

REAL-TIME TEXTURE-BASED 3D OBJECT TRACKING FOR ADVANCED ROBOTIC MANIPULATION

handed in
MASTER'S THESIS

Mariam Elsayed

Human-centered Assistive Robotics
Technical University of Munich
Univ.-Prof. Dr.-Ing. Dongheui Lee

| | |
|----------------------|-----------------------|
| Supervisor: | M. Sc. Manuel Stoiber |
| Start: | 15.12.2020 |
| Intermediate Report: | 25.05.2021 |
| Delivery: | 21.09.2021 |

December 3, 2020

MASTER'S THESIS
for
Mariam Elsayed
Student ID 03672563, Degree EI

Real-Time Texture-Based 3D Object Tracking for Advanced Robotic Manipulation

Problem description:

The Institute of Robotics and Mechatronics of the German Aerospace Center has been developing robotic hands for more than 15 years, which are now comparable to human hands both in size and robustness and can perform tasks in a human environment. Many of these tasks involve the in-hand manipulation of objects, for example when using a screwdriver or a pen. In order to plan and react, the pose of the object has to be known. For this, a multi-modality tracker was developed that uses both region and depth information from an RGBD camera to determine the pose of the object. The tracker also computes an uncertainty in form of a covariance matrix that makes it possible to fuse the pose with kinematic information from the fingers using an Extended Kalman Filter.

Building on this work, the topic of this thesis is the development and integration of an additional tracking modality that uses texture information to improve the accuracy and robustness of tracking for a wide variety of objects [1]. The approach should be integrated with the existing tracker to create a method that considers texture, region, and depth information for robust pose estimation of different objects in various settings. The performance of the developed method should be evaluated on the YCB dataset [2], and if possible on the humanoid robot system "David".

Tasks:

- Literature review on existing texture-based tracking methods, as well as on SLAM approaches
- Design and implementation of the tracking modality
- Integration of the modality into the existing tracker
- Implementation of a method that creates a 3D map of the texture of an object during tracking. The 3D map can be saved and used to improve the tracking performance of known objects.
- Experimental validation
- Thesis documentation

Bibliography:

- [1] V. Lepetit and P. Fua. *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey*. 2005.
[2] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

Supervisor: M. Sc. Manuel Stoiber
Start: 15.12.2020
Intermediate Report: 15.03.2021
Delivery: 15.06.2021

(D. Lee)
Univ.-Professor

Abstract

In many applications, the 6 DoF pose of an object is required. This includes robotic applications, such as in-hand manipulation by an anthropomorphic robot and applications in augmented reality to link the real-world with augmented objects. For these purposes, optical tracking has been used due to its low financial cost and flexible algorithms. However, it is still a challenge to have a versatile tracker that can provide accurate pose estimates for objects with different properties. Specifically, applications of in-hand manipulation pose constraints on the tracker, like the computational efficiency and the ability to handle possible occlusions. These challenges have motivated methods that rely on different information, such as edges or object silhouette. While these approaches work well for complex objects, they struggle with objects that have non-distinct shapes and result in pose ambiguities.

To minimize this problem, we develop a texture-based tracking modality that considers feature points on the object surface to estimate the pose. Each new pose usually depends on the pose estimation from a previous frame, leading to possible error accumulation. While this does not greatly impact the performance in many scenarios, the drift error can be minimized by including a fixed reference of the object characteristics. For this purpose, we additionally develop a framework to generate a sparse feature map of a tracked object, which acts as a fixed reference during further tracking instances. The complete texture-based tracking modality is then combined with an existing region-based tracker in a multi-modality framework. This exploits the advantages of both methods and results in an accurate, robust tracker. Finally, we perform a threefold evaluation on the RBOT dataset, YCB video dataset, and a self-recorded sequence to assess the performance of the tracker and compare it to the current state of the art. The results show that the combined tracker delivers superior tracking performance to existing methods.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 2 | Problem Statement | 7 |
| 2.1 | The <i>David</i> System | 7 |
| 2.2 | Requirements for the Vision-based Tracker | 8 |
| 2.3 | Vision-based object tracking | 8 |
| 2.4 | Multi-modality tracker | 9 |
| 2.4.1 | Region modality | 10 |
| 2.4.2 | Depth modality | 10 |
| 2.4.3 | Extended Kalman Filter framework | 11 |
| 2.5 | Aim of this Work | 11 |
| 3 | Related Work | 13 |
| 3.1 | Texture-based object tracking | 13 |
| 3.1.1 | Feature-point-based tracking | 13 |
| 3.2 | Mapping | 15 |
| 3.2.1 | Structure from motion | 15 |
| 3.2.2 | SLAM | 15 |
| 4 | Pose estimation | 17 |
| 4.1 | Preliminaries | 17 |
| 4.1.1 | Pose parametrization | 17 |
| 4.1.2 | Image projection | 19 |
| 4.2 | Feature correspondences | 20 |
| 4.3 | Energy function | 22 |
| 4.4 | Optimization | 25 |
| 5 | Sparse feature map | 27 |
| 5.1 | Data collection | 27 |
| 5.1.1 | Geodesic grid | 28 |
| 5.1.2 | Grid matching | 28 |
| 5.2 | Point optimization | 29 |
| 5.2.1 | Point clustering | 29 |
| 5.2.2 | Bundle adjustment | 30 |

| | | |
|----------|---|-----------|
| 5.2.3 | Iterative Closest Point | 31 |
| 5.2.4 | Point projection | 32 |
| 5.3 | Integration into the tracker | 33 |
| 6 | Implementation | 35 |
| 6.1 | Efficiency optimization | 35 |
| 6.2 | Occlusion handling | 37 |
| 6.2.1 | Implicit occlusion handling | 37 |
| 6.2.2 | Explicit occlusion handling | 37 |
| 6.3 | Parameter tuning | 38 |
| 7 | Evaluation | 41 |
| 7.1 | Evaluation on the RBOT dataset | 41 |
| 7.1.1 | Design | 41 |
| 7.1.2 | Results | 43 |
| 7.2 | Evaluation on the YCB video dataset | 45 |
| 7.2.1 | Design | 45 |
| 7.2.2 | Results | 46 |
| 7.3 | Evaluation on a recorded sequence | 47 |
| 7.3.1 | Design | 47 |
| 7.3.2 | Results | 49 |
| 8 | Discussion | 53 |
| 9 | Conclusion and outlook | 55 |
| | List of Figures | 57 |
| | List of Tables | 59 |
| | Bibliography | 63 |

Chapter 1

Introduction

Many applications nowadays require 3D tracking of complex objects. These applications range from augmented reality to robotic manipulation. In augmented reality, synthetic objects are overlaid in real-world environments, which requires tracking the relevant objects to estimate the size and position of the synthetic ones. For humanoid robots or - generally - robotic arms, it is also necessary to estimate the object's pose to be grasped accurately. Specifically for manipulating objects within the robotic hand, an accurate pose of the object is essential. One system that facilitates in-hand manipulation is the anthropomorphic robot system *David*, a lightweight robot system that can grasp and manipulate objects in its hands. It was developed at the *Robotics and Mechatronics Center (RMC)* of the *German Aerospace Center (DLR)*. It is shown in Fig. 1.1. Its hands are comparable to their human counterparts in size and robustness, and the system is capable of performing tasks in a human environment. The 6 degrees of freedom (DoF) object pose, consisting of translation and rotation, is required for all these applications.

Optical tracking - using color cameras - is desirable for such applications because of its relatively low financial cost. It offers flexibility to develop different methods targeted for specific applications. However, this comes with challenges like motion blur, occlusions, lighting changes and object ambiguities. To address those issues, different approaches have been proposed in the past with different strengths depending on the application.

