines temporalen Trackers basierend auf dreidimensionalen Daten. Das Ziel ist es nicht nur robust in öffentlichen Benchmarks zu sein, sondern auch eine breite Palette von echten Anwendungen zu erstellen.

Ein Temporaltracker ist ein Frame-To-Frame-Algorithmus, der die Pose eines Objekts von einem Frame zum nächsten propagiert und dann für den aktuellen Frame verbessert. Temporaltracker werden häufig in Rekonstruktionssystemen verwendet um ein einzelnes Objekt oder eine statische Szene zu verfolgen. Wenn jedoch mehrere, sich unabhängig bewegende Objekte verfolgt werden müssen werden oft Detektionsmethoden verwendet die jedes Bild einzeln betrachten und darin, unabhängig vom vorherigen Bild, die Position und Pose bestimmen.

Einerseits hat ein temporaler Tracker das Problem, dass die Pose im ersten Bild manuell oder von einem Detektor bestimmt werden muss, andererseits sind temporale Tracker sehr effizient, da nur die Änderung der Pose von einem Bild zu nächsten bestimmt werden muss. Deshalb ist die Kombination beider Ansätze intuitiv sinnvoll, da ein Detektor den Tracker initialisieren kann. Da eine große Robustheit gegenüber Verdeckungen und anderen Objekten in der Szene erreicht werde muss benötigen die meisten temporalen Tracker etwa 100ms pro Frame mit GPU Unterstützung. Da für jedes weitere Objekt ein Tracker benötigt wird, skaliert die Laufzeit linear mit der Anzahl der Objekte. Das bedeutet, dass ab etwa 5 Objekten in der Szene, Detektionsansätze effizienter werden als temporale Tracker.

Wir formulieren einen lernbasierten 3D Tracker der mit Hilfe von Random Forests die nur von 3D CAD Modellen lernen, die Änderung der Objekttransformation zwischen zwei Tiefenbildern vorhersagt. Im Durchschnitt benötigt der Tracker weniger als 2ms pro Bild und Objekt in der Szene auf nur einem CPU Kern. Deshalb können mehr als einhundert unabhängige Objekte mit einer Effizienz von 30 Bildern pro Sekunde und 8 CPU Kernen verarbeitet werden. Das ist deutlich schneller als jeder Detektionsansatz. Auf Grund dieser Performanz erzielen wir eine erfolgreiche Vereinigung von Detektion und Tracking, d.h. mit geringer Latenzzeit.

Die erste Variante des Trackers funktioniert mit spezifischen Objektinstanzen, für die das Objekt als CAD Modell gegeben sein muss. Diese können wir dann aber erweitern um über verschiedene Instanzen einer Objektklasse generalisieren zu können. Hier zeigen wir als Anwendung die Bestimmung der Orientierung des Kopfes des Benutzers ohne dabei die Kopfform des Anwenders kennen zu müssen.

Da Tracking mit einem genauen Modell präziser ist als die generalisierte Variante, führen wir eine Methode ein, die es erlaubt die Form des Objekts an die aktuelle Instanz anzupassen. Wir benutzen diese Methode um die Form und Haltung der Hand eines Benutzers genau bestimmen zu können. Dafür werden nur wenige Tiefenbilder benötigt.

Contents

List of Figures

List of Tables

Part I

Introduction

1 Introduction

1.1 Overview

With the constant emergence of affordable depth cameras as well as wearable and mobile devices with depth sensors, the value of a real-time object pose estimation with 3D data has found its way into mainstream. It uncovers a wide-range of applications such as robotic perception, augmented reality (AR) and human-computer interaction.

The goal of object pose estimation is to estimate the 6 d.o.f. transformation of an object. This includes 3 d.o.f. for translation and 3 d.o.f. for rotation. Algorithms that aim at the same goal are classified into two categories. The first is the object detection and pose estimation (alternatively, tracking-by-detection) that localizes the object of interest in the scene and simultaneously approximate its pose. The other is an object temporal tracker that relays the transformation from one frame to the next such that the problem is simplified to refine the pose from the previous frame.

When we compare the two categories, the advantage of the former against the latter is its independence from the previous frames. This implies that, when one of the frames fail to estimate the correct pose, the succeeding frames are not affected by this failure. Conversely, the latter utilizes the pose from the previous frames, which make it vulnerable to tracking failures and loses the object on the next frames. On the other hand, the advantage of a temporal tracker is the simplification of the problem to a frame-to-frame pose refinement. Theoretically, this allows the tracker to be significantly faster than the detector.

The complementary nature of the two allows them to be combined synergically. Intuitively, when we personally interact with an object in a scene, we detect an object and temporally track it across time. But in most object pose estimation framework, the temporal tracker is not implemented. Although the temporal tracker is theoretically faster than a detector, this does not hold in practice. Most robust temporal trackers [28, 74, 119] requires approximately more than 100 ms to track an object with a GPU implementation. Thus, when we have several independently moving objects in the scene, the time required to track all of them drastically increases such that the real-time performance is lost. Therefore, in practice, detecting objects in each frame independently becomes more efficient.

Fast Learner

A template tracker estimates the homography of a template in a sequence of intensity images. Using a temporal tracker, we operate on updating the homography from one frame to the next, assuming that the location of the template in the initial frame is given. These methods aim at attaining a fast 2D temporal tracker with a fast learning method. Hence, the speeds enable these approaches to learn and track templates using mobile devices.

Deformable Tracker

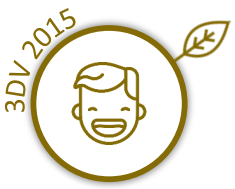
Attempting to move farther from the 2D rigid template tracking with homographies, we explore the possibility of increasing the number of control points that distorts the template in order to achieve a deformable tracker. With more control points, the distortion of the template follows the 2D free form deformation with cubic b-splines interpolation. Towards the end of this work, we evaluate the algorithm in the context of tracking facial deformations.

Chameleon Tracker

By observing the similarities of the objective functions from different registration algorithms, we generalize our learning-based tracker in order to perform not only 2D template tracking but also 3D model-based tracking using depth images. We boast the high speed of 2ms per frame with a single CPU core, which is specifically a milestone for 3D trackers. Furthermore, we also claim that this work is the first learning-based 3D tracker that estimates the 6 d.o.f. pose of a rigid object.

Versatile Tracker

Although the Chameleon tracker is very fast with a very low computational cost, it has a lot of constraints in other aspects. This work alleviates these constraints by generating a tracker that has a fast learning time, low memory footprint and more robust to occlusions but still preserving the tracking time and computational cost. All the advancement in performance led the tracker to achieve scalability with respect to multiple objects at a real-time framerate.



3D Head Pose Estimation

An interesting end-to-end application of the versatile tracker is the 3D head pose estimation. In this context, the tracker is initialized by a face detection through the RGB image. The tracker then converts this location into the 3D space using the depth image and continuously estimate the 6 d.o.f. pose of the head in the scene. Contrary to learning a tracker for a specific object, the challenge here is to learn a tracker that generalizes for an arbitrary user.

Hand Shape Personalization

The cost of generalizing a tracker to estimate the pose for a set of objects or users is the decrease in tracking accuracy. Given a set of depth images, this work proposed a calibration procedure in order to personalize a detailed hand model for a specific user. Thereafter, the tracker takes this model and approximate its deformed pose. Notably, this work is the first to show that a personalized model improve the accuracy in deformable hand tracking.

Type of Algorithms

In this thesis, we focus on achieving an object temporal tracker that scales well with a large number of objects in the scene. Our tracker efficiently tracks in less than 2 ms per frame per object with only one CPU core. At the same time, it is proven to be robust against clutter and large occlusions.

As a starting point, we extensively explore the learning-based 2D template trackers [60, 61, 62, 65, 132] and investigate the similarity of its 2D registration with 3D registration [133]. From there, we discovered the first learning-based temporal tracker that relies on depth images and estimates the 6 d.o.f. pose of an object. An essential attibute of such approach is its tracking time of 2 ms.

We then continue studying this method and simplify it to consider other aspects of the algorithm [134]. By changing the objective function and the learning strategy, we accomplish a versatile tracker that is capable of tracking 108 objects in the scene at 30 fps with 7.4 MB memory consumption of each object, and achieving a remarkable robustness against large occlusions. It also decreased the learning time by two orders of magnitude. As a consequence, we also introduce an online learning scheme where the algorithm tracks and learns the object for each frame.

The temporal tracker has been further studied to estimate the head pose. By initializing the tracker through a face detector [143], we temporally relay the tranformation from one frame to the next [135]. The challenge in this case is to acquire a tracker that can estimate the head pose for an arbitrary user. In effect, this work is the successful attempt to lean and track a class of objects.

Alas, the effects of generalizing the tracker is a decrease in accuracy. Thus, it would be more beneficial to track an oject from a model specified to the object of interest. We then propose a fast and reliable calibration procedure to acquire a detailed shape of an object within a class. This work is used in the context of hand pose estimation in order to personalize a detailed hand shape for a specified user [131]. Towards the end of this work, we have quantitatively proven that tracking with a calibrated model generate a more accurate pose.

Finally, an overview of all the contributions included in this thesis is illustrated on pages 4-5. In addition, the complete list of publications is in Appendix A. Notably, different methods of the object temporal tracker [133, 134] and the head pose estimation [135] have been presented in Demo sessions of CVPR 2014, ECCV 2014 and CVPR 2016. After combining the temporal tracker with an object detector and pose estimator, our first seamless object detection and tracking was demonstrated at CVPR 2016. In this framework, the object detection and pose estimation automatically initialize the temporal tracker for multiple instance of the object present in the scene. It highlights the low latency of the tracker such that the users cannot sense the delay in tracking.

1.2 Motivation to Generalize

The motivation of the work is to encompass different aspects in order to transition from our theoretical ideas towards solving problems of the real-world applications. From one chapter to the next, this thesis narrates a constant improvement and a constant simplification of the algorithm with the goal of achieving *generalization* – "make

(something) more widespread or widely applicable"[1].

A typical pipeline to reach practical applications goes through the theory which is a collection of principles that explains or describes a phenomenon, and algorithm engineering that implements and optimizes the theory. Although algorithm engineering bridges the gap between algorithm theory and practical applications, the thesis focuses on strengthening the theoretical foundation. Evidently, we do not devalue the importance of a good implementation. However, we believe that the only way to achieve generalization is through a strong theoretical foundation.

In this context, the goal of generalization manifests in different forms. We begin by generalizing the 2D template tracking from [8, 48, 60, 61, 62, 65] for a generic learning-based registration problem. The idea is based on the observation of the similarities between the objective functions from different registration tasks. As a result, while moving away from the input RGB images of the 2D template trackers and going to 3D data, this theory led to the emergence of the first 3D learning-based temporal tracking approach that learns solely from the rendered synthetic images of the object's 3D CAD model and estimates the full 6 d.o.f. pose of the object from real depth images. Notably, the tracker performs domain generalization where it learns from synthetic images or perfect data, and tracks on real images with sensor noise, missing data, clutter and occlusions. In addition, after learning from a specific camera model, the tracker also generalizes to any depth sensor, *i.e*. without the requirement to use identical camera models with similar intrinsic parameters as the learned camera model.

When tracking the 3D object, the next goal is generalizing the algorithm to attain the same robustness and accuracy for different objects as well as for different environmental scenarios. This is an example where a strong theoretical foundation of the algorithm becomes essential. If we engineer the algorithm for an object or some objects, it loses its capacity to perform in the same way for other objects, which effectively constraints the approach to the specified objects.

Regarding the environmental factors, we need to handle occlusions from the object's surroundings. This frequently occurs in bin-picking scenarios where the objects are on top of each other or tracking hand-held objects where the object is partially occluded by the hand. After observing different kinds of occlusions on the images, we theorize that occlusions are 2D obstructions from an edge of the object. Following this idea, we adapt the learning component of the tracker and obtain remarkable results in achieving robustness to large occlusions.

When we step back and look at the big picture, the tracker must not be constrained to a specific hardware requirement and must generalize to any computational device. A simple example of such is the implementation on GPU for a real-time performance. With the emergence of depth sensors specifically suited for mobile devices, the real-time performance on GPU on a desktop computer does not translate to the real-time performance on mobile devices. Another aspect to consider is the fact that object pose estimation is merely a component of a larger framework. For instance, in an AR application, tracking is used as input to superimpose 3D models on the user's view. Tracking then is not the final result but rather an initial information. If tracking requires a large computational power, real-time performance on AR is no longer possible. Contrary to this, our tracker runs in less than 2 ms per frame per object using a single core CPU. Therefore, we solve this generalization by running an extremely fast tracker with the

[1] Definition of *generalize* from Oxford dictionary.

minimum computational requirement possible.

Other than computational power, we should also consider the memory footprint since this is a learning-based approach. This aspect is important when we track multiple objects at the same time. We explicitly address this aspect so that the tracker only utilize approximately 7.4 MB per object. Thus, considering that we are only using one core, this work relies on its theoretical merits in order to achieve the robustness and accuracy in a very short time with a small memory footprint.

In addition to the 3D object tracker, we also discuss the generalization of the head pose estimation to track an arbitrary user. Thus, when a user is in front of a camera, we can immediately initialize the tracker to temporally estimate the head pose. Conversely, we also propose a calibration procedure for the hand pose estimation. In this case, the calibration is done prior to tracking, where the algorithm deforms the mean hand model in order to generate a detailed hand shape model. Thereafter, the tracker uses the resulting personalized model to ensure a good fit to user's hand on the depth image.

1.3 Thesis Outline

For the next chapters, we provide a short overview and the corresponding related works associated to them. Most of the methods and materials of the thesis are published or under-submission for a major conference or journal. Note that the complete list of papers is listed in Appendix A.

Chapter 2. We review the 2D template tracking approach using linear predictors [65] with the derivation of the objective function from [8, 48]. This is our starting point.

Chapter 3. Based on [65], this chapter discusses two main improvements. One deals with instantaneous learning procedures to learn templates on-the-fly while the other discusses on the extension to 2D deformable trackers. The related works are:

- Holzer, S., Ilic, S., Tan, D.J., Navab, N.: Efficient learning of linear predictors using dimensionality reduction. In: Asian Conference on Computer Vision, pp. 15–28. Springer (2012) – *Best Application Paper Honorable Mention*
- Holzer, S., Pollefeys, M., Ilic, S., Tan, D.J., Navab, N.: Online learning of linear predictors for real-time tracking. In: European Conference on Computer Vision, pp. 470–483. Springer (2012)
- Tan, D.J., Holzer, S., Navab, N., Ilic, S.: Deformable template tracking in 1ms. In: British Machine Vision Conference. Citeseer (2014)

Chapter 4. After observing the similarites of the objective function from 2D image registration and 3D point-based registration, we look into generalizing the work of [8, 48, 65] to perform a generic learning-based registration using random forest [21]. As a result, we evaluated the approach as a 2D template tracker as well as a 3D model-based tracker. The former is similar to [60, 62, 65] but has the advantage of performing better in the presence of occlusion and extreme global illumination changes. On the other hand, the latter has the largest impact in relation to its related works. To the best of our

knowledge, this is the first learning-based temporal tracking approach that operates on depth images and estimates the complete 6 d.o.f. pose of an object. The other approach that performs the same goal is the iterative closest point algorithm (ICP) [15, 118, 123], which is an energy-based approach. Another interesting attribute of this apporach is the tracking time of 2 ms per frame with only one CPU core. The related works are:

- Tan, D.J., Ilic, S.: Multi-forest tracker: A chameleon in tracking. In: Conference on Computer Vision and Pattern Recognition, pp. 1202–1209. IEEE (2014)

Briefly, we also discuss that the 2D tracker from this work is part of our tool tracking framework for surgical application in:

- Rieke, N., Tan, D.J., Tombari, F., Vizcaino, J.P., di San Filippo, C.A., Eslami, A., Navab, N.: Real-time online adaption for robust instrument tracking and pose estimation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 266–273. Springer (2015)

- Rieke, N., Tan, D.J., di San Filippo, C.A., Tombari, F., Alsheakhali, M., Belagiannis, V., Eslami, A., Navab, N.: Real-time localization of articulated surgical instruments in retinal microsurgery. Medical image analysis (2016)

- Rieke, N., Tan, D.J., Alsheakhali, M., Tombari, F., di San Filippo, C.A., Belagiannis, V., Eslami, A., Navab, N.: Surgical tool tracking and pose estimation in retinal microsurgery. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 266–273. Springer (2015) – *Young Scientist Award*

In the succeeding chapters, we primarily focus on the 3D tracker due to its appeal to a wide-range of new applications that includes robotic perception, augmented reality and human-computer interaction.

Chapter 5. Although the 3D tracker is fast, it does not consider other aspects such as memory consumption, learning time, robustness to large occlusions and scalability to multiple objects, that are useful in terms of applications. This work explicitly addresses these problems while keeping the tracking time low. We achieve this by re-interpreting the objective function as a point-to-plane registration problem and taking occlusion into account. With these, we were able to simplify the learning procedure which takes two orders of magnitude faster than the previous work. Due to the fast learning strategy, we also introduce an online learning procedure where we begin by identifying the object of interest in the first frame, and continuously track and learn in the succeeding frames in order to capture different views of the object. Using only 8 CPU cores, the online learning framework runs at 30 fps. Notably, the model-based tracker attains a memory footprint of about 7.4 MB and scales up to tracking 108 objects present in the scene at 30 fps. The related works are:

- Tan, D.J., Tombari, F., Ilic, S., Navab, N.: A versatile learning-based 3d temporal tracker: Scalable, robust, online. In: International Conference on Computer Vision (2015)

Chapter 6. When we thought of utilizing the temporal tracker as a head pose estimation, we realize the importance of generalizing the tracker to run for different users. When any user is in front of the camera, the tracker can be initialized and must temporally estimate the 6 d.o.f. head pose. Thus, based on a set of 3D head models, we learn a random forest to accomplish this task. Notably, this is the first attempt to generalize the tracker for a class of objects instead of a specific model for a given user. However, a generalized tracker is constrained to track the facial structure of the user since this is common structure that is visible in both the models and the real depth images. As a consequence, the generalized tracker cannot handle extreme poses where the facial structures are no longer visible. Due to this, we also propose a subject-specific online learning approach that captures the unique head structure of the user. In this way, the head can rotate and track the regions where the generalized tracker had not learned. Another option is to build a multi-camera system so that, if the structure to track in not visible in one or more cameras, the other cameras can still track the head pose. Interestingly, the tracker is well-suited for such system and can aggregate the predictions from different cameras with ease. The related works are:

- Tan, D.J., Tombari, F., Navab, N.: A combined generalized and subject-specific 3d head pose estimation. In: International Conference on 3D Vision. IEEE (2015)
- Tan, D.J., Tombari, F., Navab, N.: Real-time accurate 3d head tracking and pose estimation with consumer rgb-d cameras. International Journal of Computer Vision (2017) – *Submitted*

Chapter 7. Considering that a generalized tracker tries to utilize a mean model that averages the structure of different users, this work proposes a fast and reliable calibration procedure that captures a detailed personalized shape model of a specific user. In the context of hand pose estimation, the mean model is replaced by the personalized model in order to improved the accuracy of a generalized tracker. This work follows an energy-based minimization on the golden energy from [124] which is a pixel-wise difference between the input depth image and the rendered depth image. The related works are:

- Tan, D.J., Cashman, T., Taylor, J., Fitzgibbon, A., Tarlow, D., Khamis, S., Izadi, S., Shotton, J.: Fits like a glove: Rapid and reliable hand shape personalization. In: Conference on Computer Vision and Pattern Recognition (2016)

Chapter 8. We finally conclude the thesis with the future directions.

Part II

2D Template Tracking

2

2D Template Tracking – From Energy Minimization to Learning

2.1 Motivation

Template tracking is an extensively studied field in computer vision. The main task of template tracking is to follow a template in an image sequence. This is implemented 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. 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 [7, 8, 13, 24, 36, 48, 85, 87, 125] 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 [47, 58, 65, 66, 90, 92, 102, 148] 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 [65] proposed a 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.

The motivation of this chapter is to derive the learning-based approach as proposed by Jurie and Dhome [65]. We consider this approach as a starting point for which we improve in Chapter 3, and we generalize to different registration problems in Chapter 4 so to not only perform a 2D template tracker but also a 3D model-based tracker that estimates the 6 d.o.f. pose of a rigid object.

2.2 Related Work

The existing template tracking approaches can be categorized in mainly three different sets of methods: tracking-by-detection (TBD) [51, 52, 53, 57, 100], template tracking based on energy minimization [7, 8, 13, 24, 32, 36, 48, 85, 87, 112, 125], and methods that utilize learning [47, 65, 66, 90, 92, 102, 148]. 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*. [66] demonstrated that linear predictors are superior to Jacobian approximation and Holzer *et al*. [58] showed an experiment where linear predictors are superior to Efficient Second-order Minimization (ESM) [13].

Tracking-by-Detection-based approaches. Some of the most prominent work on patch-based TBD was recently proposed by Hinterstoisser *et al*. [51, 52, 53]. Their former two methods, called Leopard [51] and Gepard [52], 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) [53]. 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) [100] 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) [57] proposed a detection based approach which builds on finding closed contours and matches them using a similar approach as [100] 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 [85]. They consist of different update rules of the warp function [7, 24, 36, 48, 85, 125], handling of occlusions and illumination changes [48], as well as considering different orders of approximation of the error function [13, 87]. The different update rules of the warp function can be classified into four types, namely, the additive approach [85], the compositional approach [125], the inverse additive approach [24, 48] and the inverse compositional approach [7, 36], 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

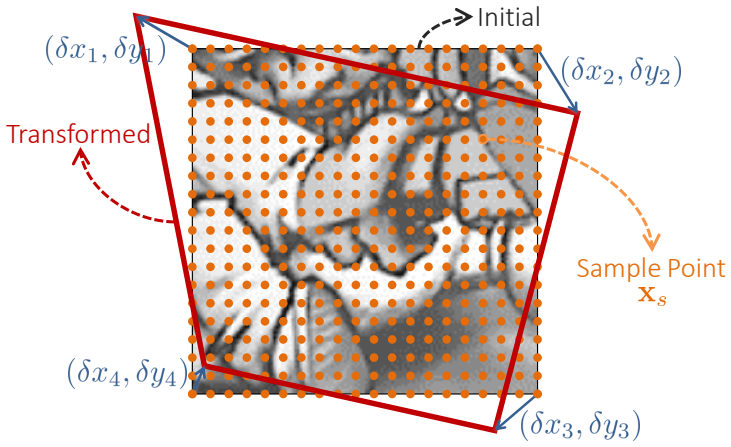


Figure 2.1: A template is represented by a set of regularly placed sample points located at $\mathbf{x}_s$. Its pose is parameterized through the displacements of the four corner points represented by the vectors $\{(\delta x_c, \delta y_c)\}_{c=1}^{4}$ from its initial state to the transformed.

occlusions was addressed by Hager and Belhumeur [48]. 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 [13, 87]. A more detailed overview of energy-based tracking methods is given by Baker and Matthews [8].

Learning-based approaches. In contrast to energy minimization approaches, Jurie and Dhome [65] 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 [66]. Other authors such as Gräßl *et al*. [46] demonstrated how linear predictors can be made invariant to illumination changes. In addition, to further increase accuracy in tracking, they [47] also formulated a method on how to select the points for sampling from the image data. Zimmermann *et al*. [148] 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*. [58] 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 [92] stepped back from linear predictors by presenting an approach that fits the sampling region to pre-trained samples using general regression.

2.3 2D Template Tracking

A frame-to-frame tracking algorithm estimates the transformation parameters of a template throughout a sequence of images. Let us assign $\mathbf{I}_t$ and $\mathbf{T}_t$ as the image and transformation at time t, respectively. Without loss of generality, the template is defined

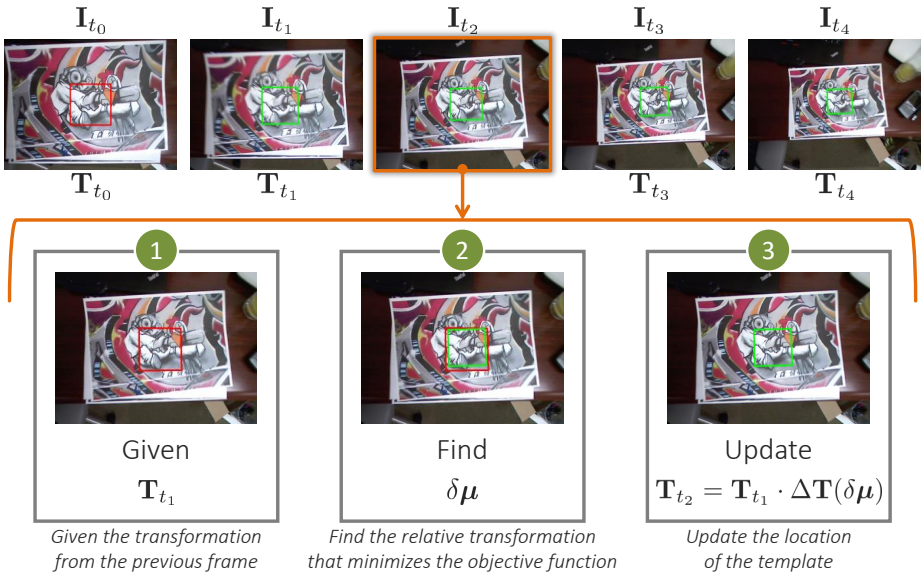


Figure 2.2: For every frame in the video sequence, the tracker follows a three step proceduce where the algorithm (1) utilizes the location of the template from the previous frame; (2) finds the relative transformation between consecutive frames; and, (3) updates the current location of the template using the relative transformation.

as a $w \times h$ region on an image with $n_s = w \cdot h$ pixels. Notably, this method is not restricted to this sample point arrangement or rectangular shapes.

Instead of using the full-resolution template, we apply a uniform subsampling as shown in Figure 2.1 to obtain a grid of n_s sample points written as $\{\mathbf{x}_s\}_{s=1}^{n_s}$ at its initial state. The intensities at the sample points are concatenated into the intensity vector $\mathbf{i} = [I(\mathbf{x}_s)]_{s=1}^{n_s}$.

Assuming a rigid transformation of the template, the template transformation $\mathbf{T}$ is a homography with 8 d.o.f. to represent the current perspective distortion of a planar template. Since a set four corresponding points is sufficient to estimate a homography, the location of the template is described through its four corners $\{(x_c, y_c)\}_{c=1}^{4}$. These are aggregated to form the transformation vector $\boldsymbol{\mu} = [x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4]^\top$.

In contrast to tracking-by-detection, frame-to-frame tracking propagates the transformation from one frame to the next. For instance in Figure 2.2, the transformation from the previous frame $\mathbf{T}_1$ is given. The problem then is diverted to estimate the relative transformation between $\mathbf{T}_1$ and $\mathbf{T}_2$ in order to find $\mathbf{T}_2$. A similar process then applies to all the subsequent frames.

2.3.1 Objective Function

The goal of frame-to-frame tracking is to update the parameters at $t + \tau$ using the given parameters $\boldsymbol{\mu}_t$ at time t [48]. Thus, we seek for the change in parameters $\delta\boldsymbol{\mu}$ by minimizing the registration energy

$$\varepsilon(\delta\boldsymbol{\mu}) = \left\| \mathbf{i}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}, t + \tau) - \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0) \right\|^2 \tag{2.1}$$

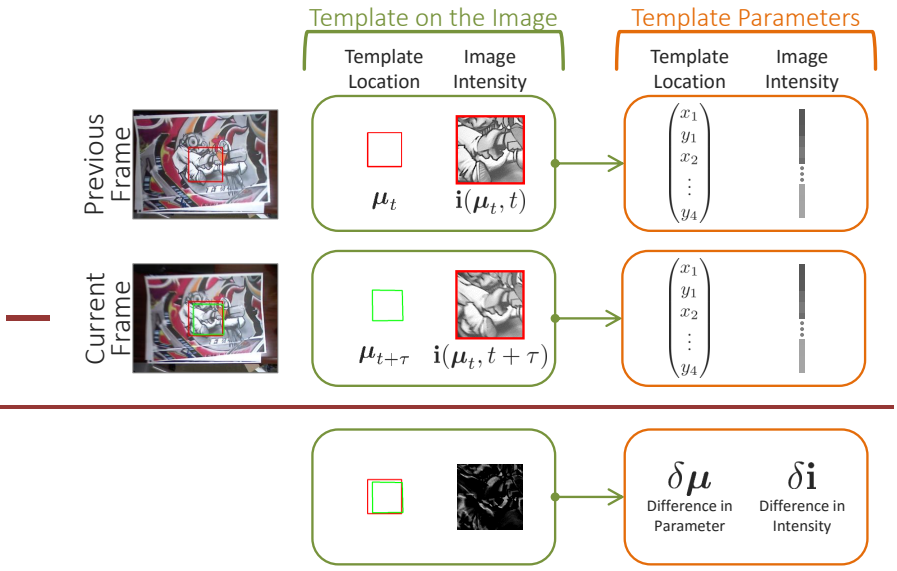


Figure 2.3: Visual representation of the objective function between two consecutive frames where we are estimating for $\boldsymbol{\mu}_{t+\tau}$ by finding the difference in parameter $\delta\boldsymbol{\mu}$ between $\boldsymbol{\mu}_t$ and $\boldsymbol{\mu}_{t+\tau}$.

that compares the intensities at the location of the tempaltes at $t+\tau$ and t_0. Here, we define

$$\mathbf{i}(\boldsymbol{\mu}, t) = [\mathbf{I}_t(\mathbf{T}(\boldsymbol{\mu}) \cdot \mathbf{x}_s)]_{s=1}^{n_s} \ . \tag{2.2}$$

When comparing (2.1) to Figure 2.3, we assume that, at t, the parameters $\boldsymbol{\mu}_t$ is at the ground truth location of the template on the image such that

$$\boldsymbol{\mu}_t = \arg\min_{\boldsymbol{\mu}} \left\| \mathbf{i}(\boldsymbol{\mu}, t) - \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0) \right\|^2 \tag{2.3}$$

is optimized. This implies that the intensities in $\mathbf{i}(\boldsymbol{\mu}_t, t) \approx \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0)$, which is constant.

Using Taylor series approximation, the first term in (2.1) is expanded as

$$\mathbf{i}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}, t+\tau) \approx \mathbf{i}(\boldsymbol{\mu}_t, t) + \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)\delta\boldsymbol{\mu} + \tau\frac{\partial \mathbf{i}}{\partial t}(\boldsymbol{\mu}_t, t) \tag{2.4}$$

where $\mathbf{J}_{\boldsymbol{\mu}_t} = \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)$ is the Jacobian matrix of $\mathbf{i}$ with respect to $\boldsymbol{\mu}$. A forward difference approximation on $\frac{\partial \mathbf{i}}{\partial t}(\boldsymbol{\mu}_t, t)$ simplifies the equation as

$$\mathbf{i}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}, t+\tau) \approx \mathbf{i}(\boldsymbol{\mu}_t, t) + \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)\delta\boldsymbol{\mu} + \tau\left[\frac{\mathbf{i}(\boldsymbol{\mu}_t, t+\tau) - \mathbf{i}(\boldsymbol{\mu}_t, t)}{\tau}\right] \tag{2.5a}$$

$$= \mathbf{i}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)\delta\boldsymbol{\mu} \ . \tag{2.5b}$$

Putting back (2.5b) to (2.1),

$$\varepsilon(\delta\boldsymbol{\mu}) \approx \left\| \mathbf{i}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}_t}\delta\boldsymbol{\mu} - \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0) \right\|^2 \ . \tag{2.6}$$

In order to find $\delta\boldsymbol{\mu}$ with the minimum ε, $\bigtriangledown_{\delta\boldsymbol{\mu}}\varepsilon$ is set to zero. After assigning $\check{\varepsilon} = \mathbf{i}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}_t}\delta\boldsymbol{\mu} - \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0)$ so that $\varepsilon \approx \|\check{\varepsilon}\|^2 = \check{\varepsilon}^\top\check{\varepsilon}$,

$$\bigtriangledown_{\delta\boldsymbol{\mu}}\varepsilon \approx 2\check{\varepsilon}\cdot\bigtriangledown_{\delta\boldsymbol{\mu}}\check{\varepsilon} = 0\,. \tag{2.7}$$

Since $\bigtriangledown_{\delta\boldsymbol{\mu}}\check{\varepsilon} = \mathbf{J}_{\boldsymbol{\mu}_t}$ is not necessarily zero, then

$$\check{\varepsilon} = \mathbf{i}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}_t}\delta\boldsymbol{\mu} - \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0) = 0 \tag{2.8}$$

$$\Rightarrow \delta\boldsymbol{\mu} = -\mathbf{J}^+_{\boldsymbol{\mu}_t}\delta\mathbf{i}(\boldsymbol{\mu}_t, t+\tau) \tag{2.9}$$

where $\mathbf{J}^+_{\boldsymbol{\mu}} = \left(\mathbf{J}^\top_{\boldsymbol{\mu}}\mathbf{J}_{\boldsymbol{\mu}}\right)^{-1}\mathbf{J}^\top_{\boldsymbol{\mu}}$ and $\delta\mathbf{i}(\boldsymbol{\mu}, t) = \mathbf{i}(\boldsymbol{\mu}, t) - \mathbf{i}(\boldsymbol{\mu}_{t_0}, t_0)$ as shown in Figure 2.3. Similar to [65], we aim to replace the time-consuming $-\mathbf{J}^+_{\boldsymbol{\mu}_t}$, that defines the relation between the parameter update $\delta\boldsymbol{\mu}$ and the change in intensity $\delta\mathbf{i}$. Instead, [65] learns a linear predictor $\mathbf{A}$ offline such that tracking is simplified to the matrix multiplication of

$$\delta\boldsymbol{\mu} = \mathbf{A}\,\delta\mathbf{i}(\boldsymbol{\mu}_t, t+\tau) \tag{2.10}$$

with a constant matrix $\mathbf{A}$.

2.3.2 Learning a Linear Predictor

At $t = t_0$, the image I_{t_0} is randomly transformed through $\delta\boldsymbol{\mu}_\omega$ where $\delta\boldsymbol{\mu}_\omega$ is a small random disturbances on the template's corners. The transformed template mimics its motion from the previous frame to the current frame. We then assign I_ω as a transformed version of the template such that $\mathbf{T}(\delta\boldsymbol{\mu}_\omega)\cdot\mathbf{x}_s$ is the ground truth location of the sample points on I_ω. In effect, for every random set of values of $\delta\boldsymbol{\mu}_\omega$, this introduces a variation of the image intensity differences $\delta\mathbf{i}_i = \mathbf{i}(\mu_{t_0}, \omega) - \mathbf{i}(\mu_{t_0}, t_0)$. Hence, with n_ω random perturbation, the set of $\{(\delta\mathbf{i}_\omega, \delta\boldsymbol{\mu}_\omega)\}_{\omega=1}^{n_\omega}$ is formulated as our learning dataset.

Based on (2.10), an important property of $\mathbf{A}$ is that, for each element of the learning dataset, $\delta\boldsymbol{\mu}_\omega = \mathbf{A}\delta\mathbf{i}_\omega$ holds. Then, when concatenating the vectors to construct $\mathbf{Y} = \left[\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \cdots, \delta\boldsymbol{\mu}_{n_\omega}\right]$ and $\mathbf{H} = [\delta\mathbf{i}_1, \delta\mathbf{i}_2, \cdots, \delta\mathbf{i}_{n_\omega}]$, the relation $\mathbf{Y} = \mathbf{AH}$ also holds. It follows that $\mathbf{A}$ is learned by minimizing

$$\arg\min_{\mathbf{A}} \sum_{\omega=1}^{n_\omega} \|\delta\boldsymbol{\mu}_\omega - \mathbf{A}\delta\mathbf{i}_\omega\|^2 \tag{2.11}$$

which results in the closed-form solution [65]

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top\left(\mathbf{H}\mathbf{H}^\top\right)^{-1}\,. \tag{2.12}$$

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.

2.3.3 Tracking with Linear Predictor

The location of the template is estimated by using a parameter update vector $\delta\boldsymbol{\mu}_{t+\tau}$ that corrects the transformation of $\mathbf{x}_s$ from the frame at t to $t+\tau$. Using the image difference vector

$$\delta\mathbf{i}_{t+\tau} = \mathbf{i}(\boldsymbol{\mu}_t, t+\tau) - \mathbf{i}(\mu_{t_0}, t_0)\ , \tag{2.13}$$

where $\mathbf{i}(\boldsymbol{\mu}_t, t+\tau)$ is a collection of image intensities in the current frame with the sample point locations of the previous frame, the relation between $\delta\mathbf{i}_{t+\tau}$ and $\delta\boldsymbol{\mu}_{t+\tau}$ in (2.10) can be defined using the linear predictor [65] with

$$\delta\boldsymbol{\mu}_{t+\tau} = \mathbf{A}\delta\mathbf{i}_{t+\tau}\ . \tag{2.14}$$

Therefore, to track the reference template, we compute $\delta\boldsymbol{\mu}_{t+\tau}$ using the given vector $\delta\mathbf{i}_{t+\tau}$ and the learned matrix $\mathbf{A}$. Having learned from the same coordinate system as $\mathbf{I}_{t_0}$, the homography follows a compositional update with $\mathbf{T}_{t+\tau} = \mathbf{T}_t \cdot \Delta\mathbf{T}(\delta\boldsymbol{\mu}_{t+\tau})$ where $\Delta\mathbf{T}(\delta\boldsymbol{\mu})$ is a transformation from the initial state of the four corners at $\boldsymbol{\mu}_{t_0}$ to its transformed state at $\boldsymbol{\mu}_{t_0} + \delta\boldsymbol{\mu}$ as shown in Figure 2.1.

Similar to an energy minimization approach, learning-based tracking is also iterative. For improved tracking performance, a multi-predictor approach is implemented such that multiple linear predictor $\{\mathbf{A}_l\}_{l=1}^{n_l}$ 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. Typically, there are $n_l = 5$ levels with three iterations for each of them.

2.4 Conclusion

Due to the matrix multiplications during tracking, the work from Jurie and Dhome [65] manages to track at a very high speed that can reach 1000 fps. We consider this as the most attractive benefit of their work. However, it also falls under some limitations such as offline learning and rigid transformations. In the next chapter, we propose several improvements on the 2D template tracking approach using linear predictors. This includes a fast learning scheme that can instantaneously learn new templates [60, 61, 62] while tracking and an extension of the rigid transformation to a deformable model in order to allow more degrees of freedom on the template's motion [132].

3

Improvements on the Learning-based 2D Template Tracker

3.1 Motivation

Template tracking aims to follow a reference template over a sequence of images. Assuming that the reference template is located at t_0 of a video sequence, its location is defined by an 8×1 initial parameter vector $\boldsymbol{\mu}_{t_0}$ that corresponds to concatenation of the 2D points of the four corners of the template. Within the template, n_s 2D points define the location of the sample points. By collecting the intensities at the n_s sample points, an $n_s\times1$ vector $\mathbf{i}_{t_0}$ corresponds to the image intensities of the template to be matched across frames. Then, at time t, the parameter vector $\boldsymbol{\mu}_t$ defines the location of the template in the current frame which, in effect, transforms the location of the sample points. Henceforth, tracking is accomplished by computing $\boldsymbol{\mu}_t$.

The learning-based tracker maps the relation between the location of the template and the intensities within the template. The approach first computes a vector $\delta\mathbf{i} = \mathbf{i}_t - \mathbf{i}_{t_0}$ 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 $\mathbf{i}_t$ stores the image values extracted from the current image and is extracted by computing the sample point locations using the template pose of the previous image frame.

Jurie and Dhome [65] learns a linear predictor $\mathbf{A}$ to compute the unknown $\delta\boldsymbol{\mu}$ based on the given vector $\delta\mathbf{i}$ with the relation

$$\underbrace{\delta\boldsymbol{\mu}}_{8\times1} = \underbrace{\mathbf{A}}_{8\times n_s}\underbrace{\delta\mathbf{i}}_{n_s\times1}. \tag{3.1}$$

Considering a constant $\mathbf{A}$ in tracking, (3.1) allows the tracker to be extremely fast with less than 1 ms per frame.

However, in order to compute $\delta\boldsymbol{\mu}$, one needs to precompute the matrix $\mathbf{A}$. This is done by collecting a set of n_ω random transformations, where n_ω is significantly larger than n_s, 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 $\mathbf{Y} = \left[\delta\boldsymbol{\mu}_1, \delta\boldsymbol{\mu}_2, \ldots, \delta\boldsymbol{\mu}_{n_\omega}\right]$ and $\mathbf{H} = [\delta\mathbf{i}_1, \delta\mathbf{i}_2, \ldots, \delta\mathbf{i}_{n_\omega}]$. Using the relation from (3.1), the matrices also formulate the relation

$$\underbrace{\mathbf{Y}}_{8\times n_\omega} = \mathbf{A} \underbrace{\mathbf{H}}_{n_s\times n_\omega} \tag{3.2}$$

which solves $\mathbf{A}$ through a closed-form solution

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top \underbrace{\left(\mathbf{H}\mathbf{H}^\top\right)^{-1}}_{n_s\times n_s}. \tag{3.3}$$

Notably, the typical values of n_s is 400 while n_ω is 1200. A more detailed derivation is written in Chapter 2.

Based on the derivation above, the motivations of this chapter is to alleviate the limitations of [65] with regards to (1) the learning time of the linear predictor and (2) the rigid transformation of the template. Since learning takes a long time due to the inversion of a large $n_s \times n_s \approx 400\times400$ matrix in (3.3), Section 3.3 improves on the learning strategy to achieve an instantaneous learning of an entire template [60, 61, 62]. This 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.

Moreover, having the four corners in $\boldsymbol{\mu}$, the transformation of the template is limited up to the perspective distortion. Section 3.4 attempts to alleviate this limitation by incorporting a deformable model in the parameter vector [132].

3.2 Related Work

Jurie and Dhome [65] 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. The same authors extended their approach to handle occlusions [66]. Invariance to illumination changes was introduced by Gräßl *et al*. [46]. They [47] also formulated a method on how to select the points for sampling from image data to further increase accuracy in tracking. Zimmermann *et al*. [148] 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*. [58] 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 [92] 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 schemes in Section 3.3,

which is different to the one proposed by Jurie and Dhome [65] and enables online learning of templates.

3.2.1 Comparison of Deformable Trackers

Literature regarding deformable template tracking is classified into two well-known categories – feature-based methods [29, 104, 105, 146, 147] where features are used to characterize the images and to estimate the deformation from one image to the other; and, pixel-based methods [11, 43, 64, 88] where the intensity of the images is directly used to find the deformation.

Feature-based methods. The steps in feature-based methods can be summarized into feature correspondence search, outlier rejection mechanism, and finally deformation model estimation. To detect and match features, most of the literature [104, 105, 121, 146, 147] use keypoint descriptors such as SIFT [84], SURF [12] or randomized trees [78]. Furthermore, given the correctly matched features from the reference image to the target image, the thin plate splines of Bookstein [19] is commonly used to estimate the deformation between the matched keypoints because it offers a closed-form solution. Another approach with a closed-form solution is from Salzmann and Fua [121] that reconstructs a 3D model from dense correspondences between the reference template configuration, and the input image relying on the easy to learn local surface deformation models and non-linear constraints between vertices of the local surface parts.

Unlike rigid transforms where we estimate a 2D homography with a maximum of 8 d.o.f. and easily remove incorrectly matched features by using RANSAC [41], deformable transforms have significantly more degrees of freedom which makes it difficult to identify whether a matched features is an inlier or an outlier. Thus, a number of works [104, 105, 80, 141] have focused on the outlier rejection for deformable transforms. One of the prominent works is from Pizarro *et al*. [105] where they assume that the deformation is locally smooth by using a Delaunay triangulation and iteratively build a set of strong inliers. Another is from Pilet *et al*. [104] where, aside from the quadratic deformation energy that penalizes local surface curvature, they introduce a correspondence energy that includes a ridged shape robust estimator with a decreasing radius of confidence such that, as the radius of confidence decreases, the estimator becomes more selective in choosing inliers. In addition, Tran *et al*. [141] show how to utilize a "simple" RANSAC algorithm with Radial Basic Function (RBF).

Therefore, the problem in using feature-based methods is that they depend on the repeatability of the features and require a sufficiently large number of features in the reference image to have enough inliers when matched with the target image. This generates a time-consuming outlier rejection and constraints their application to large templates. As a result, feature-based template matching has reached real-time application at only 10 fps [104].

Pixel-based methods. On the other hand, less attention has been given to deformable template matching using pixel-based methods to perform in real-time applications. These methods are usually associated with the energy minimization-based image registration, where their energy is composed of the data term that expresses the similarity

measure of two images and the smoothing term that defines the rigidity of the deformation. Notable works in this field is the self-occlusion reasoning of Gay-Bellile *et al*. [43] where they added a term called shrinker that shrinks the deformation to prevent it from folding at self-occluded regions.

On one hand, the main advantage of using an energy-based image registration is that it can achieve a subpixel accuracy [64] and can handle self-occlusion [43]; on the other, it requires an extensive amount of time to complete as well as a good initialization to avoid local minima.

Works that have accomplished real-time application include the Active Appearance Models (AAM) [30, 91]. It relies on applying Principal Component Analysis (PCA) on hundreds of labelled images to find the mean and eigenvectors of the shape, which defines the geometric structure, and appearance, which includes the image intensities. In this way, their model can be represented by the linear combination of the resulting mean and eigenvectors. Another real-time method that exploits dense pixel information and handles deformations is the work of Malis [88]. He mainly concentrates on energy-minimization based template tracking called Efficient Second-order Minimization (ESM) which is designed for tracking rigid motion of planar templates and extends it with thin plate splines to handle surface deformations. However, their method for deformable surfaces runs at 3.5 seconds per frame.

Nonetheless, we introduce a pixel-based method with linear predictors in Section 3.4 that extends the rigid template tracking of Jurie and Dhome [65] to handle non-rigid deformations where we use the Free-Form Deformations (FFD) based on cubic B-Splines [45, 77] as our model. Based on our evaluations, our approach is stable for several types of deformations and can compete against the feature-based method of [104] in terms of precision. However, our method is running at less than 1 ms per frame which makes it the fastest deformable 2D tracking method up-to-date.

3.3 Fast Learning Strategy

When we look at (3.2), due to the pseudo-inverse of $\mathbf{H}$ that involves the inverse of an $n_s \times n_s$ matrix $\mathbf{H}\mathbf{H}^\top$, the approach for learning the linear predictor $\mathbf{A}$ as proposed by Jurie and Dhome [65] is very time consuming and not applicable for learning on-the-fly. Therefore, this section proposes two methods that are simple yet powerful to learn much faster than [65]. These involve a reformulation of (3.2) [61, 62] and an image compression through DCT [60, 61]. Towards the end of this section, the goal is to propose a learning strategy that can instantaneously learn a template such that, for a multi-template tracking application, a new template is learned on-the-fly even while tracking the previously learned templates. Notably, one of our papers [60] is a recipient of the *Best Application Paper Honorable Mention* from the 2012 Asian Conference on Computer Vision (ACCV).

3.3.1 Reformulation

To reduce learning time, the first approach uses the pseudo-inverse of $\mathbf{Y}$, instead of $\mathbf{H}$, in order to generate a much faster learning process. Using this approach, (3.2) leads to

$$\mathbf{I} = \mathbf{A}\mathbf{H}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1} \tag{3.4a}$$

$$= \mathbf{A}\mathbf{B} \tag{3.4b}$$

where $\mathbf{B} = \mathbf{H}\mathbf{Y}^\top(\mathbf{Y}\mathbf{Y}^\top)^{-1}$ is an $n_s \times 8$ matrix and $\mathbf{I}$ is an identity matrix. Hence, solving for $\mathbf{A}$,

$$\mathbf{A} = (\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top. \tag{3.5}$$

The pseudo-inverse is applied differently in (3.4a) and (3.5), 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 [103].

The computation of the matrix $\mathbf{A}$ involves the inversion of two matrices. However, since both $\mathbf{Y}\mathbf{Y}^\top$ and $\mathbf{B}^\top\mathbf{B}$ are 8×8 matrices, computing the inverse of two 8×8 matrices is much faster in comparison to inversion of an $n_s \times n_s$ matrix. In fact, $\mathbf{Y}\mathbf{Y}^\top$ can be precomputed when the parameters of the random perturbations in the learning dataset are kept constant for different templates. As a consequence, only a single 8×8 matrix has to be inverted online.

Since the linear mapping denoted by $\mathbf{A}$ 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 (3.1) in tracking. Unlike the normalization used on $\mathbf{H}$ in Section 2.3.2 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 in the evaluation (see Figure 3.4).

Solving (3.3) from [65] actually corresponds to approximating $\mathbf{Y}$ by orthogonally projecting it on $\mathbf{H}$. On the other hand, solving (3.5) in this 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 [65], the noise has more effect. This makes their approach more sensitive to noise. We also prove this in the experiments.

Updating the learned A. Similar to [51], we also investigate on the capability to add new training samples to $\mathbf{A}$ after learning by using Sherman-Morrison formula. Hinterstoisser *et al*. [51] rely on (3.3) and efficiently update 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 [65] do. We propose to derive $\mathbf{S}$ from an existing $\mathbf{A}$ using (3.3) by

$$\mathbf{A} = \mathbf{Y}\mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top)^{-1} \tag{3.6a}$$

$$= \mathbf{Y}\mathbf{H}^\top\mathbf{S} \tag{3.6b}$$

$$= \mathbf{D}\mathbf{S} \tag{3.6c}$$

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

$$\mathbf{S} = \mathbf{D}^\top(\mathbf{D}\mathbf{D}^\top)^{-1}\mathbf{A}. \tag{3.7}$$

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.

Later, the evaluation in Figure 3.4 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

$$\hat{\mathbf{S}} = \left(\mathbf{S}^{-1} + \delta\mathbf{i}_{n_\omega+1}\delta\mathbf{i}_{n_\omega+1}^\top\right)^{-1} = \mathbf{S} - \frac{\mathbf{S}\delta\mathbf{i}_{n_\omega+1}\delta\mathbf{i}_{n_\omega+1}^\top\mathbf{S}_I}{1+\delta\mathbf{i}_{n_\omega+1}^\top\mathbf{S}\delta\mathbf{i}_{n_\omega+1}}, \tag{3.8}$$

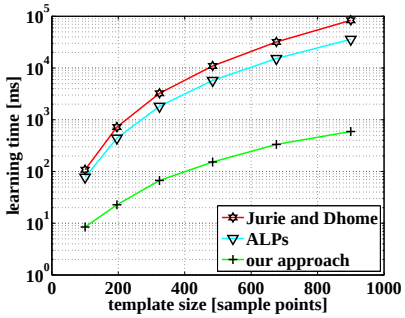
where $\delta\mathbf{i}_{n_\omega+1}$ is a vector of image value differences obtained from a new random transformation applied to the sample points. Note that before computing the updated linear predictor using (3.6b), the matrices $\mathbf{H}$ and $\mathbf{Y}$ are updated by concatenating them with the new training samples. For the normalization of the parameter differences, the same normalization as applied to the original learning is implemented.

Computational Complexity

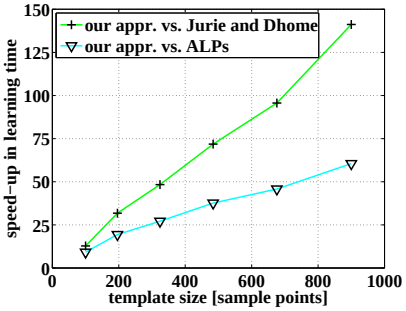
In the first evaluation, the computational complexity of this approach is compared to the approach of Jurie and Dhome [65] and Holzer *et al*. [58]. Our algorithm is divided into three parts – learning linear predictor matrices, tracking using the learned $\mathbf{A}$ and updating it while tracking. This section mainly focuses on the amount of time that each part requires to finish in relation to the number of sample points used.

Learning. The main contribution is reflected on the learning time. We show in Figure 3.1(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 [65] and Holzer *et al*. [58]. This difference is emphasized in Figure 3.1(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 [65] and more than 50 times faster than the approach of Holzer *et al*. [58].

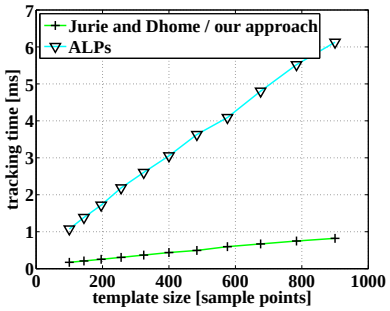
Tracking. Both the original approach [65] and the proposed approach have similar tracking time because the time needed to de-normalize the parameter updates in our approach is negligible. Furthermore, the measure of tracking time per frame with respect to template size in Figure 3.1(c) demonstrates that our approach can easily reach frame rates higher than 1000 fps even with large templates. In contrast to Holzer *et al*. [58], 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*. [13] takes about 10 ms for tracking the same template.



(a) Learning time



(b) Speed-up



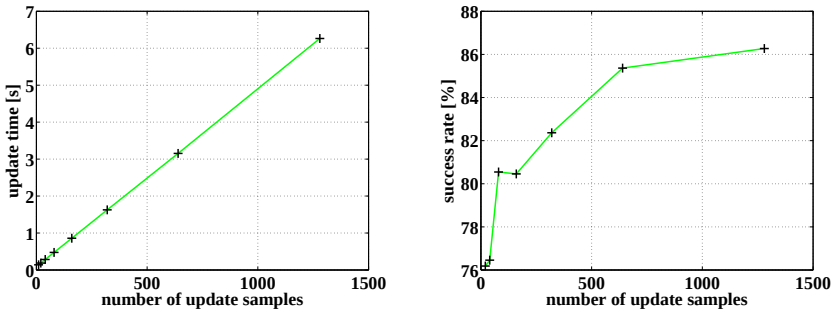
(c) Tracking time

Figure 3.1: (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 [65], by Holzer *et al*. [58] (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.

Updating. The updating process is a way of adding new training samples to a learned linear predictor during tracking. Figure 3.2(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 Figure 3.2(b), we show an exemplary improvement of tracking robustness when updating the linear predictor with a specific number of update samples. The next section discusses more on the tracking robustness in updating.

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. Accuracy is measured 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



(a) Update time (b) Success rate

Figure 3.2: (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.

Internet (see Figure 3.3). 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, the image is corrupted with noise sampled from a Gaussian distribution before applying the random transform.

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. Note that a similar evaluation is also conducted in Sections 3.3.2 and 3.4.

Here, the proposed approach is compared to the methods of Jurie and Dhome [65], Holzer *et al*. [58], and Benhimane *et al*. [13]. For the proposed approach, we considered three different cases:

- *unnormalized*, where no normalization on the parameter differences is implemented;
- *normalized*, where the parameter differences is normalized before learning $\mathbf{A}$ and de-normalize them during tracking; and,
- *updated*, where The parameter differences is normalized and the linear predictor is updated 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 (in-plane rotation), scale and viewpoint

Figure 3.3: The dataset used for synthetic experiments.

change (out-of-plane rotation). 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, the test image is corrupted 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 predictor use $3 \cdot 18 \cdot 18 = 972$ training samples; while for the non-linear approach of Benhimane *et al*. [13], the complete template is used without subsampling.

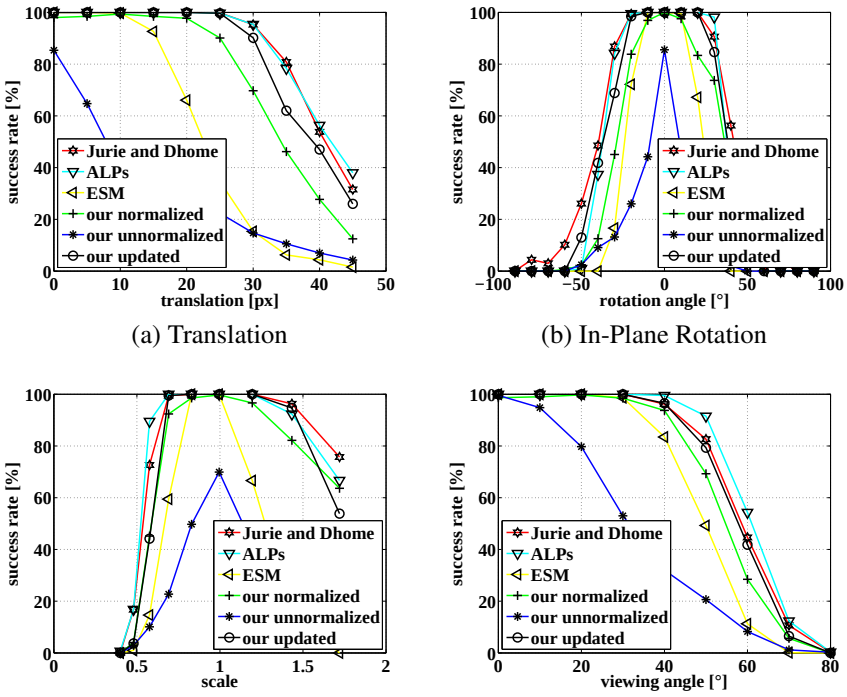
Normalization of parameter differences. As previously mentioned, normalizing the parameter difference matrix $\mathbf{Y}$ before learning is important for our approach. To emphasize this, we included the results of the unnormalized approach in Figure 3.4. It clearly shows that the unnormalized approach 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 [65]. On the other hand, the outcome from Holzer *et al*. [58] also shows that it performs similarly well as Jurie and Dhome [65]. 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*. [13].

Number of samples points. In Figure 3.5, we compare the tracking robustness in relation to the number of sample points. Note that all the results show similar behavior across different transformations. The normalized approach replicates the results of Jurie and Dhome [65] 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. Figure 3.2(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 predictor to the transformed images, we checked how often the translation was correctly estimated. This was done for linear predictor matrices 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 Figure 3.4 that updating brings the tracking performance closer to the original learning approach of [65].

Sensitivity to Noise. A comparison among the different methods with respect to noise sensitivity is presented in Figure 3.6. 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 Figure 3.6 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 Figure 3.4, we illustrate in Figure 3.6(a) that the tracking robustness of our



(a) Translation (b) In-Plane Rotation

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

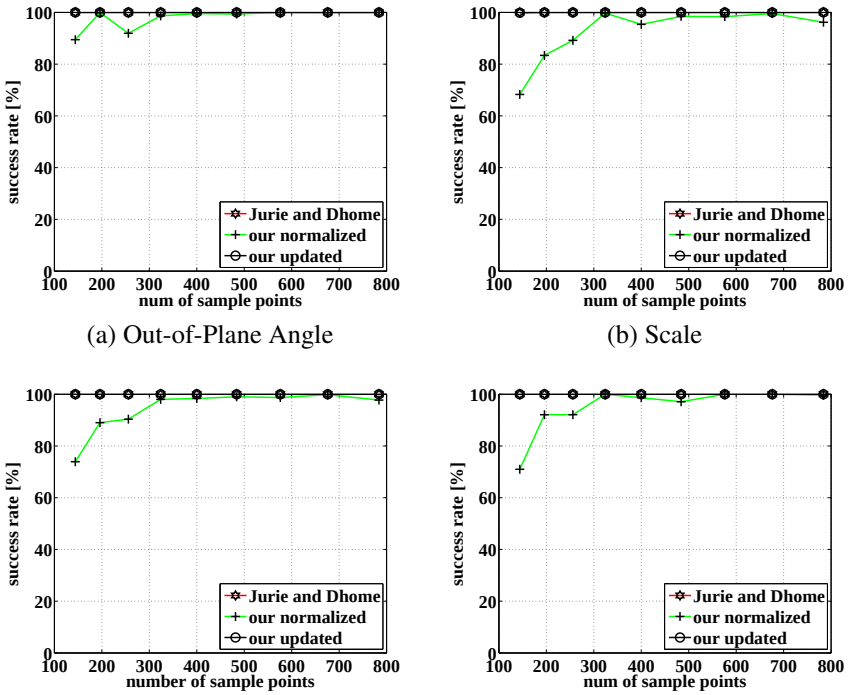
Figure 3.4: Comparison of the approach of Jurie and Dhome [65], Holzer *et al.* [58] (ALPs), Benhimane *et al.* [13] (ESM), as well as our approach with and without normalization, and with updated predictors. We consider four different types of motions as specified. The success rate indicates the percent of successful estimation of the applied motions.

approach outperforms the original approach in terms of sensitivity to noise. An evidence for this is shown in Figure 3.6(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 [65] 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.

3.3.2 Dimensionality Reduction

Considering that the main drawback of learninig through [65] in Section 3.1 is the inversion of a large matrix $\mathbf{H}\mathbf{H}^\top$, the idea behind the second learning approach is to compress the image difference vectors $\delta\mathbf{i}_i$ before using them to learn the linear predictor. By reducing the dimensionality of $\delta\mathbf{i}_i$ from n_s 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



(a) Out-of-Plane Angle (b) Scale

(c) Translation (d) In-Plane Rotation

Figure 3.5: Comparison of the success rate in tracking with respect to the number of sample points for the approach of Jurie and Dhome [65], as well as our approach with normalization and with updated predictors.

(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

$$\mathbf{U} = \mathrm{DCT}(\mathbf{V}) = \mathbf{C}\mathbf{V}\mathbf{C}^\top \tag{3.9}$$

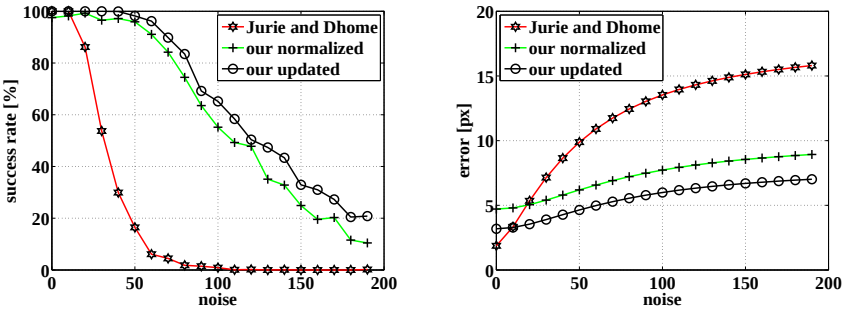
where the elements of the matrix $\mathbf{C}$ are defined as

$$\mathbf{C}_{i,j} = \sqrt{\frac{\alpha_i}{k}} \cos\left[\frac{\pi(2j+1)i}{2k}\right] \tag{3.10}$$

with

$$\alpha_i = \begin{cases} 1 & \text{if } i = 0 \ , \\ 2 & \text{otherwise} \ . \end{cases} \tag{3.11}$$

After transforming $\delta\mathbf{i}_i$ as $\delta\hat{\mathbf{i}}_i = \mathrm{DCT}(\delta\mathbf{i}_i)$, we form $\hat{\mathbf{H}} = [\delta\hat{\mathbf{i}}_1, \delta\hat{\mathbf{i}}_2, \ldots, \delta\hat{\mathbf{i}}_{n_\omega}]$. However, since we reshaped the samples from a 2D template into a vector, (3.9) cannot be directly



(a) Success rate (b) Average Error

Figure 3.6: Comparison of our approach to the approach of Jurie and Dhome [65] 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.

applied to $\delta\mathbf{i}_i$. Therefore, we formulate an $n_s \times n_s$ 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 $\delta\mathbf{i}_i = \text{reshape}(\mathbf{V}_i)$, we compute $\mathbf{W}_{DCT}$ as

$$\mathbf{W}_{DCT} = (\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_{n_s}) \tag{3.12}$$

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. By setting a single element to 1, the set of matrices $\{\mathbf{B}_s\}_{s=1}^{n_s}$ are a base of the image space of the template and the set of vectors $\{\mathbf{b}_s\}_{s=1}^{n_s}$ are the DCT projections of this base. This way, we can directly compute the 2-dimensional DCT of our image difference vectors as

$$\delta\hat{\mathbf{i}}_i = \mathbf{W}_{DCT}\delta\mathbf{i}_i \;\;\Rightarrow\;\; \hat{\mathbf{H}} = \mathbf{W}_{DCT}\mathbf{H}\,. \tag{3.13}$$

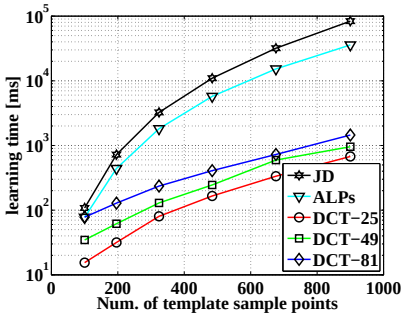
In relation to the original learning formula in (3.3), we reformulate this by using the relation

$$\mathbf{H} = (\mathbf{W}_{DCT})^{-1}\hat{\mathbf{H}}\,. \tag{3.14}$$

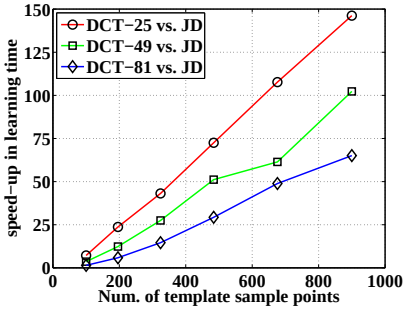
After subsituting $\mathbf{H}$ from (3.14) to (3.2) and solve for the linear predictor $\mathbf{A}$ as

$$\begin{aligned}
\mathbf{A}\mathbf{W}_{DCT}^{-1}\hat{\mathbf{H}} &= \mathbf{Y} \\
\mathbf{A}\mathbf{W}_{DCT}^{-1} &= \mathbf{Y}\hat{\mathbf{H}}^\top\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1} \\
\mathbf{A}\mathbf{W}_{DCT}^{-1}\mathbf{W}_{DCT} &= \mathbf{Y}\hat{\mathbf{H}}^\top\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1}\mathbf{W}_{DCT} \\
\mathbf{A} &= \mathbf{Y}\hat{\mathbf{H}}^\top\left(\hat{\mathbf{H}}\hat{\mathbf{H}}^\top\right)^{-1}\mathbf{W}_{DCT}\,.
\end{aligned} \tag{3.15}$$

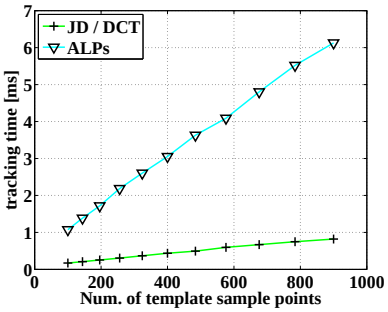
To reduce the necessary computational load, we apply a dimensionality reduction by defining an $n_r \times n_s$ submatrix $\mathbf{W}_{DCT}^{(n_r)}$ with $n_r < n_s$, such that the necessary matrix inversion is no longer applied to an $n_s \times n_s$ matrix but rather to an $n_r \times n_r$ matrix.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



(a) Learning time



(b) Speed-up



(c) Tracking time

Figure 3.7: Comparison of timings for the approach of Jurie and Dhome [65] (JD), Holzer *et al*. [58] (ALPs), and ours (DCT-x). (a) Comparison of learning time. (b) Obtained speed-up of our approach with respect to Jurie and Dhome [65]. (c) Comparison of tracking time.

final linear predictor is then computed as

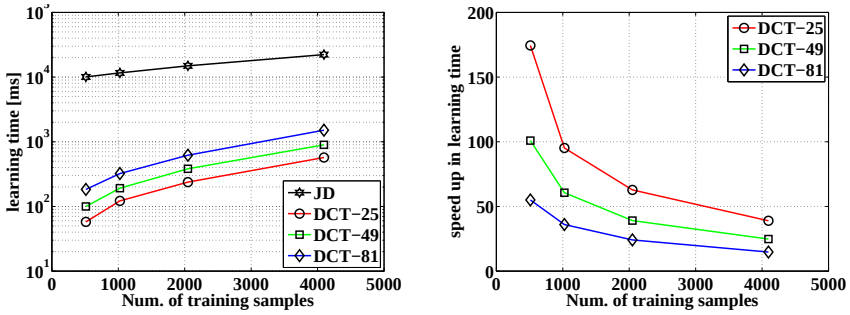
$$\mathbf{A}^{(n_r)} = \mathbf{Y}\hat{\mathbf{H}}^{(n_r)\top}\left(\hat{\mathbf{H}}^{(n_r)}\hat{\mathbf{H}}^{(n_r)\top}\right)^{-1}\mathbf{W}_{DCT}^{(n_r)}. \tag{3.16}$$

with $\hat{\mathbf{H}}^{(n_r)} = \mathbf{W}_{DCT}^{(n_r)}\mathbf{H}$. Later, our evaluation shows 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.

Computational Complexity

Similar to Section 3.3.1, 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 [65] as well as Holzer *et al*. [58].

Learning. In Figure 3.7(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 [65] and Holzer *et al*. [58]. As



(a) Learning time (b) Speed-up

Figure 3.8: 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 [65] (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.

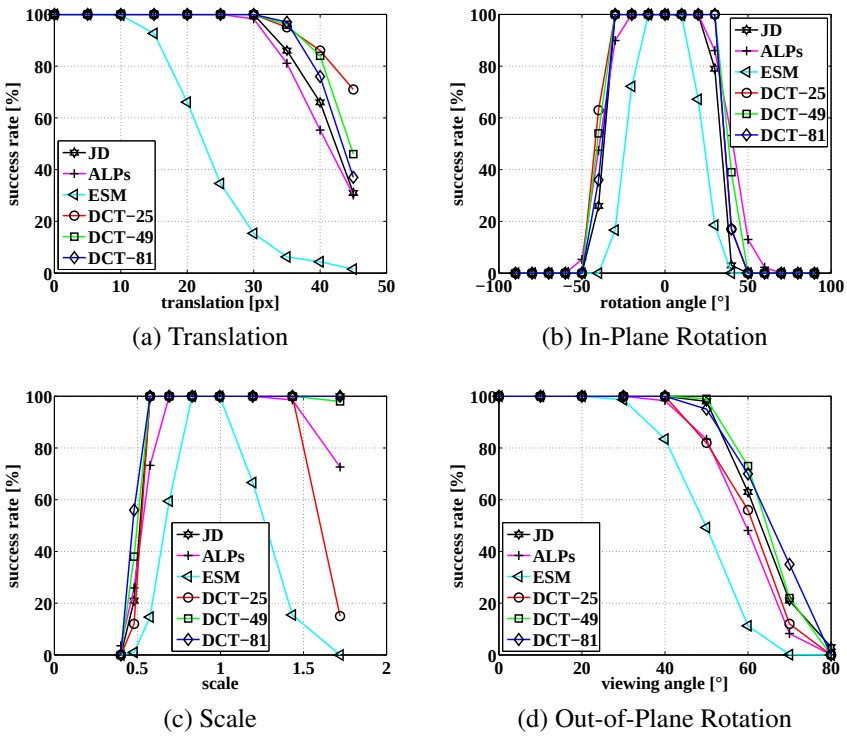
expected, using less DCT coefficients for learning decreases the necessary time. This difference is emphasized in Figure 3.7(b) where it is evident that for templates with more than 800 sample points (*e.g.* 30×30), the proposed approach is more than two orders of magnitude faster, *i.e.* almost 150 times faster, than [65] if 25 DCT coefficients are used. Using 81 DCT coefficients, it is still approximately 70 times faster.

Figure 3.8 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 $\mathbf{A}$. Although this comes hand in hand with a decrease in tracking performance, we show in the next section that our approach is much more robust against this kind of reduction compared to [65].

Tracking. Both [65] 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 Figure 3.7(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.* [58], 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.* [13] takes about 10 ms for tracking the same templates.

Robustness

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.



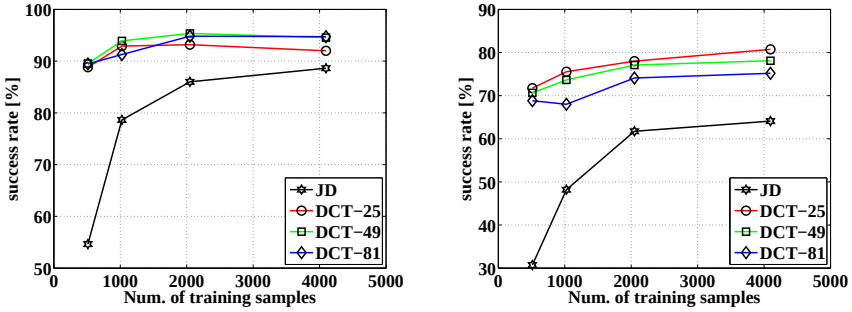
(a) Translation (b) In-Plane Rotation

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

Figure 3.9: Comparison of tracking performance for the approaches proposed by Jurie and Dhome [65] (JD), Holzer *et al.* [58] (ALPs), Benhimane *et al.* [13] (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.

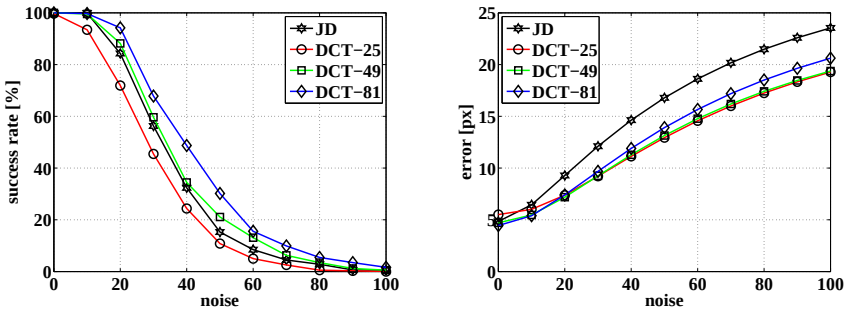
Applying these disturbances to the test images, tracking is considered successful if the mean pixel distance between the reference template corner points and the tracked template corner points, which are back-projected into the reference view, is less than 5 pixels. Hence, robustness is measured as the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

Number of sample points. In Figure 3.9, 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 [65], Holzer *et al.* [58], and the non-linear approach of Benhimane *et al.* [13]. Hereby, the training stage as it is applied for the methods based on **A** leads to significantly better results than obtained using the non-linear approach of Benhimane *et al.* [13]. Comparing our approach with that of Jurie and Dhome [65] 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 [58] tends to give slightly worse results



(a) [30, 40] (b) [35, 45]

Figure 3.10: 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.



(a) Success rate (b) Average error

Figure 3.11: Comparison of sensitivity to noise for the approach proposed by Jurie and Dhome [65] and our approach.

than that of [65]. 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 Figure 3.10), we see that the approach of Jurie and Dhome [65] lacks robustness when reducing the number of training samples while our approach still keeps high tracking performance even with reduced training examples. This property is useful to even further decrease the learning time if necessary, as we showed in Figure 3.8.

Sensitivity to Noise. The results presented in Figure 3.11 compare our proposed approach with Jurie and Dhome [65] 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. Figure 3.11 (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 [65]. Looking at Figure 3.11 (b), we see that our approach, in general, gives a smaller mean error in the tracking results.

3.4 Deformable Template Tracking

The limitation to track rigid templates is due to the parameterization of the transformation. Considering the location of the template's four corners in $\boldsymbol{\mu}$, its 8 d.o.f. is bounded to distort up to a perspective transformation. In this section, we investigate on adding more points on the template that controls its distortion in order to perform deformable tracking.

3.4.1 Deformable Model

Our 2D deformation model is based on the Free-Form Deformations (FFD) using cubic B-Splines. This model is composed of control points that are uniformly arranged in a $K \times L$ grid around the template. Each control point stores a displacement vector that defines the movement of the template. Therefore, the deformation Ψ of a pixel $\mathbf{x} = (x, y)^\top$ is computed as

$$\Psi \circ \mathbf{x} = \mathbf{x} + \mathcal{D}(\mathbf{x}) \tag{3.17}$$

such that the displacement vector $\mathcal{D}(\mathbf{x})$ of the pixel $\mathbf{x}$ is interpolated from the displacement vector $\mathbf{d}_{i,j}$ at the control points (i, j) using [45]

$$\mathcal{D}(\mathbf{x}) = \sum_{l=0}^{3} \sum_{m=0}^{3} B_l(u) B_m(v) \mathbf{d}_{i+l,j+m} \tag{3.18}$$

where $i = \lfloor x/\delta x \rfloor - 1$, $j = \lfloor y/\delta y \rfloor - 1$, $u = x/\delta x - \lfloor x/\delta x \rfloor$ and $v = y/\delta y - \lfloor y/\delta y \rfloor$ with the control point spacings $\delta x = W/(K-1)$ and $\delta y = H/(L-1)$, while B_l and B_m are the cubic B-Spline interpolation coefficients [77].

3.4.2 Tracking with Linear Predictor

In tracking, the paremeters of the deformation model is applied on $\boldsymbol{\mu}$. Instead of locating the four corner points of the template, we generalize to an arbitrary n_c control points the defines the distortion of the template. However, a similar tracking approach applies. Looking at Section 3.1, the image intensity difference vector $\delta\mathbf{i}$ is computed to find the parameter updates $\delta\boldsymbol{\mu} = [\mathbf{d}_c]_{c=1}^{n_c}$ for all the n_c control points in the deformation model. The relation between the two vectors follow (3.2). Notably, the same learning strategies to estimate $\mathbf{A}$ also apply to the deformable tracker as Section 2.3.2 through [65] and Section 3.3 through [60, 61, 62].

Homography Update. Since the estimated displacements are relative to the coordinate system of the reference template, we keep track of a homography $\mathbf{T}_t$ which approximately maps the current control points from the predicted pose of the template in the current frame to the reference template. $\mathbf{T}_t$ is estimated by computing the homography between the corner point positions in the reference coordinate system and their

location obtained in the previous frame. To apply the displacements obtained from $\mathbf{A}$, we initially warp all control points back into the reference coordinate system.

Coarse-to-Fine Strategy. Several linear predictor are learned for a single template that compensate for large to small deformations such that each $\mathbf{A}_l$ refines the previously estimated parameter vector. The coarse-to-fine approach is applied in a hierarchical way. At the first two levels of hierarchy, the linear predictor are learned with the coarse 2×2 control points; while, in the last three, they are learned with 3×3 control points. Note that between the second and third level, the number of considered control points increases and we linearly interpolate the positions of the additional control points. Furthermore, each linear predictor is iteratively used to generate better results. In this work, three iterations for each linear predictor are performed.

3.4.3 Comparison

We perform quantitative evaluations on our approach where each template is 150×150 pixels. Learning is parameterized with $n_\omega = 5 \cdot n_s$ training samples where n_s is the number of sample points.

The first quantitative evaluation uses a dataset of 20 images and evaluates the tracking robustness of our method with respect to rigid and non-rigid transformations. It also includes the comparison with the fast learning methods from Section 3.3 where we modify (3.3) to avoid the inversion of the large matrix $\mathbf{H}\mathbf{H}^\top$.

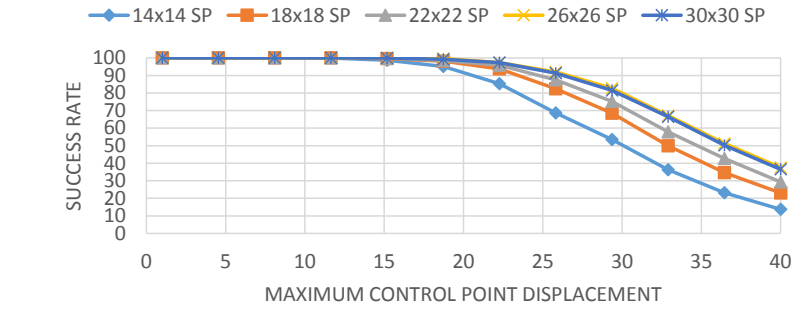
Another quantitative evaluation involves three video sequences that consider problems such as camera noise and motion blur from real deformations. These sequences are used to test the precision of our algorithm where we compute the distance of 10 manually selected points on the template from their tracked positions to the ground truth. Consequently, our results are compared to the ones from the feature-based approach of Pilet *et al*. [104].

Robustness

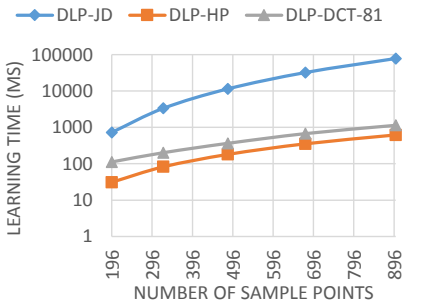
A similar evaluation as Section 3.3 is used to test the robustness through the dataset from Figure 3.3. In addition to the rigid transformation, including translation, scale, in-plane rotation and out-of-plane rotation, we also distort the test images using non-rigid transformations produced from FFD where control points are located on the whole image. This FFD is parameterized using the maximum displacement that a control point can randomly move without overlapping each other.

The objective is to remove environmental factors such as noise and motion blur; to have full control of the warping parametrization; to determine the range of movements that our approach can handle; and, to find the optimum parameters. Furthermore, we look into the rigid transforms to find out whether using deformable warping parametrization in $\delta\boldsymbol{\mu}$ compromises the linear transformations that [65] can manage.

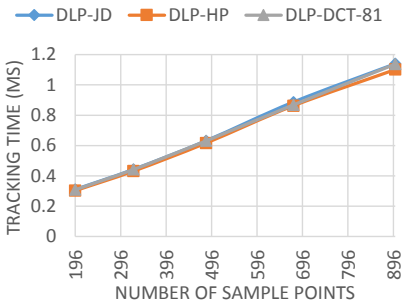
In these evaluations, two parameters are important – time and robustness. The optimum case is to perform at high-speed with a fast learning procedure without deteriorating the tracking robustness. We define tracking robustness as the percent of successfully recovered template deformations after generating random warps on each



(a) Tracking Robustness



(b) Learning time



(c) Tracking time

Figure 3.12: This shows the (a) tracking robustness, (b) learning time and (c) tracking time with respect to the number of sample points $n_s = K \times K$ when using different learning modalities – JD [65], HP [62] and DCT [59]. Using the optimum $26 \times 26 = 676$ sample points arrangement, we can learn the template in 353.38 ms and track in 0.87 ms.

image of the dataset. Here, tracking is successful if the average distance of the back-warped sample points of the tracked template to the original sample points on the image is less than 1.5 pixels.

Number of sample points. From the previous section, we know that the sample points subsample the template and, in effect, replaces the template in all computations. Thus, it is crucial to determine the optimum number of sample points such that the time-performance ratio is optimal. We evaluate the tracking robustness in Figure 3.12(a) with respect to different number of sample points. It shows that there is no significant improvement between $26 \times 26 = 676$ and $30 \times 30 = 900$ sample points. Thus, the optimum number of sample points for our application is 26×26.

Contrary to the rigid transformations from [65], the deformable tracker requires more sample points because it has more parameters. This is one of the reason why we use a maximum of 3×3 control points as discussed in Section 3.4.2. Another reason is because 3×3 control points are sufficient to give a realistic tracking performance as shown in Figure 3.17.

Furthermore, Figure 3.12(b-c) shows the average learning and tracking time while using different learning techniques [59, 62, 65]. For a 26×26 sample point arrangement,

we can learn at a minimum of 353.38 ms using [62] and track at approximately 0.87 ms. In comparison to other deformable template tracking algorithms, we claim that this method achieves real-time tracking at the minimum time.

Synthetic evaluation. Using the optimum 26×26 samples points, we compare the performance of our algorithm with the rigid template tracker of Jurie and Dhome [65] using synthetic rigid and non-rigid deformations of the images of the dataset. The objective is to identify how well our method can handle rigid transformations in comparison to the rigid template tracking method [65] and how our method perform under non-rigid deformations using different learning methods.

For the rigid transforms in Figure 3.13(a-d), the results show that there is no significant difference between [65] and our method. This signifies that our performance does not deteriorate in rigid transformations. On the other hand, it is not surprising that [65] fails in non-rigid deformations in Figure 3.13(e-f). In Figure 3.13(f), our method starts to fail after 30 pixel movements. This is because, if the displacements of the 9×9 control points are large, the resulting image has a smudge-like deformation which is not a realistic surface deformation.

Accuracy

Three real video sequences with a duration of 200 frames are used to evaluate the accuracy. These are taken using a Logitech Webcam C525. The first two sequences consist of a textured template that undergoes a flag-like deformation with different textures, while the last one is a face sequence with relatively low texture undergoing arbitrary facial expressions.

Our aim is to track the location of 10 points that have been manually selected in all sequences. To evaluate these points, we compute the average distance between the tracked points, using our method and the tracking-by-detection method of Pilet *et al*. [104], from the manually provided ground truth point locations. The points are initialized in the sequences as shown in Fig 3.15. For [104], the authors kindly provided the results for the sequences.

To generate a realistic flag-like deformation, we induce a fast motion on one corner of the template which results in a motion blur. Throughout the first two sequences, the largest mean error is 3.6 pixels for our approach as shown in Figure 3.14 and 7.0 pixels for [104]. Notably, when we backproject the template using the results of our approach, the resulting images are very similar to the template even if frame 166 from Sequence 1 suffers from motion blur. When comparing these two approaches, we can see that the errors in tracking are very similar but our approach is 100 times faster.

On the face sequence, all distance errors are less than 5 pixels as plotted in Figure 3.14 while we show in Figure 3.15 that the frame with the highest error is the one when the head is tilted up. According to the authors of [104], their work does not work on faces because their geometric model assumes a planar surface bending smoothly without holes.

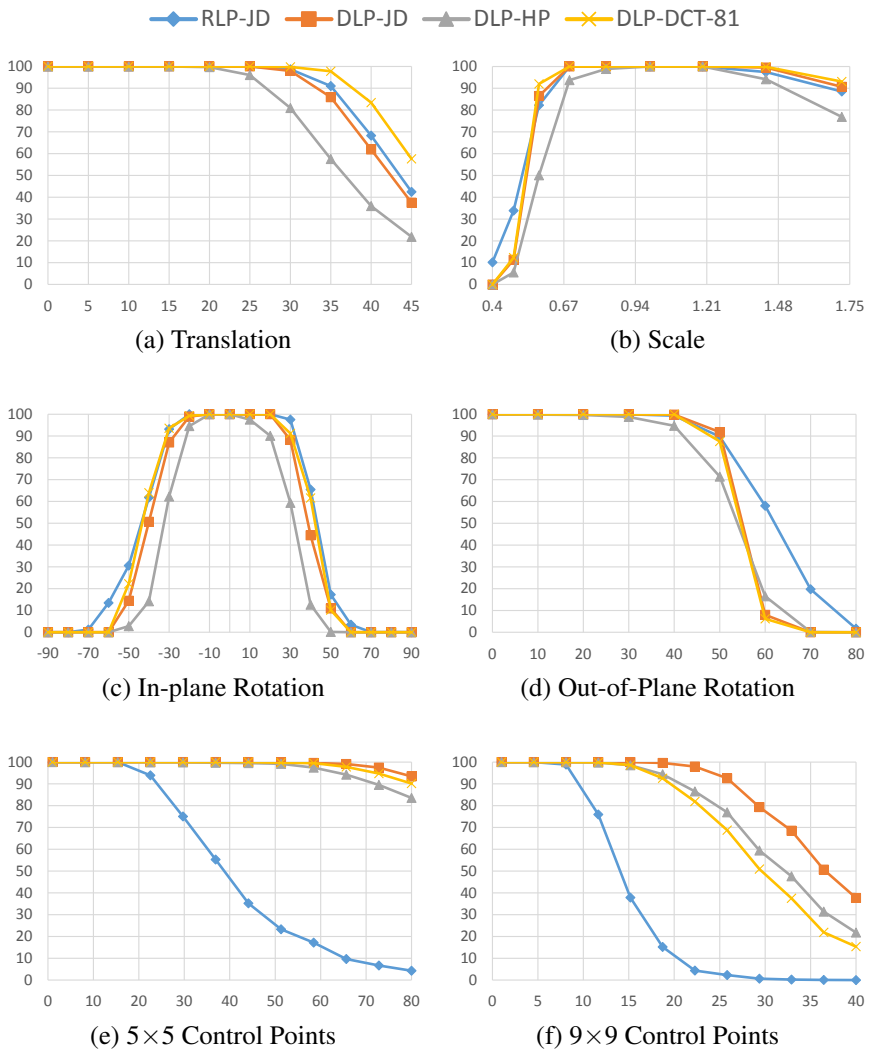


Figure 3.13: Comparison of tracking robustness between rigid linear predictor (RLP-JD) [65], and the deformable linear predictor (DLP) using different learning approaches – JD [65], HP [62] and DCT-81 [59] with $n_r = 81$ coefficients. It evaluates using (a-d) rigid as well as non-rigid transforms using FFD with (e) 5×5 and (f) 9×9 control points on the entire image where the x-axis shows the maximum control point displacement. Note that 5×5 and 9×9 does not refer to the control points of the template but rather to the deformation of the entire image.

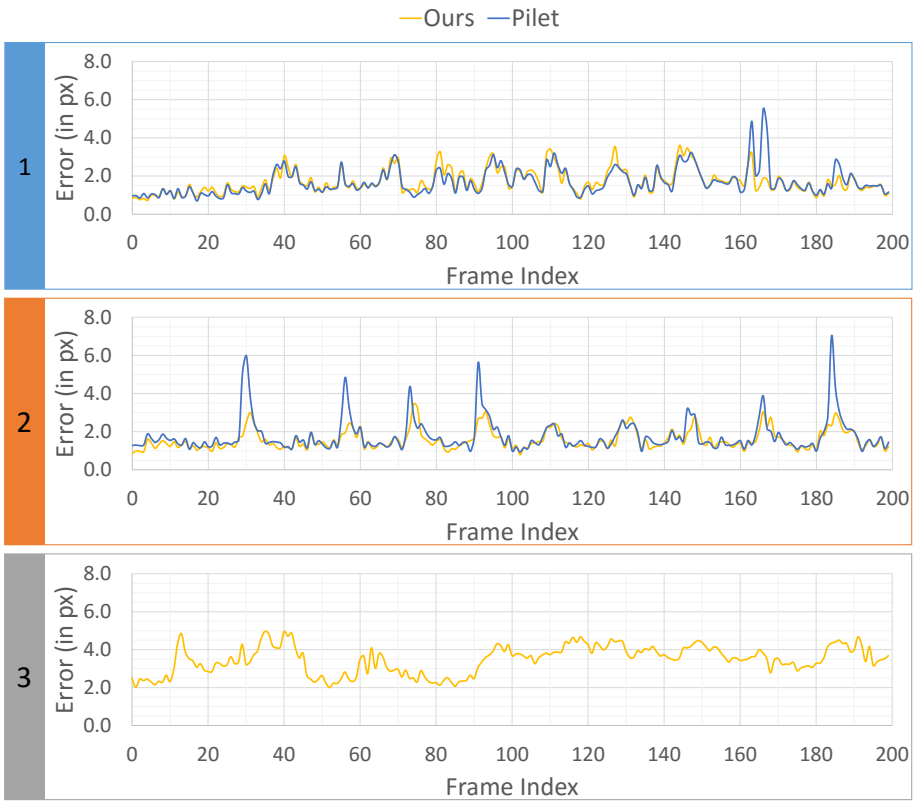


Figure 3.14: These graphs plot the resulting average distance error (in pixels) of the tracked points to its ground truth from the three sequences.