In this work we aim to extend a versatile tracker that is able to track objects of different shapes and textures and fulfill the needed requirements. In the following chapter, we will first describe the *David* system. The requirements and constraints for a vision-based tracker used in the *David* system are then discussed to define the problem and the aim of this work. An overview of the related work follows in Chapter 3. In Chapter 4, we provide the mathematical basis for the remainder of this thesis. Following this, we present the tracker formulation with a feature-point-based modality. Then, in Chapter 5, we introduce a novel sparse feature map formulation that reduces drift and enhances the tracking performance. Chapter 6 provides implementation details of the tracker and in Chapter 7, we evaluate the



Figure 1.1: The *DLR* robot system *David* [GAB⁺11]

different components of the tracker on a simulated and real-world dataset and on a self-recorded sequence and show the results. This is followed by a discussion in Chapter 8. Finally, Chapter 9 concludes this work and gives an outlook on future research.

Chapter 2

Problem Statement

This chapter first introduces the *David* system on which the tracker has to operate. Tracking diverse objects for a real-time application like in-hand manipulation poses different requirements for the vision-based tracker. We will outline such requirements in the section that follows and give a short overview over existing tracking methods. Then, we will present the tracker and the extended Kalman filter (EKF) framework that is currently used on the *David* system. We conclude the chapter with a clear definition of the aim of this work.

2.1 The *David* System

The *David* system, developed by the *DLR*, is an anthropomorphic lightweight robot system with arms similar to human arms in size, weight, and performance. The overall system has 41 DoF, 76 actuators, and 165 position sensors. It was developed to have variable stiffness actuation similar to human muscles. The system has an agonist and an antagonist actuator for each DoF in its fingers, making the joint stiffness naturally adjustable independently of its position. This direct physical relationship between force, stiffness, and displacement makes it possible to infer joint torques from the displacement of the elastic elements in the actuators. All of these components make the *David* system well suited for in-hand manipulation.

To perform manipulation tasks, the robot has to know the object pose. For this, the *David* system is equipped with an *Azure Kinect DK*, a Red Green Blue Depth (RGBD) camera that provides color and depth information of the working area. The *Azure Kinect DK* has a 1 MP depth camera, providing depth images with a resolution of 1280×1024 pixels and a 12 MP 4K color camera. [Mic21] The camera runs at 30 Hz. With these specifications, it provides a reasonable basis for the vision-based object tracking algorithm.

2.2 Requirements for the Vision-based Tracker

Performing in-hand manipulation requires having an accurate estimate of the 6 DoF that define the 3D translation and orientation of the object relative to the camera. This application poses different constraints to the tracker, which will be presented in this section.

Efficiency Firstly, the pose calculation must occur in real-time, as the robot needs this information to react. The entire control cycle, including pose estimation and performing an in-hand manipulation step, should utilize the total frequency of the camera, which is 30 Hz. The tracking module should be much faster than the camera frequency to allow smooth movement and prevent information loss in online applications.

Occlusion handling Secondly, the tracker has to be robust to occlusions. In the application of in-hand manipulation, objects are usually occluded by the hand itself. This type of occlusion is typically known and can be handled explicitly by computing a mask over the occluding object. The tracker should provide the possibility to handle unknown occlusions as well. Implicit occlusion handling can utilize the known object model and depth information from the camera to examine inconsistencies between expected and measured depth values.

Robustness Thirdly, the complete tracker has to be robust to different object shapes and textures, including untextured objects and textured and symmetrical ones. Especially for symmetrical objects, texture information has to be utilized to estimate the correct object orientation.

Illumination invariance Lastly, the tracking system must be robust to illumination changes, including lighting differences, shadows from occlusions, and shadows from the object itself. Especially when relying on object texture, different illumination settings can present a challenge and should be handled accordingly.

2.3 Vision-based object tracking

While model-free object tracking methods exist, such as [LKL15, SLK14, 2], in this work, we focus on model-based tracking, in which a 3D model of the tracked object is assumed to be known. Due to the variety of applications and challenges that exist, many methods have been proposed for model-based tracking [LF05, YJS06, 2]. Region-based approaches use statistical models to partition a camera frame into a foreground and a background region, as in [SPS⁺20, ZZZ⁺20, TSSC18, Pri12, 4]. With the resulting silhouette and a known object model, the pose of the object is

estimated. As only the object’s silhouette is considered, these methods face difficulties in tracking objects with local ambiguities or rotational symmetries. In optical flow methods, the motion of a physical point is projected onto a sequence of images. It is assumed that the intensity of the 2D projection is constant and that intensity changes originate from 3D pose changes. Optical flow is prone to drift because it only considers relative change [LF05]. In edge-based approaches, such as [HZSQ20, bug, SPHI14, 3], primitives - or edges - are extracted from the image and matched with primitives from the known object model. These primitives can be straight lines or more complex curves. Minimizing the projection error gives an estimate of the pose. Because only edges are considered, these approaches can be affected by background clutter and object texture.

Machine learning approaches do either standard pose tracking or so-called tracking-by-detection, which does not include previous information to estimate the next pose. In [MKK⁺20], a Convolutional Neural Network (CNN) architecture is proposed with attention modules to handle background clutter and occlusions. Other examples include [DMX⁺21, WMRB20, WXZ⁺20, 3]. While these methods deliver promising results, until now, machine learning approaches are still not fast enough to fulfill our efficiency constraints and they require complex computations on a Graphical Processing Unit (GPU). To our knowledge, no method was proposed yet to compute a pose uncertainty with machine learning approaches, which makes it difficult to reliably fuse results with other sensory outputs.

In feature-point-based methods, such as [WRM⁺10, VLF04a, 2], feature points are detected and matched in consecutive frames. The relative object pose can then be computed from the displacement of feature points between two frames and their corresponding 3D points on the object model. Feature points are usually regions of high-contrast or gradient change. As these methods use local information in the image, they are robust to different object characteristics. The feature points are defined according to their local region, which makes them robust to illumination changes. Because these regions are small enough, an illumination change would likely affect the whole area and, thus, have little effect on the relative intensity differences. However, if no textured 3D model is used, similar to optical flow, they struggle with drift because the error of frame-to-frame motion estimation accumulates over time. In addition to the previous approaches that use RGB color cameras, there are also depth-based methods that utilize depth input, such as [NIH⁺11]. The pose is typically obtained by matching the depth image to the known object model, e.g. using an Iterative Closest Point (ICP) algorithm. Most approaches, like [RPK⁺17, KTIN17, 2], use combined RGB and depth information.

2.4 Multi-modality tracker

To enable robust tracking, a multi-modality tracker has been developed at the *DLR* that combines two of the methods presented in the previous section. It was im-

plemented on the *David* system. Given an object model, the current tracker uses a region modality introduced in [SPS⁺20] and a depth modality based on the ICP algorithm in [NIH⁺11] to determine the pose of the object. This pose is then fused with kinematic data from the robot fingers using an Extended Kalman filter (EKF), as explained in [PCSAS18]. In this section, we will explain the region and depth modalities in more detail and present the extended Kalman filter framework.

2.4.1 Region modality

The region modality relies on RGB images to get information about the objects contour and was presented in [SPS⁺20]. With an initial pose estimate, the object contour is first projected into image space to define a foreground region and a background region describing the object and background. Distributions of color histograms for foreground and background areas known from previous frames are used to calculate the posterior probabilities for both regions. Each projected contour point becomes the center of a correspondence line perpendicular to the contour. The formulation then optimizes the contour points along the correspondence lines according to the posterior probabilities. The optimization is done with a regularized Gauss-Newton algorithm using the gradient and Hessian of the probability formulation.

2.4.2 Depth modality

As opposed to the region modality, the depth modality uses depth images for its pose estimation. It builds on the ICP-based tracking framework in [NIH⁺11]. The aim is to find the object pose that best matches the object model to the depth image. The first step is to find point correspondences between the model and the image. This is done by projecting the model points into the image and assigning each one the closest image point as the correspondence. The correspondence points are then weighted according to their distance and optimized using a Gauss-Newton algorithm. The result is an estimated object pose that best fits the model projection to the depth image.