3.5 Qualitative Results

There are numerous applications to a fast learning strategies from Section 3.3. Achieving an instantaneous template learning, the proposed approaches are capable of learning and tracking multiple templates on-the-fly as shown in Figure 3.16(a-b). Such framework allows the tracker to be more robust against occlusions since, while some templates are not visible, other templates can utilize their arrangement to determine the homography of the occluded templates.

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. Together with the fast learning and fast tracking algorithms, this is especially the case for cameras used in mobile devices as demonstrated in Figure 3.16(b-d).

On the other hand, the deformable tracker is applied to wave-like deformations on a template in Figure 3.17. It also showcases a face tracking framework where the user produces different facial expressions that deforms the template.

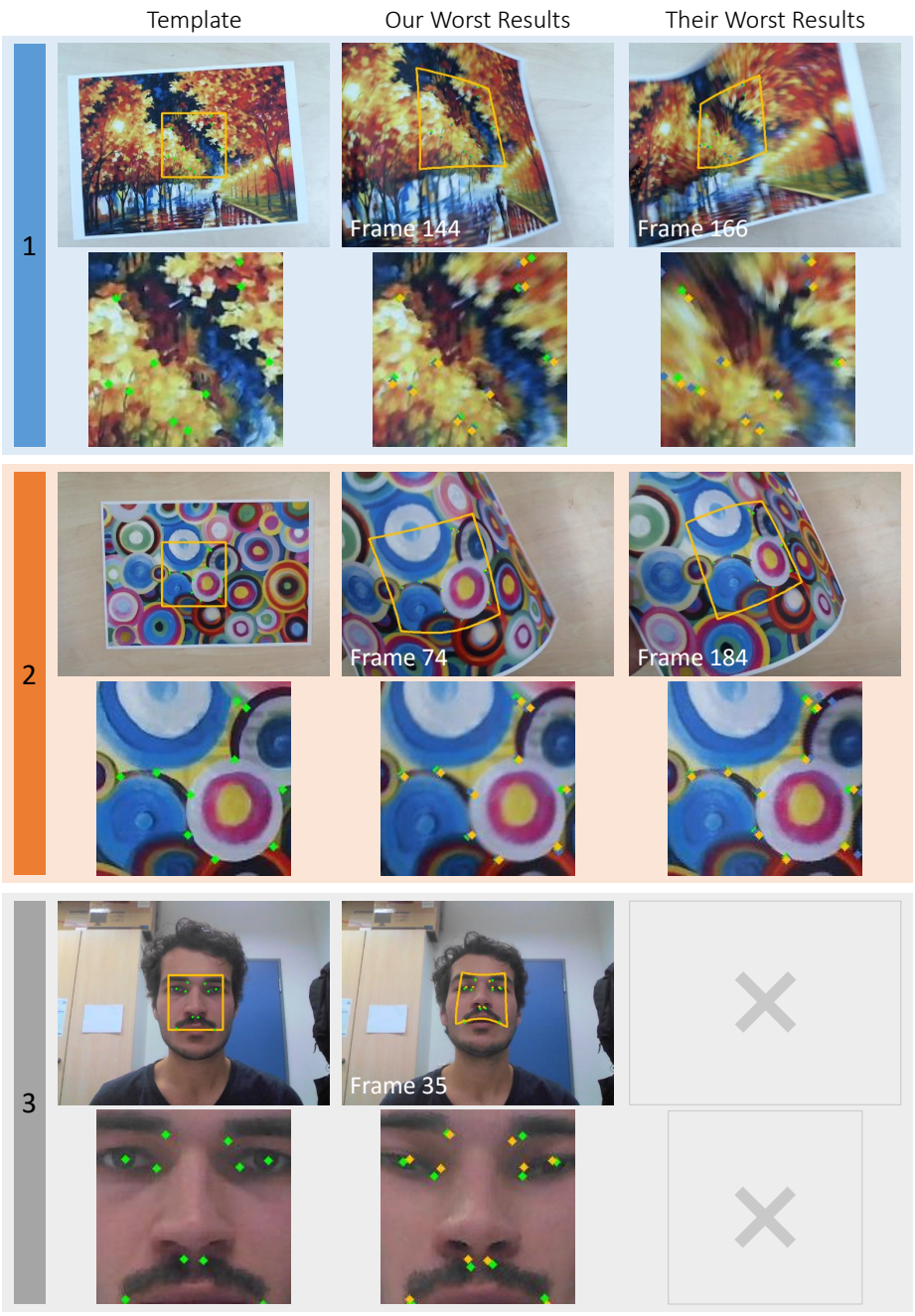


Figure 3.15: These figures show the learned template and the worst results from [104] as well as our approach. For each pair of images, the rectangular image on the left is a frame from the video sequence while the square image on the right is the backprojected template using our approach. Moreover, the ten points in all these images are labelled with green for ground truth, blue for the results from [104] and yellow for our results.

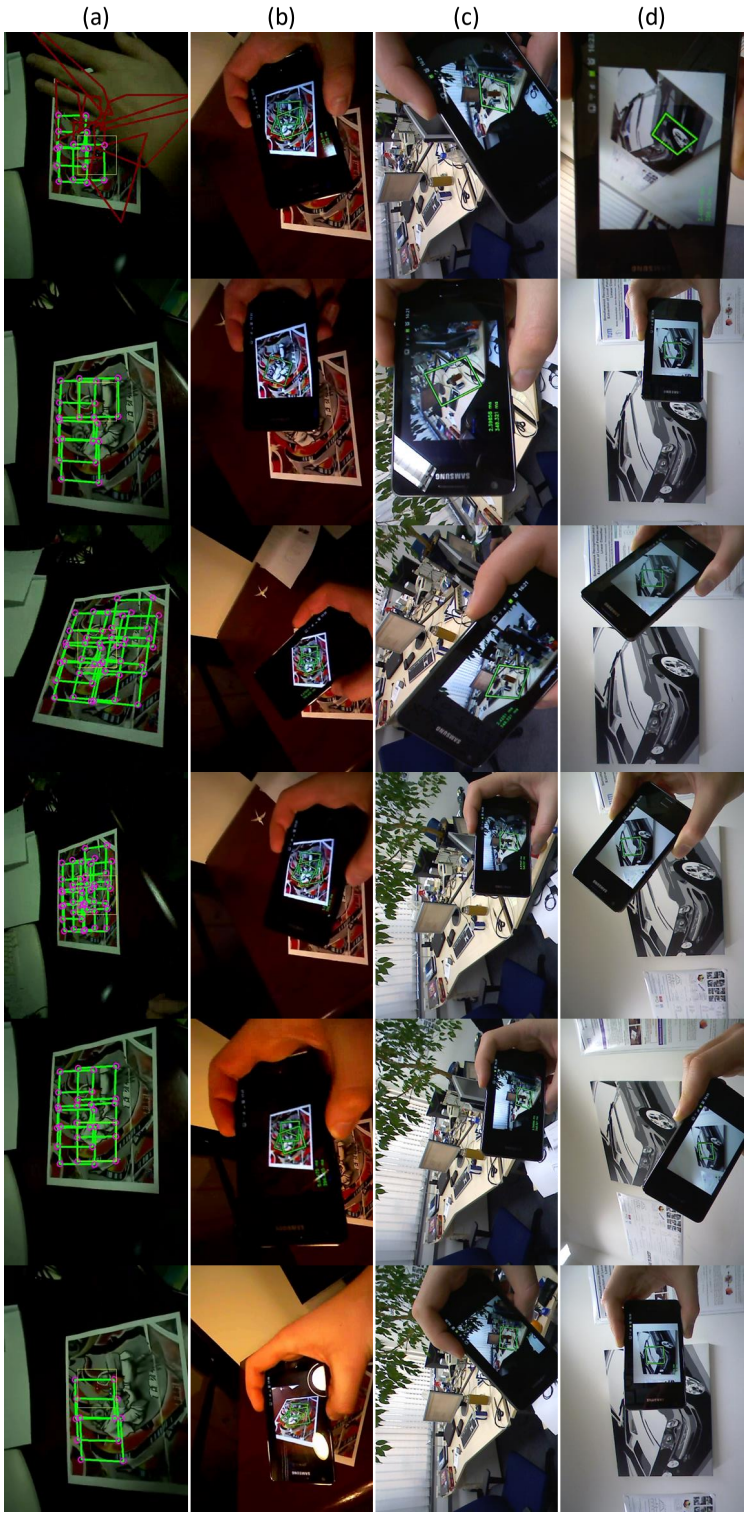


Figure 3.16: Qualitative evaluation of the fast learning strategies as an application to track multiple templates in (a-b) and an application to mobile devices in (b-d).

Figure 3.17: Qualitative evaluation of the deformable tracker as an application to template tracking in (a-b) and an application to track face expressions in (c-d).

3.6 Conclusion

In this chapter, we propose some improvements on the linear predictors from Jurie and Dhome [65]. On one hand, the first is based on the reformulation of the learning scheme and DCT in order to attain instantaneous learning of an entire template. These works allow us to learn and track multiple templates in real-time. By retaining a geometric structure between templates, the multiple templates can handle occlusions even if a template single template is sensitive to it. We also discover the robustness of the template to noise that allows the algorithm to track in low lighting conditions. With fast learning and robustness to noise, we demonstrate that these approaches are suitable for mobile devices. On the other, the second improvement is a simple extension to deformable tracking by replacing the predicted linear transformation parameters into a deformation model. For this work, we demonstrate the capacity of the algorithm to track facial deformations through a rectangular template.

Part III

3D Tracking with Depth Images

4

Chameleon Tracker – A Multi-Forest Approach

4.1 Motivation

Jurie and Dhome [65] proposes a temporal tracking approach for 2D templates where the location of template is relayed from one frame of a video sequence to the next frame. Based on image registration, their work learns a linear predictor that relates the changes of parameters between consecutive frames with the changes of image intensity on the template. Thus, in tracking, they utilize a matrix multiplication between the learned linear predictor and the changes of image intensity to determine the template's motion.

While [65] have accomplished an extremely fast tracker for 2D template tracking and the extensions from our previous works [60, 61, 62, 132] in Chapter 3 have improved it for different use cases, the applicability of such trackers is limited. Essentially, it has not found a real application where it can thrive. One limitation is the requirement of a textured template which is not always the case. Furthermore, having implemented image intensity normalization and multi-template tracking do not solve the problems in handling non-global illumination changes and occlusion on a single template.

The motivation of this chapter is to generalize the tracker at its core – the objective function. A significant observation is based on the similarity of error functions from different registration problems such as the 2D intensity-based registration from template tracking and the 3D point-based registration from object tracking. Therefore, in this chapter, we aim at building a tracker that can generalize to different registration problems. We coined the generalized framework as the *chameleon* tracker due to its adaptive nature.

In addition to the 2D template tracking, the chapter also explores the possibility of utilizing the generalized framework on a 3D model-based registration problem. Given the 3D CAD model of an object, the goal is to track this object across the depth images of a video sequence by estimating the full 6 d.o.f. pose with three rotation and three translation parameters in 3D space.

The depth images are taken from consumer depth cameras such as Microsoft Kinect, Asus Xtion, Intel RealSense, Google Tango and Structure Sensor. Although they are cheap and readily available, there are special considerations in handling the 3D data

from these sensors. The challenge of a 3D model-based tracker is that it learns from a perfect model and tracks in an imperfect depth image with noise and holes (*i.e*. missing data).

The holes on the image are invalid pixel values where the sensor failed to determine the depth value. We associate the holes as small occlusions on several parts of the object. Since the template tracking approaches that uses a linear predictor [60, 61, 62, 65, 132] are highly sensitive to occlusions, we propose to change the machine learning approach to random forest [21]. In this way, we have independent trees that looks at different regions of the object such that, when occlusions occur, some trees are affected while the others are not. Notably, we have initially tried to use linear predictor on the 3D data. This version of the tracker worked well on synthetic evaluation but failed on real images.

Keeping these problems in mind, when we use the chameleon tracker on 2D templates, we achieve robustness against occlusions as well as extreme illumination changes which are limited in [60, 61, 62, 65, 132]. On the other hand, when we apply the chameleon tracker on 3D data, we have successfully achieved the first learning-based 3D temporal tracker that operates on depth images alone while maintaining a very low tracking time with less than 2ms per frame. In effect, the 3D tracker unlocks a range of applications in the robotic perception, human-computer interaction and augmented reality to be discovered.

4.2 Related Work

Frame-to-frame tracking can be divided into two categories that are based on energy minimization and learning. The main difference between them is that the former is generally slower and is more sensitive to local minima; whereas, the latter requires an extensive training procedure. To have a more focused comparison of our approach with other methods in these categories, the scope of this section is limited to model-based frame-to-frame tracking using intensity (or RGB) images, depth images or both.

Energy minimization-based approach. The work of Lucas and Kanade [85] has significantly triggered an advancement in the field of 2D template tracking. Baker and Matthews [8] has summarized these through four different update rules – additive approach [85], compositional approach [125], inverse additive approach [24, 48], and inverse compositional approach [7, 36]. Among them, the inverse additive and inverse compositional approaches have decreased tracking time by switching the roles of the source and target images to evaluate several computations prior to tracking.

On the other hand, ICP [15] and variations of it [118, 123] has dominated the research field regarding 3D object tracking. However, ICP has problems when foreign objects such as clutter or hands are close to or occlude the object of interest. For instance, when tracking hand-held objects in [50], the authors segments the hand through the intensity image to remove the point clouds associated to it before running ICP. Another approach in 3D object tracking includes the level sets of Ren *et al*. [110] that uses a probabilistic method to statistically determine occlusions. But this also uses intensity images as an appearance model to help handle occlusions.

Learning-based approach. Using the objective function of an energy-minimization approach, Jurie and Dhome [65] builds up from the work of Hager and Belhumeur [48] to learn linear predictors to find the relation of the intensities and parameters by randomly warping the 2D template using different parameters of the warping function. Then, when the template is moved, the changes in intensities can predict the warping parameters. Another work is from Mayol and Murray [92] where they use general regression to fit the sampling region to pre-trained samples.

There have been successful attempts to handle occlusion using 2D templates. In [58, 66, 148], they represent a template using smaller templates so that, when the template is partially occluded, only a few smaller templates are affected. However, in addition to occlusions from other objects where a specific region of the template is affected, noise and missing data from the sensor can be interpreted as small occlusions, appearing as small curves or small blobs, that extend throughout the template. As a result, this could affect a significant number of small templates used in [58, 66, 148] to handle occlusions. Furthermore, unlike textured images, tracking small templates is unstable in 3D because, if we divide the surface of an object into small portions, most of them have a very similar structure to their neighbors which makes tracking ambiguous.

To the best of our knowledge, we have implemented the first learning-based tracking algorithm that uses depth images alone. Nevertheless, our algorithm is a generic approach that is applicable to both intensity and depth images. Furthermore, it is robust to occlusions, illumination changes as well as fast motion, and runs in less than 2 ms per frame with one CPU core.

4.3 Generalizing the Tracker

This work is rooted from the objective function of Hager and Belhumeur [48] in Section 2.3.1 where they relate the image intensities of a template and transformation parameters by the pseudo-inverse of a Jacobian matrix [48]. Another work from [65] replaces the relation by a linear predictor so that tracking becomes very fast. However, in contrast them, our method uses random forests in lieu of the matrices, which generalizes it to any input function and not constrained to 2D intensity images.

4.3.1 Reformulating the Objective Function

Given an arbitrary input function Ω_t at time t (*e.g.* intensity images or point clouds), the location of the reference template is represented by n_s sample points $\{\mathbf{x}_s \in \mathbb{R}^N\}_{s=1}^{n_s}$ at t_0 such that the template is described by the set of values $\{\Omega_{t_0}(\mathbf{x}_s),\ \forall\, \mathbf{x}_s\}$. As the template moves across time, the sample points are transformed as $\Phi(\boldsymbol{\mu}) \circ \mathbf{x}_s$, where the vector $\boldsymbol{\mu}$ contains the parameters of the transformation function Φ and its initial value is $\boldsymbol{\mu}_{t_0} = \mathbf{0}$. Therefore, at time t, the objective function minimizes

$$\varepsilon(\boldsymbol{\mu}_t) = \sum_s |\Omega_t(\Phi(\boldsymbol{\mu}_t) \circ \mathbf{x}_s) - \Omega_{t_0}(\mathbf{x}_s)|^2 \tag{4.1}$$

such that, at time $t+\tau$, the parameter vector $\boldsymbol{\mu}_t$ updates to $\boldsymbol{\mu}_{t+\tau} = \boldsymbol{\mu}_t + \delta\boldsymbol{\mu}$ by minimizing

$$\varepsilon(\delta\boldsymbol{\mu}) = \sum_s |\Omega_{t+\tau}(\Phi(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}) \circ \mathbf{x}_s) - \Omega_{t_0}(\mathbf{x}_s)|^2 \tag{4.2}$$

where $\delta\boldsymbol{\mu}$ is the parameter update vector. To simplify this equation, we assign $\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = [\Omega_t(\Phi(\boldsymbol{\mu}) \circ \mathbf{x}_s)]_{s=1}^{n_s}$ as a collection of $\Omega_t(\cdot)$; hence, (4.2) is rewritten in vector form as

$$\varepsilon(\delta\boldsymbol{\mu}) = \left\| \boldsymbol{\Omega}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}, t+\tau) - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0) \right\|^2 . \tag{4.3}$$

Using Taylor Series Approximation, we have

$$\begin{aligned} &\boldsymbol{\Omega}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}, t+\tau) \\ &\approx \boldsymbol{\Omega}(\boldsymbol{\mu}_t, t) + \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)\delta\boldsymbol{\mu} + \tau\frac{\partial\boldsymbol{\Omega}}{\partial t}(\boldsymbol{\mu}_t, t) && \text{(4.4a)} \\ &= \boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)\delta\boldsymbol{\mu} && \text{(4.4b)} \end{aligned}$$

where $\mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)$ is the Jacobian matrix of $\boldsymbol{\Omega}$ with respect to $\boldsymbol{\mu}$, and $\frac{\partial\boldsymbol{\Omega}}{\partial t}(\boldsymbol{\mu}_t, t)$ is estimated using the forward difference approximation. We substitute (4.4) into (4.3) as

$$\varepsilon(\delta\boldsymbol{\mu}) \approx \left\| \boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}_t}\delta\boldsymbol{\mu} - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0) \right\|^2 \tag{4.5}$$

where $\mathbf{J}_{\boldsymbol{\mu}_t} = \mathbf{J}_{\boldsymbol{\mu}}(\boldsymbol{\mu}_t, t)$. Finally, to find $\delta\boldsymbol{\mu}$ with the minimum ε, we set the $\bigtriangledown_{\delta\boldsymbol{\mu}}\varepsilon$ to zero, which implies that

$$\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) + \mathbf{J}_{\boldsymbol{\mu}}\delta\boldsymbol{\mu} - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0) = 0 . \tag{4.6}$$

Then, solving for $\delta\boldsymbol{\mu}$,

$$\begin{aligned} \delta\boldsymbol{\mu} &= -\mathbf{J}^+_{\boldsymbol{\mu}_t}[\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0)] && \text{(4.7a)} \\ &= -\mathbf{J}^+_{\boldsymbol{\mu}_t}\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) && \text{(4.7b)} \end{aligned}$$

where $\mathbf{J}^+_{\boldsymbol{\mu}} = \left(\mathbf{J}^\top_{\boldsymbol{\mu}_t}\mathbf{J}_{\boldsymbol{\mu}_t}\right)^{-1}\mathbf{J}_{\boldsymbol{\mu}_t}$ and $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = \boldsymbol{\Omega}(\boldsymbol{\mu}, t) - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0)$. Therefore, the pseudo-inverse of the Jacobian matrix $-\mathbf{J}^+_{\boldsymbol{\mu}_t}$ in (4.7) represents the relation from the given $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau)$ to the parameter update $\delta\boldsymbol{\mu}$. In this way, $\delta\boldsymbol{\mu}$ updates the transformation function as

$$\begin{aligned} \Phi(\boldsymbol{\mu}_{t+\tau}) &= \Phi(\boldsymbol{\mu}_t) \circ \Phi(\boldsymbol{\mu}_{t_0} + \delta\boldsymbol{\mu}) && \text{(4.8a)} \\ &= \Phi(\boldsymbol{\mu}_t) \circ \Phi(\delta\boldsymbol{\mu}) . && \text{(4.8b)} \end{aligned}$$

Instead of finding the non-linear relation $-\mathbf{J}^+_{\boldsymbol{\mu}_t}$ in (4.7), we formulate a learning method using random forests to find the relation between $\delta\boldsymbol{\Omega}$ and $\delta\boldsymbol{\mu}$.

4.3.2 Learning a Tracker

We use regression forests to learn how different values of $\delta\boldsymbol{\mu}$ affect the template in Ω_{t_0} through $\delta\boldsymbol{\Omega}$. Subsequently, when the input function $\Omega_{t+\tau}$ and the parameters $\boldsymbol{\mu}_t$ are given, the forests use $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau)$ to predict $\delta\boldsymbol{\mu}$ and update the transform to $\Phi(\boldsymbol{\mu}_{t+\tau})$. Contrary to [48], this learning scheme is not restricted to 2D images and can handle different types of input function Ω without the necessity or difficulty to derive different Jacobian matrices.

This method assumes independence between the parameters in $\boldsymbol{\mu}$ and learns one forest for each parameter separately. As a result, there are n_p forests where n_p is the number of parameters in $\boldsymbol{\mu}$ and each forest consists of n_t trees.

Learning dataset. This process begins by creating a training dataset where we produce n_ω random values of $\delta\boldsymbol{\mu}^\omega$ to transform the input Ω_{t_0} to Ω^ω, such that the location of the sample points in Ω^ω is computed as $\Phi(\delta\boldsymbol{\mu}^\omega) \circ \mathbf{x}_s$. It follows that we can define the learning dataset

$$\mathcal{S} = \left\{ \left(\delta\boldsymbol{\Omega}^\omega, \delta\boldsymbol{\mu}_p^\omega \right) \ \forall\, \omega \right\} \tag{4.9}$$

that is used to construct the p-th forest where the vector $\delta\boldsymbol{\Omega}^\omega = \delta\boldsymbol{\Omega}(\mu_{t_0}, \omega)$ is the input sample of the forest and the scalar $\delta\boldsymbol{\mu}_p^\omega$ is the p-th parameter in $\delta\boldsymbol{\mu}^\omega$. In general, the objective of using such a synthetic training dataset is to generate $\mathcal{S}$; thus, there are several ways of creating $\mathcal{S}$ and using a synthetic dataset by transforming Ω_{t_0} is just one of them.

Before training a tree, we randomly select n_r points from the n_s sample points of the template and only use these points to construct the tree. The goal is to impose randomness and to help handle cases when some sample points are not available or have incorrect values in Ω (*e.g.* occlusion). Hence, we assign the features $\{\theta_r\}_{r=1}^{n_r}$ as the indices of the θ_r-th sample point.

Learning the forests. Given the learning dataset $\mathcal{S}$, each feature is used to split the samples that arrive in the node $\mathcal{S}_N$ into two subsets $\mathcal{S}_l$ and $\mathcal{S}_r$ that goes to its left and right child, respectively. These subsets are defined as

$$\mathcal{S}_l = \left\{ (\delta\boldsymbol{\Omega}^\omega, \delta\boldsymbol{\mu}_p^\omega) \in \mathcal{S}_N \,|\, \delta\boldsymbol{\Omega}_{\theta_r}^\omega \geq \kappa_{\theta_r} \right\} \text{ and} \tag{4.10a}$$

$$\mathcal{S}_r = \left\{ (\delta\boldsymbol{\Omega}^\omega, \delta\boldsymbol{\mu}_p^\omega) \in \mathcal{S}_N \,|\, \delta\boldsymbol{\Omega}_{\theta_r}^\omega < \kappa_{\theta_r} \right\} \tag{4.10b}$$

where $\delta\boldsymbol{\Omega}_{\theta_r}^\omega$ is the θ_r-th element of $\delta\boldsymbol{\Omega}^\omega$, and κ_{θ_r} is the threshold. Furthermore, to evaluate the split, we use the information gain to determine whether it produced less random or more homogeneous subsets, that is written as

$$G(\theta_r) = \sigma(\mathcal{S}_N) - \sum_{i \in \{l,r\}} \frac{|\mathcal{S}_i|}{|\mathcal{S}_N|} \sigma(\mathcal{S}_i) \tag{4.11}$$

where $\sigma(\mathcal{S}_i)$ is the standard deviation of all $\delta\boldsymbol{\mu}_p^\omega$ in $\mathcal{S}_i$. Among all θ_r, we look for the best feature with the highest information gain. The node stores the best feature and its threshold, and passes the subsets to its children. Note that the choice of the threshold κ_{θ_r} depends on the values of all $\delta\boldsymbol{\Omega}_{\theta_r}^\omega$ in $\mathcal{S}_N$. It can either be a single threshold such as the median, or multiple threshold candidates such as linearly spaced values. Moreover, if multiple thresholds are used, each of them is also evaluated using (4.11) and the one with the highest information gain is stored.

The tree continuously splits the samples and grows until at least one of these stopping criteria is true:

(1) the maximum depth of the tree is reached;

(2) the number of samples $|\mathcal{S}_N|$ is small;

(3) the standard deviation $\sigma(\mathcal{S}_N)$ is sufficiently low; or,

(4) the information gain of the best feature is less than a threshold.

Consequently, the node is considered as a leaf and stores the mean and standard deviation of all $\delta\boldsymbol{\mu}_p^{\omega}$ that reached this leaf. Similarly, the same process occurs for all n_t trees in each forest and for all n_p forests.

In the succeeding chapters, after creating a different learning dataset $\mathcal{S}$, the same procedure applies in learning the forests – the branches enfoce the splitting function from (4.10) with the best information gain from (4.11), while the leaves stores the mean and standard deviation of the parameter.

4.3.3 Tracking with Forests

Looking at (4.7) for $t+\tau$, the curent input function $\Omega_{t+\tau}$ and the parameter vector $\boldsymbol{\mu}_t$ from t are given in order to solve the parameter update $\delta\boldsymbol{\mu}$. We compute the input sample $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau)$ of the the forests to predict $\delta\boldsymbol{\mu}$ that updates the parameters.

Starting from the root node of each tree, the input sample uses the splitting parameters in the node to determine whether to go to the left or right child, and continuously traverse from the parent node to the child node until it reaches a leaf where the learned mean and standard deviation that predict a parameter of $\delta\boldsymbol{\mu}$ are stored. As a result, each of the n_p parameters have n_t predictions.

To find the final prediction of a parameter in $\delta\boldsymbol{\mu}$, a percentage of the n_t predictions with the lowest learned standard deviation are aggregate by taking the average of the learned means. Finally, the transform is updated by employing (4.8a) and also update the location of the sample points in $\Omega_{t+\tau}$. Lastly, we iteratively repeat the entire process to refine the previously predicted parameters.

4.4 Registration on Different Domains

With a generalized $\boldsymbol{\Omega}$, this section investigates on two input domains for tracking. The first is the 2D template tracking using intensity images in Section 4.4.1. Similar to the works of [60, 61, 62, 65] in Chapters 2 and 3, the objective is to track a rectangular template through a homography. However, instead of relying on the linear predictor, we utilize a random forest approach that makes an individual template robust against extreme illumination changes and occlusions.

The second investigates on a 3D model-based tracker in Section 4.4.2. To the best of our knowledge, the closest approach that achieves the same goals in ICP [14, 118, 123]. But the existing ICP algorithms are based on energy-minimization while our work relies on learning. Thus, for the 3D tracker, we claim that this is the first method that estimates the full 6D pose of the object through a learning framework using depth images.

4.4.1 2D Template Tracking

Section 4.3 is applied as a 2D template tracking algorithm using intensity images Ω. Similar to Chapters 2 and 3, we parameterize a rectangular template using the 2D location of its four corners $\{\mathbf{x}_c\}_{c=1}^4$ at t_0, and define its motion by the displacement of these corners $\{\delta\mathbf{x}_c\}_{c=1}^4$. From these, the transformation function $\Phi(\boldsymbol{\mu})$ is denoted as a homography that transforms $\{\mathbf{x}_c\}_{c=1}^4$ to $\{\mathbf{x}_c+\delta\mathbf{x}_c\}_{c=1}^4$ where $\boldsymbol{\mu} = [\delta\mathbf{x}_c]_{c=1}^4$and $\Phi(\boldsymbol{\mu}_{t_0}) = \mathbf{I}_{3\times3}$. Moreover, we position the sample points $\{\mathbf{x}_s\}_{s=1}^{n_s}$ on an $n_s = n_g \times n_g$ regular grid fitted into the template.

$\mathbf{\Omega}(\boldsymbol{\mu}, t) = [\Omega_t(\Phi(\boldsymbol{\mu}) \cdot \mathbf{x}_s)]_{s=1}^{n_s}$ is then a collection of image intensities on the image Ω_t at the transformed sample points, and $\delta\mathbf{\Omega}(\boldsymbol{\mu}_{t_0}, \omega) = [\Omega^{\omega}(\mathbf{x}_s) - \Omega_{t_0}(\mathbf{x}_s)]_{s=0}^{n_s}$ is the input sample of the forest where $\Omega^{\omega}(\mathbf{x}_s)$ is the warped template and $\Omega_{t_0}(\mathbf{x}_s)$ is the reference template. Hence, in training, the synthetic data are transformations of the image Ω_{t_0} with n_ω random motions from $\delta\boldsymbol{\mu}$. Moreover, after predicting the parameters $\delta\boldsymbol{\mu}$ in tracking, we update the homography as $\Phi(\boldsymbol{\mu}_{t+\tau}) = \Phi(\boldsymbol{\mu}_t) \cdot \Phi(\delta\boldsymbol{\mu})$.

To handle illumination changes, we normalize the n_r sample points in each tree such that the intensities on these points have zero mean and unit standard deviation.

Parametrization. Using this concept, we use a 250×250 template and train $n_t = 50$ trees for each $n_p = 8$ forests with 25×25 grid enclosed on the template and $n_s = 625$ sample points. For the synthetic dataset, we generate $n_\omega = 50{,}000$ transformed images where the corners moves in the range of [-85, 85] pixels. The thresholding in each branch uses ten linearly spaced values and the stopping criteria in each node includes maximum depth of 20, minimum number of samples of 40, minimum standard deviation of 0.5, and minimum information gain of 0.01. In tracking, we aggregate 20% of the prediction with the lowest standard deviation and run 15 iterations.

Experimental Results

Using the same dataset in Figure 3.3 and evaluation in Chapter 3, we measure the tracking robustness of our algorithm and compare it with Jurie and Dhome [65] where they learn the relation between image intensity difference and parameter update vector using linear predictor. The choice in comparing with [65] is due to its similarity with our approach where both are rooted from [48] and replace the pseudo-inverse of the Jacobian matrix with a learning algorithm. Moreover, it is reported in Chapter 3 that the overall performance of [65] is either comparable to or better than [13, 58, 59].

Here, we learn the template at the center of the image and track this template after warping the image. The tracking robustness is computed as the percent of successfully tracked templates where the average distance of the tracked corners to its ground truth location is less than 5.0 pixels when backwarped. In addition, we compare our tracking robustness with different values of n_r and, based on the plots in Figure 4.1, the performance converges when $n_r = 25$. From these evaluations, the average tracking time of our approach is approximately 1.47 ms when using one core of the CPU, while linear predictor [65] runs at 0.47 ms. Thus, their approach is 1 ms faster than ours which is almost negligible.

With different types of image warping in Fig 4.1(a-d), we show that our 2D experiment has similar tracking robustness as the linear predictor [65]. For the evaluation with respect to noise in Figure 4.1(e) where we randomly translate the image by a maximum distance of 35 pixels after inducing varying levels of Gaussian noise, it illustrates that we are slightly better in tracking than [65].

However, our learning approach is much more robust to occlusion than [65] as shown in Figure 4.1(f) where the image is translated after replacing a percent of the template's region by a different image; while, in Figure 4.2(c), we demonstrate its performance after occluding the template with different objects. Note that this property becomes a requirement when some of the pixel values from the sensor is not available such as shadows from depth sensors – a thorough investigation in using our algorithm

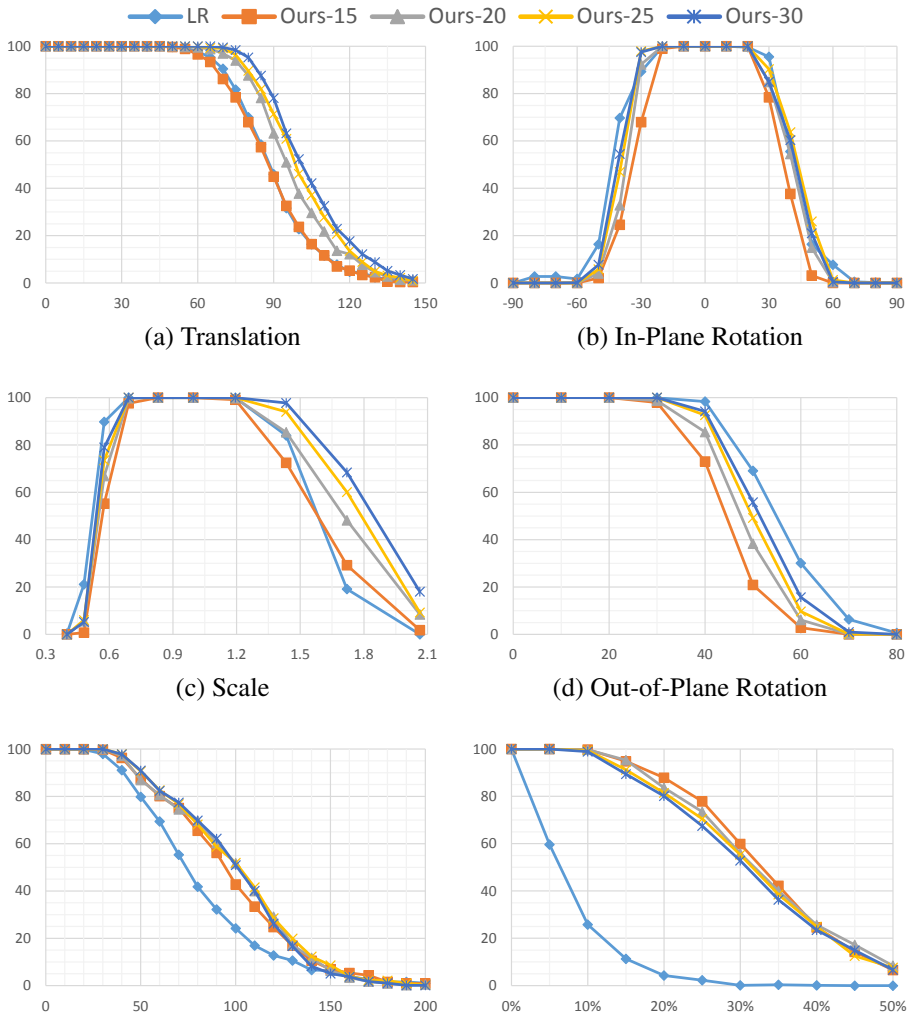


Figure 4.1: These plots show the tracking robustness of our algorithm with varying n_r (15, 20, 25, 30) and compare it with linear predictor (LR) [65] under different (a-d) transformations, (e) levels of Gaussian noise and (f) percentage of occluded region.

for depth images is conducted in Section 4.4.2.

Therefore, although there are similarities with [65], our algorithm has proven to be a *chameleon* since its range of application is not limited to a specific sensor and its tracking robustness is not compromised by this generality. With regards to 2D tracking, we have demonstrated that both algorithms are equally robust in tracking performance, but we are more robust in the presence of occlusion and strong illumination changes.

Qualitative Results

Based on the quantitative evaluation, this section aims to reinforce the claims through qualitative tests. The evaluation begins with a simple tracking example in Figure 4.2(a). Knowing that [65] is sensitive to occlusions, we demonstrate that our forest-based tracker in the presence of occlusion in Figure 4.2(b). In addition, we also demonstrate in Figure 4.2(c-d) that we are also robust to strong illumination changes which [65] cannot handle.

Application

The forest-based 2D template tracker has been applied as part of the surgical tool tracking frameworks in [114, 115, 116]. In these works, we aim to develop a tracker that is robust to illumination changes, different types of light sources and different variations of reflectance from the metallic tool. Contrary to other template tracking approaches such [60, 61, 62, 65, 132, 133], instead of learning a single template, a unique attribute of [114, 115, 116] is that it learns from multiple templates in order to generalize for different environmental changes.

Learning from multiple templates in one linear predictor [60, 61, 62, 65] is not possible since it requires a single template to compare its $\delta\mathbf{i}$. In this framework, we utilize the intensities from the sample points instead of the difference in intensity. This is based on the observation that the threshold in splitting uses an element of the learning dataset. The intensity of un-perturbed template in the intensity difference vector is always contant across all the dataset and does not affect the information gain in splitting – thus, it is negligible when learning through random forest.

Notably, one of our papers [114] is a recipient of the *Young Scientist Award* from the 2015 conference on Medical Image Computing and Computer Assisted Intervention (MICCAI).

4.4.2 3D Object Tracking

Through the generalized objective function from (4.3), we designed a model-based tracking method that finds the pose of a 3D rigid object using the depth image $\mathbf{D}_t$. The first goal is to derive an objective function that resembles the error function in (4.3). When accomplished, we follow a similar learning and tracking algorithm as Sections 4.3.2 and 4.3.3.

Here, the object coordinate system are used, where the centroid of the model's vertices is the origin, and the camera coordinate system, where the camera center is the origin. The camera is parameterized by a 3×3 intrinsic matrix $\mathbf{K}$ and a 4×4 transformation matrix $\mathbf{T}_{t_0}$ that relates the camera coordinate system and the object coordinate system at t_0. Moreover, the sample points $\{\mathbf{X}_s\}_{s=1}^{n_s}$ are 3D homogeneous

Figure 4.2: Qualitative results of our 2D template tracking algorithm – (a) tracking with perspective transform, (b) tracking with partial occlusion, and (c-d) tracking under strong illumination changes with a time-lapse during sunset in (d).

points on the model as seen by the camera coordinate system. The corresponding points in the object coordinate system are then computed as $\{\mathbf{T}_{t_0}^{-1}\mathbf{X}_s\}_{s=1}^{n_s}$.

Looking from the object coordinate system, the rigid transform of the object is defined by the 4×4 matrix

$$\mathbf{T}_{\text{obj}}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma) \tag{4.12}$$

where $\mathbf{t} = (t_x, t_y, t_z)^\top$ is the translation vector; α, β and γ are the yaw, pitch and roll angles, respectively; and, the parameter vector $\boldsymbol{\mu}$ is composed of the three translation parameters and three rotation parameters. Therefore, after transforming the object, the sample points in the camera coordinate system transforms as

$$\mathbb{P}_{\boldsymbol{\mu}}(\mathbf{X}_s) = \mathbf{T}_{t_0}\mathbf{T}_{\text{obj}}(\boldsymbol{\mu})\mathbf{T}_{t_0}^{-1}\mathbf{X}_s \tag{4.13}$$

where

$$\mathbb{P}_{\boldsymbol{\mu}_{t_0}}(\mathbf{X}_s) = \mathbf{X}_s\ . \tag{4.14}$$

Then, the projection of the points into the camera's image is given as $\mathbf{x}_s^d = [\mathbf{K}|\mathbf{0}] \cdot \mathbb{P}_{\boldsymbol{\mu}}(\mathbf{X}_s)$ where $\mathbf{x}_s^d$ is given in 2D homogeneous coordinates.

If we denote $\mathbb{D}_t(\mathbf{x})$ as the backprojection of the pixel $\mathbf{x}$ in the depth image $\mathbf{D}_t$, then, to find the optimum $\delta\boldsymbol{\mu}$ at $t+\tau$, we minimize the sum of the distances

$$\left\| \mathbb{D}_{t+\tau}([\mathbf{K}|\mathbf{0}]\mathbb{P}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbf{X}_s)) - \mathbb{P}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbf{X}_s) \right\|^2 \tag{4.15}$$

$$= \left\| \mathbb{P}^{-1}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbb{D}_{t+\tau}([\mathbf{K}|\mathbf{0}]\mathbb{P}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbf{X}_s))) - \mathbf{X}_s \right\|^2 \tag{4.16}$$

for all sample points. However, we observed that the difference in the x- and y-coordinates are close to zero in (4.16). Hence, we simplify the error function by only using the difference in the z-coordinates

$$\varepsilon(\delta\boldsymbol{\mu}) = \sum_s |\Phi_{t+\tau}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}) \circ \mathbf{X}_s - \mathbf{X}_s|_z^2 \tag{4.17}$$

that resembles the generalized objective function in (4.3) where

$$\Phi_t(\boldsymbol{\mu}) \circ \mathbf{X}_s = \mathbb{P}_{\boldsymbol{\mu}}^{-1}\left(\mathbb{D}_t\left([\mathbf{K}|\mathbf{0}]\mathbb{P}_{\boldsymbol{\mu}}\left(\mathbf{X}_s\right)\right)\right) \tag{4.18}$$

is a time-varying transformation function that is dependent on the depth image $\mathbf{D}_t$, and the operator $|\cdot|_z$ takes the z-coordinate of the point. When we compare (4.17) with (4.3), the vector

$$\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = [|\Phi_t(\boldsymbol{\mu}_t) \circ \mathbf{X}_s|_z]_{s=1}^{n_s} \tag{4.19}$$

is described as a collection of the z-coordinates of the transformed sample points where $\boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0) = [|\mathbf{X}_s|_z]_{s=1}^{n_s}$.

In training, the input sample can be further simplified as

$$\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, \omega) = [|\mathbb{D}^\omega([\mathbf{K}|\mathbf{0}]\mathbf{X}_s) - \mathbf{X}_s|_z]_{s=1}^{n_s} \tag{4.20}$$

because of (4.14). This implies that $\delta\boldsymbol{\Omega}$ in the learning dataset $\mathcal{S}$ only changes with respect to the depth image across ω. As a consequence, to create the set $\mathcal{S}$ for training, we render n_ω depth images with different parameters of $\delta\boldsymbol{\mu}^\omega$ in $\mathbf{T}_{\text{obj}}$. Moreover, from the synthetic depth image with $\mathbf{T}_{\text{obj}}(\boldsymbol{\mu}_0)$, the model on the image is enclosed by an $n_g \times n_g$ regular grid, where the points on the model are backprojected and are used as the sample points. On the other hand, in tracking, the prediction $\delta\boldsymbol{\mu}$ updates $\mathbf{T}_{\text{obj}}$ in the transformation function Φ through

$$\mathbf{T}_{\text{obj}}(\boldsymbol{\mu}_{t+\tau}) = \mathbf{T}_{\text{obj}}(\boldsymbol{\mu}_t) \cdot \mathbf{T}_{\text{obj}}(\delta\boldsymbol{\mu}) \,. \tag{4.21}$$

Multi-View Tracking. The problem with our sample point arrangement is that it is restricted to one view of the object. For instance, if the object keeps rotating in one direction, at some point, the tracker becomes unstable and fails since the number of visible sample points continuously decreases. Due to this, we individually learn random forests for n_c views of the object where the camera is located around the object using an icosahedron. It follows that the c-th camera view has its own set of sample points $\{\mathbf{X}_s^c\}_{s=1}^{n_s^c}$ and transformation matrix $\mathbf{T}_c$, and this produces $6{\cdot}n_c \cdot n_t$ trees. Therefore, using the object coordinate system in tracking, we find the closest learned camera view to the current position of the camera. Mathematically, to switch from one view to the other, we modify (4.13) to

$$\mathbb{P}_{\boldsymbol{\mu}}^c(\mathbf{X}_s^c) = \mathbf{T}_0 \mathbf{T}_{\text{obj}}(\boldsymbol{\mu}) \mathbf{T}_c^{-1} \mathbf{X}_s^c \tag{4.22}$$

where $\mathbf{T}_0$ is the initial object transformation matrix in tracking.

Parametrization. Based on this, we use $n_t = 100$ trees for each $n_p = 6$ parameters with a $40{\times}40$ grid enclosed on the model as seen from the depth image where only points that lie on the model are used as sample points. In training, we render $n_\omega =$ 50,000 depth images where the object randomly transforms with a translation that ranges in [-35, 35] mm for each axis, and the angles α, β and γ in [-30°, 30°]. Moreover, each tree randomly chooses $n_r = 20$ sample points, each branch uses ten linearly spaced thresholds and each node checks the stopping criteria with depth of 20, 40 samples, standard deviation of 0.5, and information gain of 0.01. This is done for $n_c = 42$ camera views of the model. Then, in tracking, we aggregate 15% of the predictions with the lowest standard deviation and run ten iterations.

Experimental Results

We created a test dataset with four sequences using PrimeSense's Carmine 1.09 camera to track different objects – vise, driller, cat and bunny – in cluttered environments as shown in the first row of Figure 4.4. We compare our results to the ICP implementation from PCL [1] where the vertices of the model are used as input source, and LineMod [54], which is the state-of-the-art 3D object detector, that is taken from the original implementation of the authors. Note that LineMod refines the pose of the object after template matching using ICP.

To generate ground truth transforms, we placed the objects on top of a board with markers on its edges; thus, we can transform the object's model using the ground truth and compare it with the location of the tracked object through the mean distance of the

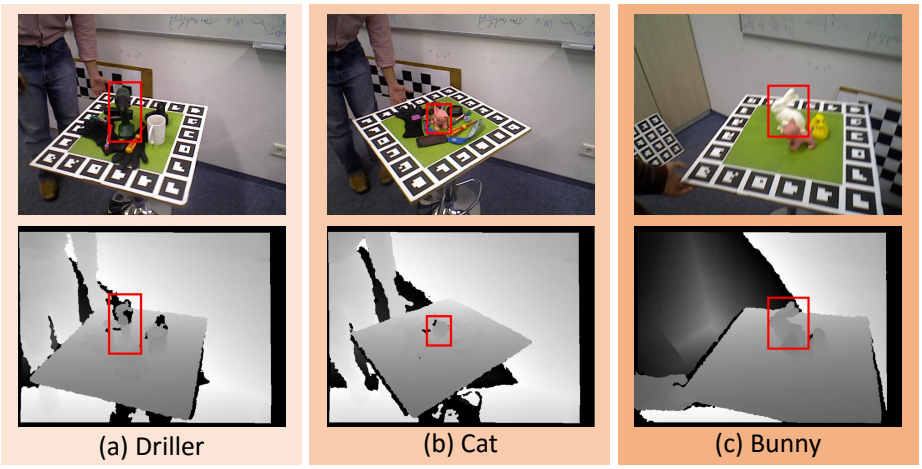


Figure 4.3: These images show the frames (a) when our approach becomes unstable in the driller sequence due to the lack of depth data, and (b-c) when ICP fails in the cat and bunny sequences. Note that the object of interest is marked in red.

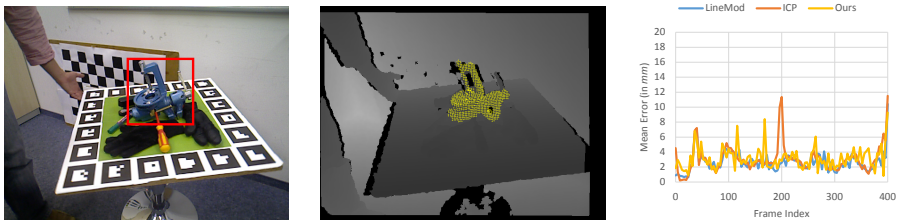
model's vertices as plotted in the second row of Figure 4.4. Moreover, in the vise, driller and cat sequences, the camera stayed static while the board rotates; but, in the bunny sequence, both the camera and board moves. Hence, from this dataset, the tracking results can be summarized as follows:

1. **Vise**. All approaches works well in this sequence even if there is a small occlusion from the red screwdriver as illustrated in Figure 4.4(a).

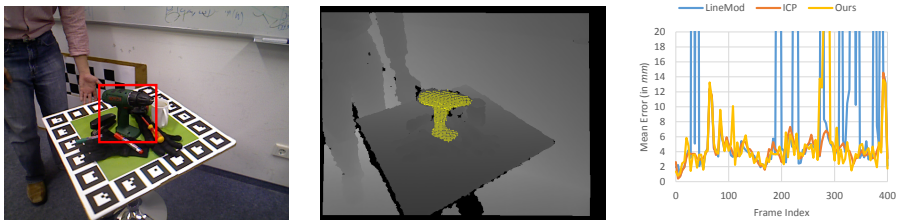
2. **Driller**. For this sequence, there are some depth images that does not have sufficient information for a portion of the object as shown in Figure 4.3(a). This affects both LineMod and our method. However, it is important to mention that our method became unstable in a small section of the sequence but it did not lose tracking; while, LineMod frequently fails in detecting the object.

3. **Cat**. As the cat rotates, the depth image loses information of its tail as shown in Figure 4.3(b) and its relatively large spherical head dominates the ICP algorithm. Due to its shape, ICP stayed static and fails to track the rotation of the cat. This results in a tracking failure for the rest of the images in the sequence. Regarding LineMod and our algorithm, they work well in this sequence and smoothly track the cat's rotation without getting trapped in a local minimum.

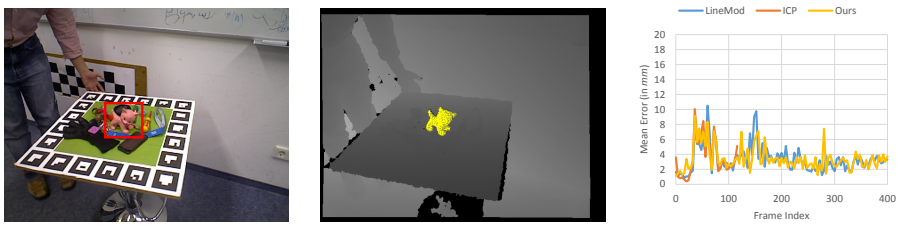
4. **Bunny**. There are two essential criteria to consider in this sequence. The first is the fast motion of both object and camera which creates a motion blur in the colored image as well as noise in the depth image as shown in Figure 4.3(c), and the second is the closeness of the surrounding objects to the object of interest as depicted in Figure 4.4(d). In this sequence, the bunny is occluded by the cat, then the duck. Therefore, when the cat occluded a large portion of the bunny, ICP started incorporating the cat as part of the bunny and loses tracking. For LineMod, the detector also fails when the objects occlude the bunny, which is indicated



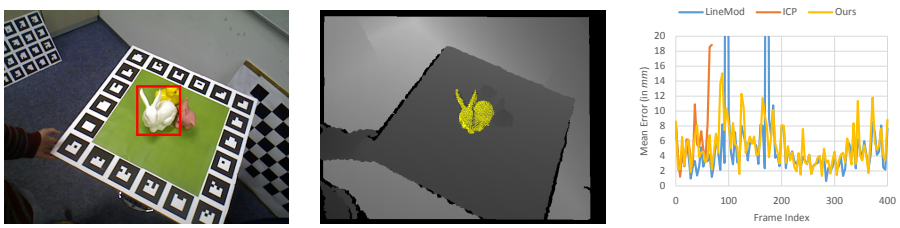
(a) Vise



(b) Driller



(c) Cat



(d) Bunny

Figure 4.4: The first column shows the setup of the four sequences that are used to evaluate the 3D tracking algorithm where the object of interest is mark in red; the second shows the mean distance error for each sequence using LineMod [54] where their peaks indicate detection failures, PCL's ICP [1] where ICP fails at frame 116 in (c) and frame 72 in (d), and our approach; and, the last row shows our tracking results in the corresponding depth image.

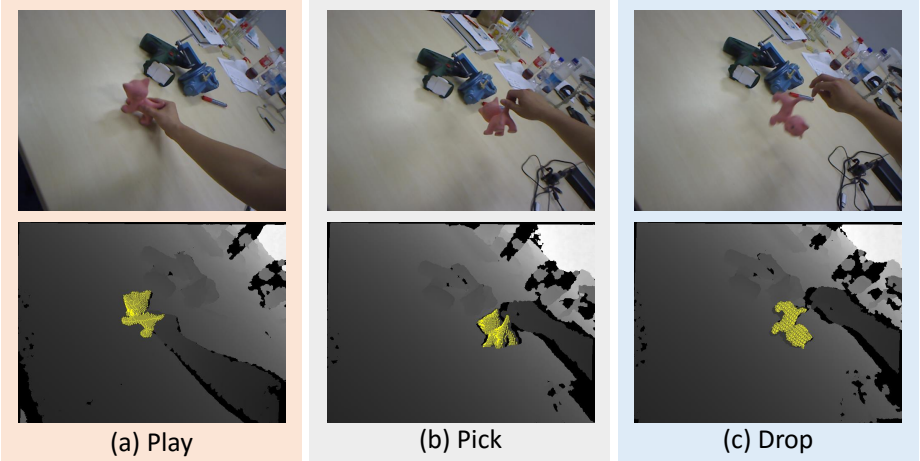


Figure 4.5: These images show examples of our 3D tracking algorithm where the actor (a) plays with the cat; (b) picks it up; then, (c) drops it. More examples are in the *Supplementary Materials*.

by the two peaks in Figure 4.4(d). Finally, our approach became unstable but completely recovers after the occlusion.

On average, our tracking time is 1.75 ms for the vise, 1.76 ms for the driller, 1.84 ms for the cat and 1.84 ms for the bunny when using only one CPU core. For ICP, the average tracking time is 189.09 ms for the vise, 72.24 ms for the driller, 65.04 ms for the cat and 225.72 ms for the bunny. Moreover, the average tracking time for LineMod is approximately 119 ms for all models.

Taking the results from the dataset into account, we have exhibited different scenarios and show that, in comparison to LineMod and ICP, our 3D algorithm can avoid local minima, is more stable in the presence of both noise and occlusion; and, is at least 35 times faster than ICP [1] and two orders of magnitude faster than LineMod [54].

Qualitative Results

Furthermore, we illustrate some tracking examples in Figure 4.5 from a video where the actor plays with the cat, picks it up through its tail, turns it around, then drops it. These examples show how well the tracker handles occlusion. For instance, when the cat is occluded by the hand in Figure 4.5(a), it is not necessary to do any pre-processing procedure, such as segmenting the hand, even if the object is relatively small. Additionally, when the cat is turned or dropped, it also shows how well our algorithm handles fast motion.

Through the examples in Figures 5.9 and 5.10, we demonstrate that we have a good tracking performance in fast motion as well as partial occlusion. An interesting characteristic of the tracker is the capacity to handle close-range occlusions such as hand-held devices. In contrast, ICP usually fails in such scenario since it tries to incorporate the neighboring objects into the energy minimization as validated in Figure 5.10(b).

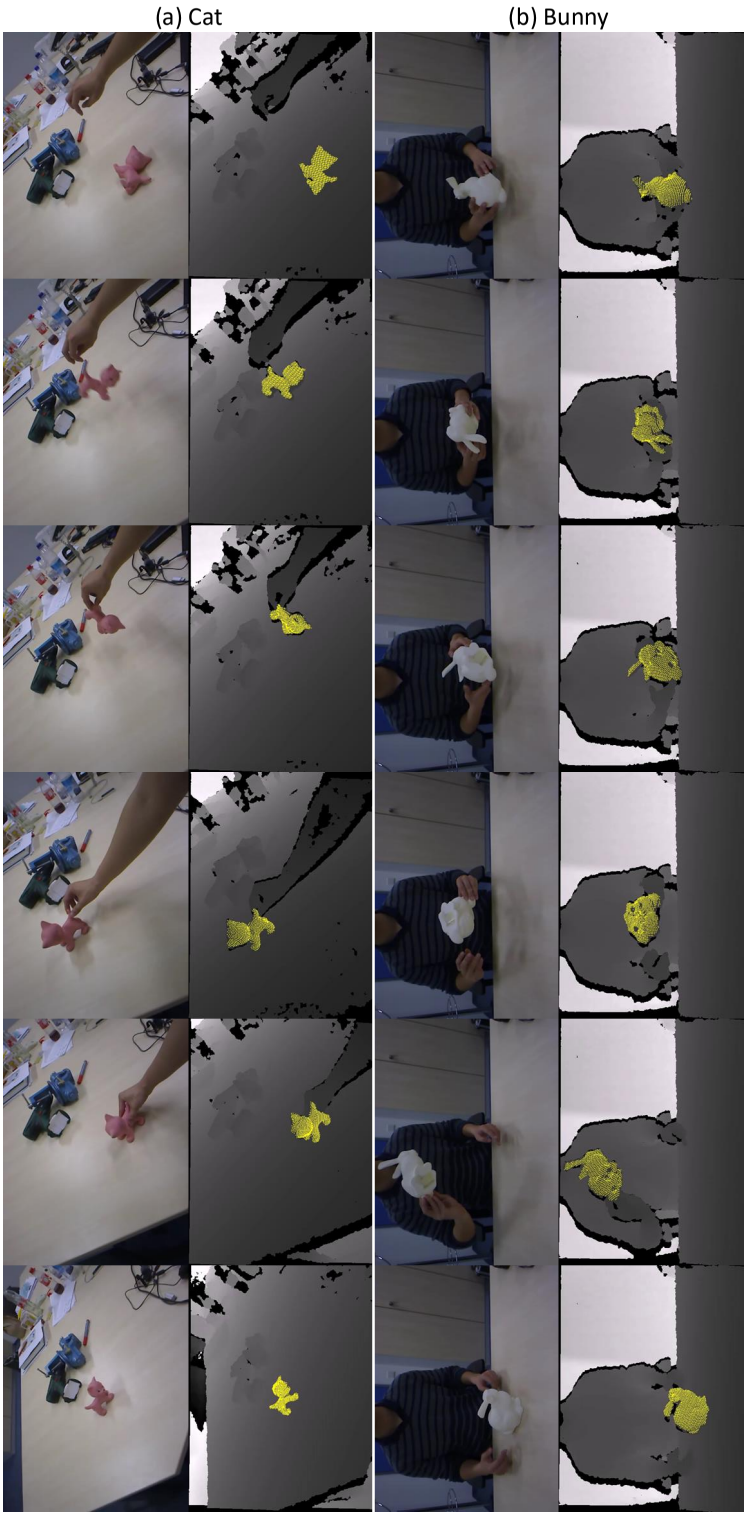


Figure 4.6: Qualitative evaluation on the 3D model-based tracker. Note that only the depth image is used for tracking.

Figure 4.7: Qualitative evaluation on the 3D model-based tracker. Note that only the depth image is used for tracking.

4.5 Conclusion

After generalizing the objective function, the novelty of this chapter is the transition from tracking 2D templates to 3D objects with the estimation of the full 6 d.o.f. pose. Achieving a robust algorithm and an extemely fast tracking time with a low computational cost (*i.e*. less than 2 ms using 1 CPU core), the 3D tracker paves the way to applications such as robotic applications and human-computer interaction, wherein the tracker is a significant component of a larger framework. In these applications, robustness avoids tracking failures and losing the object of interest which then requires the re-initialization of the frame-to-frame tracker, while the fast tracking time allows the resulting application to attain low latency.

5

Versatile 3D Tracker with Online Learning Capabilities

5.1 Motivation

Real-time 3D tracking is now the enabling technology for a range of applications in the field of augmented reality, robotic perception as well as human-machine interaction. In these cases, tracking multiple objects at real-time becomes an inherent requirement. Moreover, when using 3D sensors on mobile devices, low computational cost and memory consumption are required.

This chapter is inspired by our learning-based approach [133] from Chapter 4 that uses depth images only. Our previous work [133] is a temporal tracking algorithm based on random forest [21] that runs at 2 ms per frame with a single CPU core. At the moment, this is the only method that has achieved this efficiency in 3D tracking with an extremely low computational requirement compared to the literature concerning 3D tracking-by-detection [2, 37, 54] and 3D temporal tracking [14, 27]. However, it poses problems in the robustness against large occlusions and large holes that results in tracking errors and failures, memory consumption that limits tracking to a maximum of 9 objects and long learning time that limits its applicability to model-based tracking.

The motivation is to overcome the limitations from [133] to satisfy the diverse requirement of most 3D tracking applications. In order to transition from achieving theoretical goals into a versatile algorithm that solves real-world problems, we summarize the following required attributes to accomplish this objective:

(1) *Robustness.* To avoid tracking failures, the tracker must be robust against sensor noise such as holes and artifacts, commonly present in depth data, as well as robust against partial occlusion from the environment.

(2) *Tracking time and computational cost.* Due to the theoretical efficiency, the tracker must be faster than any tracking-by-detection method. Moreover, it must specify the computational cost to attain this speed.

(3) *Memory consumption.* The amount of memory the tracker consumes from RAM

for a single target should be small enough to allow simultaneous tracking of multiple targets along the same sequence.

(4) *Scalability to multiple objects.* An increase in the number of simultaneously tracked objects causes an increase in (2) tracking time and computational cost, and (3) memory consumption in comparison to tracking a single object. Moreover, it emphasizes how additional objects affect the (1) robustness of the algorithm.

In addition, for all learning-based methods, it is also essential to consider the:

(5) *Learning time.* This includes the creation of the learning dataset from loading or rendering images to extracting the input (samples) and output (labels) parameters, and the construction of the machine learning data structure. It is particularly important for online tracking, where the object has to be incrementally learned in the successive frames at real-time.

Therefore, the novelty of this work is that it satisfies all of the aforementioned attributes simultaneously, while achieving better results against the other methods individually.

Hence, in contrast to [133], the proposed tracker overcomes their limitations through an algorithm that (1) is more robust to holes and partial occlusions, (3) has a very low memory footprint, (4) is scalable to track a hundred objects in real-time and (5) has a fast learning time, while keeping the existing attributes regarding (2) low tracking time with a low computational expense. Due to the fast learning time, we extended the work to online model-free tracking. In this case, the tracker initializes the geometry of the target from a single depth frame and adapts it to changing geometry and unseen camera viewpoints while tracking.

Our main theoretical contribution is two-fold. On one hand, we propose a novel occlusion handling strategy that adapts the choice of the input samples being learned. In effect, this notably increases the overall robustness, as proven through the state-of-the-art results reported by our method on benchmark datasets (see Section 5.4). On the other hand, in lieu of the learning strategy employed by [133], we propose to use only one depth image to create the entire learning dataset, or the entire set of samples and labels for each camera view. This leads to a novel formulation of the learning strategy, which allows a much denser sampling of the camera viewpoints with respect to [133]. As a consequence, we achieve not only a high scalability, but also a remarkably low memory footprint and fast learning time, that allows our proposal to be deployed in an online 3D tracking context, which initializes the geometry of the target from a single depth frame, and adapts it to changing geometry and unseen camera viewpoints while tracking.

5.2 Related Work

If we limit our scope to temporal trackers that estimate the object's pose using solely depth images, there are only two existing methods – the energy-minimization such as Iterative Closest Point (ICP) algorithms [14, 27] and a learning-based algorithm [133]. Most works [2, 55, 63, 95] have applied ICP as an integral component of their algorithms; while, others [42, 107, 118, 123] have developed it to different extensions. Nonetheless, to the best of our knowledge, there have been only one learning-based object temporal tracking algorithm that relies solely on depth images [133].

Furthermore, there are several works that have utilized the RGB-D data. This includes the hand-held object tracking [50] that uses RGB to remove the hand before running ICP. Moreover, the particle filter approaches [28, 74] extends existing RGB trackers to include the depth data. Another work [111] uses level-set optimization with appearance and physical constraints to handle occlusions from interacting objects; but, they only conduct their experiments on texture-less objects with simple geometric structure such as prisms or spheres. Among the RGB-D methods [28, 50, 74, 111], it is common to implement them in GPU for real-time tracking. In effect, their runtime depends on the type of GPU that they use, which creates a problem to track more objects while still keeping the real-time performance.

5.3 3D Temporal Tracker

Tracking aims at solving the registration problem between the 3D points on the object and the 3D points from the depth image representing the current frame. To register these two sets of points, the error function is defined as the signed displacement of a point correspondence

$$\epsilon_j^v(\mathbf{T};\mathbf{D}) = \mathbf{N}_v \cdot \left(\mathbf{T}^{-1}\mathcal{D}(\mathbf{x}_j) - \mathbf{X}_j\right) \tag{5.1}$$

where $\mathbf{X}_j$ is a point on the object in the object coordinate system, $\mathbf{N}_v$ is a unit vector that defines the direction of the displacement (see (5.3)), $\mathbf{T}$ is the object transformation from the camera (see (5.2)), $\mathbf{x}_j$ is the projection of $\mathbf{T}\mathbf{X}_j$, and $\mathbf{D}$ is the depth image with $\mathcal{D}(\mathbf{x})$ as the back-projection function of the pixel $\mathbf{x}$. As notations, we include a tilde as $\tilde{\mathbf{x}}$ to denote inhomogeneous coordinates while $\mathbf{x}$ as homogeneous.

The objective of tracking is to locate the object in the image by finding the transformation that registers the points on the object to the points from the depth image. Specifically, object temporal trackers seek the transformation $\hat{\mathbf{T}}_t$ from the frames at time $t-1$ to t and transform $\mathbf{X}_j$ by $\mathbf{T}_t = \prod_{i=0}^{t} \hat{\mathbf{T}}_i$. In the current frame $\mathbf{D}_t$, it utilizes the displacement of the points $\epsilon_j^v(\mathbf{T}_{t-1};\mathbf{D}_t)$ to determine the relative transformation $\hat{\mathbf{T}}_t$ that minimizes $\epsilon_j^v(\mathbf{T}_t;\mathbf{D}_t)$.

Instead of aggregating the errors as $\sum_j |\epsilon_j^v|^2$ in energy minimizers, we take the individual values of the signed displacements $\epsilon_j^v(\mathbf{T}_{t-1};\mathbf{D}_t)$ from n_j points on the object $\{\mathbf{X}_j\}_{j=1}^{n_j}$ as the input to the random forest [21] and predict the transformation parameters of $\hat{\mathbf{T}}_t$. However, similar to energy minimizers, the tracker runs several iterations on each frame to refine the predicted pose.

Parametrization. The rigid transfomation $\mathbf{T}$ is constructed with the Euler angles α, β and γ, and the translation vector $\tilde{\mathbf{t}} = (t_x, t_y, t_z)^\top$ such that

$$\hat{\mathbf{T}} = \mathbf{R}_x(\alpha) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma) \cdot \begin{bmatrix} \mathbf{I}_{3\times3} & \tilde{\mathbf{t}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{5.2}$$

with the parameter vector $\boldsymbol{\tau} = [\alpha, \beta, \gamma, \tilde{\mathbf{t}}^\top]^\top$.

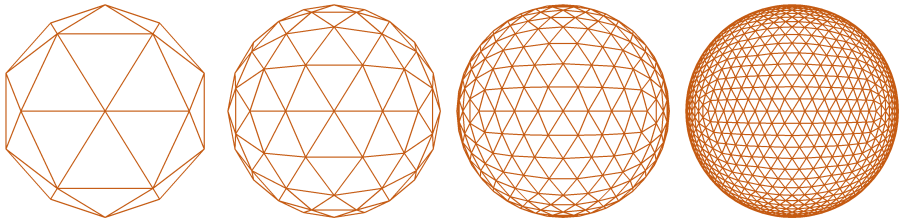


Figure 5.1: The geodesic grids, which locate the camera around the target object, are derived from recursively dividing an icosahedron with 12 vertices to (a) 42, (b) 162, (c) 642 and (d) 2562 vertices.

Dense camera. When the object moves during tracking, its viewpoint changes and the visible points on the object also vary accordingly. Thus, to ensure the capacity to track the object from different viewpoints, the algorithm learns the relation between the error function and the transformation parameters from different viewpoints or camera views. It follows that, in tracking, the closest camera views have the highest similarity to the current frame and only the trees from these views are evaluated to predict the relative transformation.

For instance, in model-based tracking, n_v views of the object's model are synthetically rendered by positioning the camera on the vertices of a densely-sampled geodesic grid [120] around the object. This is created by recursively dividing an icosahedron into equally spaced n_v vertices, as shown in Figure 5.1. By increasing n_v, the distance between neighboring cameras is decreased. In effect, the trees from multiple neighboring camera views predict the output parameters, instead of evaluating a number of trees from one view in [133]. Consequently, we can significantly decrease the number of trees per view in comparison to [133]. Thus, each view independently learns one tree per parameter using the corresponding rendered image. This produces a total $6n_v$ trees in the forest from all views.

Although one can argue to increase the number of camera views for [133], this is impractical because of the time required to generate the increased number of rendered images. As an example, when using 642 views in Figure 5.1(c), they need a total of 32.1M images for the learning dataset from all camera views, while our method needs 642 images, *i.e.* one for each camera view.

Whether using synthetic or real depth images, the input to learning from one view is a depth image $\mathbf{D}_v$ and its corresponding object transformation $\hat{\mathbf{T}}_v$. In the object coordinate system, the location of the camera $\tilde{\mathbf{X}}_v$ is

$$\tilde{\mathbf{X}}_v = -\tilde{\mathbf{R}}_v^\top \tilde{\mathbf{t}}_v \;\Rightarrow\; \mathbf{N}_v = \left(\frac{\tilde{\mathbf{X}}_v^\top}{\|\tilde{\mathbf{X}}_v\|_2}, 0 \right)^\top \tag{5.3}$$

where $\tilde{\mathbf{R}}_v$ is the 3×3 rotation matrix and $\tilde{\mathbf{t}}_v$ is the translation vector of $\hat{\mathbf{T}}_v$. From this, we define the unit vector $\mathbf{N}_v$ from (5.1) as the vector that points towards the camera center. Instead of the normal to the object's surface, the advantage of using (5.3) is evident with real depth images, where the normal to the object's surface becomes expensive to compute and prone to large errors due to sensor noise.

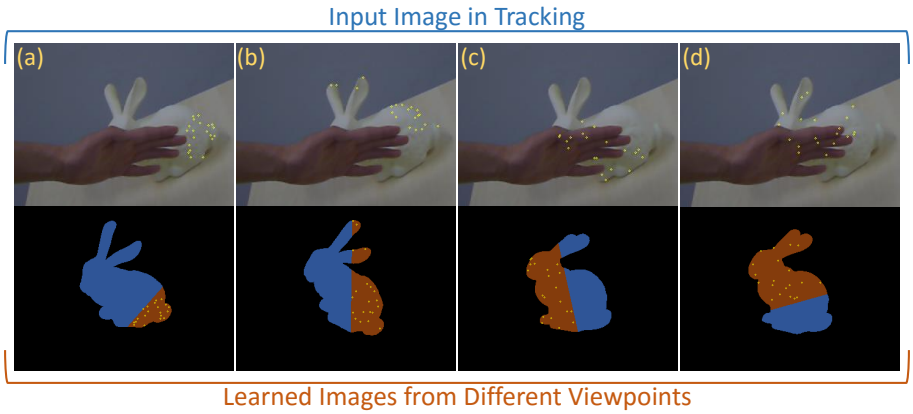


Figure 5.2: First row: occluded object when tracking. Second row: learned views where the occluded region is in blue and the points on the object, which are projected in the first row, are in yellow. Note that (a-b) are not affect by occlusion while (c-d) are affected.

5.3.1 Learning from One Viewpoint

While looking at the object from a given viewpoint v, the depth image $\mathbf{D}_v$ and the corresponding object transformation $\hat{\mathbf{T}}_v$ are taken as the input to learning. Using $\mathbf{D}_v$ and $\hat{\mathbf{T}}_v$ from only one view of the object, the visible points on the object are extracted to create the learning dataset and, eventually, learn the trees. Among the pixels $\{\mathbf{x}_i\}_{i=1}^{n_i}$ from $\mathbf{D}_v$ that are on the object, n_j points are selected, back-projected and transformed to the object coordinate system. These are the set of points on the object $\chi_v = \{\mathbf{X}_j\}_{j=1}^{n_j}$ that are used to compute the displacements in (5.1). As a consequence, we are tracking the location of χ_v across time by transforming them with $\mathbf{T}_t$.

Occlusion handling. Even though randomly selecting a subset of points on the object endows the tracker with robustness against small holes on the depth image [133], occlusions still affect its performance. By observation, we describe an occlusion on an image as a 2D obstruction that covers a portion of an object starting from an edge of the object's silhouette, while the other regions are visible to the camera, as demonstrated in Figure 5.2. Using this observation, the object on the image are divided into two regions using a line with a unit normal vector $\mathbf{n}_l$, where the pixels from one region is selected for χ_v and $\mathbf{n}_l$ is a random unit vector within the 2π unit circle. Thereupon, the pixels are sorted based on $d_i = \mathbf{n}_l \cdot \mathbf{x}_i$ such that the pixels with a lower value are located on one edge of the object while the pixels with a higher value are on the opposite edge. Hence, only the first 10% to 70% of the sorted pixels are included for the selection of χ_v, where we randomly choose the percentage of pixels. In effect, occlusion is handled by discarding a subregion of the object and selecting the set of points χ_v from the remaining subregion as illustrated in Figure 5.2.

Dataset. To build the learning dataset from $\mathbf{D}_v$, $\hat{\mathbf{T}}_v$ and χ_v, the rotation angles and translation vector in $\boldsymbol{\tau}_r$ of (5.2) are randomly parametrized to compose $\hat{\mathbf{T}}_r$ and formulate

$\mathbf{T}_r = \hat{\mathbf{T}}_v \hat{\mathbf{T}}_r^{-1}$. By transforming $\mathbf{X}_j$ by $\mathbf{T}_r$, it emulates the location of the points from the previous frame such that the current frame needs a transformation of $\hat{\mathbf{T}}_r$ to correctly track the object. Consequently, $\mathbf{T}_r$ is used to compute the displacement vector $\boldsymbol{\epsilon}_r^v = [\epsilon_j^v(\mathbf{T}_r; \mathbf{D}_v)]_{j=1}^{n_j}$. After imposing n_r random parameters, the accumulation of $\boldsymbol{\epsilon}_r^v$ and $\boldsymbol{\tau}_r$ builds the learning dataset $\mathcal{S} = \{(\boldsymbol{\epsilon}_r^v, \boldsymbol{\tau}_r)\}_{r=1}^{n_r}$. In this way, the forest aims at learning the relation between $\boldsymbol{\epsilon}$ and $\boldsymbol{\tau}$; so that, when $\boldsymbol{\epsilon}$ is given in tracking, the forest can predict $\boldsymbol{\tau}$.

Learning. Given the dataset $\mathcal{S}$, learning aims at splitting $\mathcal{S}$ into two smaller subsets to be passed down to its children. The tree grows by iteratively splitting the inherited subset of the learning dataset $\mathcal{S}_N$ and passing down the resulting $\mathcal{S}_l$ and $\mathcal{S}_r$ to its left and right child. The objective is to split using $\boldsymbol{\epsilon}$ while optimizing a parameter in $\boldsymbol{\tau}$ to make the values more coherent which is measured by the standard deviation $\sigma(\mathcal{S})$ of the parameter from all $\boldsymbol{\tau}$ in $\mathcal{S}$.

To split $\mathcal{S}_N$ into $\mathcal{S}_l$ and $\mathcal{S}_r$, we follow the same procedure as Section 4.3.2. An element of the vector $\boldsymbol{\epsilon}$ across all $\mathcal{S}_N$ is thresholded such that all values that are less than the threshold goes to $\mathcal{S}_l$ while the others go to $\mathcal{S}_r$. All of the n_j elements of $\boldsymbol{\epsilon}$ and several thresholds that are linearly space between the minimum and maximum values of the each element across $\mathcal{S}_N$ are tested to split the dataset. These tests are evaluated based on the information gain computed from (4.11) where the test with highest information gain gives the best split. As a result, the index of the element in the vector and the threshold that gives the best split are stored in the node.

The tree stops growing if the size of the inherited learning dataset is too small or the standard deviation of the parameter is less than a threshold. Then, this node is a leaf and stores the mean and standard deviation of the parameter.

Consequently, the same learning process is applied for each of the parameters in $\boldsymbol{\tau}$ to grow one tree per parameter. It is also applied to all of the n_v views of the object.

5.3.2 Tracking an Object

When tracking an object at time t, the given input is the current frame $\mathbf{D}_t$, the object transformation from the previous frame $\mathbf{T}_{t-1}$ and the learned forest with $6n_v$ trees. Ultimately, the forest predicts the parameters of $\hat{\mathbf{T}}_t$ and updates the object transformation from $\mathbf{T}_{t-1}$ to $\mathbf{T}_t$.

From the n_v views of the object, a subset of the trees are selected such that the object's viewpoint shows the highest similarity with the current frame. To compare the camera views in the model coordinate system, the camera location

$$\tilde{\mathbf{X}}_{t-1}^c = -\tilde{\mathbf{R}}_{t-1}^\top \tilde{\mathbf{t}}_{t-1} \tag{5.4}$$

is estimated based the pose from $t-1$, where $\tilde{\mathbf{R}}_{t-1}$ is the 3×3 rotation matrix of $\hat{\mathbf{T}}_{t-1}$ and $\tilde{\mathbf{t}}_{t-1}$ is its translation vector. It follows that the unit vector pointing towards the camera is derived as

$$\mathbf{N}_{t-1} = \left(\frac{(\tilde{\mathbf{X}}_{t-1}^c)^\top}{\|\tilde{\mathbf{X}}_{t-1}^c\|_2}, 0 \right)^\top . \tag{5.5}$$

Then, the comparison between the current view of the object from the learned views is measured through the angular distance between $\mathbf{N}_{t-1}$ and $\mathbf{N}_v$ for all views. Thus, the subset of trees chosen for evaluation is composed of the trees with the camera view that are within the neighborhood of $\mathbf{N}_{t-1}$ such that the angular distance is less than τ_n.

To evaluate on the v-th view, $\boldsymbol{\epsilon}^v_{t-1} = [\epsilon^v_j(\mathbf{T}_{t-1}; \mathbf{D}_t)]^{n_j}_{j=1}$ is constructed as the input to the trees. The threshold for $\boldsymbol{\epsilon}^v_{t-1}$ at each node guides the prediction to the left or right child until a leaf is reached. Each leaf stores the predicted mean and standard deviation of a parameter. After evaluating the trees from all neighboring views, the final prediction of a parameter is the average of the 20% predicted means with the least standard deviation. As a result, the average parameters are used to assemble the relative transformation $\hat{\mathbf{T}}_t$ and we execute n_k iterations.

It is noteworthy to mention that, by taking the trees from a neighborhood of camera views and by aggregating only the best predictions, our algorithm can effectively handle large holes and occlusions. Indeed, as demonstrated in Figure 5.2, some trees are affected by occlusion, the others can efficiently predict the correct parameters.

5.3.3 Online Learning

When tracking an object in real scenes, there are situations when its 3D model is not at hand, which makes model-based tracking impossible. To track in these scenarios, we propose to deploy 3D *online* tracking, where, starting from a single 3D pose on an initial depth image, the target object is adaptively learned through the successive frames while being tracked, under unseen camera viewpoints and appearance changes. In contrast to learning a model-based tracker, only the depth image $\mathbf{D}_v$ is given while the corresponding ground truth object transformation $\hat{\mathbf{T}}_v$ is unknown.

For this approach, it is necessary to attain not only tracking efficiency but also learning efficiency. Our proposed tracking algorithm, through its attributes in terms of efficiency and memory footprint, suits nicely to this application. In particular, from one frame to the next, we propose to incrementally add new trees to the forest from different object viewpoint. To achieve this goal, the online learning is initialized by defining the object to learn in the first frame and a 3D bounding box that encloses the object. It follows that the centroid of the box is the origin of the object and the object transformation of the initial frame $\hat{\mathbf{T}}_0$ is the translation from the camera center to the centroid. The bounding box defines the constraints of the object in 3D space and segments the object for learning. Thereafter, Section 5.3.1 is used to learn with the segmented image from $\mathbf{D}_t$ and the object transform $\mathbf{T}_t$ as input. The initial frame needs to learn n_t trees per parameter to stabilize the forest for tracking the object in the next frames; while, the succeeding frames learn one tree per parameter. In this case, the geodesic grid from Figure 5.1 is used to avoid re-learning trees from similar viewpoints. Thus, we find the closest vertex of the grid from the camera location in the object coordinate system and impose to have only one set of trees in each vertex.

5.4 Experimental Results

This section evaluates the proposed tracking algorithm by taking into consideration, one at a time, the five essential attributes already discussed in Section 5.1 – (1) robustness,

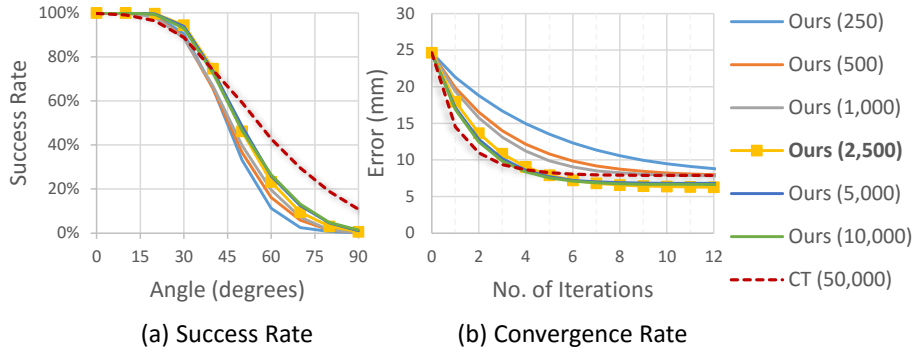


Figure 5.3: (a) Success rate and (b) convergence rate of our proposal with varying sizes of the learning dataset compared against CT [133].

(2) tracking time and computational cost, (3) memory consumption, (4) scalability to multiple objects and (5) learning time.

5.4.1 Robustness

To evaluate the robustness of our algorithm, we use three benchmark datasets [28, 54, 133]. The evaluation of the first dataset [54] determines the optimum parameters utilized throughout Section 5.4 and compares against the Chameleon Tracker (CT) [133]; the second [28] compares the accuracy of the transformation parameters against the RGB-D particle filter approaches [28, 74, 119]; finally, the third [133] compares the robustness of our approach against other trackers [1, 133] based on depth images only. Notably, across all the datasets, our work only uses the depth images of the RGB-D sequences.

Optimum Parameters. The driller dataset from [54] is composed of its model and 1,188 real RGB-D images with the ground truth pose of the object in each image. This evaluation focuses on the robustness of the algorithm to track an object in the current frame given its pose in the previous frame. To mimic the transformation of the previous frame, the ground truth pose is randomly translated and rotated using the Rodrigues' rotation formula [38, 117]. Thereafter, the tracker estimates the object's pose and the error of the estimated pose is computed based on the average distance between the corresponding vertices from the ground truth pose and the estimated pose.

From this error, the effects of different parameters on the tracker are observed through the success rate and the convergence rate. According to [54], a successfully estimated pose has the error value below 0.1 of the object's diameter. Moreover, the convergence rate takes the average error across the entire dataset for each of the iterations. These evaluations aim at finding the optimum parameters that produce the best results and to compare with CT [133].

In learning, there are two main aspects that affect the performance of the trees. These are the size of the learning dataset n_r and the number of camera views from the geodesic grid n_v. With regards to the size of the learning dataset, Figure 5.3 illustrates that there is no significant difference in both success rate and convergence rate between 2500, 5000 and 10000; while, with the number of camera views, Figure 6.2 shows that

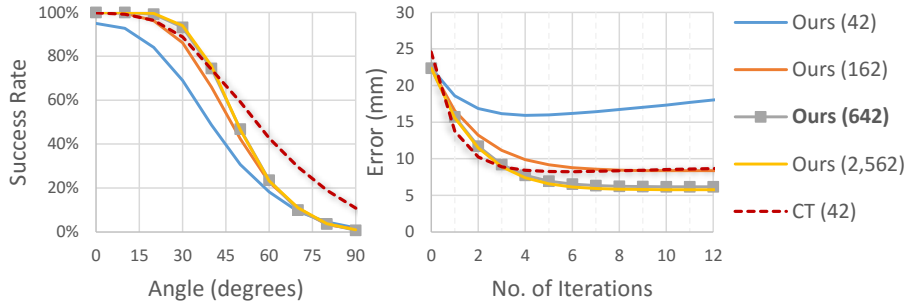


Figure 5.4: (a) Success rate and (b) convergence rate of our proposal with different number of camera views in the geodesic grid compared against CT [133].

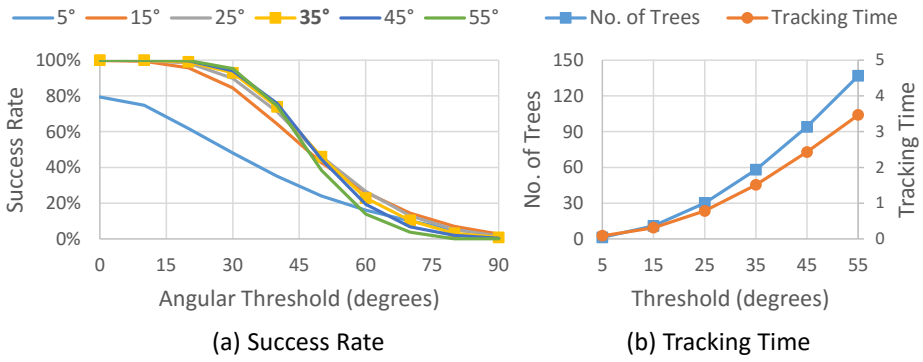


Figure 5.5: (a) Success rate, and (b) tracking time and number of trees with respect to the angular distance threshold within the neighborhood of the camera location that is used in tracking.

increasing from 642 to 2562 does not change the performance of the tracker. Thus, the optimum parameters for learning is 2,500 pairs of samples and labels with 642 camera views. Furthermore, based on the convergence rate in Figure 5.3(b) and Figure 6.2(b), 10 iterations ensures that the tracker converges to a low error value. We also look into the angular distance threshold θ of the neighboring trees when tracking. In Figure 5.5(a), the success rate starts to converges with a threshold of 35°. On average, this corresponds to evaluate 58 trees from Figure 5.5(b). For the rest of the evaluation, we use the parametric values of $n_r = 2500$, $n_v = 642$, $n_j = 20$ and $\theta = 35°$.

Compared to CT [133], we have a higher success rate when the relative motion is below 40°, while their success rate is higher above 40° in Figure 5.3. Considering that an object temporal tracker estimates the transformation of the object between two consecutive frames, the success rates below 40° are, in terms of application, more relevant than the ones above. Furthermore, the error in their convergence rate initially drops faster than ours but we converge to a lower error value after as few as 4 iterations.

Synthetic Dataset. We evaluate our tracker on the publicly available synthetic dataset of [28]. It consists of four objects: each object has its model and 1,000 RGB-D images with the ground truth pose of the object. This evaluation aims at comparing the accuracy between the RGB-D particle filter approaches [28, 74, 119] and our method in estimating the rigid transformation parameters, *i.e.* the translation in the x-, y- and z-axis, and the roll, pitch and yaw angles.

Table 5.1 shows that we remarkably outperform PCL [119], and Choi and Christensen [28] over all sequences. With respect to [74], there is no significant difference in the error values: on average, we are 0.01 mm better in translation and 1.01° better in rotation. However, the difference between the two algorithms lies in the input data and the learning dataset. On one hand, they use RGB-D images while we only use the depth; on the other, their learning dataset includes the object's model on different backgrounds while we only learn using the object's model. The latter implies that they predefine their background in learning and limit their application to tracking objects in known (or similarly structured) environments. Due to this, their robustness in Table 5.1 depends on the learned background and, to achieve these error values, they need to know the object's environment beforehand. This is different from our work because we only use the object's model without any prior knowledge of the environment.

Real Dataset. This evaluation aims at comparing the robustness of the trackers that use depth images only, so to analyze in details the consequences in terms of tracking accuracy arising from the typical nuisances present in the 3D data acquired from consumer depth cameras.

To achieve our goal, we use the four real video sequences from [133] (see Figure 5.6(a-d)) as well as an additional sequence with higher amount of occlusions and motion blur (see Figure 5.6(e)). Each sequence is composed of 400 RGB-D images and the ground truth pose of the marker board. Across the frames of the sequence, we compute the average displacement of the model's vertices from the ground truth to the estimated pose. Moreover, we compare the robustness of ICP [1], CT [133] and our approach in the presence of large holes from the sensor, close-range occlusions from the surrounding objects and motion blur from the camera movement. We also compare our approach using sample points with random selection and with the selection to handle

	Errors	PCL	C&C	Krull	Ours	*Online*
(a) *Kinect Box*	t_x	43.99	1.84	**0.83**	1.54	2.25
	t_y	42.51	2.23	**1.67**	1.90	3.92
	t_z	55.89	1.36	0.79	**0.34**	1.82
	Roll	7.62	6.41	1.11	**0.42**	3.40
	Pitch	1.87	0.76	0.55	**0.22**	1.00
	Yaw	8.31	6.32	1.04	**0.68**	2.23
	Time	4539	166	143	**1.5**	1.1
(b) *Milk*	t_x	13.38	0.93	**0.51**	1.23	0.86
	t_y	31.45	1.94	1.27	**0.74**	1.02
	t_z	26.09	1.09	0.62	**0.24**	0.42
	Roll	59.37	3.83	2.19	**0.50**	1.66
	Pitch	19.58	1.41	1.44	**0.28**	1.14
	Yaw	75.03	3.26	1.90	**0.46**	1.29
	Time	2205	134	135	**1.5**	1.3
(c) *Orange Juice*	t_x	2.53	0.96	**0.52**	1.10	1.55
	t_y	2.20	1.44	**0.74**	0.94	1.64
	t_z	1.91	1.17	0.63	**0.18**	1.55
	Roll	85.81	1.32	1.28	**0.35**	2.94
	Pitch	42.12	0.75	1.08	**0.24**	2.37
	Yaw	46.37	1.39	1.20	**0.37**	4.71
	Time	1637	117	129	**1.5**	1.2
(d) *Tide*	t_x	1.46	0.83	**0.69**	0.73	0.88
	t_y	2.25	1.37	0.81	**0.56**	0.81
	t_z	0.92	1.20	0.81	**0.24**	0.36
	Roll	5.15	1.78	2.10	**0.31**	0.86
	Pitch	2.13	1.09	1.38	**0.25**	1.03
	Yaw	2.98	1.13	1.27	**0.34**	2.51
	Time	2762	111	116	**1.5**	1.2
Mean	Translation	18.72	1.36	0.82	**0.81**	1.42
	Rotation	29.70	2.45	1.38	**0.37**	2.10
	Time	2786	132	131	**1.5**	1.2

Table 5.1: Errors in translation (mm) and rotation (degrees), and the runtime (ms) of the tracking results, evaluating with the synthetic dataset [28], of PCL [119], Choi and Christensen (C&C) [28], Krull *et al*. [74], and our approach with the model-based offline learning (Ours) as well as the image-based online learning (Online).