As both modalities depend on a regularized Gauss-Newton framework for optimization, they can be jointly optimized by adding their gradients and Hessians together. This formulation enables us to incorporate a new modality following the same structure, which is relevant for this thesis. In addition, the computation of the Hessian provides a measure for pose uncertainty that is a necessary requirement for an extended Kalman filter (EKF).

2.4.3 Extended Kalman Filter framework

The EKF framework is based on [PCSAS18] and extends the pose estimation from the vision-based tracker to consider joint angles and velocities from the robotic system. It aims to estimate the object pose as well as joint position biases of the robot. These quantities are joined together in one vector \mathbf{y} . The framework consists of two main steps: a prediction step and an update step. In the prediction step, the estimated mean and covariance of \mathbf{y} are calculated based on a known motion model and measured joint velocities. Then, the estimated object pose from the vision-based tracker is used to update the mean and covariance in the following step. With this formulation, information from the camera and the joint sensors is fused to compute a pose estimation.

2.5 Aim of this Work

The existing tracker framework that was presented in the previous section considers silhouette information from the region modality, and, with the availability of a depth camera, the second modality utilizes depth related information. However, in both modalities no texture information is considered. This presents a disadvantage when tracking objects with a non-distinct silhouette and a symmetrical shape. Building on the idea of a multi-modality vision-based tracker, we can extend this tracker to additionally consider texture.

In this work, we formulate a texture-based tracking modality that can be merged with other tracking methods depending on the application and requirements. While the system can be applied to different methods, in this work we focus on the combination of the region and texture modality. Such a tracker only requires an RGB camera without a depth component and has wide applications.

One of the shortcomings of texture-based tracking is drift, as was presented in Sec. 2.3. Small pose estimation errors can accumulate with each frame and cause an estimation drift over a large sequence of frames. The problem of drift also exists in visual odometry applications, in which the robot's pose and orientation have to be determined from images of its surroundings. The problem was minimized by doing Simultaneous Localization And Mapping (SLAM). Estimating a map allows for making frame-to-map comparisons. After detecting a loop closure, the algorithm can correct large accumulated errors at once and minimize drift. In this work, we adopt a similar concept and derive a novel sparse feature map formulation that acts as a reference for the texture-based modality during tracking.

The first part of the work is the development of the texture-based tracking modality. We then design and implement the method to generate a sparse feature map of the object. Having a feature map - or rather a model - improves and accelerates tracking for known objects. With a feature map, the algorithm can compare frame-to-map instead of frame-to-frame to minimize drift. Providing a method for map creation using tracking data eliminates the need for texture model creation, e.g., using a 3D

scanner, and is easily integrated into the regular tracking framework.

Finally, the complete texture-based tracking modality is combined with the region modality from [SPS⁺20]. Developing and integrating the texture modality into the combined tracker improves its robustness.

Chapter 3

Related Work

The literature on 3D pose estimation is vast. We have previously motivated our choice for texture-based tracking and defined the aim of this work. In this chapter, we will first overview the relevant state of the art from texture-based tracking methods. Then, we will present relevant research in object and environment mapping, which provides a good reference for the sparse feature map formulation that is part of this work.

3.1 Texture-based object tracking

Image texture describes the spatial disparity of pixel intensities within a certain region of the image. Such information can be used to indicate the position of an object and is the basis for texture-based tracking. This can be categorized into dense tracking that considers all image pixels and feature-point-based tracking that relies on a sparse representation of distinct features. Dense methods have been mostly employed in environment mapping algorithms, such as [KSC13, WLM⁺15, 2], while feature-point-based methods are more common in object tracking applications. One reason for this tendency is that dense methods are more affected by illumination variance, which is more visible in object tracking than mapping large spaces. In the following section, we will present the state of the art of the feature-point-based methods.

3.1.1 Feature-point-based tracking

Feature-point-based methods rely on a sparse representation of the image data in the form of feature points. A feature point is composed of a keypoint and the corresponding descriptor. The keypoint contains the 2D position of the feature and different properties such as scale and rotation. The descriptor includes the visual description of the region around the keypoint and is used to compare the similarity between keypoints to match them. The terms keypoint, feature, and

interest point are sometimes used interchangeably in the literature. Feature-point-based tracking consists of three main steps: feature detection, feature matching, and pose calculation. Feature detection includes keypoint detection and computing the respective descriptors. For 2D tracking methods, the matching step consists of matching keypoints between two frames according to their descriptors. In our 3D object tracking application, this step firstly includes finding 2D-3D correspondences. Finally, the set of correspondences is used to calculate the object pose. There have been different solutions to the three stages.

Over the years, several feature detectors have been developed, which offer different trade-offs between accuracy, robustness, and speed. Lowe proposed the Scale-Invariant Feature Transform (SIFT) descriptor that relies on local gradients [Low99, Low04]. It quickly gained popularity thanks to its high accuracy and robustness. However, the computationally expensive high-dimensional descriptor makes the algorithm not fast enough for most real-time applications. Another detector named Speeded Up Robust Features (SURF) was introduced in [BTVG06], which is faster than SIFT while maintaining similar performance. Much faster algorithms became possible with the introduction of binary descriptors, as in the algorithms Binary Robust Invariant Scalable Keypoints (BRISK) [LCS11] and Oriented FAST and Rotated BRIEF (ORB) [RRKB11]. Both detectors have similar performance. BRISK computes the descriptors by making binary intensity comparisons between the center pixel and the surrounding ones in a circular sampling pattern. In contrast, the Binary Robust Independent Elementary Features (BRIEF) descriptor, used in ORB, performs binary intensity comparisons between random pixel pairs in the patch around the feature point.

To compute correspondences of the detected features to the 3D object points, one possible method is to match detected features between two consecutive frames and then, using the estimated pose and camera parameters, to reconstruct the model point. Because it relies on the estimated pose, it can accumulate estimation errors. Another common method is to first include an offline training phase. The keyframes of the model are obtained with the associated pose [SL04, VLF04b, LZ13, SPS16, 4]. These are then used to reconstruct a textured object model by performing Structure from Motion (SfM) [SSS06]. Finally, the descriptors for each point are computed. During tracking, features are detected in an image and then matched against those in the model. While this group of methods performs well during the online tracking phase and reduces drift, it requires a lot of offline work for each new object, as each keyframe has to be segmented before the SfM step and the scaling has to be set manually [SPS16].

Having computed a set of correspondences, the next step is to estimate the pose. This problem is known as the Perspective-n-Point (PnP) problem. The basic form requires three correspondences (P3P) and was first investigated by [Gru41]. A more recent review of PnP solutions can be found in [Lu18]. To find the best set of correspondences and avoid outliers, PnP is often combined with Random Sample Consensus (RANSAC) [FB81, LZ13]. However, this method is very computationally

expensive. An alternative to RANSAC is to include an M-estimator, like a Tukey or Huber norm [LF05], to penalize large correspondence errors.

3.2 Mapping

Mapping is a task that is closely related to motion estimation. In previous years, robot motion estimation, known as visual odometry [NNB04], and environment reconstruction were treated separately. Reconstruction by vision relied on Structure from Motion (SfM) algorithms, as was the case in object tracking. With the introduction of visual SLAM [DRMS07], the performance of the motion tracking, as well as the reconstruction, improved. In the following section, we will first present the development of SfM and its current applications in object tracking, then we will give an insight on the state of the art SLAM algorithms.