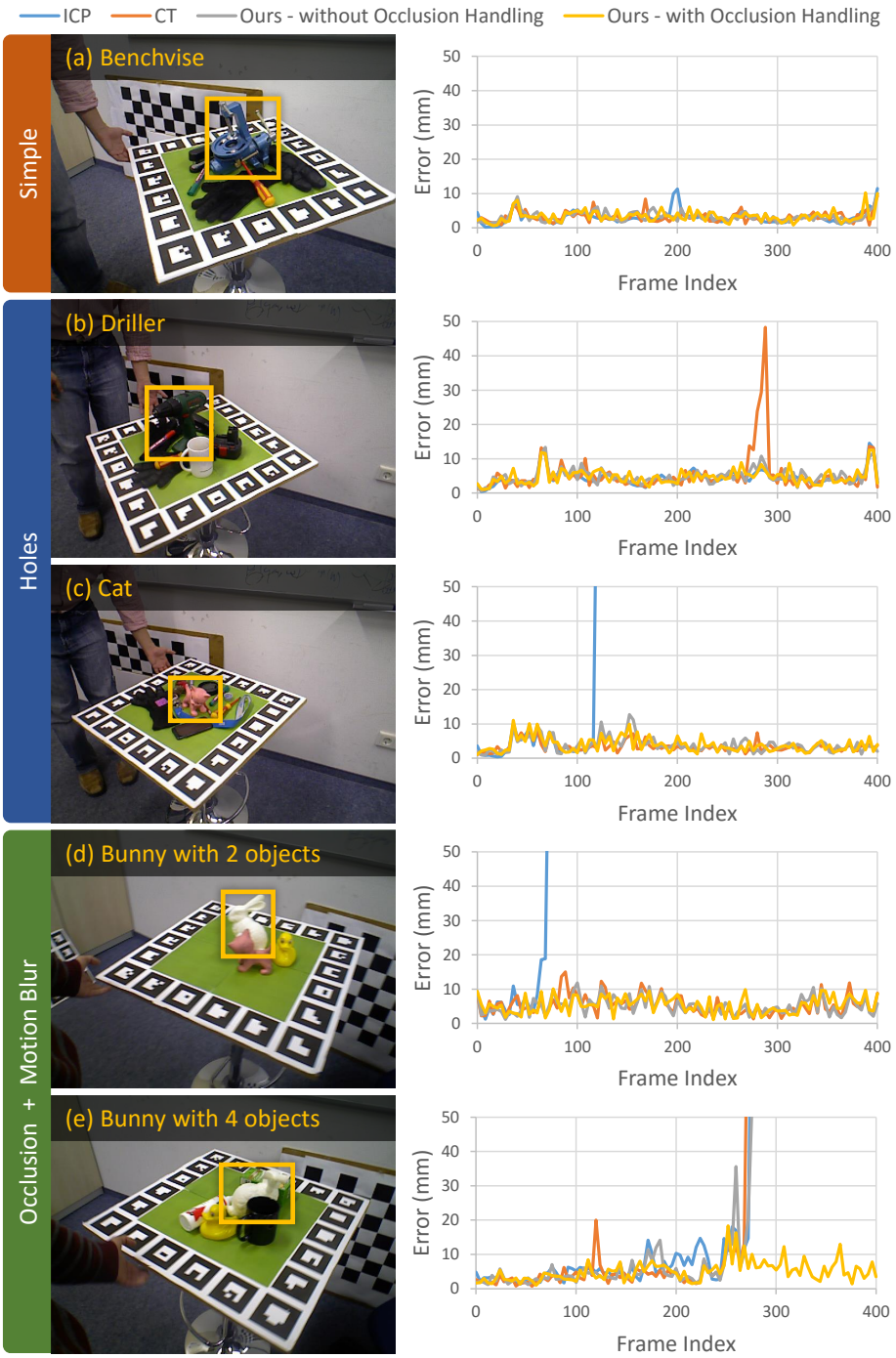


Figure 5.6: Tracking comparison on the dataset of [133] among ICP [1], CT [133], and our approach with and without the occlusion handling sample points selection.

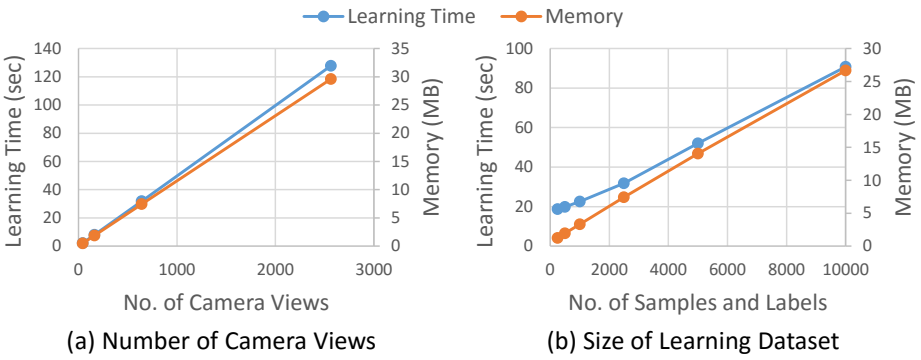


Figure 5.7: Learning time and memory usage with respect to (a) the number of camera views and (b) the size of the learning dataset.

occlusions.

The first sequence in Figure 5.6(a) is a simple sequence with small holes and small occlusions, where all trackers perform well. Next, the *driller* sequence illustrates the effects of large holes due to the reflectance of its metallic parts. It generates instability on CT [133] that is highlighted by the peak in Figure 5.6(b). Even if it did not completely lose track of the object, this instability affects the robustness of the tracker in estimating the object's pose. On the contrary, ICP [1] and both versions of our method track the driller without any instability.

As reported in [133], the toy *cat* sequence in Figure 5.6(c) causes ICP [1] to get trapped in a local minimum. When the cat continuously rotate until its tail is no longer visible due to holes and self-occlusion, the relatively large spherical shape of its head influences the error in the pose estimation and stays in that position for the succeeding frames. In contrast, CT [133] and our method track the toy cat without getting trapped in a local minimum.

The last two sequences in Figure 5.6(d-e) present two important challenges. First, they exhibit close-range occlusions where the surrounding objects are right next to the object of interest. This introduces the problem in determining whether nearby objects are part of the object of interest or not. The second is the constant motion of both the camera and the object. This induces motion blur on the depth image which, in turn, distorts the 3D shape of the object.

Since the surrounding objects are close to the object of interest, ICP [1] fails in both sequences. When occlusions occur, it starts merging the point clouds from the nearby objects into the object of interest and completely fails tracking. For CT [133], it becomes unstable when occluded but completely recovers in Figure 5.6(d). But, when larger occlusions are present such as Figure 5.6(e), tracking fails. In comparison, our method with the random sample point arrangement has a similar robustness as [133]. However, our method, by modeling the sample points to handle occlusions smoothly, is able to track the object without any instability or failures. Among the competing methods, it is the only one that is able to successfully track the *bunny* in the last sequence.

5.4.2 Tracking Time and Computational Cost

As witnessed in Figure 5.5(b), the tracking time increases with respect to the number of trees evaluated or the number of camera views included. Using the optimum parameters from Section 5.4.1, the algorithm runs at 1.5 ms per frame on an Intel(R) Core(TM) i7 CPU, where only one core is used. This is comparable to the time reported by CT in [133]. With regards to the competing approaches [28, 74, 119] in Table 5.1, their work takes about 100 times longer than ours while producing slightly higher error values. Among them, [28, 74] optimize their runtime through GPU.

5.4.3 Memory Consumption

Our memory consumption increases linearly with the number of camera views and the size of the learning dataset for each view, as shown in Figure 5.7(a) and (b), respectively. With the parameters from Section 5.4.1, our forests needs 7.4 MB. Compared to [133] which uses 821.3 MB, our memory requirement is two orders of magnitude less. Most of the related works do not mention or disregard this measurement from their papers, but we argue that it is an important aspect especially with regards to scalability towards tracking multiple objects.

5.4.4 Scalability to Multiple Objects

When tracking multiple objects, we utilize independent trackers for each object. It follows that the tracking time and the memory usage increase linearly with respect to the number of objects, where an increased computational expense, *i.e*. additional CPU cores, divides the resulting tracking time by the number of cores. Furthermore, the independence of the trackers for different objects keeps the robustness of the algorithm unaffected and the same as Section 5.4.1.

Considering a typical computer with 8 GB RAM and 8 CPU cores, a memory consumption of 7.4 MB for each object allows us to include more than 1,000 objects into RAM. In contrast to CT [133] where they use 821.3 MB for each object and reached a maximum limit of 9 objects, our tracker can include at least two orders of magnitude more objects in memory than [133].

To demonstrate the remarkable scalability of our approach, we synthetically rendered 108 moving objects with random initial rotations in a 3D video sequence as shown in Figure 5.8; apply one tracker for each object, requiring a total memory footprint of, approximately, 108×7.4 MB $= 799.2$ MB; and, track them independently. By using 8 CPU cores, our work tracks all 108 objects at 33.7 ms per frame, *i.e*. yielding a frame rate of 30 fps. Interestingly, our memory requirement for 108 objects is less than that required for just one object by CT; while, our tracking time for 108 objects is less than the GPU implementations of [28, 74] that track one object at 130 ms per frame. It is important to mention that we had to resort to a rendered input video given the difficulty of recreating a similar scenario with so many moving objects under real conditions. Nevertheless, since we only aim at evaluating the scalability of our approach, we can expect an identical performance under real conditions.

Therefore, although scalability is linear with respect to the number of objects, we highlight the extremely low magnitude of all the important components – (2) tracking

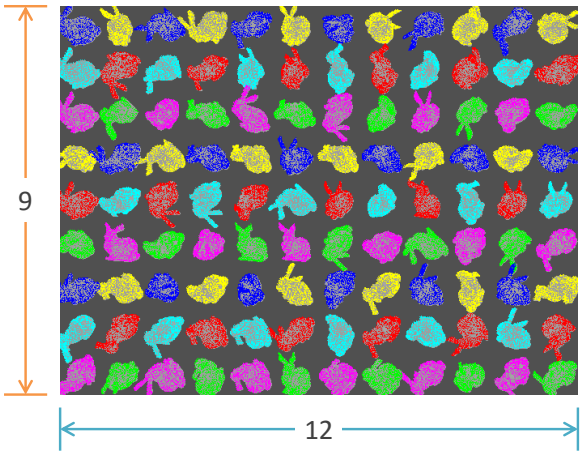


Figure 5.8: Evaluation on 108 moving objects in a 640×480 depth image.

time and computational cost, and (3) memory consumption – that makes tracking a hundred objects at real-time possible.

5.4.5 Learning Time

The learning time has a linear relation with respect to the number of camera views and the size of the learning dataset as shown in Figure 5.7. Thus, with the optimum parameters from Section 5.4.1 and 8 CPU cores, it requires 31.8 seconds to learn the trees from all of the 642 camera views using 2,500 pairs of sample and label. This is significantly lower than the 12.3 hours of CT [133]. Even with an increased number of camera views or a larger learning dataset in Figure 5.7, our learning time remains below 140 seconds.

Online learning. One of the most interesting outcomes of the fast learning time is the online learning where the tracker does not require the object's 3D model as input to learning.

We use the dataset of [28] to evaluate our online learning strategy. In the first frame, we use the ground truth transformation to locate the object and start learning with 50 trees per parameter, which takes 1.3 seconds. The succeeding frames continues to learn one tree per parameter, which takes 25.6 ms per frame. In Table 5.1, the average tracking error of the online learning is comparable to the results of Choi and Christensen [28]. It performs worse than the model-based trackers with offline learning of Krull *et al.* [74] and ours, but performs better than PCL [119]. Furthermore, the combined learning and tracking time, which is approximately 26.8 ms per frame using 8 CPU cores, is still faster than the competing approaches [28, 74, 119] that only execute tracking.

5.4.6 Failure Cases

Since we are tracking the geometric structure of the objects through depth images, the limitation of the tracker is highly related to its structure. Similar to ICP [14, 27], highly

symmetric objects loses some degrees of freedom in relation to its axis of symmetry. For instance, a bowl that has a hemispherical structure loses one degree of freedom because a rotation around its axis of symmetry is ambiguous when viewed from the depth image. Therefore, although our algorithm can still track the bowl, it fails to estimate the full 3D pose with six degrees of freedom.

Specific to online learning, large holes or occlusions in the initial frames create problems where the forest has not learned enough trees to describe the object's structure. Due to this, drifts occur and tracking failures are more probable.

5.5 Qualitative Results

We demonstrate the applicability of the tracker on different scenarios. The first example in Figure 5.9(a) showcases the ability to track the object under large occlusion.

Considering the very efficient memory footprint, tracking time and computational cost, the algorithm can independently track multiple distinct objects in the scene at real-time performance. Although scalability with the number of objects is linear, the low requirement of the tracker allows it to simultaneously track three objects in Figure 5.9(b).

The other examples in Figure 5.10 illustrates the online learning capabilities of the work where the model is not given. In this case, we manually initialize a 3D bounding box on the object of interest then continuously learn new trees in the forest for the unseen views and track the object in the scene.

An interesting application of the online learning scheme is the head pose estimation in Figure 5.10(b). Evidently, this is not an optimum framework for a head pose estimation because different subjects follow different coordinate systems depending on the way the 3D bounding box is initialized. In next chapter, we investigate this application further in order to allow a standard coordinate system for different subjects as well as incorporating subject-specific structures through online learning.

5.6 Conclusion

In addition to the extremely fast tracker achieved in Chapter 4, this chapter investigates various aspects of the tracker in order to make it versatile for different uses. By replacing the learning method, the resulting tracker highlights the capacity to be robust even under large occlusions, to have low memory footprint, to track multiple objects in the scene and to learn an object fast. The speed in learning allows us to learn different views of an unknown object online.

Figure 5.9: Qualitative results of the (a) robustness to occlusion and (b) scalability to track multiple objects.

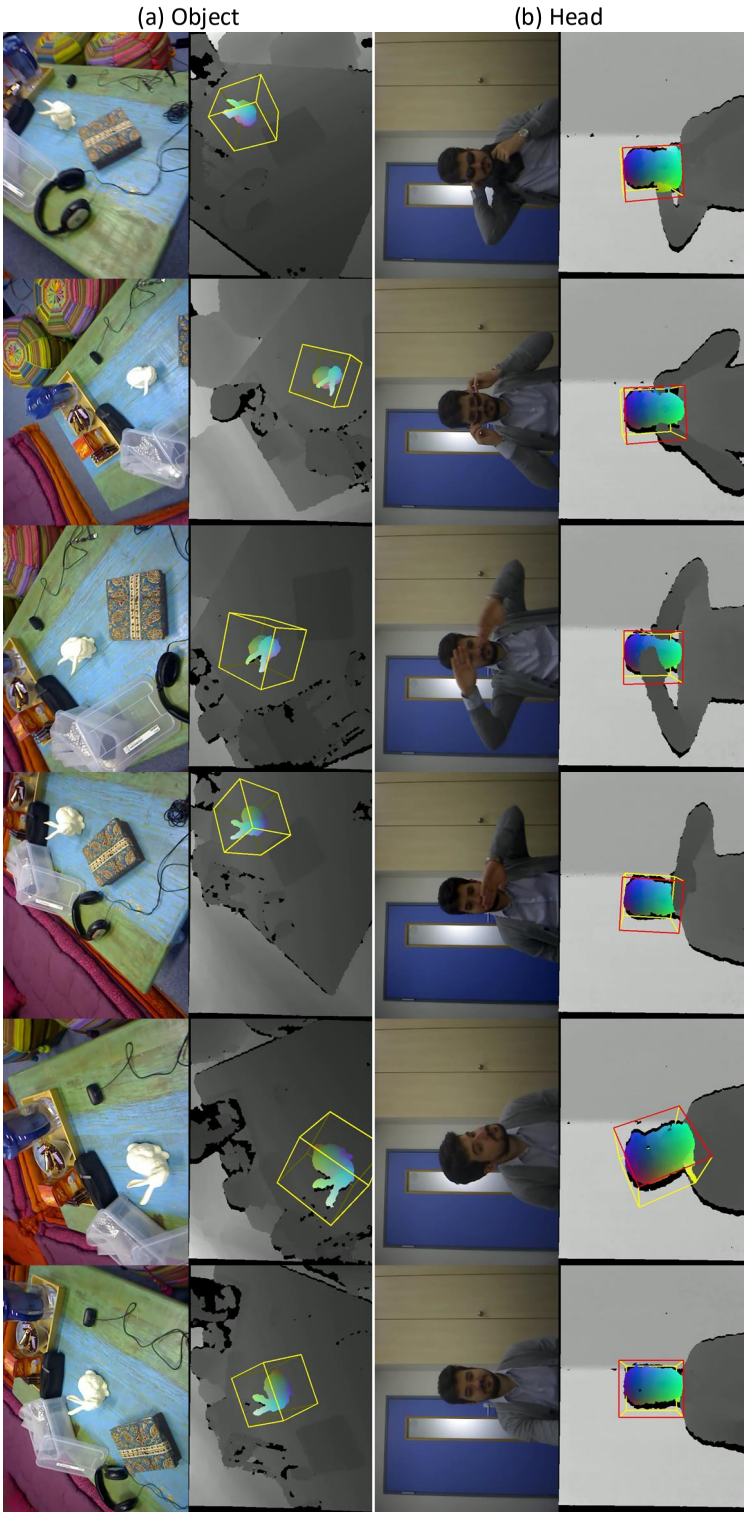


Figure 5.10: Qualitative results of the online learning framework applied on (a) an object and (b) a head.

6 3D Head Pose Estimation

6.1 Motivation

Head pose estimation aims at approximating the rotation and translation of the head and tracking it through a sequence of images. The use of 3D information can yield the 6 d.o.f. head pose in the 3D space rather than just the 2D pose on the image plane, which makes this task increasingly relevant for a high number of computer vision and augmented reality applications relying on accurate 3D head pose. For instance, 3D head pose is useful to develop Human-Computer Interaction (HCI) interfaces that estimate input commands from the user by means of specific configurations of its facial features and head movements, this being particularly exploited also by the gaming industry. In addition, head pose estimation is relevant for human behavior analysis and gaze analysis applied to faces, *e.g*. to automatically understand when the driver of a vehicle falls asleep. It is a key step for augmented reality applications, *e.g*. in the fashion industry for virtual mirrors, as well as in the context of 3D avatar creation for video conferencing and special effects.

The motivation is to showcase a complete framework that highlights the achievements from our previous works [133, 134] in Chapters 4 and 5. This work aims at achieving head pose estimation from RGB-D data by means of a frame-to-frame temporal tracking approach, that incorporates the temporal information so to estimate the head's pose throughout the video sequence. The algorithm uses depth images and random forest [21] to estimate the pose of an object from one frame to the next. However, in contrast to tracking an object with a precise 3D CAD model available offline as [133, 134] in Chapters 4 and 5, this work addresses the problem of adapting and generalizing the tracker to subject-specific variations – including slightly non-rigid – of the head structures with respect to a given pre-defined 3D model of the head. It involves combining a generalized model-based tracker with an online learning method to capture subject-specific structures that makes the tracker more robust against noise, occlusions and facial deformations.

Moreover, as a temporal tracking approach, it requires an initialization when a new face appears in the sequence or when the tracker loses the target. For this reason, we integrate an initialization stage based on a face detector [143] through the RGB images that are available from the RGB-D sequences. This allows our method to handle a

variety of situations such as the absence of people in some frames of the sequence, as well as the simultaneous presence of more than one person in the scene.

For the purpose of 3D head pose estimation, the proposed algorithm focuses on achieving a wide range of applications by satisfying all the following fundamental criteria:

(1) *Robustness*. Accuracy is the most important aspect to consider in head pose estimation given that it will serve as input to the aforementioned applications that need to observe and elaborate the interaction of the user with nearby objects or with other users. It follows that the accuracy must be robust in the presence of typical occlusions such as head-wear, props, eyeglasses and hand gestures, as well as robust in the presence of typical sensor-induced data artifacts such as holes and noise.

(2) *Efficiency*. Since most applications require real-time capabilities and feedback, efficiency is another fundamental characteristic of 3D head pose systems. We evaluate the efficiency of the algorithm in terms of the runtime per frame with the required processing power and memory consumption. Also, the ideal algorithm should attain low runtime with a low processing power.

(3) *Lax users*. The algorithm has to be capable of estimating the pose within the camera's field of view such that the users can move freely without being mindful of its position with respect to the camera. It must not restrict the users to stay on a specific distance from the camera or remain in a static position that is close to the principal point of the image.

(4) *Ease of use*. When running the algorithm, the users does not have to possess any special skill or to perform any prerequisite step or movement. It implies that the algorithm does not require any a priori input information about the user.

Hence, our approach is guided by these criteria to ensure that our algorithm is easily adaptable for several diverse applications. Conforming to them, our evaluation on the two benchmark datasets for 3D head pose estimation [22, 39] demonstrate the remarkable accuracy, efficiency and robustness of our approach, which outperforms the state-of-the-art methods in terms of accuracy under different nuisances while retaining a much higher efficiency without the need of GPU processing.

In reference to having *lax users*, this work extends the head pose estimation algorithm [135] to perform in a multi-camera system such that the head is tracked even when undergoing full-occlusions from individual cameras. Notably, our method is not limited to perform on identical camera models as the one used in learning. Since the tracker entirely relies just on 3D information, the tracker only needs one forest to track the head from different camera models despite having large variations in their intrinsic parameters.

6.2 Related Work

A number of works [39, 108, 113, 122] already exist specifically aimed at estimating the 6 d.o.f. pose of the head's rigid transformation by means of depth data acquired

from consumer RGB-D cameras. In particular, these methods are based on the tracking-by-detection paradigm, where the head detection and pose estimation is carried out in each frame independently from the previous ones.

A survey on different head pose estimation algorithms is available in [93, 94]. Several works on head pose estimation that rely on RGB images only focus on tracking facial features or landmarks instead of estimating the 3D pose of the head [6, 31, 33, 69, 73, 130], or on discretizing the pose to generate crude approximations [44, 67]. Unlike a rigid object with a constant structure that has a one-to-one relation from the image space to 3D space, the head structure differs from one subject to the next, which means that the landmarks does not have a corresponding 3D points and their estimated pose is inaccurate due to the lack of depth perception of the subject in RGB images.

As for methods based on depth images, Breitenstein *et al*. [22] proposed to build a set of hypotheses by means of high-resolution depth images to locate the nose through 3D shape signatures and evaluate the hypotheses by computing the error from the reference pose image. Due to its computational requirements, this work uses GPU to run in real-time. Later, Fanelli *et al*. [40] also used high-resolution images but achieved real-time performance without GPU implementation and can handle small occlusions. Another relevant work from Fanelli *et al*. [39] proposes a tracking-by-detection framework based on random forest from consumer depth camera data. Their splitting features involves locating two 2D patches with random offsets and sizes from a pixel and computing the difference of the mean depth values enclosed in the patches. Each pixel predicts the yaw, pitch and roll angles for the head's rotation and a 3D vector to locate the nose.

With a similar 2D image features in learning as well as the parametrization of the three angles and a vector, several forest-based methods [108, 113, 122] continued the work of [39]. Among them, Schulter *et al*. [122] proposes Alternating Regression Forests (ARFs) that relates the trees in the forest by optimizing a global loss function, where their approach on head pose estimation uses the same 2D features as [39]. In [108], they evaluate four 2D patches as features in learning and introduce the Probabilistic Locally Enhanced Voting (PLEV) to locally aggregate the predictions of their Hough forest. Finally, Riegler *et al*. [113] presents a combination of Hough Forests and Convolutional Neural Network for head pose estimation, which they call Hough Networks (HN). This approach extracts overlapping 2D patches in a regular grid on the image, where each patch is classified as foreground or background, and use the foreground patches to predict the pose parameters.

These learning-based methods [39, 108, 113, 122] use 2D features to directly predict the 3D pose. However, instead of using 2D features, our algorithm relies on the 3D data. The advantage of the using the 3D data is the capacity to use the forest for different camera models. This becomes essential especially with online learning for a multi-camera system with distinct camera models (*i.e*. different camera intrinsic parameters). After learning the subject-specific structures from one camera online, we use the same forest for all the other cameras in the system. If we rely on 2D features, it becomes necessary to utilize independent forest for each camera model. Note that there is an important benefit to use different camera models in the multi-camera system so that the depth sensors do not corrupt the data from other sensors. An example of such system is using a time-of-flight camera and a structured light camera in Figures 6.14 and 6.15.

There are also other 3D head pose estimation algorithms that rely on 3D data. In this case, they implement an energy minimization based method such as the iterative

closest point (ICP) [10, 23, 89, 93] or particle swarm optimization (PSO) [93, 101, 109] to register the model to the depth image. In general, ICP [14, 27] requires a good initialization to converge to the solution while PSO [70] has a slow convergence depending on the amount of particles. Thus, a notable work in this field is from Meyer *et al*. [93], where they combine ICP and PSO to avoid local minima due to the poor initialization from ICP and to speed-up the convergence rate of PSO. However, they reported a tracking rate of 160 ms per frame or approximately 6 frames per second on a GPU. Their efficiency is significantly low compared to our method that runs at 2 ms per frame with only CPU.

The value of having an efficient tracking framework is rooted from the wide-range of applications where the head pose estimation is simply an initial step for a more elaborate goal such HCI and AR. Efficiency is also valuable to capture the subject's fast motion. Considering these, an example where the computational efficiency becomes essential is estimating the head pose of a subject when undergoing fast motion from a multi-camera system as illustrated in Figures 6.14 and 6.15. Here, two cameras capture a subject as he jumps up in the air and lands on a mattress. Due to gravity, the entire motion takes only a few milliseconds to complete.

Moreover, although the standard datasets from [22, 39] restrict the location of the subject to be close to the principal point of the image, we illustrate several examples in Figures 6.11 and 6.12 where the lax users can freely roam around the image. Furthermore, while other approaches generalize their learning method to estimate the head pose of any subject, our work emphasizes the value of a combined generalized and subject-specific online learning framework to become robust in handling occlusions and extreme poses, where a significant amount of the facial features are not visible.

6.3 Tracking Framework

As the subject moves in a sequence of RGB-D images, the goal is to determine the 3D pose of the head in each frame in order to describe the subject's motion. Considering that we are using a model-based tracker, the coordinates of our head model in each frame will be referred to the model coordinate system. Therefore, the 3D pose is defined as the 4×4 rigid transformation matrix $\mathbf{T}$ that maps the points from the model coordinate system to the camera coordinate system. In this work, the rigid transformation $\mathbf{T}$ is parameterized the same as (5.2) with

$$\mathbf{T} = \mathbf{R}_z(\alpha)\cdot\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\gamma)\cdot\begin{bmatrix}\mathbf{I}_{3\times3} & \tilde{\mathbf{t}}\\ \mathbf{0}^\top & 1\end{bmatrix} \tag{6.1}$$

where α, β and γ are the yaw, pitch and roll angles while $\tilde{\mathbf{t}} = [t_x, t_y, t_z]^\top$ is the translation vector with $\boldsymbol{\mu} = [\alpha, \beta, \gamma, t_x, t_y, t_z]^\top$ is the parameter vector.

In contrast to tracking-by-detection algorithms such as [22, 39, 108, 113, 122] that estimate the pose independently for each frame, our work is a temporal tracker that relies on the propagation of the pose from one frame to the next to continuously register the 3D points from the head model to the depth image $\mathbf{D}$. Aiming at this goal, it relies on a random forest [21] that learns the relation between the transformation parameters of $\mathbf{T}$ and a set of displacements $\{\epsilon_h^v(\mathbf{T};\mathbf{D})\}_{h=1}^{n_h}$ from (5.1). The displacements are based

on n_h 3D points $\{\mathbf{X}_h\}_{h=1}^{n_h}$ on the head model

$$\epsilon_h^v(\mathbf{T};\mathbf{D}) = \mathbf{N}_v^\top \left(\mathbf{T}^{-1}\mathcal{D}(\mathbf{x}_h) - \mathbf{X}_h\right), \tag{6.2}$$

where $\mathbf{X}_h$ is the h-th point on the head model, $\mathbf{x}_h$ is the projection obtained as $\mathbf{T}\mathbf{X}_h$, $\mathcal{D}(\mathbf{x})$ is the back-projection of a pixel $\mathbf{x}$ through the depth image $\mathbf{D}$, and $\mathbf{N}_v$ is a unit vector that defines the direction of the displacement. Therefore, similar to Chapter 5, given the pose $\mathbf{T}_{t-1}$ at time $t-1$, the set of the displacements $\{\epsilon_h^v(\mathbf{T}_{t-1};\mathbf{D}_t)\}_{h=1}^{n_h}$ predicts the relative transformation from $t-1$ to t, denoted by $\hat{\mathbf{T}}_t$, and updates the pose at time t as $\mathbf{T}_t = \mathbf{T}_{t-1}\hat{\mathbf{T}}_t$.

To initialize the temporal tracker, the head is localized by means of a face detection algorithm applied solely on the RGB image. The resulting 2D bounding box is back-projected as a 3D bounding box on the depth image, which functions as the input to the tracker. Thereafter, the tracker refines the initial head pose estimation and continues tracking the head in the subsequent frames. It is noteworthy to mention that, while the face detection algorithm is applied on the RGB image, the temporal tracker only processes depth data such that the 3D pose is in metric scale.

Another important trait of our approach relies on combining together two tracking methods – a generalized model-based tracker with an offline learning stage, and a subject-specific tracker with an online learning framework. The algorithm begins with the former as it captures the common structure from multiple CAD models and learns a generalized tracker from synthetic depth images. Then, while tracking with the former, the latter continues learning new trees based on real depth fames to adapt the learned model and capture the unique structures of an individual subject through an online learning method.

The advantage of having a generalized tracker is its inherent attribute to align different head models across different subjects into one coordinate system. However, since the generalized tracker can only learn the common structure, it is limited to track the facial structure of the subject. This is because the 3D CAD models tend to have unrealistic non-frontal regions. In contrast, the subject-specific online learner incorporates the head structure that are unique to an individual. As a result, the combined method makes the tracker more robust to handle extreme poses which are not included in the generalized model as well as more robust to handle close-range occlusions such as hand gestures.

6.3.1 Initialization

With the initial frame at time t_0, the face detector of Viola and Jones [143] is applied on the RGB image to generate a rectangular region around the subject's face. Using the centroid of the rectangle $\mathbf{x}_c$, we propagate the detected face into 3D through the back-projection of the corresponding pixel value on the depth image $\mathcal{D}_{t_0}(\mathbf{x}_c)$, and initializing the head pose as

$$\mathbf{T}_0 = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathcal{D}_{t_0}(\mathbf{x}_c) \\ \mathbf{0}^\top & 1 \end{bmatrix}. \tag{6.3}$$

The initial pose is refined using the temporal tracker by finding $\hat{\mathbf{T}}_{t_0}$ to update the transformation as $\mathbf{T}_{t_0} = \mathbf{T}_0\hat{\mathbf{T}}_{t_0}$. For the next frames, the head pose is continuously

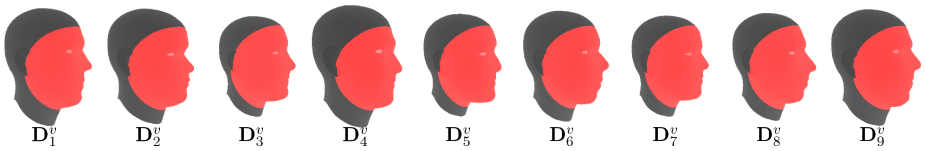


Figure 6.1: The 3D head models from different subjects from the database of [39] is rendered at a constant v-th camera view, where the common structure is highlighted in red.

estimated through the temporal tracker on the depth images. In case of tracking multiple subjects as shown in Figure 6.11(a) and (c), we impose that the back-projected pixel from the face detection is farther than 100 cm from the origin of any currently tracked subjects. Otherwise, we assume that the detected face is already being tracked.

6.4 Generalized Model-based Tracker

The goal of the generalized tracker is to temporally estimate the head pose of any subject presented on the camera, irrelevant of whether the subject is included in the set of learned CAD models or not. Using the 3D CAD models of different subjects, the algorithm learns a random forest [21] to correlate the effects of the transformations on the points on the model. As a result, the tracker is capable of estimating the head on any given subject from real depth images.

6.4.1 Common Structure

Considering that the algorithm learns from 3D CAD models and tracks from real depth images, it is necessary to determine a common structure of the head that is constantly visible among them. When comparing the CAD models of different subjects in Figure 6.1 and the depth images in Figure 6.3-6.5, the face is the only common structure across all images. This highlights the requirement of the generalized tracker to estimate the pose of the head as a result of estimating the pose of the face.

Therefore, assuming the alignment of different head models, the common structure is defined as the set of 3D points on the models that are at most τ_s away from the origin as illustrated in Figure 6.1. In relation to the error function in (6.2), the selection of the points $\{\mathbf{X}_s\}_{s=1}^{n_s}$ to be tracked must satisfy this constraint.

6.4.2 Camera Views

When the subject moves during tracking, the visible points on the model vary from viewpoint to viewpoint. Instead of learning one forest for the entire model, the tracker learns one forest for each camera view [133, 134]. The motivation is to focus each forest on learning and predicting the transformation parameters from the points on the model that are visible from the camera.

The different camera views are generated by positioning the camera at the vertices of a geodesic grid [120] with a radius of r. As illustrated in Figure 5.1 from Chapter 5, the grid is created by recursively dividing an icosahedron into equally spaced vertices.

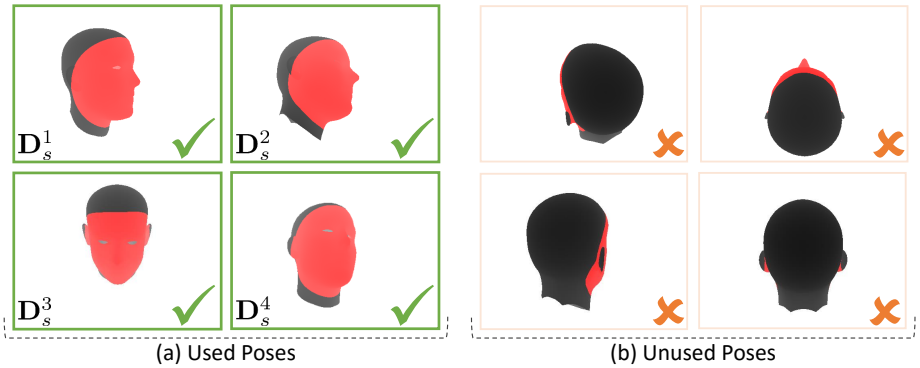


Figure 6.2: From different camera views, these are examples of the rendered depth images of a head model. Depending on the visibility of the common structure (in red), (a) are poses used for learning while (b) are not.

In this layout, the model is located at the center of the grid such that all camera views are pointing towards the model. For each view, a depth image $\mathbf{D}_s^v$ is synthetically rendered.

The generalized tracker is bounded by the common head structure. Contrary to tacking the full-view of the object from all points of the geodesic grid in [133, 134], only a subset of views are used because the common structure is not visible from all camera views. Some examples of rendered views being used and not used for learning are visualized in Figure 6.2(a) and (b), respectively. Since the common structure is defined by the face, only the frontal views of the head are rendered, *i.e.* n_v vertices of the geodesic grid that have the azimuth angle of the spherical coordinate system between $[0, \pi]$ and the polar angle between $[0, \frac{3\pi}{4}]$.

Finally, for the v-th camera view and the s-th subject, the model is transformed by means of $\mathbf{T}_v$ from the camera coordinate system and the rendered depth image is denoted as $\mathbf{D}_s^v$, such that the v-th forest learns using the depth images from the same viewpoint and across different subjects $\{\mathbf{D}_s^v\}_{s=1}^{n_s}$. Moreover, $\mathbf{N}_v$ from (6.2) is defined as the unit vector directed from the model's origin to the camera location in the geodesic grid.

6.4.3 Learning Dataset

Given the synthetic depth images at the v-th view of the n_s subjects used for training, a mean image is constructed to unify all the images. For each pixel location $\mathbf{x}$ on the image, all depth values from the n_s range maps are accumulated as $\chi = \{\mathbf{D}_s^v(\mathbf{x})\}_{s=1}^{n_s}$. Since most values of each range map will represent invalid depth measurements (*i.e.* not on the model), we take a subset of χ that have valid pixels and define each pixel at $\mathbf{x}$ on the mean image $\bar{\mathbf{D}}^v$ as

$$\bar{\mathbf{D}}^v(\mathbf{x}) = \begin{cases} \frac{1}{n_m} \sum_{m=1}^{n_m} \mathbf{D}_m^v(\mathbf{x}) & \text{if } n_m > \frac{n_s}{2} \\ \infty & \text{otherwise} \end{cases} \tag{6.4}$$

where n_m is the number of valid depth measurements for pixel $\mathbf{x}$. In practice, we invalidate a pixel on the mean image if it does not have at least half valid measurements across the n_s training subjects.

The points on $\bar{\mathbf{D}}^v$ that satisfy the constraint of common structure are collected and n_h points are randomly selected among them. After transforming them to the model coordinate system by $\mathbf{T}_v^{-1}$, the results are the set of points $\{\mathbf{X}_h^v\}_{h=1}^{n_h}$. In relation to the error function in (6.2), the goal is to register $\{\mathbf{X}_h^v\}_{h=1}^{n_h}$ on the depth image.

To build the learning dataset, we need the points on the model $\{\mathbf{X}_h^v\}_{h=1}^{n_h}$ and observe how different transformation parameters affect the error function. By introducing a transformation of $\hat{\mathbf{T}}_r$, the points are transformed by $\mathbf{T}_r = \mathbf{T}_v\hat{\mathbf{T}}_r^{-1}$ which creates the error vector $\boldsymbol{\epsilon}_r^v = [\epsilon_h^v(\mathbf{T}_r; \mathbf{D}_s^v)]_{h=1}^{n_h}$ for the n_h points on s-th subject. Here, the transformation of $\mathbf{T}_r$ emulates the location of the points from the previous frame such that a transformation of $\hat{\mathbf{T}}_r$ brings the points to its ground truth location, where the error vector is in its optimum. Therefore, after n_r random transforms on the n_s subjects, the learning dataset is assembled as $\mathcal{S} = \{(\boldsymbol{\epsilon}_r^v, \boldsymbol{\mu}_r)\}_{r=1}^{n_s \cdot n_r}$, where $\boldsymbol{\mu}_r$ is the parameter vector of $\mathbf{T}_r$.

6.4.4 Occlusion Handling

Although randomly selecting a subset of points on the model makes the tracker robust against small holes on the depth images [133] in Chapter 4, occlusion still affects the tracker's performance. Hence, instead of a random selection on the entire model, our work [134] in Chapter 5 proposes to incorporate the knowledge of occlusions as part of the selection.

We observe that most cases of occlusion consist of a 2D obstruction that covers a portion of a subject's head starting from an edge of its silhouette, while keeping the other parts of the head visible to the camera. Due to this, we propose to divide the model on the image into two regions on the image plane, such that one region is used for the selection of the points while the other is assumed to be occluded.

Using a random unit normal vector within the 2π unit circle, the pixels are sorted based on $d_i = \mathbf{n}_l \cdot \mathbf{x}_i$ such that the pixels with a lower value are located on one edge of the head while the pixels with a higher value are on the opposite edge. The first 10% to 70% of the sorted pixels are included for the selection, where the percentage of pixels is randomly chosen. Therefore, occlusion is handled by discarding a subregion of the object and selecting the set of points from the remaining subregion as illustrated in Figure 5.2.

6.4.5 Learning the Forests

Considering the learning dataset $\mathcal{S} = \{(\boldsymbol{\epsilon}_r^v, \boldsymbol{\mu}_r)\}_{r=1}^{n_s \cdot n_r}$, each tree in the forests learns the relation of $\boldsymbol{\epsilon}_r^v$ and a parameter of $\boldsymbol{\mu}_r$. The tree is constructed by continuously splitting $\mathcal{S}$ into two subsets and passing the subsets down to the children. The objective is to split using $\boldsymbol{\epsilon}$ while optimizing a parameter in $\boldsymbol{\mu}$ to make its values more homogeneous.

Without loss of generality, we denote the set of samples that arrive on the node N as $\mathcal{S}_N$. Similar to (4.10), the feature θ_N is an index of the vector $\boldsymbol{\epsilon}_r^v$ such that $\boldsymbol{\epsilon}_r^v[\theta]$ takes the scalar value of the θ-th index in the vector and the threshold κ_N is a scalar value that splits $\mathcal{S}_N$ into $\mathcal{S}_l$ and $\mathcal{S}_r$ that goes to the left and right child, respectively.

In order to find the optimum split, all of the n_h elements of $\boldsymbol{\epsilon}$ and several thresholds that are linearly spaced between the minimum and maximum values of each element across $\mathcal{S}_N$ are tested to split the dataset. The feature and threshold that best splits the set

is measured by the information gain from (4.11). Hence, the optimum pair of (θ_N, κ_N) is the one with the highest information gain. After iteratively splitting $\mathcal{S}_N$ to produce deeper trees, the splitting stops either when the tree reaches its maximum depth, or when $\sigma(\mathcal{S}_N)$ is low, which means that the parameters are homogeneous. Finally, this node is a leaf and stores the mean and standard deviation of the parameter in all $\boldsymbol{\mu}$ from $\mathcal{S}_N$. The same process is applied to different parameters in the $\boldsymbol{\mu}$ as well as to different camera views of the model.

6.4.6 Tracking

At time t, the given input is the current frame $\mathbf{D}_t$, the head pose from the previous frame $\mathbf{T}_{t-1}$ and the learned forest with $6n_v$ trees. During tracking, the forest then predicts the relative transformation $\hat{\mathbf{T}}_t$ between two consecutive frames and updates the head pose from $\mathbf{T}_{t-1}$ to $\mathbf{T}_t$.

Considering the n_v views of the head, a subset of the trees are chosen such that the viewpoint of the learned CAD model shows the highest similarity with the current frame. Using (5.5), $\mathbf{T}_{t-1}$ generates the unit vector $\mathbf{N}_{t-1}$ that points to the camera in the object coordinate system. Then, the relation between the current view of the object from the learned views is measured through the angle between $\mathbf{N}_{t-1}$ and $\mathbf{N}_v$ for all views. Thus, the subset of trees chosen for evaluation is composed of the trees with the camera view that are within the neighborhood of $\mathbf{N}_{t-1}$, where the angle is less than τ_n.

On the v-th view, $\boldsymbol{\epsilon}^v = [\epsilon_h^v(\mathbf{T}_{t-1}; \mathbf{D}_t)]_{h=1}^{n_h}$ is constructed as the input to the trees. The splitting parameters on the branches maneuver the input towards a leaf. Each leaf stores the predicted mean and standard deviation of a parameter. After evaluating the trees from all neighboring views, the final prediction of a parameter is the average of the predicted means with the least standard deviation. As a result, the average parameters are used to assemble the relative transformation $\hat{\mathbf{T}}_t$ and to update the pose with $\mathbf{T}_t = \mathbf{T}_{t-1}\hat{\mathbf{T}}_t$. Lastly, we iteratively refine the predictions for each frame.

6.4.7 Failure Detection

The disadvantage of a temporal tracker is the dependence of the current frame from the previous frame. Thus, when the tracker fails, all the succeeding frames are affected. Contrary to our work, the tracking-by-detection algorithms [22, 39, 108, 113, 122] assume the independence of the frames in the sequence. It follows that, when they fail to estimate the pose in one frame, the pose of the succeeding frames are not affected. Therefore, determining when the tracker fails is a crucial component of our algorithm.

There are several ways to determine if the tracker fails. For rigid objects, we can observe the point-to-point error from (6.2) and take the average of the errors as

$$E(\mathbf{T}_t, \mathbf{D}) = \frac{1}{n_v'} \frac{1}{n_h} \sum_{v=1}^{n_v'} \sum_{h=1}^{n_h} \epsilon_h^v(\mathbf{T}; \mathbf{D}) \tag{6.5}$$

for all the n_h points within the neighborhood of n_v' camera views during tracking. However, the goal of our tracker is different from the rigid objects in [133, 134] because this work is generalized to estimate the pose of different geometric structure from different subjects and it has to be robust against facial deformations. Although (6.5)

can be used to evaluate when the subject has a neutral facial expression, a simple facial deformation such as opening the mouth enforces a failed frame.

Since we have learned the geometric structure from multiple subjects in our forest and the trees in the forest are independent from each other, we assume that, if tracking is successful, the prediction from individual trees are approximately the same; otherwise, the predictions are random. We then measure the confidence of the aggregated prediction by the standard deviation of the best predictions when tracking. The confidence of the prediction is low if the standard deviation of at least one transformation parameter is less thatn τ_c. After having a low confidence for n_f consecutive frames, we conclude that tracker failed. Subsequently, the tracker performs re-initialization using Section 6.3.1.

6.5 Subject-Specific Online Learning

Since the generalized tracker only learns the facial structure of the subjects as shown in Figure 6.2(b), it is limited to track the frontal view of the head . As the subject moves and facial structure becomes less visible such as Figure 6.2(a), which we refer as *extreme poses*, tracking fails due to the lack of structure on the CAD models that are consistent across various subjects and the differences with the real depth images. Therefore, we introduce a subject-specific online learning method into the generalized tracker. This aims to alleviate the limitation of the tracking the facial structures and incorporate specific head structure that is unique to an individual.

The online learning method relies on the results from the tracker. At time t, the depth image $\mathbf{D}_t$ and the resulting pose $\mathbf{T}_t$ are the input to learning. Imposing (5.5) on $\mathbf{T}_t$, $\mathbf{N}_t$ is derived as the unit normal vector for (6.2). To avoid learning the background, the head is segmented by enclosing it into a 3D bounding box and removing all pixels outside this region. Compared to learning the generalized tracker, the n_h points in $\{\mathbf{X}_h^v\}_{h=1}^{n_h}$ are randomly selected within the 3D points on the segmented depth image with the occlusion handling scheme. Using $\hat{\mathbf{T}}_t^{-1}$, the points are then transformed to the model coordinate system. To build the learning dataset, n_r random transformations $\mathbf{T}_r$ are imposed on these points by transforming with $\mathbf{T}_r = \mathbf{T}_t\hat{\mathbf{T}}_r$ to generate the error vector $\boldsymbol{\epsilon}_r = [\epsilon_h(\mathbf{T}_r; \mathbf{D}_s^v)]_{h=1}^{n_h}$ and assemble the learning dataset $\mathcal{S} = \{(\boldsymbol{\epsilon}_r, \boldsymbol{\tau}_r)\}_{r=1}^{n_r}$. Thereafter, learning using $\mathcal{S}$ is carried out in the same way as the model-based in Section 6.4.5.