3.2.1 Structure from motion

The aim of SfM algorithms is to recover the 3D pose of a scene or object, referred to as the structure, by estimating the camera motion relative from a collection of 2D images of that scene. The problem was first defined and solved for the case of two cameras in [LH81]. In [TK92], the authors presented a solution using a factorization method, in which the measurement matrix of some tracked feature points is factorized into camera motion and object shape. Later approaches [Har93, TMHF00, 2] introduced bundle adjustment to solve SfM problems. In bundle adjustment, both motion and structure are refined together to minimize the reprojection error. One of the most notable applications of bundle adjustment is [AFS⁺11], in which the team attempted to reconstruct the city of rome from a vast collection of images from the internet.

3.2.2 SLAM

One of the first approaches to solve the SLAM problem was the algorithm EKF-SLAM [LDW91]. It is based on an Extended Kalman Filter and provided the basis for new SLAM implementations. In [MTKW02], the FastSLAM algorithm was introduced and extended the EKF formulation with a particle filter. Since then, numerous SLAM approaches were introduced that rely on the different tracking methods. The algorithm introduced in [SLK16] uses an RGBD camera input and estimates the camera motion using an Iterative Closest Point (ICP) algorithm. This framework was extended in [LD17] to further consider edge information.

One of the most notable feature-based SLAM algorithms is ORB-SLAM, which was introduced in [MAMT15] and extended in [MAT17] to ORB-SLAM2. In [CER⁺21], ORB-SLAM3 was introduced, which represents the current state of the art. In the main algorithm, features are extracted in each camera frame, assigned a depth value from the depth image, and matched across time frames. The motion is then

estimated by minimizing the reprojection error with bundle adjustment. To reduce local drift, the new frame is matched against a keyframe instead of the preceding frame. For the map creation, new RGBD images are matched against a small set of keyframes to detect loop closures when similar views appear. The map is represented as a pose graph with each keyframe as a vertex and each loop closure as an edge. Correction of accumulation errors after a loop closure is done by solving a non-linear least-squares problem distributed over the graph's edges. The framework of ORB-SLAM3 [CER⁺21] additionally allows for the creation of multiple maps when viewing a new place.

Chapter 4

Pose estimation

In the previous chapters, we have motivated our choice for feature-based methods. These rely on feature points to extract information from camera frames. In the following chapter, we first define the mathematical notation used in this work in the preliminaries section. Then, we derive the theoretical outline for a tracker that can estimate the 6 DoF object pose and the pose uncertainty from texture data. We first explain the concepts needed for feature detection, description, and matching. Then, we derive an energy function by establishing a relationship between the needed pose and the set of matched features. Finally, an optimization framework to minimize such a function with the Newton method is presented.

4.1 Preliminaries

This section defines basic mathematical concepts and notation, which will provide the basis for the following sections. First, we need to parametrize the pose transformation of an object during its movement in relation to a camera. Additionally, we need to define the projection of points between image space and model space.

4.1.1 Pose parametrization

The translation and rotation of an object are collectively called a pose. To describe a pose in Euclidean space, a minimum of six coordinates is required. Three coordinates are needed to define the translation, while different representations exist to describe the rotation [SK08].

Homogeneous transformations combine translation and rotation in a single matrix that can describe full pose variations between coordinate frames. Translation is denoted with a vector \mathbf{T} and rotation with a matrix \mathbf{R} . The relation between some coordinate frames A and B can be expressed as

$$\begin{bmatrix} {}_A\mathbf{X} \\ 1 \end{bmatrix} = \begin{bmatrix} {}_A\mathbf{R}_B & {}_A\mathbf{T}_B \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} {}_B\mathbf{X} \\ 1 \end{bmatrix}, \quad (4.1)$$

with ${}_A\mathbf{X}$ a position vector represented in coordinate frame A , ${}_B\mathbf{X}$ a position vector in coordinate frame B , ${}_A\mathbf{R}_B$ the rotation matrix from coordinate frame B to A represented in A , and ${}_A\mathbf{T}_B$ the translation vector from coordinate frame B to A represented in A . Using homogeneous transformations with the homogeneous transformation vector $\tilde{\mathbf{X}} = [\mathbf{X} \ 1]$ and the homogeneous transformation matrix ${}_A\mathbf{D}_B$, the relation in (4.1) can be written concisely as

$${}_A\tilde{\mathbf{X}} = {}_A\mathbf{D}_{BB}\tilde{\mathbf{X}}. \quad (4.2)$$

Homogeneous transformations provide a compact notation, but include redundant elements and multiplications. Minimal representations on the other hand use independent coordinates, making them suitable for optimizations. An appropriate representation for our application is the angle-axis representation. It consists of an angle θ that rotates around an axis vector \mathbf{w} . Writing the parameters as exponential coordinates $\mathbf{r} = \mathbf{w}\theta$ allows us to write the rotation matrix as

$$\mathbf{R} = \exp(\mathbf{r}_{[\times]}) = \mathbf{I} + \mathbf{r}_{[\times]} + \frac{1}{2!}\mathbf{r}_{[\times]}^2 + \frac{1}{3!}\mathbf{r}_{[\times]}^3 + \dots, \quad (4.3)$$

with $\mathbf{r}_{[\times]}$ representing the skew-symmetric matrix of vector \mathbf{r} .

This rotation matrix representation can be linearized by taking the first two terms of the expansion series into account and neglecting the higher-order ones. With the addition of the translation variation \mathbf{t} , the linear variation around a certain pose can be written as

$$\mathbf{X}^+ = (\mathbf{I} + \mathbf{r}_{[\times]})\mathbf{X} + \mathbf{t}, \quad (4.4)$$

with \mathbf{X}^+ being the result of the variation of \mathbf{X} . The vectors \mathbf{r} and \mathbf{t} form the full linear variation and can be combined in a coordinate vector $\boldsymbol{\theta}^T = [\mathbf{r}^T \ \mathbf{t}^T]$. We can write the relation (4.4) in homogeneous coordinates as

$$\tilde{\mathbf{X}}^+ = \begin{bmatrix} \mathbf{I} + \mathbf{r}_{[\times]} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\mathbf{X}} = \mathbf{D}^+(\boldsymbol{\theta})\tilde{\mathbf{X}}. \quad (4.5)$$

In the application of our tracker, the point \mathbf{X} is typically in a camera frame of reference C , and a transformation to the model frame of reference M is needed, or the inverse operation. For this, we combine the full homogeneous transformation with the representation of the linear variation in (4.5). Specifying $\boldsymbol{\theta}$ as the variation around the model coordinate frame, the transformation and variation from camera to model can be written as

$${}_M\tilde{\mathbf{X}}^+ = \mathbf{D}^+(\boldsymbol{\theta}){}_M\mathbf{D}_{CC}\tilde{\mathbf{X}}, \quad (4.6)$$

with ${}_M\mathbf{D}_C$ the homogeneous matrix from camera frame C to model frame M . The inverse operation with the same $\boldsymbol{\theta}$ results in

$${}_C\tilde{\mathbf{X}}^+ = {}_C\mathbf{D}_M\mathbf{D}^+(\boldsymbol{\theta})^{-1}{}_M\tilde{\mathbf{X}}. \quad (4.7)$$

The inverse linear variation around the model frame

$$\mathbf{D}^+(\boldsymbol{\theta})^{-1} = \begin{bmatrix} (\mathbf{I} - \mathbf{r}_{[\times]}) & (-\mathbf{I} + \mathbf{r}_{[x]})\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \quad (4.8)$$

can be derived from (4.4) using the relation $\mathbf{r}_{[x]}^T = -\mathbf{r}_{[x]}$. With equations (4.6) and (4.7) we can now express pose transformations between a camera and a model frame of reference and apply a linear variation to both the forward and inverse transformation.

4.1.2 Image projection

In addition to transforming points from a model to a camera frame of reference, we also need to map the points from a 2D image space to the 3D model space and vice versa. For this, we use the notation $\mathbf{X} = [X \ Y \ Z]^T$ for a point in model space and $\mathbf{x} = [x \ y]^T$ for the corresponding point in image space. An RGB camera frame is denoted as I_c .