During tracking, the neighborhood of camera views from learning incorporates both the generalized as well as the subject-specific trees. After comparing the resulting camera location in tracking with the camera locations from all the learned views through the angular distance between them, the trees that are within τ_n are evaluated. The final transformation parameters are the mean of the best predictions from the trees.

6.6 Multi-Camera System

Since we utilize the 3D data from the depth images, we extend our tracking approach in the case of a multi-camera system. We denote $C_1, C_2, \ldots, C_{n_c}$ as the given n_c cameras. Assuming a calibrated system, the rigid transformation $\mathbf{T}_{C_i}$ of any subject or, in general, any object from an arbitraty camera C_i can be propagated to any other j-th camera as

$\mathbf{T}^{C_j} = \mathbf{T}^{\Delta C_j}\mathbf{T}^{C_i}$, where $\mathbf{T}^{\Delta C_j}$ is the transformation between two cameras, C_i and C_j, as a result of the calibration.

The advantage of using our tracker in the multi-camera system is twofold. One is the constant pose update from $t-1$ to t for all cameras. The other is the capacity to use the same forest for all cameras, even if they have distinct intrinsic parameters.

Consider tracking the head pose with C_i alone, the transformation from $t-1$ to t updates the pose as $\mathbf{T}_t^{C_i} = \mathbf{T}_{t-1}^{C_i}\hat{\mathbf{T}}_t$. It follows that the same transformation propagates to the j-th camera as

$$\mathbf{T}_t^{C_j} = \underbrace{\mathbf{T}^{\Delta C_j}\mathbf{T}_{t-1}^{C_i}}_{\mathbf{T}_{t-1}^{C_j}}\hat{\mathbf{T}}_t = \mathbf{T}_{t-1}^{C_j}\hat{\mathbf{T}}_t \tag{6.6}$$

where $\mathbf{T}_{t-1}^{C_j}$ is the given pose for C_j at $t-1$. Therefore, $\hat{\mathbf{T}}_t$ is constant for all cameras. To generalize, in the same way as $\mathbf{T}_t$ from C_i can be used for any C_j, we can predict the parameters of the pose update $\mathbf{T}_t$ from a subset of cameras $\mathcal{C}$. This becomes necessary, for instance, when the head pose from the previous frame is out of the camera's field of view.

Independently for each camera, the tracker evaluates the neighborhood of trees within τ_n as well as the corresponding error vector for a tree. The pair $(\theta,\ \kappa)$ at the nodes of the tree guides the error vector to the mean and standard deviation of a parameter at the leaf.

Now, since the pose update is constant for all cameras, we can accumulate all the predictions from all cameras in $\mathcal{C}$ and compute the final prediction as the average of the predicted means with the least amount of standard deviation. In this way, heads that are occluded on a camera are ranked lower than a camera with a higher structural similarity as the learned head model. As a result, in addition to handling poses that are out of the camera's field of view, we can also handle full occlusions on a subset of cameras in $\mathcal{C}$ as long as there is at least one camera where the head is not occluded.

Furthermore, since the error function in (6.2) is carried out in 3D space, all the computations depend on the intrinsic parameters of a given camera through the back-projection function $\mathcal{D}(\cdot)$. This implies that the error function and, in effect, the forest can adapt to different types of cameras and are not restricted to the learned camera model. The value of this idea is to generalize the tracker not only to different subjects but also to different camera models. It also have a relatively small amount of learning dataset because it only needs one set of camera parameters for the entire dataset, instead of adding rendered images from different camera models. As a consequence, learning is faster. Moreover, based on Section 6.5, we are able to learn online from one camera while being adaptive in evaluating the same forest for all cameras in $\mathcal{C}$, regardless of whether they have the same intrinsic parameters or not. In practice, this allows us to use cameras with different technological modalities (*e.g.* time of flight and structured light) such that one does not interfere with the depth sensor of the other in constructing the depth image.

6.7 Experimental Results

This section highlights the evaluation of different tracking strategies in terms of robustness and efficiency. Based on the previous sections, we propose three possible tracking strategies:

(1) *Generalized*: the generalized model-based tracking;

(2) *Subject-Specific*: the subject-specific model-based tracking; and,

(3) *Combined*: the combined generalized tracker with the subject-specific online learning.

Although the first two methods learn offline from 3D CAD models, the difference is that the generalized tracker learns from multiple CAD models of various subjects while the subject-specific tracker learns from a specialized CAD model of an individual. The subject-specific then follows the same procedure as Chapter 5.

Note that the evaluation of the subject-specific model-based learning is restricted to datasets such as [39] that have the head model of a specific subject beforehand for offline learning. The third method starts the generalized tracker to track the head while learning online with to integrate the individual's subject-specific structures.

6.7.1 Robustness

We evaluate the robustness of the trackers on the BiWi Kinect Head Pose Database [39] and the ETH Face Pose Range Image Dataset [22]. These datasets are arguably the most relevant public datasets with ground truth for head pose estimation from 3D data. As shown in Figures 6.3 and 6.5, the images in both datasets include also the subject while turning his/her head.

Throughout the paper, the 3D CAD models for the offline learning are taken from the BiWi database [39]. The geodesic grid is constructed with 642 vertices, where 332 of them are used after filtering through the azimuth and polar angles in the spherical coordinate system. Moreover, the head models are rendered with the camera intrinsic parameters of Primesense PSDK 5.0 Device which has a focal length of 570.3 and the principal point at (320, 240). In each camera view, we use $n_r = 2500$ random transformation for each subject and $n_h = 20$ points on the model to learn one tree per parameter with a maximum depth of 20 and maximum standard deviation of 0.1. The same learning parameters are used for the online learning method. Moreover, we use the OpenCV face detector [20] on the RGB images to initialize the tracker.

BiWi Kinect Head Pose Database

The BiWi Kinect Head Pose Database [39] consists of 24 sequences from 20 different people with a total of approximately 15K images, where all images are labeled with the ground truth head pose. For each sequence, one of the 20 subjects stands approximately 1 meter away from the camera and close to the principal point of the image. Each sequence also includes the 3D CAD model of the subject's head. Some frames of a sequence are shown in Figure 6.3.

The evaluation of this dataset follows the same experimental framework as [39], where we learn using 18 subjects and evaluate on the remaining two. In effect, the

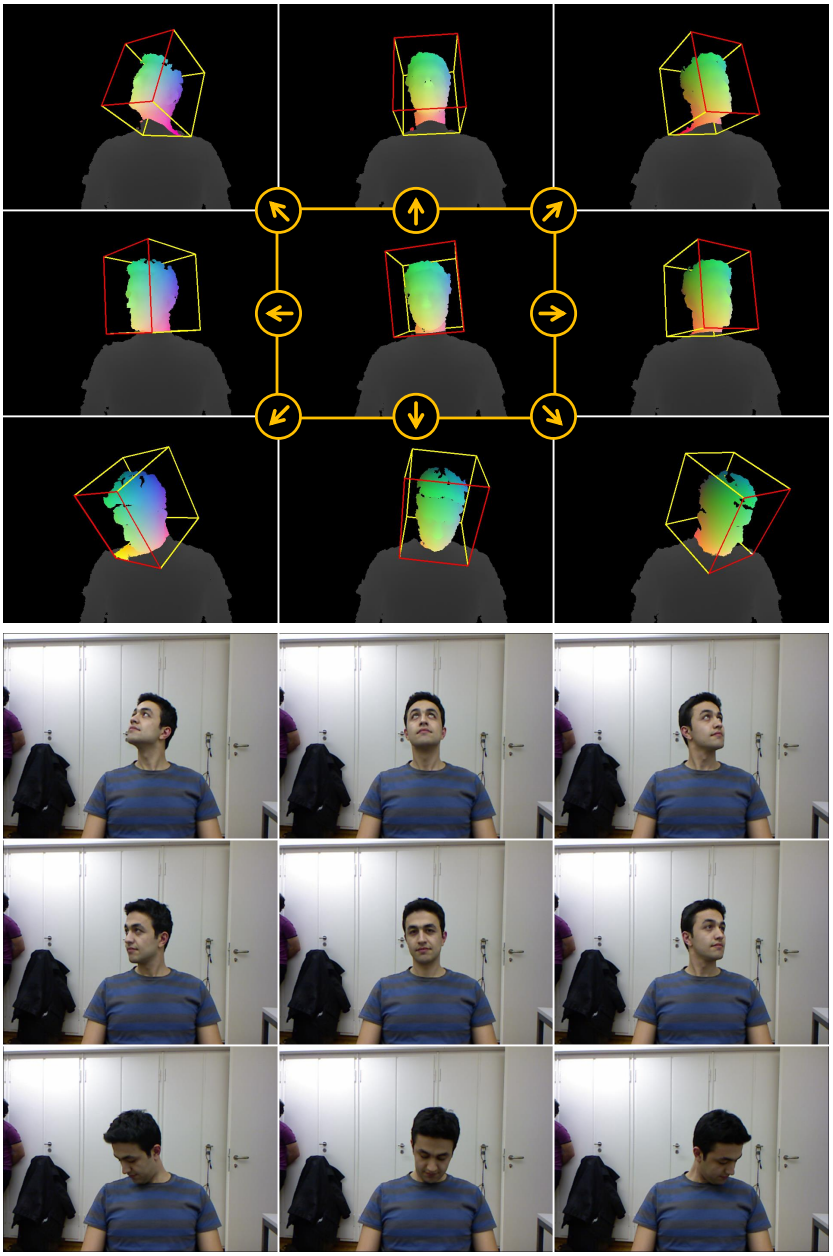


Figure 6.3: Some RGB-D frames of a sequence in the BiWi Kinect Head Pose Database [39]. Note that the depth images show the 3D bounding box from our head pose estimation results.

learning dataset of the generalized tracker renders depth images through the given 3D model of the head from the 18 subjects in the dataset, whereas all competing approaches use real images in the database. For the real images, the sequences are captured by Microsoft Kinect with a focal length of 575.8 and principal point at (320,240).

The objective of the evaluation is to estimate the head pose while the subject rotates his head within $\pm 75^\circ$ yaw, $\pm 60^\circ$ pitch and $\pm 50^\circ$ roll angles. We then measure the error of the estimated pose from the ground truth by computing the difference of the translation in the x-, y- and z-axis, as well as the rotation in the yaw, pitch and roll angles for each frame. To aggregate the errors, the mean and standard deviation are calculated across all frames in Table 6.1. Note that it excludes the frames that failed to estimate the pose correctly. These are the poses with a translation error above 20 mm [108] or 50 mm [39, 113, 122].

Conversely to the failure case, we also measure the success rate of the tracker as the percentage of frames where the error in rotation is less than 20° and the error in translation is less than 20 mm in Table 6.2. With our results, we also plot the success rates with varying thresholds for each transformation parameter in Figure 6.4. Finally, we compare our results based on the three methods with those reported in the papers of the four state of the art methods, namely Hough Forest (HF) [39], Hough Forest with Probabilistic Locally Enhanced Voting (PLEV) [108], Alternating Regression Forests (ARF*) [122] and Hough Networks (HN) [113]. For this evaluation, we utilize all of the three proposed tracking strategies since the individual models for each subject is given.

Generalized. When evaluating our generalized tracker on the BiWi dataset [39], Table 6.1 illustrates that our tracker outperforms the best result [108] from other methods by 4.5 mm in translation and 4.9° in rotation. Hence, our tracker decreases the errors in both rotation and translation by half. In addition, we achieve better results across all parameters without any failure case. These results entail a 100% success rate with a threshold of 20 mm for translation and 20° for rotation in Table 6.2.

Our method improves the best results shown in [113] by a 5% increase in translation and 11% increase in rotation. Based on Figure 6.4, we achieved 100% success rate even with a threshold of 10 mm and 10°.

Subject-specific. Interestingly, this database [39] is a special case because they computed the ground truth pose as by-product of creating an accurate 3D CAD model for each subject. As a result, we can learn a subject-specific model-based tracker using the individual's CAD model and numerically measure the loss in generalizing the tracker.

In Table 6.1, the subject-specific tracker improves the state-of-the-art by 5.5 mm in translation and 5.9° in rotation while maintaining a 100% success rate in Table 6.2 and Figure 6.4. This tracker achieves the best overall performance across all methods. It is not surprising that the subject-specific model-based tracking produces less error than the generalized tracker. On average, the subject-specific tracker is better than the generalized tracker by 0.9 mm for translation and 0.9° for rotation.

Therefore, this evaluation emphasizes that the cost of generalizing the tracker is less than 1 mm and 1° because the subject-specific tracker can only be used in special cases where an accurate 3D CAD model of a specific individual is accessible. The reconstruction of the CAD models for individual subjects is done beforehand but requiring an a priori knowledge of the subject hinders its application to directly track the

	Translation (*mm*)	t_x (*mm*)	t_y (*mm*)	t_z (*mm*)	**Rotation** (*degrees*)	Yaw (*degrees*)	Pitch (*degrees*)	Roll (*degrees*)	Fail (%)
HF [39]	12.8 ± 6.8	6.9 ± 6.7	7.4 ± 5.6	4.7 ± 3.4	14.3 ± 10.0	5.7 ± 6.1	9.7 ± 8.9	5.9 ± 5.2	5.0
PLEV [108]	7.2 ± 12.1	–	–	–	7.3 ± 5.9	4.1 ± 6.9	3.9 ± 4.0	3.2 ± 3.0	5.0
HN [113]	8.1 ± 5.3	3.8 ± 4.5	4.6 ± 4.0	3.7 ± 3.0	9.8 ± 8.0	3.8 ± 3.7	6.7 ± 6.6	4.3 ± 4.9	1.0
ARF* [122]	10.8 ± 6.9	5.5 ± 5.6	6.2 ± 6.1	4.1 ± 3.1	12.2 ± 9.0	5.5 ± 5.5	7.8 ± 7.9	5.0 ± 4.4	3.0
Generalized	2.7 ± 0.9	0.9 ± 0.6	1.3 ± 1.0	**1.8 ± 0.9**	2.4 ± 1.5	**1.7 ± 1.7**	**1.5 ± 1.5**	**1.4 ± 1.5**	**0.0**
Combined	**2.6 ± 0.8**	**0.8 ± 0.6**	**1.1 ± 0.9**	1.9 ± 0.8	**2.3 ± 1.4**	**1.7 ± 1.7**	**1.5 ± 1.4**	1.5 ± 1.7	**0.0**
Subj.-Specific	1.7 ± 0.7	0.6 ± 0.5	1.0 ± 0.8	0.8 ± 0.5	1.4 ± 0.8	0.9 ± 1.1	0.8 ± 0.7	0.9 ± 1.0	0.0

Table 6.1: Based on the evaluation on the BiWi Kinect Head Pose Database [39], the error values compares the accuracy of different head pose estimation algorithms [39, 108, 113, 122] against our trackers. For the other methods, the failure case are the percentage of frames where the error in the translation is above 20 mm for [108] or 50 mm for [39, 113, 122].

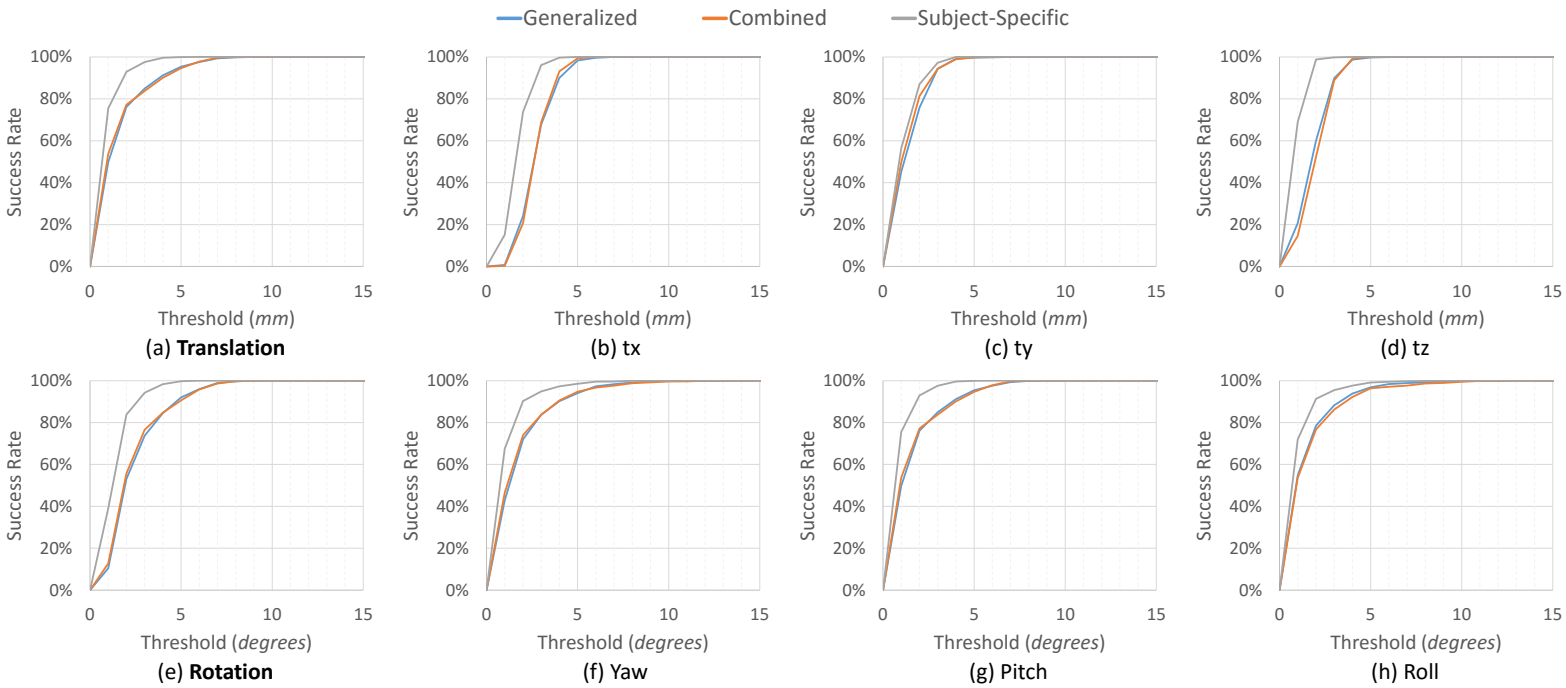


Figure 6.4: Using the evaluation on the BiWi Kinect Head Pose Database [39], these show the success rates with varying thresholds for the error in translation (in mm).

	Translation ($\leq$ 20 mm)	**Rotation** ($\leq 20^\circ$)
HF [39]	82.99%	73.29%
PLEV [108]	95.00%	–
HN [113]	95.11%	88.86%
ARF* [122]	88.30%	80.37%
Generalized	**100.00%**	**100.00%**
Combined	**100.00%**	**100.00%**
Subj.-Specific	100.00%	100.00%

Table 6.2: Success rate of different methods where the error in translation is less than 20 mm or the rotation angle is less than 20°. The list of competing methods are the same as Table 6.1.

head of any user in the camera's field of view. Hence, the subject-specific model-based learning method is time-consuming and limited in real applications.

Combined. One of the main novelties of our work is to incorporate subject-specific structures through online learning by incrementally accumulating trees from different viewpoints of the subject's head while tracking with the generalized method. When combining the generalized model with the subject-specific online learning, it outperforms the other methods in Table 6.1. Compared to the generalized tracker, the combined method produces slightly better results with a 0.1 mm decrease in translation error and a 0.1° decrease in rotation error. Similarly, the success rates is also 100% with approximately the same curves as the generalized method in Figure 6.4 for all transformation parameters.

It is a noteworthy observation that the online learning is initially guided by the generalized tracker. This implies that the combined method cannot have a significantly better accuracy than the generalized tracker, such as the subject-specific model-based tracker.

ETH Face Pose Range Image Dataset

We also evaluate on the ETH Face Pose Range Image Dataset [22]. The dataset consist of over 10K depth images from 20 subjects where they rotate their heads between $\pm 90^\circ$ yaw and $\pm 45^\circ$ pitch while being captured by a stereo enhanced structured light sensor [144]. Figure 6.5 shows some frames of a sequence in the dataset. Unlike the BiWi database [39], the ETH dataset does not include any 3D CAD model, which implies that we do not perform any subject-specific model-based tracking on this dataset. Since it does not have any 3D CAD model for learning, the same forest as the evaluation of [39] is used in this section.

Another problem of evaluating this dataset is the lack of RGB image for initialization. However, since only the head is visible on all images as shown in Figure 6.5, we replace $\mathbf{x}_c$ in (6.3) with the centroid of all pixels on the head. Furthermore, the depth images in the dataset is given as 640×480 image with each pixel represented as a 3D point. Since our formulation of the error function in (6.2) considers the projective geometry

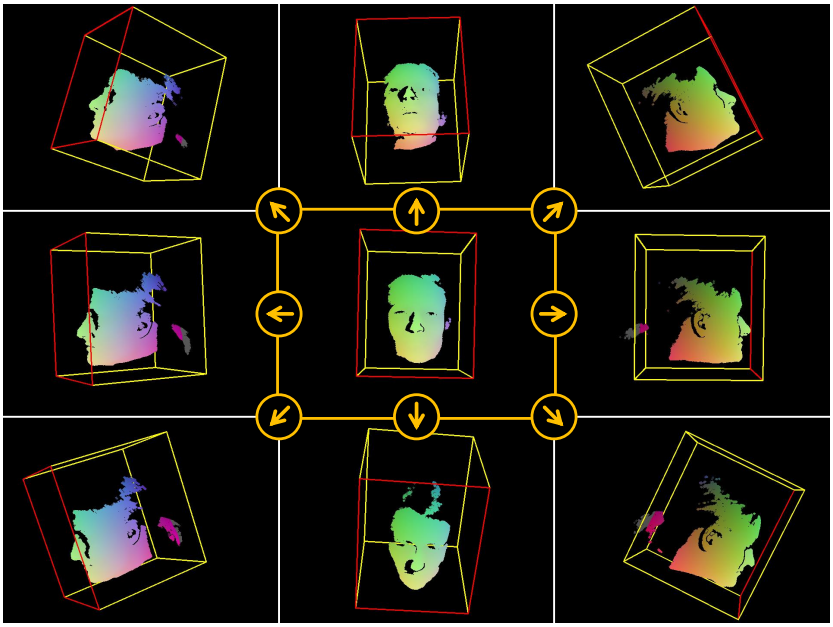


Figure 6.5: Some depth frames of a sequence in the ETH Face Pose Range Image Dataset [22]. Note that the depth images show the 3D bounding box from our head pose estimation results.

with the camera intrinsic parameters in order to project and back-project the points, we algebraically compute the intrinsic parameters by comparing the 3D points with its location on the image. Then, the entire dataset uses the intrinsic parameters of 1180.6 and 1183.0 as the focal lengths in the x and y directions and (346.2, 282.3) as the location of the principal point.

To evaluate the dataset, the ground truth head pose for each frame is represented by the location of the nose tip and a vector for the head's facing direction. However, considering that our approach does not compute for the location of the nose tip, we only compare the direction of the vector given by the dataset as θ and ϕ. Here, the success rate is then the percentage of frames where the angular error is below 10°.

Table 6.3 summarizes the evaluation on the ETH dataset. When tracking with the generalized method, our work is worse in ϕ with a difference of 0.2° against Meyer *et al.* [93] but with a better performance in θ with an improvement of 0.1°. Overall, we have a 0.3% higher success rate. However, when using the combined method, we outperform the state-of-the-art results by 0.3° in ϕ and 0.4% in success rate.

This dataset is essential because it highlights the failure cases of our tracker as illustrated Figure 6.6. Although the tracker can handle occlusions, it still needs a region of the head to learn online and to track. When comparing Figure 6.6 to the similar poses in Figure 6.5, it is evident that a large portion of the head is no longer visible on the images of the failure cases. In addition, we also highlight that, even with the large difference in focal length between the learned model which is 570.3 and the focal length of the dataset which is approximately 1180.6, we still produce better results. A more elaborate evaluation on the sensitivity to the changes in focal length is in Section 6.7.1.

	Direction θ (*degrees*)	Direction ϕ (*degrees*)	Success (%)
Breitenstein [22]	6.1	4.2	80.8
HF [39]	5.7	5.1	90.4
Meyer [93]	2.9	**2.3**	98.9
Generalized	2.8	2.5	99.2
Combined	**2.6**	**2.3**	**99.3**

Table 6.3: Based on the evaluation of the ETH Face Pose Range Image Dataset [22], the error values compare the accuracy of different head pose estimation algorthms [22, 39, 93] against our tracker. The head pose from [22] is represented through the face direction given as (θ, ϕ).

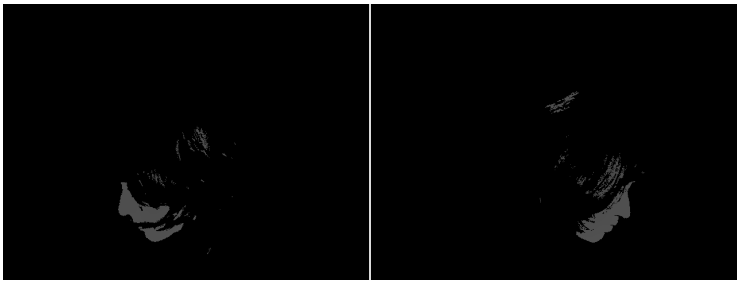


Figure 6.6: Failure case on the ETH Face Pose Range Image Dataset [22].

Sensitivity Analysis

To avoid overfitting the parameters to the subject's motion from a specific dataset, we implement a synthetic evaluation in order to assess the sensitivity of our robustness to different parameters. Using the frames from the test sequences of the BiWi dataset [39], we multiply the ground truth pose at each frame with a random rigid transformation, so to mimic the pose from the previous frame. We then conclude that tracking is successful if the rotation angle between the estimated pose and the ground truth is less than 20° and the translation displacement is less than 20 mm. These thresholds are similar to the ones used in [39].

Tree depth. In learning, the depth of the trees are essential in identifying whether the forest over-splits or under-splits the learning dataset. In Figure 6.7, we demonstrate that there are still small improvements between the tree depth of 16 to 20 but no improvement between 18 and 20. Since the success rate did not deteriorate at 20, we chose 20 as the maximum depth of the trees in the forest.

Tracking parameters. When tracking, there are two essential parameters that affects the performance of the tracker. First, the neighborhood of views (τ_n) is essential in order to compare the similarity of the learned view and the current frame. We show in Figure 6.8 that the success rate starts to converge with a threshold of 35° for both translation and rotation. The other essential parameter in tracking is the percentage of

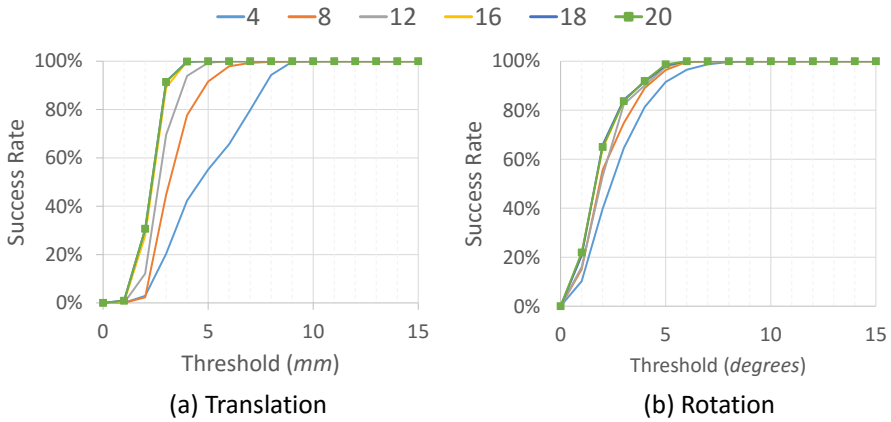


Figure 6.7: Success rate with varying maximum depth in learning the trees.

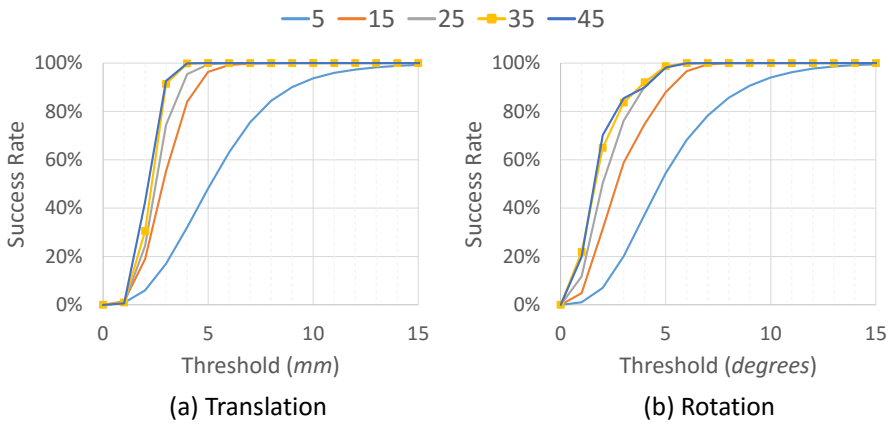


Figure 6.8: Success rate with varying angular threshold (τ_n) in tracking.

the predictions from different trees to aggregate in getting the final pose. Similarly, the success rate converges at when taking 50% of the predictions in Figure 6.9.

Iterations. As an iterative method, we also look into the number of iterations required to converge. Figure 7.4(a-b) illustrates that we reached convergence after six iterations. Nevertheless, we are using 10 iterations in our tracker to ensure convergence. The figure also demonstrates that the generalize and combined methods have similar convergence rate while the subject-specific method converges to a lower error value, which validates our results in Table 6.1.

Then, we look into the frame-to-frame evaluation of the BiWi dataset [39] to understand the effects of the number of iterations in tracking based on the real frame-to-frame movements of a subject. Here, we set the total number of iterations for the entire evaluation instead of looking at the i-th iteration. In Figure 7.4(c-d), four iterations is satisfactory to achieve the converged error.

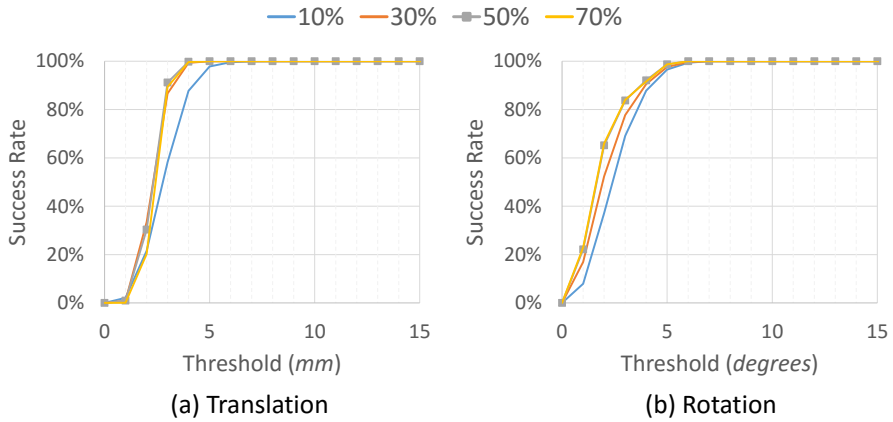


Figure 6.9: Success rate with varying percentage of predictions to aggregate in tracking.

Learned Focal Length	**Translation** (*mm*)	**Rotation** (*degrees*)
300.0	1.6 ± 0.6	1.0 ± 0.5
400.0	1.6 ± 0.7	1.1 ± 0.6
500.0	1.6 ± 0.7	1.1 ± 0.6
600.0	1.6 ± 0.6	1.1 ± 0.6
700.0	1.6 ± 0.8	1.3 ± 0.9
800.0	1.6 ± 0.6	1.1 ± 0.6
900.0	1.5 ± 0.6	1.2 ± 0.6
1000.0	1.4 ± 0.6	1.1 ± 0.5
1100.0	1.5 ± 0.6	1.1 ± 0.5

Table 6.4: When evaluating one sequence of the BiWi Kinect Head Pose Database [39], this is the comparison of the error in translation and rotation with different focal lengths when rendering images for the learning dataset. Note that the focal length of the sequence is 575.8.

Focal length. We evaluate on the effects of changing the focal length when rendering the synthetic depth images for the learning dataset. In this case, we are using the subject-specific model-based tracker and investigate using a sequence of the database [39]. Table 6.4 demonstrates that, as the focal length increases, there is no significant change on the errors in both translation and rotation. It shows the stability of the tracker to be used for different camera models, especially when the camera model used for the learning dataset and the camera model used for capturing the test sequence are distinct. This aspect is important especially since we are using a single tracker for all the cameras in the multi-camera system. We are also using the trees learned online from one camera of the multi-camera system to the rest of the system.

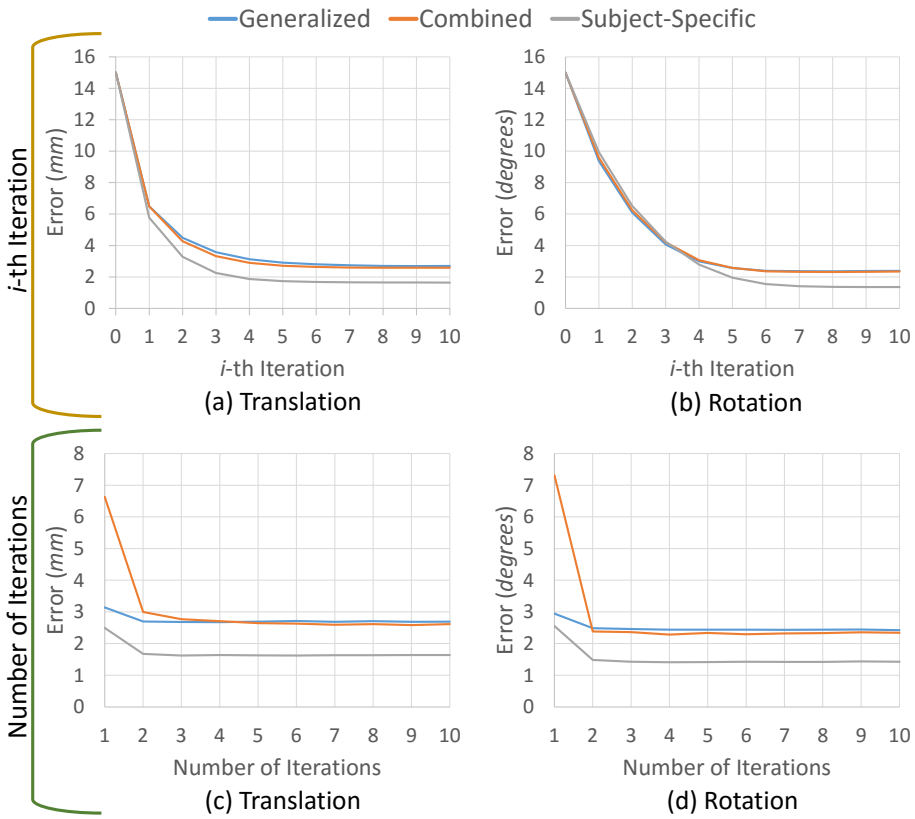


Figure 6.10: (a-b) Convergence rate of the tracker using the synthetic evaluation such that it plots the error for translation and rotation at the i-th iteration. (c-d) Given the number of iterations, the plots show the average error on the evaluation of the BiWi Kinect Head Pose Database [39].

6.7.2 Efficiency

In order to find the runtime and the corresponding computational cost, all evaluations in this section are conducted using Intel(R) Core(TM) i7-3820QM CPU with 16 GB RAM.

For initialization, the face detection algorithm of [20] runs at approximately 9.1 ms per frame. Thereafter, our tracker takes over to find the head pose estimate. Table 6.5 summarizes the tracking performance of the three tracking strategies. Among them, the most efficient is the subject-specific model-based tracker. It requires 18.8 seconds to learn the forest offline with 8 CPU cores and 1.1 ms to track the head with a single CPU core.

Since the generalized tracker learns offline from multiple 3D CAD models of different subjects, it requires to learn 9.0 minutes and tracks at approximately 1.5 ms with a single core. Furthermore, the combined method needs multiple cores to have sufficient time to learn online. As a consequence, with 8 CPU cores, additional trees in the generalized tracker for the subject-specific online learning takes 28.1 ms per frame while tracking takes around 1.4 ms. Note that not all tracked frames are used for

		Time	Architecture
Other Methods	Breitenstein [22]	23.5 ms	GPU
	HF [39]	17.8 ms	CPU (1 core)
	Meyer [93]	160.0 ms	GPU
	PLEV [108]	–	–
	HN [113]	66.7-100.0 ms	GPU
	ARF* [122]	–	–
Our Work	*Generalized*	**1.5 ms**	CPU (1 core)
	Offline Learning	9.0 minutes	CPU (8 cores)
	Subj.-Specific	**1.1 ms**	CPU (1 core)
	Offline Learning	18.8 seconds	CPU (8 cores)
	Combined	**1.4 ms**	CPU (8 cores)
	Online Learning	28.1 ms/frame	CPU (8 cores)
	Multi-Camera	**1.8 ms**	CPU (8 cores)

Table 6.5: Timings and the corresponding architecture for different head pose estimation methods [22, 39, 93, 108, 113, 122]. It also includes the timings for our three proposed tracking strategies with their learning time, and for tracking in the multi-camera system.

learning new trees, and the geodesic grid limits the number of trees for the viewpoint that are close to each vertex.

In addition, Table 6.5 also compares the efficiency of different head pose estimation methods [22, 39, 93, 113] that evaluated the BiWi database and ETH dataset in Section 6.7.1. Compared to their work, our trackers are approximately one to two orders of magnitude faster while using only the CPU. It is noteworthy to mention that most competing methods run in real-time because the scene in the dataset is almost empty: each frame includes one person, visualized from head to torso in the BiWi database [39] while visualized only the head in ETH dataset [22]. Since they employ a tracking-by-detection framework that jointly estimates the pose and segments the head in every frame independently, their runtime increases when the scene is not controlled and the detector is required to distinguish the face from the surrounding objects, such as a scene where a person is sitting on a couch or holding tools. Nonetheless, since we are utilizing a temporal tracker, our runtime is constant and is independent of the surrounding objects in the scene.

6.8 Qualitative Results

To demonstrate the robustness of our algorithm on lax users and its ease of use, we show several qualitative results in Figures 6.11 and 6.12. It includes users jumping and dancing as well as candid shots like observing two users in a Skype conversation that accentuates real-world applications. All qualitative results exhibit the capacity of the subjects to roam around the camera's field of view. This is in contrast to the methods of [22, 39, 93, 108, 113, 122] that relies on constraining the subject to a specified location within the camera's field of view.

Notably, Figure 6.11(a) illustrates the initialization and the failure detections. One of the interesting attribute of our work is the capacity to handle close-range occlusions from external objects in Figure 6.11(b) or from hand gestures in Figure 6.11(c). Due to the subject-specific online learning, our work can also handle extreme poses in Figure 6.11(d) which are not included in learning the generalized tracker. In addition, the tracker is stable to track the subject is moving fast in Figure 6.11(e).

We have implemented the tracker in a multi-camera system with the Primesense PSDK 5.0 Device and the Microsoft Kinect 2.0 in Figure 6.14-6.15. The advantage of using two distinct depth sensors is the avoidance of having one interrupt the other's data acquisition. In this framework, the tracker combines the prediction of the transformation parameters from both cameras. As a consequence, we are capable of tracking the head pose when the subject is not visible from one camera as demonstrated in Figure 6.14. Furthermore, the most significant advantage of using our tracker in this system is its capacity to transfer the learned data from one camera model to the other. Thus, after learning online from the depth images acquired using the Primesense PSDK 5.0 Device, the tracker can estimate the head from both cameras while the subject undergoes a fast motion such as a subject jumping on a mattress in Figure 6.15.

6.9 Conclusion

We applied our tracker on a head pose estimation algorithm from a given video sequence of RGB-D images. Here, the RGB image is used in the first frame to detect the face [143] and initialize the tracker, while the depth image is used to temporally track the head pose throughout the sequence using a random forest algorithm.

This paper highlights three types of tracking – the generalized model-based tracking, the subject-specific model-based tracking as well as the combined generalized tracker with the subject-specific online learning. The choice of using one of the trackers depends on the application at hand. Notably, the difference between the model-based learning and online learning is that: (1) the former tracks the common facial structure while the latter tracks the head; and, (2) the former has a unified reference coordinate system while the latter has local coordinate system that varies from one tracker to the rest.

Therefore, for a controlled environment where the face is visible such as the dataset in [22, 39], the generalized model-based tracker is sufficient; and, if the subject's head model is given such as [39], the subject-specific model-based tracker is the best option. However, for a wider range of applications where the subjects occlude their faces with hand gestures or move freely around the camera's field of view as shown in Figures 6.11 and 6.12, the choice of using the combined method becomes necessary due to its robustness to handle large occlusions and extreme poses.

Attaining similar characteristics as Chapters 4 and 5, the tracking time is less than 2 ms per frame with only one CPU core, contrary to some methods that use GPU. Overall, when comparing to other head pose estimation algorithms [22, 39, 93, 108, 113, 122], this is the fastest and the most computationally efficient, while maintaining a low tracking error.

Due to its efficiency, we also highlight the capacity of the framework to perform head pose estimation in a multi-camera system. Other than the remarkable efficiency achieved, the significance of this work is the capacity of the tracker to estimate the pose even when the head is not visible from some cameras.

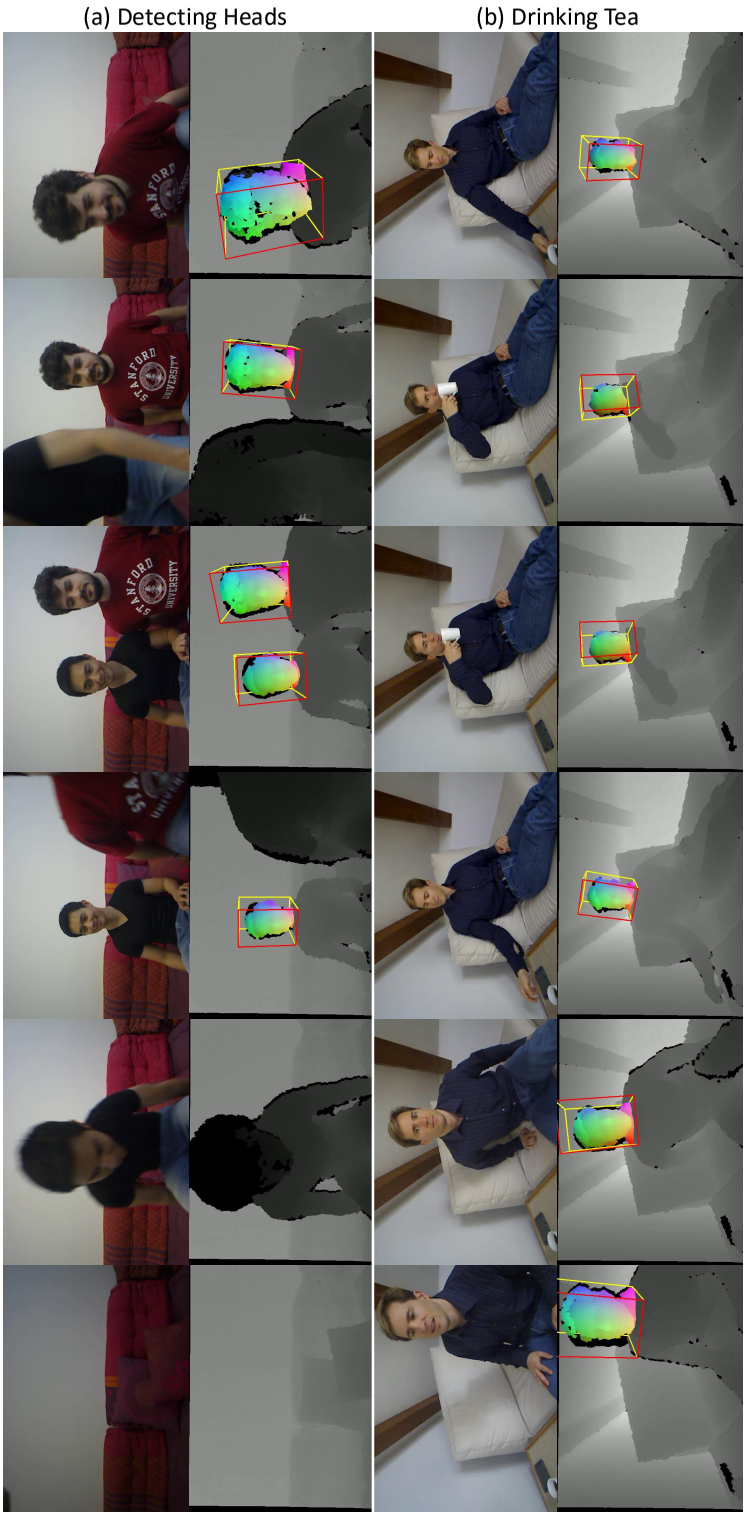


Figure 6.11: These video sequences are taken while (a) detecting the head from multiple users and (b) observing a subject drinking tea.

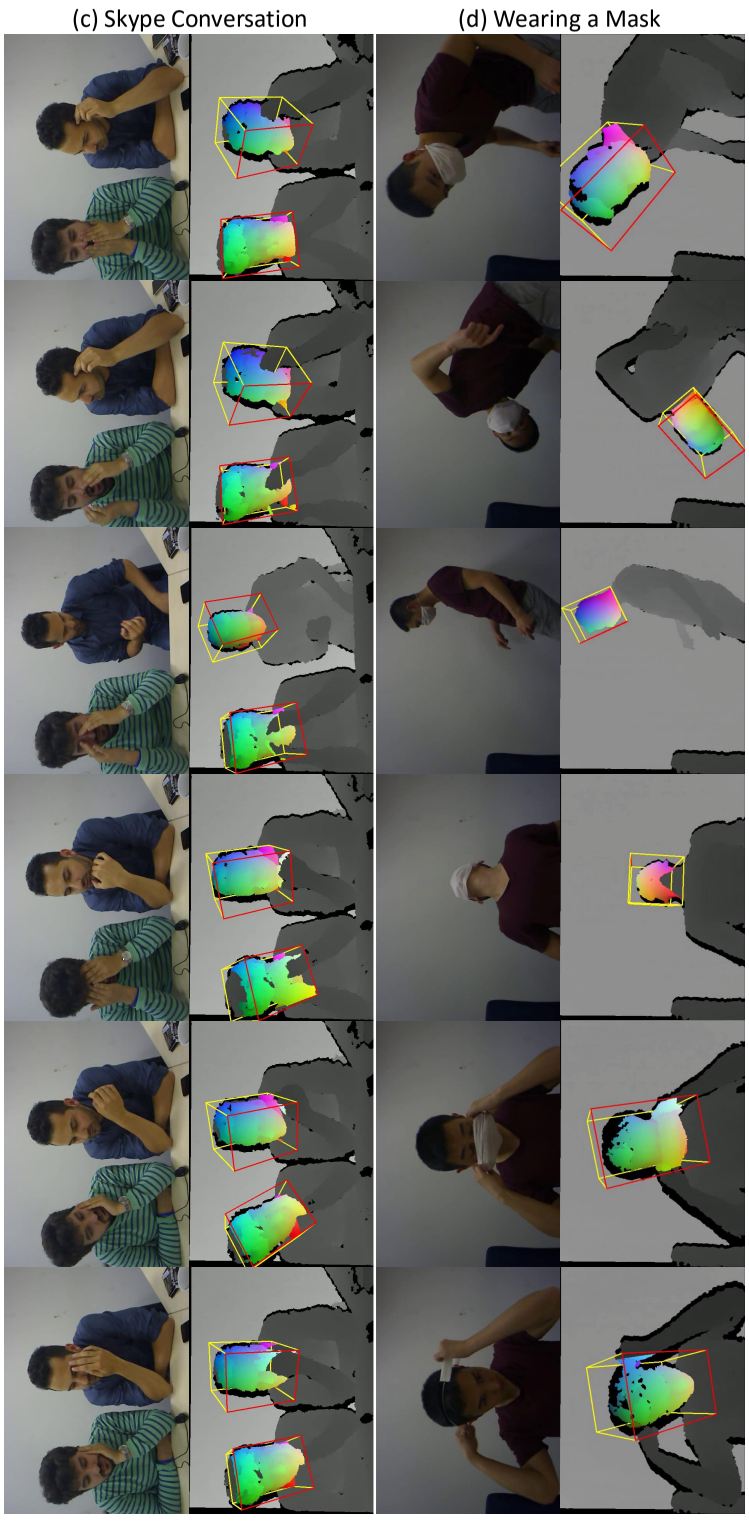


Figure 6.12: These video sequences are taken while observing (c) having a Skype conversation with multiple subjects and (d) a subject wearing a mask and moves around.

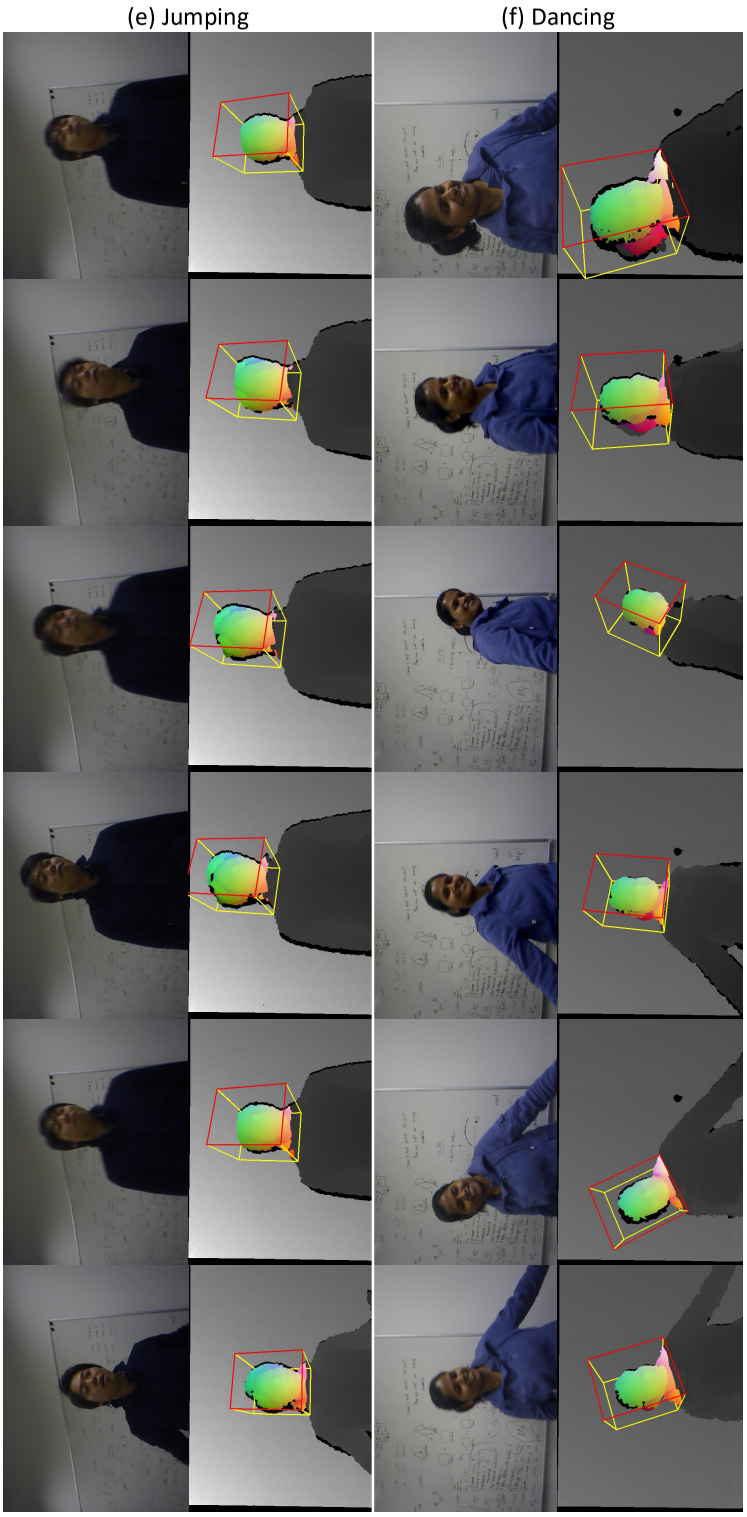


Figure 6.13: These video sequences are taken while observing (e) a subject jumping and (f) a subject dancing.

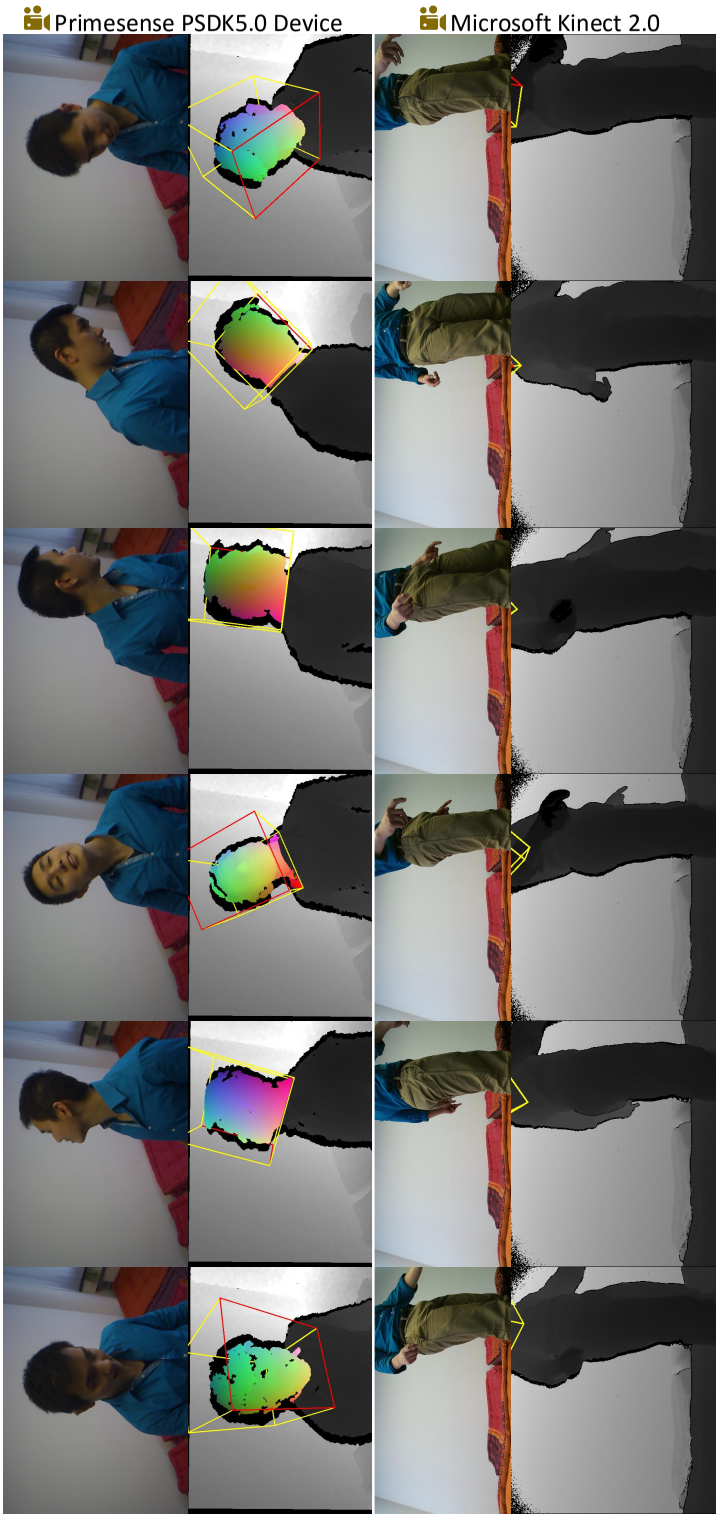


Figure 6.14: Multi-camera system with the Primesense PSDK 5.0 Device (top) and the Microsoft Kinect 2.0 (bottom). These video sequences are taken while (a) the subject is not visible in one of the cameras.

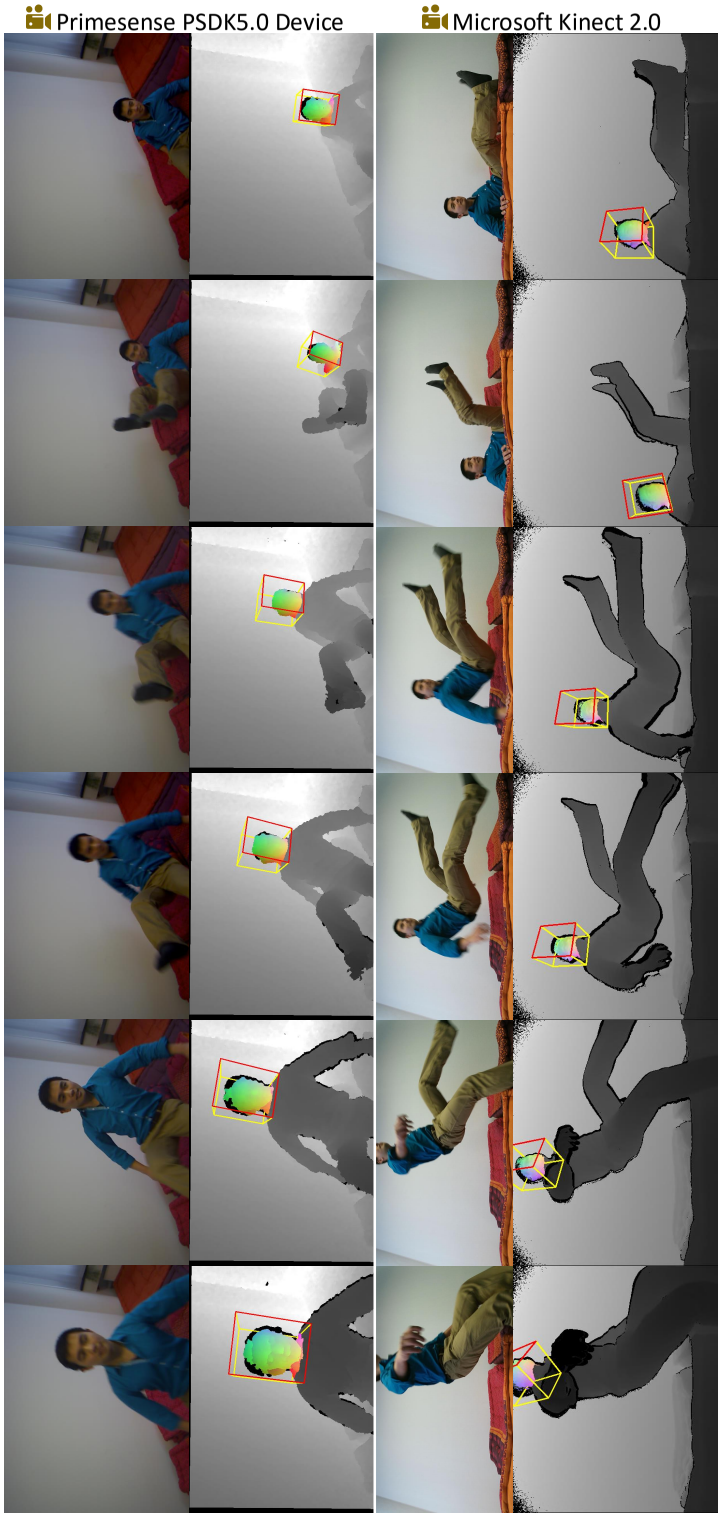


Figure 6.15: Multi-camera system with the Primesense PSDK 5.0 Device (top) and the Microsoft Kinect 2.0 (bottom). These video sequences are taken while (b) the subject jumps as seen from two cameras.

7

Hand Shape Personalization as Prior to Hand Tracking

7.1 Motivation

From Chapter 6, we compared three tracking techniques to determine the pose of the a user's head: (1) *generalized tracker* that fits a mean model to different users; (2) *subject-specific tracker* that fits the user's personalized model to the specified user; and (3) *combined tracker* that uses the generalized tracker to track an arbitrary user and an online learning approach to learn poses that are not learned. The first two methods are purely based on 3D CAD models and the last incorportates an online learning approach to learn ther user's unique structure while tracking. Comparing the second and third techniques that learn subject-specific structures, the second learns from a given reconstructed 3D CAD model during the offline learning while the third learns the head structure during tracking. Among them, that chapter demonstrates that the best tracking results is when we utilize a personalized model instead of a generalized model. Evidently, the cost of generalization from the first and third techniques increases their pose estimation error.

The motivation of the chapter is to determine a method that calibrates a personalized model in a fast and reliable manner so that we can attain the best performance in tracking. Moving from head pose estimation to the deformable hand tracking, we proposed a method that personalizes the hand shape for a specific user as a calibration procedure [131]. It begins by tracking the hand through a mean model and capture a set of keyframes. The hand model is then deformed to optimize the structure of the user's hand based on the given keyframes. After a fast and reliable calibration, the resulting personalized hand shape model fits the user's hand like a glove.

The ability to accurately and efficiently reconstruct the motion of the human hand from images promises exciting new applications in immersive virtual and augmented realities, robotic control, and sign language recognition. There has been great progress in recent years, especially with the arrival of consumer depth cameras [71, 98, 99, 124, 126, 127, 129, 137, 140]. However, it remains a challenging task [128] due to unconstrained global and local pose variations, frequent occlusion, local self-similarity,

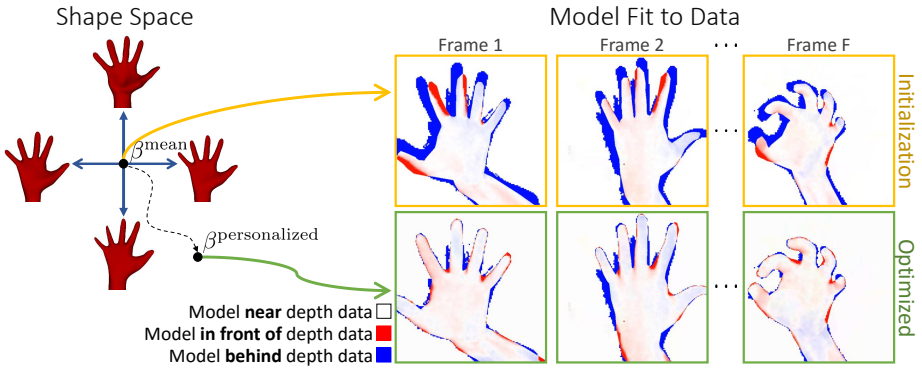


Figure 7.1: We show how to fit a deformable hand shape basis model [72] to a small set of depth images. Our method jointly optimizes over the shape $\beta \in \mathbb{R}^K$ and F poses θ_f to maximize the model's alignment to the data in F depth images. The initial hand poses are automatically determined by a hand tracker that uses the mean shape β^{mean}, but there is clearly poor alignment between model and data. After our optimization to obtain personalized shape $\beta^{\text{personalized}}$, the alignment is much better, with remaining errors largely due to sensor noise.

and a high degree of articulation.

Most recent approaches combine the best of discriminative and generative approaches: the 'bottom-up' discriminative component attempts to make a prediction about the state of the hand directly from the image data, which then guides a 'top-down' generative component by deforming the parameters of a model to try to explain the data. Discriminative methods can be faster and typically require no temporal history. In contrast, a good generative model can use its explanatory power and priors to produce what is usually a more accurate result, even in the presence of occlusion.

Generative models of hands are limited by their capacity to accurately explain the image observations. High-quality, though expensive and off-line, models have been shown to reliably fit both the pose and shape of complex sequences [9]. However, most interactive (real-time) hand tracking systems (*e.g.* [98, 127]) approximate the hand surface using primitives such as spheres or cylinders, parameterized to articulate the surface geometry. Others [124] use a detailed hand mesh model, though only attempt to fit the hand poses using a fixed template shape. To improve the model's capacity, some approaches [86, 127] allow shape deformations of primitive spheres and cylinders, but these models can only compensate for gross model-data mismatches.

Recent work [139] has investigated an off-line process for 'personalizing' a detailed 3D mesh model to an individual's hand shape using a set of depth images of the hand in varied poses, each paired with a manually-annotated initialization pose. The mesh shape is optimized to jointly explain the depth data from each frame, yielding the user's personalized model. Unfortunately, this system is likely to be too brittle and slow for an online setting, as the parameterization of each mesh vertex yields a very high-dimensional optimization problem,

A promising alternative is to create a much lower-dimensional model that parameterizes the hand shape of an entire population of individuals. Khamis *et al.* [72] take a cue from the human body shape modeling literature [5, 56] and build a detailed 3D

shape basis for human hands by parameterizing a mesh model using a small set of 'shape coefficients'. Each setting of these coefficients induces a hand model whose deformations are parameterized by a set of semantically meaningful pose parameters (*e.g.* joint angles). Unfortunately, even though Khamis *et al.* [72] show how to personalize their model for a new user, the lack of a 'background penalty' leaves local minima where the model has grown unrealistically in an attempt to explain the data. To avoid these local minima, they rely on a high-quality initialization that would be difficult to obtain reliably in an online setting. Further, they did not investigate whether the use of a personalized model was important for the accuracy of online hand tracking systems.

In this paper, we address these concerns and show how to use the trained shape basis from [72] to robustly personalize to an individual in a quick and easy calibration step. As illustrated in Figure 7.1, our approach fits a single set of shape coefficients β and per-frame poses $\{\theta_f\}_{f=1}^{F}$ to a set of F depth images (each supplied with a rough initialization pose given by a template-based hand tracking system [124]). To do so, we exploit the 'golden energy' from [124], whose 'render-and-compare' formulation implicitly penalizes protrusions into free space. The energy appears to be the combination of a smooth low-frequency function with a high-frequency, low-amplitude, piecewise-continuous function (see Figure 7.2). The discontinuities in the latter function are the result of occlusion boundaries travelling across locations being discretely sampled by each pixel. This seems to preclude gradient-based optimization, as following the exact gradient on either side of such a jump would not generally yield a good step direction.

One optimization option might be stochastic search (*e.g.* Particle Swarm Optimization) to avoid relying on derivatives, but this converges slowly and typically only works well for low-dimensional optimization problems. Our optimization space (one shape and F poses) is high-dimensional, however, and thus we would like to use a gradient-based optimizer. Although we could carefully work out the true derivatives of a continuous form of this energy [34], it is not obvious if we could compute them quickly. We thus choose to instead use an approximate derivative calculated using central differences. The step size must be right: large enough to jump over nearby occlusion boundaries, and small enough to capture the smooth global behavior of the function. We use a GPU-based tiled renderer to rapidly perform the extra function evaluations that this finite differencing requires. Given our ability to calculate the golden energy and calculate approximate derivatives, we are able to exploit Levenberg-Marquardt to minimize the energy in under a second for a small set of images (*e.g.* $F = 5$).

We can therefore demonstrate for the first time the potential for *detailed* personalization to quantifiably improve the accuracy of a real-time hand tracker. To this end, we adapt [124] to track using the personalized model, and compare template to personalized model tracking accuracy across several datasets. We show that our personalized hand tracking is able to achieve results that are competitive with the state of the art.

7.2 Related work

A large amount of work has been done constructing detailed low-dimensional models of shape and pose variation for human bodies and faces [3, 4, 17, 26, 35, 49, 79, 81, 142, 145]. While hands may be similar to human bodies in the number of degrees of freedom, hands exhibit significantly more self-occlusion. They are also much smaller, which means images from current depth cameras contain fewer foreground pixels and

suffer from more camera noise. Additionally, the space of hand poses is likely larger than that of the space of body poses. Consequently, it is only recently that similar detailed low-dimensional models were built for human hands [72]. Given various RGB-D sensor measurements, these approaches aim to find the low-dimensional shape and pose subspaces by fitting the entire set of observed data. This typically amounts to optimizing a very large number of parameters [18, 72, 83]. Despite the success of these approaches, the number of parameters prohibits their suitability for online *fitting*, although some systems may be close [83].

Recently, morphable subdivision surface models have been used to model other categories of deformation. Cashman and Fitzgibbon [25] demonstrate that extremely limited data (30 silhouette images) can be used to learn such a model for a variety of objects and animals. In more closely-related work, Taylor *et al*. [139] learn a personalized hand model from a set of noisy depth images for a single user, which was the approach adapted by Khamis *et al*. [72] to train a hand shape model on a large dataset of hands.

Other related work tackles differentiation for a render-and-compare energy function, which may at first seem unapproachable due to occlusion boundaries. When the image domain is kept continuous, however, one can show that such energies are naturally differentiable and their exact gradient can be laboriously worked out [75]. Nonetheless, current practical systems discretize the image domain by taking a point sample at each pixel, which introduces discontinuities in the energy caused by occlusion boundaries moving from pixel to pixel. In order to avoid such difficulties, it is tempting to instead approximate the gradient by peering behind these boundaries [16]. Interestingly, Oberweger *et al*. [97] side-stepped this issue completely by training a convolutional neural network to render hands, as gradients are then easily obtainable using the standard back-propagation rules for such networks.

7.3 Hand Shape Calibration

Given a set of depth images, the calibration procedure simultaneously optimizes the hand shape model from all images as well as the pose from independent images by Levenberg-Marquardt Optimization. Although the ultimate goal is solely to find the personalized hand shape model, it is also necessary to optimize the hand pose since the initial pose is simply a rough estimation of the mean hand model.

7.3.1 Shape and Pose Model

We adapt the model developed by Khamis *et al*. [72]. This model is parameterized by the hand pose $\theta \in \mathbb{R}^{28}$ and the hand shape $\beta \in \mathbb{R}^{K}$ to deform an M-vertex triangular mesh, assumed to have a fixed triangulation and hierarchical skeleton.

A vector β of shape coefficients produces a mesh of a hand in a neutral pose, but with a specific hand shape. Simultaneously, the shape also defines the position of the B bones of the skeleton. To be precise, given β, the locations of M vertices fill the columns of the $3 \times M$ matrix $V(\beta)$, and the set of bone locations fill the columns of

the $3 \times B$ matrix $L(\beta)$:

$$V(\beta) = \sum_{k=1}^{K} \beta_k V_k \quad \text{and} \quad L(\beta) = \sum_{k=1}^{K} \beta_k L_k \,. \tag{7.1}$$

The matrices $\{V_k, L_k\}_{k=1}^{K}$ thus form a linear basis for the shape of the model. These are the same bases as [72] for all values of $K \in \{1,2,3,4,5\}$ for which they trained. Note that the regularization used during the training process encouraged the first dimension (V_1, L_1) to represent something akin to a mean hand and skeleton with the other dimensions serving as offsets. We therefore call $\beta^{\text{mean}} = [1,0,\ldots,0]^\top \in \mathbb{R}^K$ the 'mean' hand shape (see Figure 7.1).

The model applies a linear blend skinning (LBS) operator $P(\theta; V, L) \in \mathbb{R}^{3\times M}$ to a mesh V and skeleton L using a set of pose parameters $\theta \in \mathbb{R}^{28}$ that include global rotation, translation, wrist and finger joint rotations. LBS is a standard tool in computer animation. A more dtailed discussion in in [72].

As a new addition to [72], the final step $\Gamma : \mathbb{R}^{3\times M} \to \mathbb{R}^{3\times M'}$ applies a single step of Loop subdivision [82] to the mesh to produce a denser mesh with M' vertices. This brings the resulting mesh into closer alignment with the true 'limit surface' that was fitted to the data in [72], while maintaining efficiency for what follows.

For notational clarity, we combine the steps together as

$$\Upsilon(\theta, \beta) = \Gamma(P(\theta; V(\beta), L(\beta)) \in \mathbb{R}^{3\times M'} \tag{7.2}$$

to denote the full deformation model that produces a subdivided mesh with shape β in pose θ.

7.3.2 The Golden Energy

One way to evaluate whether a specific combination of shape β and pose θ give rise to an image is to simply render the deformed mesh $\Upsilon(\theta, \beta)$ and compare it to the image. If this evaluation can be formulated as an energy function that assigns a low value when the rendered and observed images are close, the problem is then reduced to function minimization.

We thus define an idealized energy by simply integrating the difference between the observations and the rendering across the domain of the image $\mathcal{I}$

$$\hat{E}_{\text{gold}}(\theta, \beta) = \int_{(u,v)\in\mathcal{I}} \rho(\tilde{\mathbf{D}}(u,v) - \tilde{\mathbf{R}}(u,v;\Upsilon(\theta,\beta)))^2 \,\mathrm{d}u\,\mathrm{d}v \tag{7.3}$$

where $\rho(e) = \min(\sqrt{\tau}, |e|)$ with a constant truncation threshold τ. Here, $\tilde{\mathbf{D}}(u,v)$ and $\tilde{\mathbf{R}}(u,v;\Upsilon(\theta,\beta))$ give the observed and the rendered depth at the location (u,v), respectively. (7.3) is adapted from the *golden energy* of [124] with the following modifications:

(i) we use an L^2 penalty (instead of L^1) to allow the use of standard least-squares optimization techniques; and,

(ii) at least conceptually, we operate on a continuous pixel domain $\mathcal{I} \subseteq \mathbb{R}^2$ to model the idealized imaging process [34].

Note that we generally observe a discretized image and thus $\tilde{\mathbf{D}}(u,v)$ will be piecewise constant.

In practice, the integral in (7.3) is difficult and expensive to evaluate so practical systems instead create a discretization by rendering an image of size $W \times H$. The (discretized) golden energy is thus given by

$$E_{\text{gold}}(\theta,\beta) = \frac{1}{WH}\sum_{i=1}^{W}\sum_{j=1}^{H} r_{ij}(\theta,\beta)^2 \tag{7.4}$$

with the residual $r_{ij}(\theta,\beta)$ for pixel (i,j) defined as

$$r_{ij}(\theta,\beta) = \rho(\mathbf{D}_{ij} - \mathbf{R}_{ij}(\mathrm{Y}(\theta,\beta)) \tag{7.5}$$

where $\mathbf{D} \in \mathbb{R}^{W\times H}$ is appropriately resampled from $\tilde{\mathbf{D}}(\cdot,\cdot)$ and $\mathbf{R}_{ij}(\mathrm{Y}(\theta,\beta))$ yields the value of pixel (i,j) in the rendered depth image.

7.3.3 Objective Function

With the goal of optimizing for the user's hand shape from a sequence of depth images $\{\mathbf{D}_f\}_{f=1}^{F}$, we combine the parameterization from Section 7.3.1 and the golden energy from Section 7.3.2. To achieve this goal, we want to minimize

$$E(\beta) = \sum_{f=1}^{F} \min_{\theta} \left(E_{\text{gold}}(\theta,\beta;\mathbf{D}_f) + \lambda_{\text{prior}} E_{\text{prior}}(\theta) \right). \tag{7.6}$$

where $E_{\text{prior}}(\theta)$ is the pose prior that provides constraints on the pose in the form of the negative log-likelihood

$$E_{\text{prior}}(\theta) = (\theta - \bar{\theta})^{\top} \Sigma^{-1} (\theta - \bar{\theta}) \tag{7.7}$$

of a multivariate normal $\mathcal{N}(\bar{\theta},\Sigma)$. The mean $\bar{\theta} \in \mathbb{R}^{28}$ and covariance matrix $\Sigma \in \mathbb{R}^{28\times 28}$ were fitted to a selected set of valid hand poses $\{P_q^{\text{train}}\}_{q=1}^{Q} \subseteq \mathbb{R}^{22}$ captured using the hand tracker of [124], with the variance on the global pose set to ∞. To make the resulting value small, a pose θ must be found for each frame that yields both a low golden energy $E_{\text{gold}}(\theta)$ and a low pose prior energy $E_{\text{prior}}(\theta)$.

Using the standard 'lifting' technique (see *e.g.* [72]), we define a new lifted energy

$$E'(\theta_1,\theta_2,\cdots,\theta_F,\beta) = \sum_{f=1}^{F} \left[E_{\text{gold}}(\theta_f,\beta) + \lambda_{\text{prior}} E_{\text{prior}}(\theta_f) \right]. \tag{7.8}$$

Since $E(\beta) \leq E'(\theta_1,\theta_2,\ldots,\theta_F,\beta)$ for any value of $\{\theta_f\}_{f=1}^{F}$, we seek to implicitly minimize the former by explicitly minimizing the latter. For simplicity, we assign

$$\boldsymbol{\mu} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_F \\ \beta \end{bmatrix} \in \mathbb{R}^{28F+K} \tag{7.9}$$

as the parameter vector so that $E'(\theta_1, \theta_2, \cdots, \theta_F, \beta) = E'(\boldsymbol{\mu})$.

Note that E' has $28F + K$ parameters, and thus would be very difficult to optimize using a stochastic optimizer like PSO [124]. Instead, we use Levenberg-Marquardt, a gradient-based optimizer that can yield second-order-like convergence properties when close to the minimum.

7.3.4 Levenberg-Marquardt Optimization

Consider an arbitrary energy function E and a set of parameters to optimize in $\boldsymbol{\mu}$, the Levenberg-Marquardt Optimization assumes a sum of squares of N residuals written as

$$E(\boldsymbol{\mu}) = \sum_{i=1}^{N} (\epsilon_i(\boldsymbol{\mu}))^2 = \left\| [\epsilon_i(\boldsymbol{\mu})]_{i=1}^{N} \right\|^2 \tag{7.10}$$

where $\epsilon_i(\boldsymbol{\mu})$ is the i-th residual. The optimizer is an iterative procedure such that, for each iteration, the parameter vector is proposed to update with

$$\boldsymbol{\mu}^{\text{prop}} = \boldsymbol{\mu} - \left(\mathbf{J}^\top \mathbf{J} + \gamma \text{diag}(\mathbf{J}^\top \mathbf{J}) \right)^{-1} \mathbf{J}^\top [\epsilon_i(\boldsymbol{\mu})]_{i=1}^{N} \tag{7.11}$$

where γ is the damping factor. If $E(\boldsymbol{\mu}^{\text{prop}}) < E(\boldsymbol{\mu})$, then the update is accepted $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}^{\text{prop}}$ and the damping is decreased $\gamma \leftarrow 0.1\gamma$. Otherwise, the damping is increased $\gamma \leftarrow 10\gamma$ and the proposal is recalculated. Eventually, progress will be made as this is effectively performing a back-tracking line search while interpolating from Gauss-Newton to gradient descent.

Formulating the lifted energy E' from Section 7.3.3 as (7.11), we first rewrite the pose prior as the sum of squared residuals

$$E_{\text{prior}}(\theta) = \|\mathbf{L}(\theta - \bar{\theta})\|^2 \tag{7.12}$$

where we use the Cholesky decomposition on $\Sigma^{-1} = \mathbf{L}\mathbf{L}^\top$. The lifted energy is then reinterpreted as a vector of squared residuals with the concatination of the individual pixel differences $r_{i,j}$ in (7.5) for all F frames from the golden energy and the vector $\mathbf{L}(\theta - \bar{\theta})$ from the pose prior.

The optimizer requires the full Jacobian matrix $\mathbf{J}$ of the residuals with respect to the $28F + K$ parameters (see Figure 7.3(a-c)). Given the independence of the pose parameters across the F depth images (we do not assume any ordering or temporal continuity in the depth images, only that they come from the same individual), it follows that $28F$ columns of $\mathbf{J}$ are sparsely filled by the results of the pixel-wise derivative of the golden energy from a single image $\mathbf{D}_f$ with respect to a pose parameter in θ_f. This is combined with the Jacobian matrix of the pose prior energy. The shape coefficients, however, are the same for all images, so the column that corresponds to a shape coefficient in $\mathbf{J}$ is the concatenation of the pixel-wise derivative of the golden energy from all images.

Differentiating the golden energy. Note that the golden energy in (7.4) is only piece-wise continuous (see Figure 7.2), as moving occlusion boundaries cause jumps in the value of rendered pixels. Our desired optimization procedure requires gradients, but it is evident that the exact derivative of E_{gold} at any specific point of our approximation

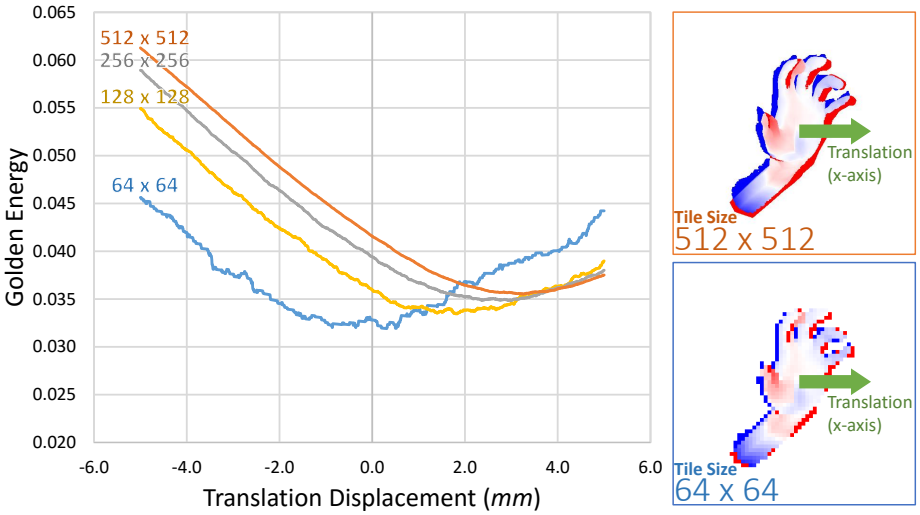


Figure 7.2: Golden energy as a function of x-axis translation, for different rendered tile sizes $W \times H$. Note the globally smooth nature but local discontinuities, which occur at an increasingly small scale with larger tile sizes.

will generally not be helpful. One option would be to return to the idealized continuous energy [75]. However, the edge overdraw antialiasing used is considerably more expensive than a simple render on the GPU. Another approximation [16] is engineered to look behind the occlusion boundary to try to anticipate what will come into view. Nevertheless, we take a different approach that lets us exploit standard GPU-accelerated rendering techniques.

To this end, we note that the curves in Figure 7.2 appear to be the combination of a well-behaved smooth function at a global scale and a low-amplitude non-smooth function at a local scale. If we could somehow recover the former, its gradient would provide a good candidate direction for minimizing (7.4). One option would be to try to smooth out the discontinuities in the approximation using a Gaussian kernel, but this would require the function's evaluation at positions across the entire basin of support of the kernel. For efficiency, we therefore attempt to approximate the function locally by fitting a line to two points that are sufficiently far from each other as to capture the dominant smooth behavior of the energy. Hence, we assign $\phi = \begin{bmatrix}\theta^\top & \beta^\top\end{bmatrix}$ with the parameters associated to an image and approximate the gradient using central differences

$$\frac{\partial E_{\text{gold}}(\phi)}{\partial \phi_k} \approx \frac{E_{\text{gold}}(\phi + \frac{\Delta_k}{2}) - E_{\text{gold}}(\phi - \frac{\Delta_k}{2})}{\epsilon_k} \tag{7.13}$$

where the constant step size ϵ_k is set empirically (see Table 7.1) and the value of the kth element of the vector $\Delta_k \in \mathbb{R}^{28+K}$ is set to ϵ_k while zero elsewhere.

As with (7.4), the residual at pixel (i, j) is only piecewise continuous, although with a sparser set of more dramatic jumps. Similarly then, we find that a central difference with a large step size allows us to approximate the derivative of the residual

$$\frac{\partial r_{ij}(\partial \phi)}{\phi_k} \approx \frac{r_{ij}(\phi + \frac{\Delta_k}{2}) - r_{ij}(\phi - \frac{\Delta_k}{2})}{\epsilon_k}\,. \tag{7.14}$$

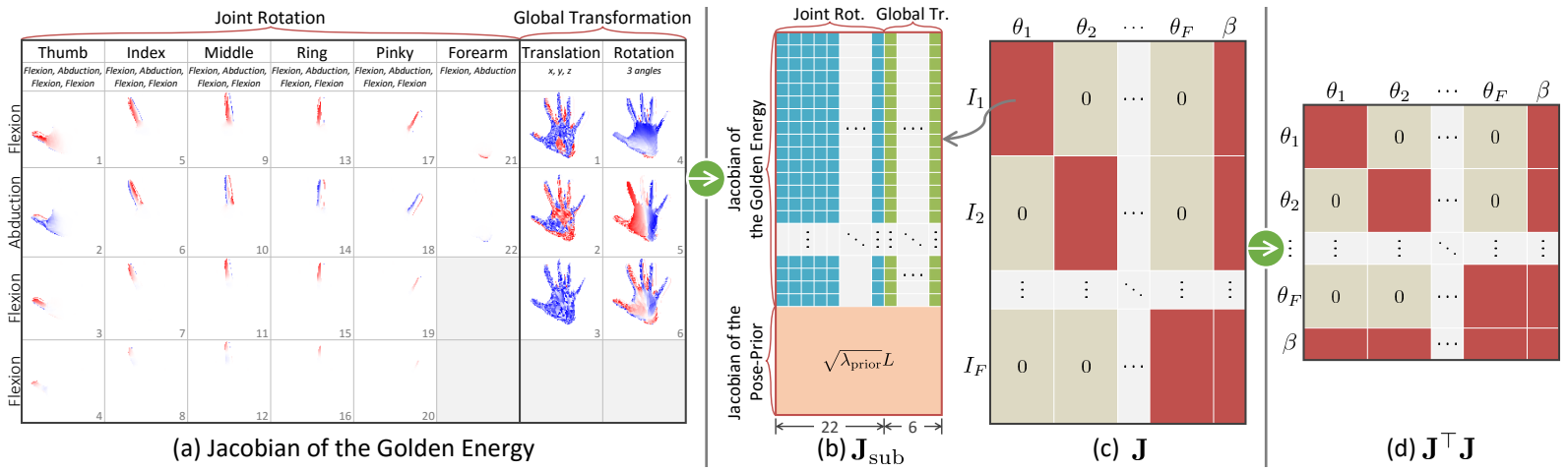


Figure 7.3: (a) Visualization of the Jacobian with respect to pose parameters θ. Each image is reshaped to form a column of $\mathbf{J}$. (b) Rows in $\mathbf{J}_{\text{sub}}$ represent subterms in the energy; columns represent the pose parameters for one frame. (c) Jacobian of the full lifted energy E', including the shape parameters β. (d) Sparsity structure of $\mathbf{J}^\top\mathbf{J}$.

Although one might be concerned about the various approximations above, our use of Levenberg-Marquardt provides a safeguard against catastrophic failure. When steps fail, the algorithm implicitly performs a back-tracking line search as it interpolates from Gauss-Newton to gradient descent. This means that in the worst case, the approximate gradient need only point uphill for progress to be made. In practice, however, we find the approximate derivatives to work quite robustly resulting in few rejected steps, indicated by the many dots (acceptances) in the convergence plots in both Figure 7.4 and Figure 7.5.

Differentiating the pose prior. Since we are computing the derivative of the residuals, the Jacobian matrix of $E_{\text{prior}}(\theta)$ with respect to the parameters is simply $\mathbf{L}$. The importance of the pose prior in our energy becomes more evident in self-occluded poses where the fingers or forearm are not visible in the rendered image. When performing a finite difference with respect to transformation parameters, zero pixel residuals can occur. Thus, without the pose prior, $\mathbf{J}$ and $\mathbf{J}^\top\mathbf{J}$ become rank-deficient. By including the pose prior, the angles of the occluded joints approach the conditional mean of the occluded joints given the visible joints as they remain unobserved by the image.

Box constraints. In addition to the pose prior, we also impose box constraints on the parameters θ to restrict the hand pose from unnatural or impossible deformations. These constraints take the form of limiting values $[P^{\min}, P^{\max}] \in \mathbb{R}^{28} \times \mathbb{R}^{28}$, which we impose using the projection Π such that $\Pi(\boldsymbol{\mu})_i = \min(\max(P_i^{\min}, x_i), P_i^{\max}))$. Then, using the Levenberg-Marquardt method with a projected step [68], we propose the following update of the parameters

$$\boldsymbol{\mu}^{\text{prop}} = \Pi\left(\boldsymbol{\mu} - \left(\mathbf{J}^\top\mathbf{J} + \gamma\,\text{diag}(\mathbf{J}^\top\mathbf{J})\right)^{-1}\mathbf{J}^\top r\right) \tag{7.15}$$

where $\mathbf{J}^\top\mathbf{J}$ is a sparse matrix as illustrated in Figure 7.3(d).

7.4 Experimental Results

We use both synthetic and real data to elucidate our effectiveness at rapidly minimizing our shape-fitting energy. We show that this shape calibration gives us an accuracy improvement on three separate datasets and that our results are competitive with the state of the art. We refer the reader to the supplementary material for more experiments and a video of the live system in action.

For all experiments, we use the step sizes in Table 7.1 to calculate finite differences, a tile size of 256×256 pixels, which gave a good balance of global smoothness and performance (see Figure 7.2), and a truncation threshold $\sqrt{\tau} = 10$cm. While one could minimize our energy using LM for *tracking* (as opposed to shape calibration), it performs only a fairly local optimization. Instead, we use an implementation of [124][1], augmented with our own pose prior.

[1] Despite statements to the contrary [86], [124] optimizes over pose only.

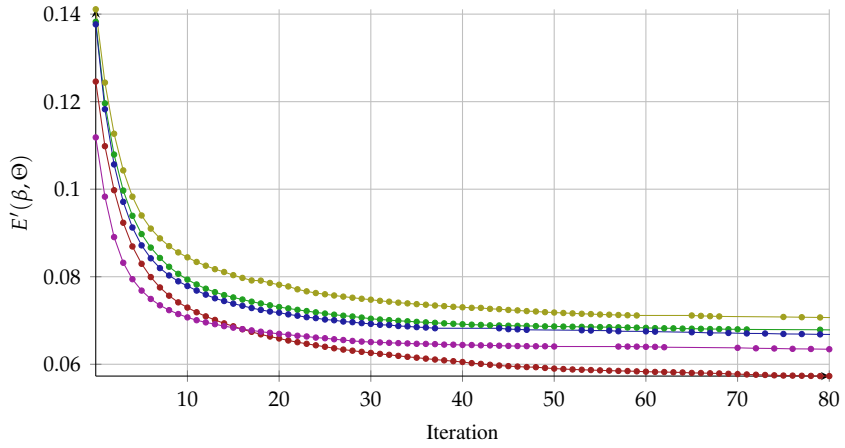


Figure 7.4: Convergence of E' for the five subjects in the FingerPaint dataset. Dots represent successful Levenberg-Marquardt iterations.

Parameter (each row maps to several ks)	Step size
X, Y and Z translations	10 mm
X rotation	5°
Y and Z rotations	2.5°
Metacarpal-phalangeal joint flexions	5°
Metacarpal-phalangeal joint abductions	5°
Proximal interphalangeal joint flexions	10°
Distal interphalangeal joint flexions	15°

Table 7.1: Step sizes ϵ_k used in central differences (7.14).