Assuming the intrinsic parameters of the camera to be known from an offline pre-calibration step, we denote the linear projection matrix as

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (4.9)$$

with f_x and f_y being the focal lengths in both directions and $p = [p_x \ p_y]^T$ being the principal point. Assuming all images have been rectified by removing lens distortion, the projection $\boldsymbol{\pi}$ of a 3D model surface point onto an image plane is

$$\mathbf{x} = \boldsymbol{\pi}(\mathbf{X}) = \begin{bmatrix} \frac{X}{Z}f_x + p_x \\ \frac{Y}{Z}f_y + p_y \end{bmatrix}. \quad (4.10)$$

The inverse operation requires an estimated or known depth value d of the point in the image plane and can be written as

$$\mathbf{X} = \boldsymbol{\pi}^{-1}(\mathbf{x}, d) = \begin{bmatrix} \frac{d}{f_x}(x - p_x) \\ \frac{d}{f_y}(y - p_y) \\ d \end{bmatrix}. \quad (4.11)$$

This formulation allows us to switch from image to model frame and will be the basis for the following chapters.

For object tracking, we have a time discrete sequence of images. An image captured at time t_k is denoted by $I_c(t_k)$, $k = 0, \dots, l$, with $I_c(t_l)$ being the current live image. The corresponding points in image space for the current image are $\mathbf{x}_i(t_l)$.

4.2 Feature correspondences

An essential step of any feature-based algorithm is finding feature correspondences between camera frames. This consists of three main steps: feature detection, feature description, and feature matching. In the detection step, distinct local regions of an image are found. These are then assigned a descriptor, which is a string of values describing the pixel values of a feature point and the surrounding region. The descriptor is used to evaluate the similarity of two points, a process needed to determine a possible match.

Several detectors and descriptors are discussed in Sec. 3.1.1. After conducting some initial trials on the efficiency and accuracy of the different detectors and descriptors, we decided to use the Oriented FAST and Rotated BRIEF (ORB) detector [RRKB11] in this work, as it provided the best trade-off in this regard. This observation was also confirmed in [GC15, MAMT15, 2].

Feature detection Features are detected by analyzing the greyscale color intensities of pixels. Certain patterns in small image patches around a pixel point indicate a feature, which can be either an edge, a corner, or a blob. Edge and corner points separate lighter pixels in one region from darker pixels on the other side, while blobs represent a dark region in a light area and vice versa. The features must be repeatable, meaning that similar features produce similar descriptors to ensure a robust match.

The ORB detector relies on the features from the Accelerated Segment Test (FAST) algorithm, developed by [RD06]. It uses a segment test, which defines a circular ring typically made up of sixteen pixels around an interest point p , as shown in Fig. 4.1. Each pixel x in the circular ring can have one of three states:

$$S(x) = \begin{cases} \text{darker,} & I(x) \leq I(p) - t \\ \text{similar,} & I(p) - t \leq I(x) \leq I(p) + t \\ \text{brighter,} & I(p) + t \leq I(x) \end{cases} \quad (4.12)$$

with $I(p)$ and $I(x)$ as the image intensities of the interest point p and pixel x on the circular ring, respectively. The point p is considered a feature point if at least twelve consecutive pixels along that circle are either *brighter* or *darker* than p by a certain threshold. The feature point, in this case, is a corner.

ORB employs a scale pyramid of the image and computes FAST features at each scale level. This way, higher scale invariance is achieved. To further improve robustness, an orientation component is added to each feature, using the so-called intensity centroid. It is assumed that the point with the highest patch intensity around the feature point is offset from the center. The vector from the center pixel to this intensity centroid may then be used to compute the orientation of the feature point [RRKB11]. The orientation is used to rotate the feature point location before computing the descriptor.

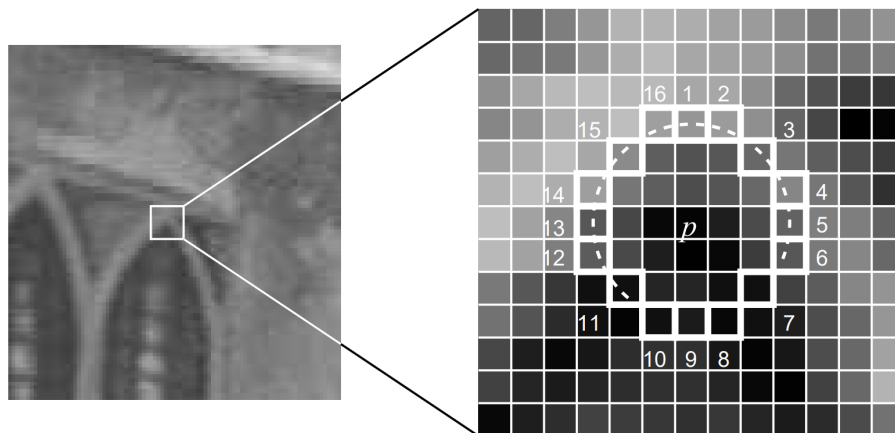


Figure 4.1: The segment test of the FAST detector. The arc shows 12 consecutive pixels brighter than the interest point p . [RD06]

Feature description After detecting features in an image frame, their corresponding descriptors are computed. One very efficient variant are binary descriptors. Binary descriptors, in general, perform intensity comparisons between pixel values around the feature. The results from the comparisons form a binary descriptor string that describes the region around the feature. Similar-looking feature points produce similar descriptors. The binary descriptors usually differ in the sampling pattern chosen to perform the point pair comparisons. The BRISK descriptor, introduced in Sec. 3.1.1, defines concentric rings of pixels around the feature as its sampling pattern and performs comparisons between neighboring point pairs. A boolean value is added to the descriptor string for each comparison. In contrast, the BRIEF descriptor performs intensity comparisons between random pixel pairs in the feature point’s vicinity. BRIEF is the descriptor applied in the ORB framework. However, for this work, we have decided to use the novel Boosted Efficient Binary Local Image Descriptor (BEBLID), which was introduced in [SB20]. Instead of comparing single pixel values, it compares mean values of small regions around the feature center, making it one of the most efficient descriptor at the time of writing.

Feature matching In each incoming camera frame, features are detected along with their descriptors. The descriptors from a camera frame are then matched with those from the preceding frame to establish point correspondences. The more similar two descriptors are, the more likely it is that they describe the same feature. The comparisons are made using brute force matching with Hamming distance. It counts the number of equal bits in both descriptors as a measure of similarity. This operation can be done efficiently using an XOR followed by a bit count. To minimize incorrect matches, we utilize Lowe’s ratio test [Low04]. A match is considered reliable if the distance to the nearest match is much smaller than the distance to the

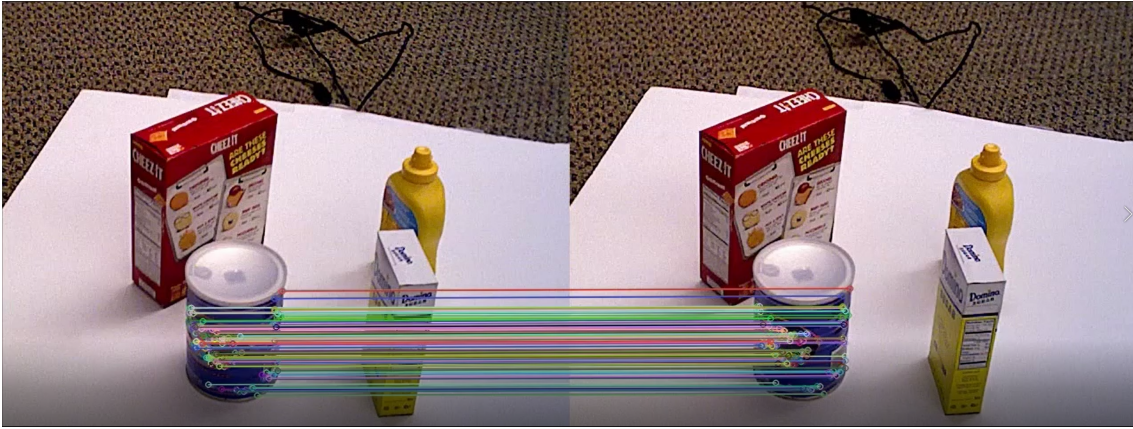


Figure 4.2: Example of feature matching from this work for the can object between two consecutive frames of the YCB dataset [XSNF17]

second nearest one, i.e., $d_1 < td_2$, with t as a threshold. In [Low04], this threshold was set to 0.7. Matches that do not fulfill this condition get discarded. An example of the correspondences found with this presented method on consecutive frames of the YCB dataset can be seen in Fig. 4.2.