7.4.1 Synthetic Ground Truth

We begin with an experiment on synthetic data to evaluate our optimization strategy and its ability to find a good hand shape. To this end, we randomly choose a ground truth shape $\beta^{\text{gt}} \in \mathbb{R}^K$. We then sample a set of $F = 40$ poses $\Theta^{\text{gt}} = \{\theta_f^{\text{gt}}\}_{f=1}^F$ from our pose prior, and render a set of depth images $\{\mathbf{D}_f\}_{f=1}^F$. We then initialize our energy minimization at the mean with $\beta = \beta^{\text{mean}}$. In Figure 7.5, we show the convergence when we optimize $E(\Theta, \beta)$. One can see in Figure 7.5 (left) that we rapidly descend the energy landscape in the first 20 iterations. This is clearly correlated with a rapid reduction of $|\beta_1 - \beta_1^{\text{gt}}|$ to near zero, which shows that we quickly obtain the correct scale. Due to the way the shape basis was trained in [72], β_1 is in a unit that roughly corresponds to the scale of the mean hand whereas the units of the other components are less interpretable. Nonetheless, one can see in the right of Figure 7.5 that once scale (*i.e.* β_1) is taken care of, the error in these components is lowered to refine detail. Figure 7.6 shows that minimizing the energy also gives strong agreement between the vertex positions $V(\beta)$ and the corresponding ground truth positions $V(\beta^{\text{gt}})$.

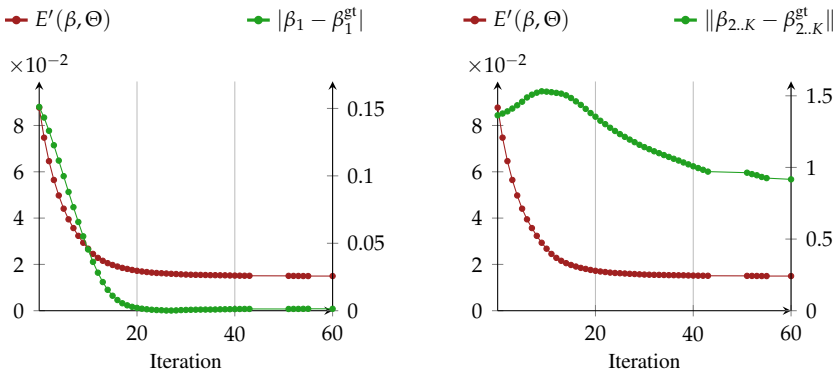


Figure 7.5: Left: Optimizing E' improves the estimate of β_1 which roughly corresponds to scale. Right: The same for the remaining coefficients of β. Dots show successful Levenberg-Marquardt steps.

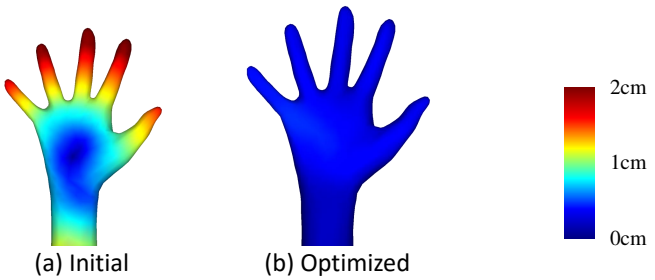


Figure 7.6: Heat maps showing the distance of each vertex to the corresponding ground truth position, for the (a) initial and (b) final iteration of the synthetic experiment (Figure 7.5).

7.4.2 Marker Localization

We now begin exploring the usefulness of our shape calibration procedure in improving tracking accuracy, for which the most common metric is prediction error for a set of marker positions that localize semantic points on the hand. As these locations differ between datasets, we need to create a mapping from the combined shape-and-pose parameters ϕ to a marker position. To do so, for each marker $t = 1, \ldots, T$ we identify four vertices on the correct region of the model using a fixed picking matrix[2] $Y_t \in \mathbb{R}^{4\times M}$, and define an affine combination of these vertices using the barycentric coordinates $w_t \in \mathbb{R}^4$ with $\sum w_t = 1$. We then solve

$$w_t = \underset{w}{\operatorname{argmin}} \sum_{f\in H} \left\| P(\theta_f; V(\beta), L(\beta))\, Y_t^\top w - G_{ft} \right\|^2 \tag{7.16}$$

where $G_{ft} \in \mathbb{R}^3$ is the ground truth location of marker t in frame f, and $H \subseteq \{1, ..., N\}$ is an equally spaced 5% sampling of the N frames in the dataset.

[2] A picking matrix contains zeros except for a single unity entry per row.

7.4.3 NYU Dataset

We test our method on the popular NYU Hand Pose dataset [140], which comprises $N = 8{,}252$ test frames with captures of two different subjects (*i.e.* only two different shapes). Each frame is provided with ground truth locations G_{ft} for 36 positions of the hand. To compute 3D error for Tompson *et al.* [140] on this dataset, we follow recent papers [96, 97, 106] that augment the inferred 2D positions with the captured depth at each location where valid, and the ground truth depth otherwise. We also obtained inferred positions from Tang *et al.* [137], Oberweger *et al.* [97] and Taylor *et al.* [138], selecting a common subset of $T = 10$ positions (2 per digit) for comparison between all methods.

We give quantitative results for four different settings (S1-4) in Figure 7.7. (S1) Since Tompson *et al.* use no temporal information to estimate hand pose, we also configure the tracker [124] to rely only on its discriminative initializer to seed each frame independently. (S2) We used $F = 20$ evenly distributed poses output by (S1) to initialize our calibration and create a $K = 5$ personalized model for each of the two subjects. We then re-ran (S1) using the appropriate personalized model. (S3) We re-enable temporal coherency in the hand tracker (a more realistic setting for tracking), and report the result using the template. (S4) We follow the same procedure as in (S2) but using poses from (S3) to create personalized models. Again, we report the result when tracking is run using the appropriate personalized model.

Notice first that our personalized tracker provides a result comparable to Tang *et al.* [137]. This machine learning approach was trained directly on the NYU training set, and thus benefits from the reduced search space induced by this largely front-facing, limited pose variation dataset. Second, the personalized tracker provides a much better result than the template tracker. We hypothesize that the superior fit of the personalized model (see Figure 7.9) creates a much deeper 'correct' local minimum closer to the true pose, making it easier to find the deep 'correct' local minimum in the next frame. In contrast, personalization does not assist as much when temporal coherence is turned off. Nonetheless, our calibration tool lets us simply upgrade the performance of a compatible tracker with a personalized model. The recent result from Taylor *et al.* [138], using the same personalized models as our result, shows the the accuracy of a *gradient-based* hand tracker when combined with our personalization.

7.4.4 Dexter Dataset

To test our ability to perform detailed surface registration, we turn to the hand part segmentation task required for the FingerPaint dataset [124]. The dataset includes sequences from five different subjects, with pixels labelled as one of 7 parts (5 fingers, the palm, and the forearm). To personalize to each subject, we first run the template-based hand tracker across each sequence. Then, for each subject, we sample $F = 30$ frames, evenly distributed throughout the dataset, and use the poses to run our shape optimization. For this dataset, we try personalizing using $K = \{1, \ldots, 5\}$ for the 5 different shape models from [72]. We then run the tracker on the dataset using the appropriate personalized models, and compare the pixel classification accuracies (see supplementary material and Figure 7.9 for examples of these personalized models). Figure 7.10 shows, as expected, the average classification accuracy increases as we increase K as the deformation model can more accurately register itself to the data.

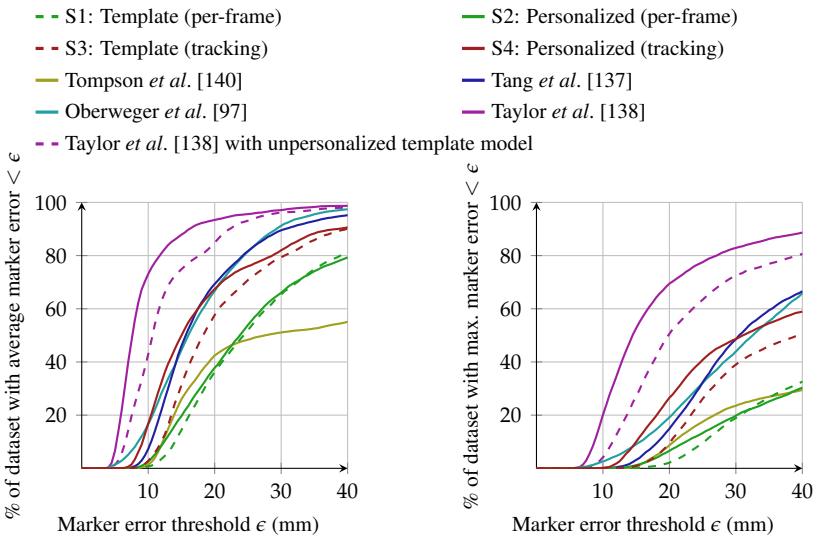


Figure 7.7: Marker localization error on NYU dataset.

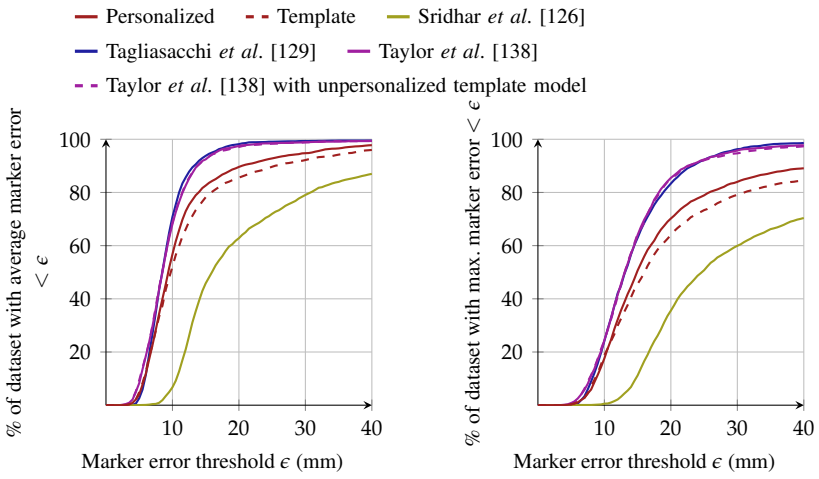


Figure 7.8: Marker localization error on Dexter dataset. The results for this dataset have been normalized so that each of the 7 sequences has equal weight.

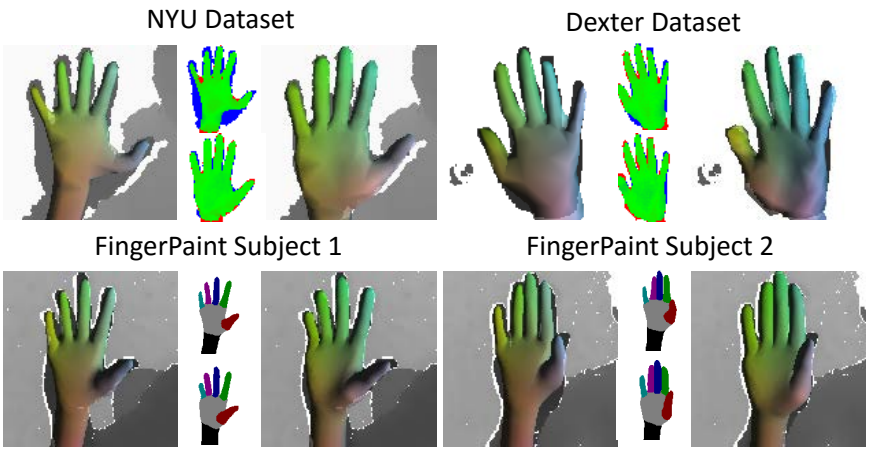


Figure 7.9: Qualitative example of fit difference between template (left and top-middle of each set) and personalized model (bottom-middle and right of each set) for one subject of the NYU (top left), the only subject of the Dexter (top right) and two subjects of the FingerPaint (bottom) datasets.

Interestingly, the $K = 1$ curve which roughly corresponds to a scaled mean hand does not always perform better than the template. We hypothesize that in these areas of the curve, any benefits to personalization are not able to compensate for the bias caused by fitting to a different dataset; in contrast the template is implicitly not biased to any dataset as it was created by hand. Note that the pose prior explains the improvement in accuracy seen between template tracking and Sharp *et al*. [124] in Figure 7.10.

7.5 Qualitative Results

We show that our shape calibration procedure can be used in an online tracker to provide rapid and reliable detailed personalized tracking for any user (see Figure 7.11). This work has been implemented with the live tracker of [124] to include the capability to perform an online personalization of the model. The system starts by using the template model to track the user's hand. The user moves their hand into F different poses, and when the user is comfortable that the tracker has a reasonable pose estimate, a button is pressed to capture both the depth frame and the pose estimate. When satisfied with these poses, the user presses a button to initiate shape calibration. Typically, this procedure takes less than a second, at which point the new personalized model is used for further tracking. A future work in this direction is to automatically conduct the calibration procedure while tracking.

7.6 Conclusion

We have presented the first online method for creating a *detailed* 'personalized' hand model for hand tracking. An easy-to-use calibration step allows a new user to rapidly transition from template to personalized tracking, yielding more robust tracking and better surface alignment that can be exploited by higher-level applications. We have

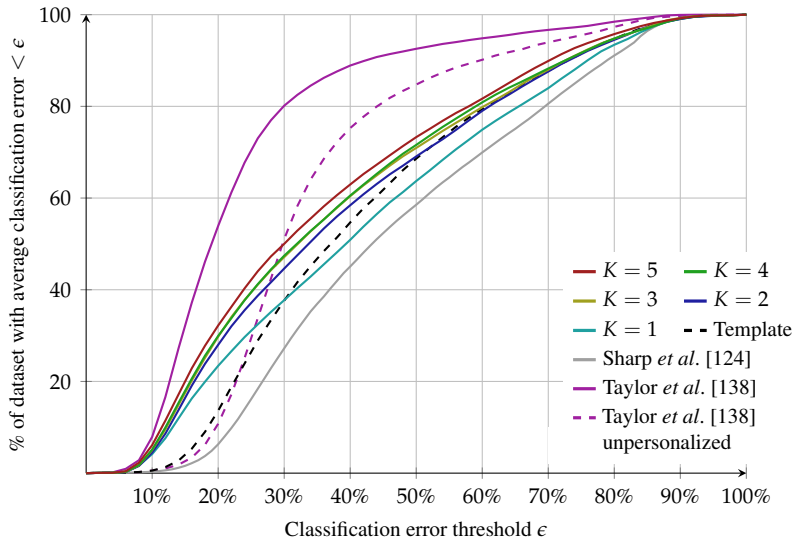


Figure 7.10: Classification error on FingerPaint dataset.

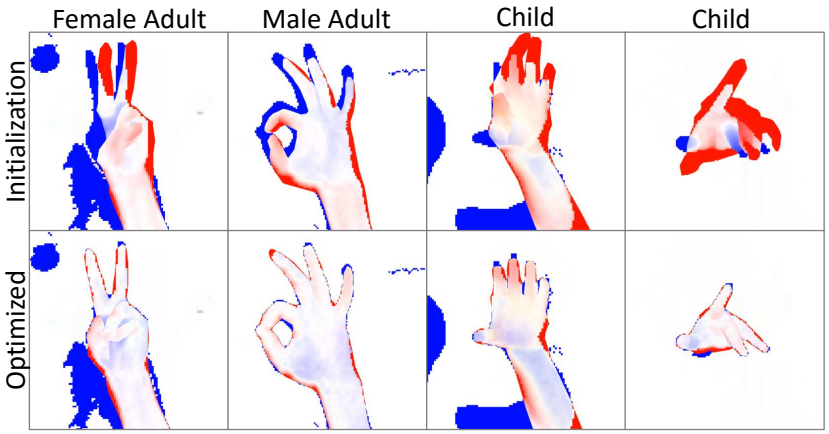


Figure 7.11: Calibration frames at initialization and after convergence of our personalization procedure. The template is the wrong shape for the female subject, too small for the male and wildly too large for the two children. After personalization, each model fits each user 'like a glove'. The truncated golden energy makes the system robust to errors in segmenting the background.

experimentally verified both of these benefits on several standard datasets, showing the increase in both marker localization and dense pixel classification accuracy one obtains when a personalized model is used in place of a poorly-fit template model. Users found our calibration system easy to use and compelling to see a detailed hand avatar. We leave it as future work to address the question of how to remove the calibration step entirely and make personalization fully automatic.

Although this work focuses on the calibration of a personalized hand model, an interesting future direction of the learning-based 3D tracker [133, 134, 135] from the previous chapters is to include deformable objects such as articulated objects or object classes. Nevertheless, this work fits perfectly as a prerequisite step in these forms of tracking.

Part IV

Conclusion

8 Conclusion

8.1 Future Directions

With the patterns taht arise from the chapters in this thesis, there are primarily two future directions in mind – tracker for deformable objects and object classes.

8.1.1 Deformable Object

The deformable learning-based temporal tracker is in line with the hand pose estimation. Note that the pose estimation in Chapter 7 was based on existing method [138] because the chapter focuses on hand shape personalization. The ideal case of the deformable tracker is to achieve the same robustness and efficiency as the rigid case. However, we foresee that the main issue lies on the deformation model and the number of parameters associated to it. Finding a generalized deformation model for a generic object is difficult. In addition, with the increase in the number of parameters, the combination of values to learn also increases drastically. With these in mind, this direction requires further investigation.

8.1.2 Object Class

Another direction is based on the head pose estimation that learns a tracker from a class of head models. The challenge in this problem is the ambiguity of the size and shape of the objects within the same class. Unlike the case of the head pose estimation where the head models have similar shapes and size, other object classes have large variations.

Evidently, in addition to the object class, we can also perform a calibration procedure based on the shape basis to identify the specific model for the object of interest. In this case, the tracker changes from a generalized model intro an adapative model such that, given the object shape basis, the tracker can modify the learned model and track the object with as good accuracy as if it was learned from a single model. Here, in addition to building an adaptive tracker, the challenge is also to create a large database of object and indentifying on a one-to-one alignment of the different models in the class.

A

Authored and Co-authored Publications

Authored:

1. Tan, D.J., Tombari, F., Navab, N.: Real-time accurate 3d head tracking and pose estimation with consumer rgb-d cameras. International Journal of Computer Vision (2017)
2. Tan, D.J., Cashman, T., Taylor, J., Fitzgibbon, A., Tarlow, D., Khamis, S., Izadi, S., Shotton, J.: Fits like a glove: Rapid and reliable hand shape personalization. In: Conference on Computer Vision and Pattern Recognition (2016)
3. Tan, D.J., Tombari, F., Navab, N.: A combined generalized and subject-specific 3d head pose estimation. In: International Conference on 3D Vision. IEEE (2015)
4. Tan, D.J., Tombari, F., Ilic, S., Navab, N.: A versatile learning-based 3d temporal tracker: Scalable, robust, online. In: International Conference on Computer Vision (2015)
5. Tan, D.J., Ilic, S.: Multi-forest tracker: A chameleon in tracking. In: Conference on Computer Vision and Pattern Recognition, pp. 1202–1209. IEEE (2014)
6. Tan, D.J., Holzer, S., Navab, N., Ilic, S.: Deformable template tracking in 1ms. In: British Machine Vision Conference. Citeseer (2014)

Co-authored:

1. Rieke, N., Tan, D.J., Tombari, F., Vizcaino, J.P., di San Filippo, C.A., Eslami, A., Navab, N.: Real-time online adaption for robust instrument tracking and pose estimation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 266–273. Springer (2015)
2. Rieke, N., Tan, D.J., di San Filippo, C.A., Tombari, F., Alsheakhali, M., Belagiannis, V., Eslami, A., Navab, N.: Real-time localization of articulated surgical instruments in retinal microsurgery. Medical image analysis (2016)

3. Rieke, N., Tan, D.J., Alsheakhali, M., Tombari, F., di San Filippo, C.A., Belagiannis, V., Eslami, A., Navab, N.: Surgical tool tracking and pose estimation in retinal microsurgery. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 266–273. Springer (2015) – *Young Scientist Award*

4. Holzer, S., Ilic, S., Tan, D.J., Pollefeys, M., Navab, N.: Efficient learning of linear predictors for template tracking. International Journal of Computer Vision **111**(1), 12–28 (2015)

5. Lallemand, J., Pauly, O., Schwarz, L., Tan, D.J., Ilic, S.: Multi-task forest for human pose estimation in depth images. In: International Conference on 3D Vision (2013)

6. Holzer, S., Ilic, S., Tan, D.J., Navab, N.: Efficient learning of linear predictors using dimensionality reduction. In: Asian Conference on Computer Vision, pp. 15–28. Springer (2012) – *Best Application Paper Honorable Mention*

7. Holzer, S., Pollefeys, M., Ilic, S., Tan, D.J., Navab, N.: Online learning of linear predictors for real-time tracking. In: European Conference on Computer Vision, pp. 470–483. Springer (2012)

Bibliography

[1] Documentation - point cloud library (pcl). http://pointclouds.org/documentation/tutorials/iterative_closest_point.php

[2] Aldoma, A., Tombari, F., Prankl, J., Richtsfeld, A., Di Stefano, L., Vincze, M.: Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In: International Conference on Robotics and Automation, pp. 2104–2111. IEEE (2013)

[3] Allen, B., Curless, B., Popović, Z.: Articulated body deformation from range scan data. ACM Transactions on Graphics **21**(3), 612–619 (2002)

[4] Allen, B., Curless, B., Popović, Z.: The space of human body shapes: reconstruction and parameterization from range scans. ACM Transactions on Graphics **22**(3), 587–594 (2003)

[5] Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., Davis, J.: SCAPE: Shape completion and animation of people. ACM Transactions on Graphics **24**(3), 408–416 (2005)

[6] Asthana, A., Zafeiriou, S., Cheng, S., Pantic, M.: Incremental face alignment in the wild. In: Conference on Computer Vision and Pattern Recognition (2014)

[7] Baker, S., Matthews, I.: Equivalence and efficiency of image alignment algorithms. In: Conference on Computer Vision and Pattern Recognition. Los Alamitos, CA, USA (2001)

[8] Baker, S., Matthews, I.: Lucas-kanade 20 years on: A unifying framework. International Journal of Computer Vision (2004)

[9] Ballan, L., Taneja, A., Gall, J., Gool, L.V., Pollefeys, M.: Motion capture of hands in action using discriminative salient points. In: European Conference on Computer Vision, pp. 640–653 (2012)

[10] Bär, T., Reuter, J.F., Zöllner, J.M.: Driver head pose and gaze estimation based on multi-template icp 3-d point cloud alignment. In: Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on, pp. 1797–1802. IEEE (2012)

[11] Bartoli, A., Zisserman, A.: Direct estimation of non-rigid registrations. In: British Machine Vision Conference (2004)

[12] Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. In: European Conference on Computer Vision (2006)

[13] Benhimane, S., Malis, E.: Homography-based 2d visual tracking and servoing. International Journal of Robotics Research (2007)

[14] Besl, P., McKay, N.D.: A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence **14**(2), 239–256 (1992)

[15] Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence (1992)

[16] Black, M., Loper, M.: OpenDR: An approximate differentiable renderer. In: European Conference on Computer Vision, pp. 154–169 (2014)

[17] Blanz, V., Vetter, T.: A morphable model for the synthesis of 3D faces. In: ACM SIGGRAPH, pp. 187–194 (1999)

[18] Bogo, F., Black, M.J., Loper, M., Romero, J.: Detailed full-body reconstructions of moving people from monocular RGB-D sequences. In: International Conference on Computer Vision, pp. 2300–2308 (2015)

[19] Bookstein, F.: Principal warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on Pattern Analysis and Machine Intelligence (1989)

[20] Bradski, G.: The opencv library. Doctor Dobbs Journal (2000)

[21] Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)

[22] Breitenstein, M.D., Kuettel, D., Weise, T., Van Gool, L., Pfister, H.: Real-time face pose estimation from single range images. In: Conference on Computer Vision and Pattern Recognition (2008)

[23] Cai, Q., Gallup, D., Zhang, C., Zhang, Z.: 3d deformable face tracking with a commodity depth camera. In: Computer Vision–ECCV 2010, pp. 229–242. Springer (2010)

[24] 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)

[25] Cashman, T.J., Fitzgibbon, A.W.: What shape are dolphins? Building 3D morphable models from 2D images. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(1), 232–244 (2013)

[26] Chen, Y., Liu, Z., Zhang, Z.: Tensor-based human body modeling. In: Conference on Computer Vision and Pattern Recognition, pp. 105–112 (2013)

[27] Chen, Y., Medioni, G.: Object modelling by registration of multiple range images. Image and vision computing **10**(3), 145–155 (1992)

[28] Choi, C., Christensen, H.I.: Rgb-d object tracking: A particle filter approach on gpu. In: International Conference on Intelligent Robots and Systems, pp. 1084–1091. IEEE (2013)

[29] Chui, H., Rangarajan, A.: A new point matching algorithm for non-rigid registration. Computer Vision and Image Understanding (2003)

[30] Cootes, T.F., Edwards, G.J., Taylor, C.J.: Active Appearance Models. PAMI **23**(6), 681–685 (2001)

[31] Cootes, T.F., Edwards, G.J., Taylor, C.J.: Active appearance models. IEEE Transactions on Pattern Analysis and Machine Intelligence (2001)

[32] Dame, A., Marchand, E.: Accurate real-time tracking using mutual information. In: Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on, pp. 47 –56 (2010). doi: 10.1109/ISMAR.2010.5643550

[33] Dantone, M., Gall, J., Fanelli, G., Van Gool, L.: Real-time facial feature detection using conditional regression forests. In: Conference on Computer Vision and Pattern Recognition (2012)

[34] de La Gorce, M., Fleet, D.J., Paragios, N.: Model-based 3D hand pose estimation from monocular video. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(9), 1793–1805 (2011). URL `http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=5719617'escapeXml='false'/>`

[35] Delaunoy, A., Prados, E.: Gradient flows for optimizing triangular mesh-based surfaces: Applications to 3D reconstruction problems dealing with visibility. International Journal of Computer Vision **95**(2), 100–123 (2011)

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

[37] Drost, B., Ulrich, M., Navab, N., Ilic, S.: Model globally, match locally: Efficient and robust 3d object recognition. In: Conference on Computer Vision and Pattern Recognition, pp. 998–1005 (2010)

[38] Euler, L.: Problema algebraicum ob affectiones prorsus singulares memorabile. Commentatio 407 Indicis Enestoemiani, Novi Comm. Acad. Sci. Petropolitanae **15**, 75–106 (1770)

[39] Fanelli, G., Dantone, M., Gall, J., Fossati, A., Van Gool, L.: Random forests for real time 3d face analysis. International Journal of Computer Vision (2013)

[40] Fanelli, G., Gall, J., Van Gool, L.: Real time head pose estimation with random regression forests. In: Conference on Computer Vision and Pattern Recognition (2011)

[41] Fischler, M., Bolles, R.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Communications ACM (1981)

[42] Fitzgibbon, A.W.: Robust registration of 2d and 3d point sets. Image and Vision Computing **21**(13), 1145–1153 (2003)

[43] Gay-Bellile, V., Bartoli, A., Sayd, P.: Direct estimation of nonrigid registrations with image-based self-occlusion reasoning. IEEE Transactions on Pattern Analysis and Machine Intelligence (2010)

[44] Geng, X., Xia, Y.: Head pose estimation based on multivariate label distribution. In: Conference on Computer Vision and Pattern Recognition (2014)

[45] Glocker, B., Komodakis, N., Tziritas, G., Navab, N., Paragios, N.: Dense image registration through mrfs and efficient linear programming. Medical Image Analysis (2008)

[46] Gräßl, C., Zinßer, T., Niemann, H.: Illumination insensitive template matching with hyperplanes. In: Proceedings of Pattern recognition: 25th DAGM Symposium. Magdeburg, Germany (2003)

[47] Gräßl, C., Zinßer, T., Niemann, H.: Efficient hyperplane tracking by intelligent region selection. In: Image Analysis and Interpretation (2004)

[48] Hager, G., Belhumeur, P.: Efficient region tracking with parametric models of geometry and illumination. IEEE Transactions on Pattern Analysis and Machine Intelligence (1998)

[49] Hasler, N., Stoll, C., Sunkel, M., Rosenhahn, B., Seidel, H.P.: A statistical model of human pose and body shape. Computer Graphics Forum **28**(2), 337–346 (2009)

[50] Held, R., Gupta, A., Curless, B., Agrawala, M.: 3d puppetry: A kinect-based interface for 3d animation. In: Proceedings of the 25th annual ACM symposium on User interface software and technology (2012)

[51] 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)

[52] Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., Lepetit, V.: Real-time learning of accurate patch rectification (2009)

[53] Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., Navab, N.: Dominant orientation templates for real-time detection of texture-less objects (2010)

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

[55] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., Navab, N.: Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In: Asian Conference on Computer Vision, pp. 548–562. Springer (2013)

[56] Hirshberg, D.A., Loper, M., Rachlin, E., Black, M.J.: Coregistration: Simultaneous alignment and modeling of articulated 3D shape. In: European Conference on Computer Vision, pp. 242–255 (2012)

[57] Holzer, S., Hinterstoisser, S., Ilic, S., Navab, N.: Distance transform templates for object detection and pose estimation (2009)

[58] Holzer, S., Ilic, S., Navab, N.: Adaptive linear predictors for real-time tracking. In: Conference on Computer Vision and Pattern Recognition (2010)

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

[60] Holzer, S., Ilic, S., Tan, D.J., Navab, N.: Efficient learning of linear predictors using dimensionality reduction. In: Asian Conference on Computer Vision, pp. 15–28. Springer (2012)

[61] Holzer, S., Ilic, S., Tan, D.J., Pollefeys, M., Navab, N.: Efficient learning of linear predictors for template tracking. International Journal of Computer Vision **111**(1), 12–28 (2015)

[62] Holzer, S., Pollefeys, M., Ilic, S., Tan, D.J., Navab, N.: Online learning of linear predictors for real-time tracking. In: European Conference on Computer Vision, pp. 470–483. Springer (2012)

[63] Huang, C.H., Boyer, E., Navab, N., Ilic, S.: Human shape and pose tracking using keyframes. In: Conference on Computer Vision and Pattern Recognition, pp. 3446–3453. IEEE (2014)

[64] Irani, M., Anandan, P.: About direct methods. Vision Algorithms: Theory and Practice (2000)

[65] Jurie, F., Dhome, M.: Hyperplane approximation for template matching. IEEE Transactions on Pattern Analysis and Machine Intelligence (2002)

[66] Jurie, F., Dhome, M.: Real time robust template matching. In: British Machine Vision Conference (2002)

[67] Kan, M., Shan, S., Chang, H., Chen, X.: Stacked progressive auto-encoders (spae) for face recognition across poses. In: Conference on Computer Vision and Pattern Recognition (2014)

[68] Kanzow, C., Yamashita, N., Fukushima, M.: Levenberg-Marquardt methods with strong local convergence properties for solving nonlinear equations with convex constraints. Journal of Computational and Applied Mathematics **172**(2), 375–397 (2004). URL `http://www.sciencedirect.com/science/article/pii/S0377042704001256`

[69] Kazemi, V., Sullivan, J.: One millisecond face alignment with an ensemble of regression trees. In: Conference on Computer Vision and Pattern Recognition (2014)

[70] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, vol. 4, pp. 1942–1948 vol.4 (1995). doi: 10.1109/ICNN.1995.488968

[71] Keskin, C., Kiraç, F., Kara, Y.E., Akarun, L.: Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In: European Conference on Computer Vision, pp. 852–863 (2012)

[72] Khamis, S., Taylor, J., Shotton, J., Keskin, C., Izadi, S., Fitzgibbon, A.: Learning an efficient model of hand shape variation from depth images. In: Conference on Computer Vision and Pattern Recognition, pp. 2540–2548 (2015)

[73] Koterba, S., Baker, S., Matthews, I., Hu, C., Xiao, J., Cohn, J., Kanade, T.: Multi-view aam fitting and camera calibration. In: International Conference on Computer Vision (2005)

[74] Krull, A., Michel, F., Brachmann, E., Gumhold, S., Ihrke, S., Rother, C.: 6-dof model based tracking via object coordinate regression

[75] de La Gorce, M., Paragios, N., Fleet, D.J.: Model-based hand tracking with texture, shading and self-occlusions. In: Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)

[76] Lallemand, J., Pauly, O., Schwarz, L., Tan, D.J., Ilic, S.: Multi-task forest for human pose estimation in depth images. In: International Conference on 3D Vision (2013)

[77] Lee, S., Wolberg, G., Shin, S.: Scattered data interpolation with multilevel b-splines. IEEE Transactions on Visualization and Computer Graphics (1997)

[78] Lepetit, V., Lagger, P., Fua, P.: Randomized trees for real-time keypoint recognition. In: Conference on Computer Vision and Pattern Recognition (2005)

[79] Li, H., Sumner, R.W., Pauly, M.: Global correspondence optimization for non-rigid registration of depth scans. Computer Graphics Forum **27**(5), 1421–1430 (2008). doi: 10.1111/j.1467-8659.2008.01282.x. URL `http://dx.doi.org/10.1111/j.1467-8659.2008.01282.x`

[80] Li, X., Hu, Z.: Rejecting mismatches by correspondence function. International Journal of Computer Vision (2010)

[81] Liao, M., Zhang, Q., Wang, H., Yang, R., Gong, M.: Modeling deformable objects from a single depth camera. In: International Conference on Computer Vision, pp. 167–174 (2009)

[82] Loop, C.T.: Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah (1987). URL `http://research.microsoft.com/apps/pubs/default.aspx?id=68540`

[83] Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., Black, M.J.: SMPL: A skinned multi-person linear model. ACM Transactions on Graphics **34**(6), #248 (2015)

[84] Lowe, D.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision (2004)

[85] Lucas, B., Kanade, T.: An Iterative Image Registration Technique with an Application to Stereo Vision. In: International Joint Conference on Artificial Intelligence (1981)

[86] Makris, A., Argyros, A.: Model-based 3D hand tracking with on-line hand shape adaptation. In: British Machine Vision Conference, pp. 77.1–77.12 (2015)

[87] Malis, E.: Improving vision-based control using efficient second-order minimization techniques. In: IEEE International Conference on Robotics and Automation (2004)

[88] Malis, E.: An efficient unified approach to direct visual tracking of rigid and deformable surfaces. In: International Conference on Intelligent Robots and Systems (2007)

[89] Martin, M., Van De Camp, F., Stiefelhagen, R.: Real time head model creation and head pose estimation on consumer depth cameras. In: 3D Vision (3DV), 2014 2nd International Conference on, vol. 1, pp. 641–648. IEEE (2014)

[90] Matas, J., Zimmermann, K., Svoboda, T., Hilton, A.: Learning efficient linear predictors for motion estimation. In: Computer Vision, Graphics and Image Processing (2006)

[91] Matthews, I., Baker, S.: Active Appearance Models Revisited. International Journal of Computer Vision **60**, 135–164 (2004)

[92] Mayol, W.W., Murray, D.W.: Tracking with general regression. Journal of Machine Vision and Applications (2008)

[93] Meyer, G.P., Gupta, S., Frosio, I., Reddy, D., Kautz, J.: Robust model-based 3d head pose estimation. In: International Conference on Computer Vision (2015)

[94] Murphy-Chutorian, E., Trivedi, M.M.: Head pose estimation in computer vision: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence (2009)

[95] Newcombe, R., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A., Kohli, P., Shotton, J., Hodges, S., Fitzgibbon, A.: KinectFusion: Real-time dense surface mapping and tracking. In: Proc. ISMAR (2011)

[96] Oberweger, M., Wohlhart, P., Lepetit, V.: Hands deep in deep learning for hand pose estimation. In: Proc. Computer Vision Winter Workshop (CVWW) (2015)

[97] Oberweger, M., Wohlhart, P., Lepetit, V.: Training a feedback loop for hand pose estimation. In: International Conference on Computer Vision, pp. 3316–3324 (2015)

[98] Oikonomidis, I., Kyriazis, N., Argyros, A.: Efficient model-based 3D tracking of hand articulations using Kinect. In: British Machine Vision Conference, pp. 101.1–101.11 (2011)

[99] Oikonomidis, I., Kyriazis, N., Argyros, A.: Tracking the articulated motion of two strongly interacting hands. In: Conference on Computer Vision and Pattern Recognition, pp. 1862–1869 (2012)

[100] 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)

[101] Padeleris, P., Zabulis, X., Argyros, A.A.: Head pose estimation on depth data based on particle swarm optimization. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on, pp. 42–49. IEEE (2012)

[102] Parisot, P., Thiesse, B., Charvillat, V.: Selection of reliable features subsets for appearance-based tracking. Signal-Image Technologies and Internet-Based System (2007)

[103] Penrose, R.: A generalized inverse for matrices. In: Proceedings of the Cambridge Philosophical Society (1955)

[104] Pilet, J., Lepetit, V., Fua, P.: Fast non-rigid surface detection, registration and realistic augmentation. International Journal of Computer Vision (2008)

[105] Pizarro, D., Bartoli, A.: Feature-based deformable surface detection with self-occlusion reasoning. International Journal of Computer Vision (2012)

[106] Poier, G., Roditakis, K., Schulter, S., Michel, D., Bischof, H., Argyros, A.A.: Hybrid one-shot 3D hand pose estimation by exploiting uncertainties. In: British Machine Vision Conference, pp. 182.1–182.14 (2015)

[107] Pomerleau, F., Colas, F., Siegwart, R., Magnenat, S.: Comparing icp variants on real-world data sets. Autonomous Robots **34**(3), 133–148 (2013)

[108] Redondo-Cabrera, C., Lopez-Sastre, R., Tuytelaars, T.: All together now: Simultaneous detection and continuous pose estimation using a hough forest with probabilistic locally enhanced voting. In: British Machine Vision Conference (2014)

[109] Rekik, A., Ben-Hamadou, A., Mahdi, W.: 3d face pose tracking using low quality depth cameras. In: VISAPP (2), pp. 223–228 (2013)

[110] Ren, C., Prisacariu, V., Murray, D., Reid, I.: Star3d: Simultaneous tracking and reconstruction of 3d objects using rgb-d data. In: International Conference on Computer Vision (2013)

[111] Ren, C.Y., Prisacariu, V., Kaehler, O., Reid, I., Murray, D.: 3d tracking of multiple objects with identical appearance using rgb-d input. In: International Conference on 3D Vision, vol. 1, pp. 47–54. IEEE (2014)

[112] 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, pp. 2953 –2958 (2011). doi: 10.1109/IROS.2011.6094650

[113] Riegler, G., Ferstl, D., Rüther, M., Bischof, H.: Hough networks for head pose estimation and facial feature localization. In: British Machine Vision Conference (2014)

[114] Rieke, N., Tan, D.J., Alsheakhali, M., Tombari, F., di San Filippo, C.A., Belagiannis, V., Eslami, A., Navab, N.: Surgical tool tracking and pose estimation in retinal microsurgery. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 266–273. Springer (2015)

[115] Rieke, N., Tan, D.J., di San Filippo, C.A., Tombari, F., Alsheakhali, M., Belagiannis, V., Eslami, A., Navab, N.: Real-time localization of articulated surgical instruments in retinal microsurgery. Medical image analysis (2016)

[116] Rieke, N., Tan, D.J., Tombari, F., Vizcaino, J.P., di San Filippo, C.A., Eslami, A., Navab, N.: Real-time online adaption for robust instrument tracking and pose estimation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 266–273. Springer (2015)

[117] Rodrigues, O.: Des lois géometriques qui regissent les déplacements d' un systéme solide dans l' espace, et de la variation des coordonnées provenant de ces déplacement considerées indépendent des causes qui peuvent les produire. J. Math. Pures Appl. **5**, 380–400 (1840)

[118] Rusinkiewicz, S., Levoy, M.: Efficient variants of the icp algorithm. In: 3-D Digital Imaging and Modeling (2001)

[119] Rusu, R.B., Cousins, S.: 3d is here: Point cloud library (pcl). In: International Conference on Robotics and Automation, pp. 1–4. IEEE (2011)

[120] Sadourny, R., Arakawa, A., Mintz, Y.: Integration of the nondivergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere 1. Monthly Weather Review **96**(6), 351–356 (1968)

[121] Salzmann, M., Fua, P.: Linear local models for monocular reconstruction of deformable surfaces. IEEE Transactions on Pattern Analysis and Machine Intelligence (2011)

[122] Schulter, S., Leistner, C., Wohlhart, P., Roth, P.M., Bischof, H.: Alternating regression forests for object detection and pose estimation. In: International Conference on Computer Vision (2013)

[123] Segal, A., Haehnel, D., Thrun, S.: Generalized-icp. In: Robotics: Science and Systems (2009)

[124] Sharp, T., Keskin, C., Robertson, D., Taylor, J., Shotton, J., Kim, D., Rhemann, C., Leichter, I., Vinnikov, A., Wei, Y., Freedman, D., Kohli, P., Krupka, E., Fitzgibbon, A., Izadi, S.: Accurate, robust, and flexible realtime hand tracking. In: Proc. CHI, pp. 3633–3642 (2015)

[125] Shum, H.Y., Szeliski, R.: Construction of panoramic image mosaics with global and local alignment. International Journal of Computer Vision (2000)

[126] Sridhar, S., Mueller, F., Oulasvirta, A., Theobalt, C.: Fast and robust hand tracking using detection-guided optimization. In: Conference on Computer Vision and Pattern Recognition, pp. 3213–3221 (2015)

[127] Sridhar, S., Rhodin, H., Seidel, H.P., Oulasvirta, A., Theobalt, C.: Real-time hand tracking using a sum of anisotropic Gaussians model. In: Proc. 3DV, pp. 319–326 (2014)

[128] Supančič III, J.S., Rogez, G., Yang, Y., Shotton, J., Ramanan, D.: Depth-based hand pose estimation: data, methods, and challenges. In: International Conference on Computer Vision, pp. 1868–1876 (2015)

[129] Tagliasacchi, A., Schröder, M., Tkach, A., Bouaziz, S., Botsch, M., Pauly, M.: Robust articulated-ICP for real-time hand tracking. Computer Graphics Forum **34**(5), 101–114 (2015)

[130] Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Conference on Computer Vision and Pattern Recognition (2014)

[131] Tan, D.J., Cashman, T., Taylor, J., Fitzgibbon, A., Tarlow, D., Khamis, S., Izadi, S., Shotton, J.: Fits like a glove: Rapid and reliable hand shape personalization. In: Conference on Computer Vision and Pattern Recognition (2016)

[132] Tan, D.J., Holzer, S., Navab, N., Ilic, S.: Deformable template tracking in 1ms. In: British Machine Vision Conference. Citeseer (2014)

[133] Tan, D.J., Ilic, S.: Multi-forest tracker: A chameleon in tracking. In: Conference on Computer Vision and Pattern Recognition, pp. 1202–1209. IEEE (2014)

[134] Tan, D.J., Tombari, F., Ilic, S., Navab, N.: A versatile learning-based 3d temporal tracker: Scalable, robust, online. In: International Conference on Computer Vision (2015)

[135] Tan, D.J., Tombari, F., Navab, N.: A combined generalized and subject-specific 3d head pose estimation. In: International Conference on 3D Vision. IEEE (2015)

[136] Tan, D.J., Tombari, F., Navab, N.: Real-time accurate 3d head tracking and pose estimation with consumer rgb-d cameras. International Journal of Computer Vision (2017)

[137] Tang, D., Taylor, J., Kohli, P., Keskin, C., Kim, T.K., Shotton, J.: Opening the black box: Hierarchical sampling optimization for estimating human hand pose. In: International Conference on Computer Vision, pp. 3325–3333 (2015)

[138] Taylor, J., Bordeaux, L., Cashman, T., Corish, B., Keskin, C., Sharp, T., Soto, E., Sweeney, D., Valentin, J., Luff, B., Topalian, A., Wood, E., Khamis, S., Kohli, P., Izadi, S., Banks, R., Fitzgibbon, A., Shotton, J.: Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences. In: ACM SIGGRAPH (2016). To appear

[139] Taylor, J., Stebbing, R., Ramakrishna, V., Keskin, C., Shotton, J., Izadi, S., Hertzmann, A., Fitzgibbon, A.: User-specific hand modeling from monocular depth sequences. In: Conference on Computer Vision and Pattern Recognition, pp. 644–651 (2014)

[140] Tompson, J., Stein, M., Lecun, Y., Perlin, K.: Real-time continuous pose recovery of human hands using convolutional networks. ACM Transactions on Graphics **33**(5), #169 (2014)

[141] Tran, Q., Chin, T., Carneiro, G., Brown, M., Suter, D.: In defence of ransac for outlier rejection in deformable registration. In: European Conference on Computer Vision (2012)

[142] Tsoli, A., Mahmood, N., Black, M.J.: Breathing life into shape: capturing, modeling and animating 3D human breathing. ACM Transactions on Graphics **33**(4), #52 (2014)

[143] Viola, P., Jones, M.J.: Robust real-time face detection. International Journal of Computer Vision (2004)

[144] Weise, T., Leibe, B., Van Gool, L.: Fast 3d scanning with automatic motion compensation. In: Conference on Computer Vision and Pattern Recognition, pp. 1–8. IEEE (2007)

[145] Ye, M., Yang, R.: Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera. In: Conference on Computer Vision and Pattern Recognition, pp. 2353–2360 (2014)

[146] Zhu, J., Hoi, S., Lyu, M.: Nonrigid shape recovery by gaussian process regression. In: Conference on Computer Vision and Pattern Recognition (2009)

[147] Zhu, J., Lyu, M.: Progressive finite newton approach to real-time nonrigid surface detection. In: Conference on Computer Vision and Pattern Recognition (2007)

[148] Zimmermann, K., Matas, J., Svoboda, T.: Tracking by an optimal sequence of linear predictors. IEEE Transactions on Pattern Analysis and Machine Intelligence (2009)