4.3 Energy function

After obtaining feature correspondences in two consecutive frames with the methods presented above, the next step relies on an energy function to estimate the object pose. In this section, we will derive such a function and calculate its gradient and Hessian. We start with the following formulation

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} E, \quad (4.13)$$

with $\boldsymbol{\theta}^*$ as the optimal pose parameters that minimize E . We first define the residual error

$$r_i = \|\mathbf{x}'_i(t_l) - \mathbf{x}_i(t_{l-1})\|_2 \quad (4.14)$$

as the distance between a feature point $\mathbf{x}'_i(t_l)$ from the live camera frame and its corresponding feature point $\mathbf{x}_i(t_{l-1})$ from the previous frame transformed into the current image. For ease of notation, we will omit the time stamps, so r_i becomes

$$r_i = \|\mathbf{x}'_i - \mathbf{x}_i\|_2. \quad (4.15)$$

To establish a relation to the pose parameters $\boldsymbol{\theta}$, we use the pose estimate from t_{l-1} and the known object model to obtain the depth value of the point \mathbf{x}_i . With this information, we compute the reconstructed 3D point \mathbf{X}_i according to Eq. (4.11) and can write the residual with the projection of this point into the current image

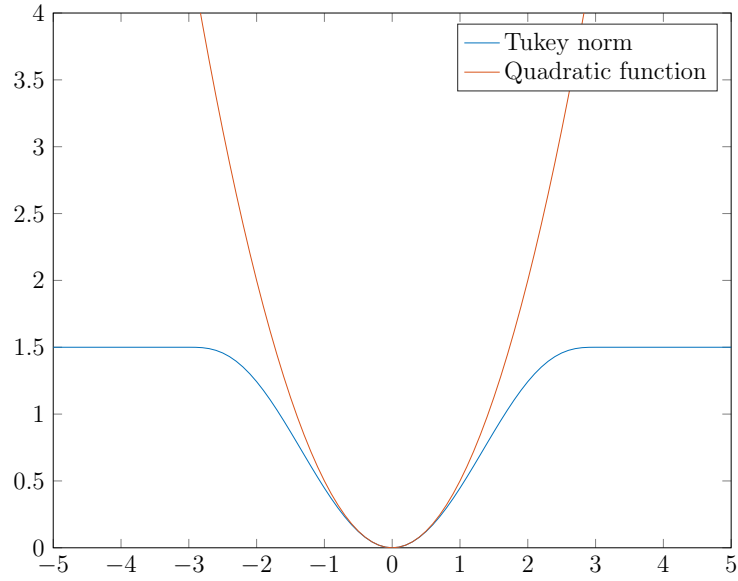


Figure 4.3: Tukey norm for $c = 3$ compared to $\frac{x^2}{2}$

frame as

$$r_i = \|\mathbf{x}'_i - \boldsymbol{\pi}(\mathbf{X}_i)\|_2. \quad (4.16)$$

We then write an energy function for the correspondence error of the feature points to be minimized. Instead of minimizing

$$E = \sum_{i=0}^N r_i^2, \quad (4.17)$$

we can use a more robust version

$$E = \sum_{i=0}^N \rho_{Tuk}(r_i), \quad (4.18)$$

with the number of matched feature points N . The function ρ_{Tuk} is the Tukey norm, shown in Fig. 4.3, that minimizes the effect of outliers causing large residual errors r_i^2 . It is defined as

$$\rho_{Tuk}(v) = \begin{cases} \frac{c^2}{6} (1 - (1 - (\frac{v}{c})^2)^3) & \text{if } |v| \leq c \\ \frac{c^2}{6} & \text{otherwise} \end{cases}, \quad (4.19)$$

where c is a constant that is typically set to a value proportional to the standard deviation of residual errors for inliers [LF05].

To minimize E with a Newton algorithm, we write it as a weighted least-squares problem

$$E = \sum_{i=0}^N w_i^2 r_i^2 \quad (4.20)$$

as proposed in [LF05], and define the weight w_i as

$$w_i = \frac{\rho_{Tuk}(r_i)^{\frac{1}{2}}}{r_i}, \quad (4.21)$$

to get (4.18). This weight is calculated at the beginning of each optimization cycle and treated as a constant during optimization. This way, we have a robust energy function that can be minimized over the pose parameters as defined in (4.13).

For the Newton algorithm, the gradient and Hessian of E in (4.20) are needed. The weight w_i is constant during an optimization cycle, so it does not affect the derivatives. The gradient is calculated as

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \sum_{i=0}^N w_i^2 \frac{\partial r_i^2}{\partial \mathbf{x}_i} \frac{\partial \mathbf{x}_i}{\partial_C \mathbf{X}_i} \frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}}, \quad (4.22)$$

using the chain rule. With the definition of r_i in (4.15) and the relation from (4.11), the following partial derivatives can be calculated.

$$\frac{\partial r_i^2}{\partial \mathbf{x}_i} = [2(x_i - x'_i) \ 2(y_i - y'_i)] \quad (4.23)$$

$$\frac{\partial \mathbf{x}_i}{\partial_C \mathbf{X}_i} = \frac{1}{Z_i^2} \begin{bmatrix} Z_i f_x & 0 & -X_i f_x \\ 0 & Z_i f_y & -Y_i f_y \end{bmatrix} \quad (4.24)$$

Finally, the derivative $\frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}}$ can be calculated using the relation of transformation and variation from (4.7) as

$$\frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}} = {}_C \mathbf{R}_M [-[M \mathbf{X}_i]_{\times} \ \mathbf{I}]. \quad (4.25)$$

The Hessian of E results in

$$\begin{aligned} \frac{\partial^2 E}{\partial^2 \boldsymbol{\theta}} &= \sum_{i=0}^N w_i^2 \frac{\partial}{\partial \boldsymbol{\theta}} \left(\frac{\partial r_i^2}{\partial \mathbf{x}_i} \frac{\partial \mathbf{x}_i}{\partial_C \mathbf{X}_i} \frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}} \right) \\ &= \sum_{i=0}^N w_i^2 \left(\frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}} \right)^T \left(\frac{\partial \mathbf{x}_i}{\partial_C \mathbf{X}_i} \right)^T \frac{\partial^2 r_i^2}{\partial \mathbf{x}_i^2} \frac{\partial \mathbf{x}_i}{\partial_C \mathbf{X}_i} \frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}} \\ &\quad + \frac{\partial r_i^2}{\partial \mathbf{x}_i} \frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}} \otimes \frac{\partial^2 \mathbf{x}_i}{\partial_C \mathbf{X}_i^2} \otimes \frac{\partial_C \mathbf{X}_i}{\partial \boldsymbol{\theta}} + \frac{\partial r_i^2}{\partial \mathbf{x}_i} \frac{\partial \mathbf{x}_i}{\partial_C \mathbf{X}_i} \otimes \frac{\partial^2 {}_C \mathbf{X}_i}{\partial \boldsymbol{\theta}^2}. \end{aligned} \quad (4.26)$$

We aim to minimize the energy function E , so its first order derivative is equal to zero for the optimal pose $\boldsymbol{\theta}^*$. And, as the energy function describes the distances between feature points to be minimized, the first-order partial derivative of (4.23) should be close to zero. As a result, we can approximate the last two additive terms

in the Hessian to zero for a more efficient implementation. We then only require the second-order derivative

$$\frac{\partial^2 r_i^2}{\partial \mathbf{x}_i^2} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad (4.27)$$

to complete the calculations. This energy formulation enables us to estimate the 3D object pose from the 2D feature points of two consecutive frames using a known object model and a pose estimate for the first frame. The Hessian of the energy function further gives information on the uncertainty of the estimated pose, which is needed for integration into an extended Kalman filter.

4.4 Optimization

To minimize the energy function defined in the previous section, we estimate a variation vector and iteratively update the pose using Newton optimization. This approach was also applied in the existing tracker introduced in Sec. 2.2. For each iteration, the variation vector $\hat{\boldsymbol{\theta}}$ is calculated as follows

$$\hat{\boldsymbol{\theta}} = \left(-\mathbf{H} + \begin{bmatrix} \lambda_r \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \lambda_t \mathbf{I}_3 \end{bmatrix} \right)^{-1} \mathbf{g}, \quad (4.28)$$

where \mathbf{H} and \mathbf{g} are the Hessian and gradient defined in the previous section, and λ_r and λ_t are the Tikhonov regularization parameters for rotation and translation respectively. Using Tikhonov regularization damps the optimization in directions with little information. Setting λ_r and λ_t to different values balances out the difference in magnitude between rotation and translation.

Using the exponential map formulation defined in Sec. 4.1.1, the pose can be updated as

$${}^c\mathbf{T}_M = {}^c\mathbf{T}_M \begin{bmatrix} \exp([\hat{\boldsymbol{\theta}}_r]_{\times}) & \hat{\boldsymbol{\theta}}_t \\ \mathbf{0} & 1 \end{bmatrix}. \quad (4.29)$$

The exponential map naturally results in an orthonormal rotation matrix, so no orthonormalization is needed. By iteratively updating this process, we can now estimate the optimal pose.

Chapter 5

Sparse feature map

The tracking framework presented in Chapter 4 takes a preceding pose and preceding camera frame as a reference and computes a relative pose change from frame to frame. Like other relative tracking algorithms, this approach is prone to drift when small estimation errors propagate throughout the tracking sequence. To minimize this behavior, we introduce a novel sparse feature map formulation. The sparse feature map includes relevant feature points that describe a specific object from different views. These points are associated with 3D points on the object surface and are optimized in relation to each other to provide a reliable reference for succeeding tracking procedures. A feature map enables tracking the object directly with reference to the map instead of relying on estimations from preceding frames. Generating a sparse feature map consists of two steps: Data collection and point optimization. Our formulation collects relevant data during one or several regular tracking sequences, which eliminates the need for specialized processes or equipment. The following chapter will explain the complete pipeline to generate and use a sparse feature map. First, we will present how relevant data is collected during tracking. Then, we will go through the point optimization steps and finally explain how this map is utilized during new tracking procedures.

5.1 Data collection

During a regular object tracking sequence, feature points are detected and used to compute an object pose. This association between feature points and object pose is the basis for a sparse feature map. Thus, we extend the tracking algorithm to utilize this data for map generation without needing a separate process. Ideally, the data should represent the object from all possible sides. This section describes the mapping of the poses on a geodesic grid and how it is used during tracking.

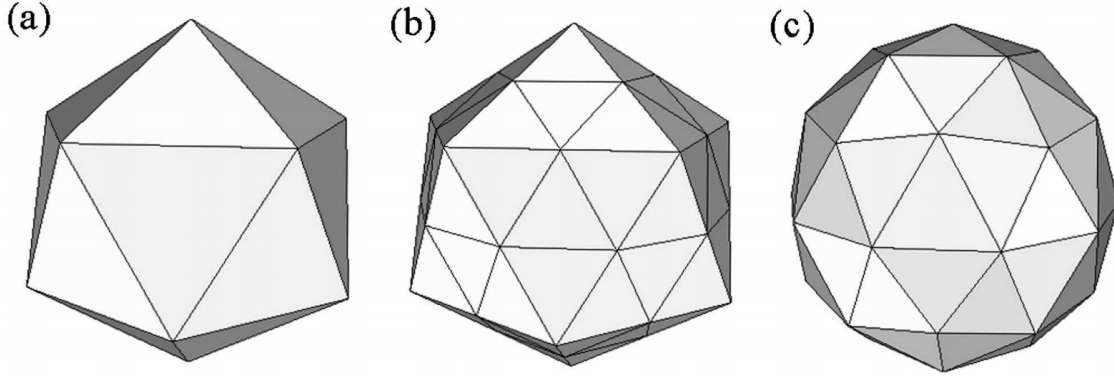


Figure 5.1: Generating a geodesic grid by recursive bisection [LR05]

5.1.1 Geodesic grid

To have a spatial reference of the object pose, we define fixed viewpoints around the model on equally spaced vertices of a geodesic grid which approximates a sphere with a specified radius. These viewpoints will be referred to as key views in the next sections. To create the geodesic grid, we start with an icosahedron, which is a polyhedron with 20 equilateral triangular faces. By bisecting each face into four more triangles, we get a geodesic grid with more faces. This bisection is repeated until the desired number of faces is achieved [SAM68]. An example of this is shown in Fig. 5.1.

Having created the desired grid, we use each of its vertices to represent an orientation in the frame of reference of a camera placed on that vertex. The camera's coordinate frame is defined at its focal point with the z axis pointing to the center of the sphere, the origin of the model coordinate frame. With this definition, we have a reference grid that represents all orientations around the object. Notice that in our tracking application, we have a fixed camera and a moving object, not the opposite. However, as we only consider the relative orientation between the model and the camera, the presented geodesic grid formulation is mathematically equivalent. It is also not affected by in-plane rotations of the object.

5.1.2 Grid matching

During a tracking process, we aim to collect relevant information from different views to utilize for map generation. In an ideal tracking scenario, we would view the object from all possible key views and have data for all of the model poses defined as vertices on the geodesic grid. To assign the data for each vertex, we compare the orientation of the estimated model pose to the orientations on the grid. The orientation of a pose is defined as

$$\mathbf{P}_{orientation} = {}_M\mathbf{R}_{CC}\mathbf{T}_M. \quad (5.1)$$

${}_M\mathbf{R}_C$ and ${}_C\mathbf{T}_M$ represent the inverse rotation and translation components of the pose ${}_C\mathbf{D}_M$, respectively. The orientation represents the rotated translation vector. To measure the distance, in this case the angle difference, between two orientation vectors, the dot product is used. With this definition, we assign the estimated pose to the closest vertex in terms of orientation. For this vertex, we then store the keypoints and the descriptors collected from the associated frame, along with the estimated model pose. These associations are then used to generate a sparse feature map in the point optimization step.

5.2 Point optimization

The data collected during tracking is based on single estimations from frame to frame. We are given a set of different viewpoints on the geodesic grid, observed feature points, and their estimated 3D reconstructions. A lot of the feature points represent the same 3D point from different views. The first step in optimizing the map points is finding such associations. Then, we optimize the estimated poses and estimated model points in relation to each other with bundle adjustment and, finally, fit them to the available object model using ICP. Ideally, all views around the object are traversed during tracking to collect information from each view. However, in realistic tracking scenarios, only a portion of the poses defined along the geodesic grid is viewed. Therefore, we include a final step in this algorithm to project the 3D points into untracked directions. This section will first present the point matching step across frames, followed by bundle adjustment and ICP, and finally, the point projection step.

5.2.1 Point clustering

Finding associations between feature points in the different views is a necessary requirement for bundle adjustment. This step aims to find groups of equivalent feature points and link them to a single 3D point on the object model. To achieve this, we first match the feature descriptors across all views and then combine them into groups by solving a maximum clique problem.

Key view matching Matching the descriptors of the key views on the grid with Hamming distance as described in Sec. 3.1.1 gives us an undirected neighbor graph of the different feature points. We denote this graph $G = (V, E)$, with V being the vertex set of G , in our case representing the set of keypoints, and E the edge set of G . An edge exists between two feature points if their descriptors match to a sufficient degree. The aim is to search this graph and find groups of points where all members are neighbors. This mathematical problem is known as the maximum clique problem [CP90, PX94, 2].

Finding maximum cliques In maximum clique problems, the goal is to find maximum complete subgraphs in which all member vertices are pairwise adjacent. The largest possible complete subgraphs are called maximum cliques. This problem is solved by iteratively increasing the clique sizes until they become maximal. A single vertex, by definition, constitutes a clique. We then consider one neighboring vertex. As they are adjacent, this second vertex is added to the clique. Then, we search along the graph; if a vertex is adjacent to *all* the existing clique members, it gets added to it. This process is repeated until no new vertex fulfills the complete subgraph condition. A pseudocode example of this algorithm is below. To remove trivial cliques, we include a filtering step to exclude cliques that are smaller than a certain threshold.

Algorithm 1: Find maximum cliques

Input : Undirected graph G
Output: Maximum cliques $C_{i..n}$
for $i \leftarrow 0$ **to** $size(V)$ **do**
 Add V_i to C_i ;
 for $j \leftarrow 0$ **to** $size(V)$ **do**
 if $C_i \cup V_j$ *is clique* **then**
 Add V_j to C_i ;
 end
 end
end

Quasi-clique expansion Finding pure maximum cliques is a very strong condition for our application. It eliminates descriptors that do not fulfill the complete subgraph condition, even if they have partial matches to the majority of members in the clique. This results in a representation that is not dense enough and can lead to the formation of different cliques that essentially describe the same feature. To mitigate this, we expand the definition to quasi-cliques [BHB07], which are *almost* complete subgraphs. For each clique, we assign a new member, if it is adjacent with a minimum of n members. This step is repeated until no new vertices fulfill this condition. A pseudocode example of this method is in Algorithm 2.

As a final step, we exclude duplicate cliques and obtain the final set of cliques. Each clique is then assigned a 3D point computed as the mean of the 3D reconstructions of the points in each clique. These points and their projections in the different views are used as initialization for bundle adjustment.

5.2.2 Bundle adjustment

Bundle adjustment is essentially a large parameter estimation problem. We denote the set of estimated 3D points obtained from the matching step as $\mathbf{X}_{i,i=1\dots n}$. The

Algorithm 2: Expand to quasi-cliques

Input : Maximum cliques $C_{i..n}$, Discarded points V
Output: Quasi-cliques $Q_{i..n}$

```

while  $size(V)$  changing do
   $Q_{i..n} = C_{i..n}$  ;
  for  $i \leftarrow 0$  to  $size(V)$  do
    for  $j \leftarrow 0$  to  $size(C)$  do
      if  $V_i$  is neighbor with  $n$  members in  $C_j$  then
        Add  $V_i$  to  $Q_j$  ;
        Remove  $V_i$  from  $V$  ;
      end
    end
  end
end

```

set of estimated poses is denoted $\mathbf{D}^{j,j=1..m}$. The observation of a point \mathbf{X}_i with object pose \mathbf{D}^j is then the point \mathbf{x}_{ij} . We want to minimize the projection error of the 3D points \mathbf{X}_i and the observed points \mathbf{x}_{ij} in all views. This can be written as

$$\min_{\mathbf{X}_i, \mathbf{D}^j} \sum_{i=1}^n \sum_{j=1}^m \rho_{Tukey}(\mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij}), \quad (5.2)$$

$$\hat{\mathbf{x}}_{ij} = \pi({}_C \mathbf{R}_{MM}^i \mathbf{X}_i + {}_C \mathbf{T}_M^i), \quad (5.3)$$

where n is the number of 3D points and m is the number of frames. $\hat{\mathbf{x}}_{ij}$ is the projection of a 3D point ${}_M \mathbf{X}_i$ into a view with pose ${}_C \mathbf{D}_M^i$. To increase robustness against possible false associations, we include the Tukey norm ρ_{Tukey} , introduced in Sec. 4.3. Minimizing this error starts from given initial parameter estimates. We use the reconstructed mean points to initialize \mathbf{X}_i and the computed poses during tracking the object as initial poses for \mathbf{D}^j . By minimizing the nonlinear least squares error function, we obtain the refined 3D points and poses. This can be done by applying trust region methods, such as Levenberg-Marquardt, or line search methods, such as Gauss-Newton optimization. As this problem is usually ill-posed due to the large number of unknown parameters, we apply Gauss-Newton to optimize it. It has been proven to converge better than trust region methods for such problems [SDS09].

5.2.3 Iterative Closest Point

Because bundle adjustment does a relative optimization, there is no absolute reference for size or scale. This may lead to shifting of the points from their original location during the optimization. To rectify this behavior and align the computed

3D points \mathbf{X}_i with the known object model points denoted $\mathbf{Q}_{k,j=1\dots p}$, we follow bundle adjustment with an ICP step. Traditionally, the ICP algorithm finds a single transformation matrix to minimize the distance between both point clouds. In our formulation, we additionally include a scaling factor. ICP is an iterative approach that alternates between the following two steps.

- Correspondence step: Find the closest model point for each 3D point \mathbf{X}_i

$$\mathbf{Q}_{corr,i} = \underset{\mathbf{Q}}{\operatorname{argmin}} \|\mathbf{R}s\mathbf{X}_i + \mathbf{T} - \mathbf{Q}\| \quad (5.4)$$

according to some rotation matrix \mathbf{R} and translation vector \mathbf{T} describing a transformation matrix \mathbf{D} and scaling term s . In the first step, these parameters are initialized with \mathbf{I} and 1 respectively.

- Alignment step: Update $\mathbf{D}(\mathbf{R}, \mathbf{T})$ and s to minimize the distance between both point clouds

$$(\mathbf{D}, s) = \underset{\mathbf{D}, s}{\operatorname{argmin}} \sum_{i=1}^n \rho_{Tuk}(\mathbf{R}s\mathbf{X}_i + \mathbf{T} - \mathbf{Q}_{corr,i}). \quad (5.5)$$

The alignment step is solved using Gauss-Newton optimization, analogous to bundle adjustment. Both steps are repeated until convergence. By applying the scale and transformation to the 3D points, we obtain 3D points that are optimized with regard to the different poses and aligned with the model.

5.2.4 Point projection

As a final step in the map generation algorithm, we verify if the optimized 3D points are also visible from other views that were not traversed during the tracking sequence. Likely, points from one view on the geodesic grid are also visible from neighboring views. To demonstrate the point projection method, we consider a candidate point \mathbf{X}_i and a view represented by a pose \mathbf{D}^j . The pose \mathbf{D}^j was not traversed on the geodesic grid, and we validate if point \mathbf{X}_i is visible from this pose by comparing the actual and expected depth values. The actual depth is calculated by transforming the 3D point into the camera frame of reference according to the view in question, as ${}_C\mathbf{D}_M^j\mathbf{X}_i$. The expected depth value is calculated by reconstructing the image point with the view pose and a depth renderer, as $\pi^{-1}(\mathbf{x}_i, d)$. If both depth values are close enough to each other, indicating that the point is on the visible surface of the object in this view, it is assigned to it. With this method, we eliminate the need for a comprehensive tracking sequence. Important features on the object have to be viewed during tracking, but not all poses on the geodesic grid need to be traversed.

With this formulation, we get a sparse feature map that relates 3D points on the model surface to their projections in different views along with their descriptors. This map can then be used as a reference during tracking.

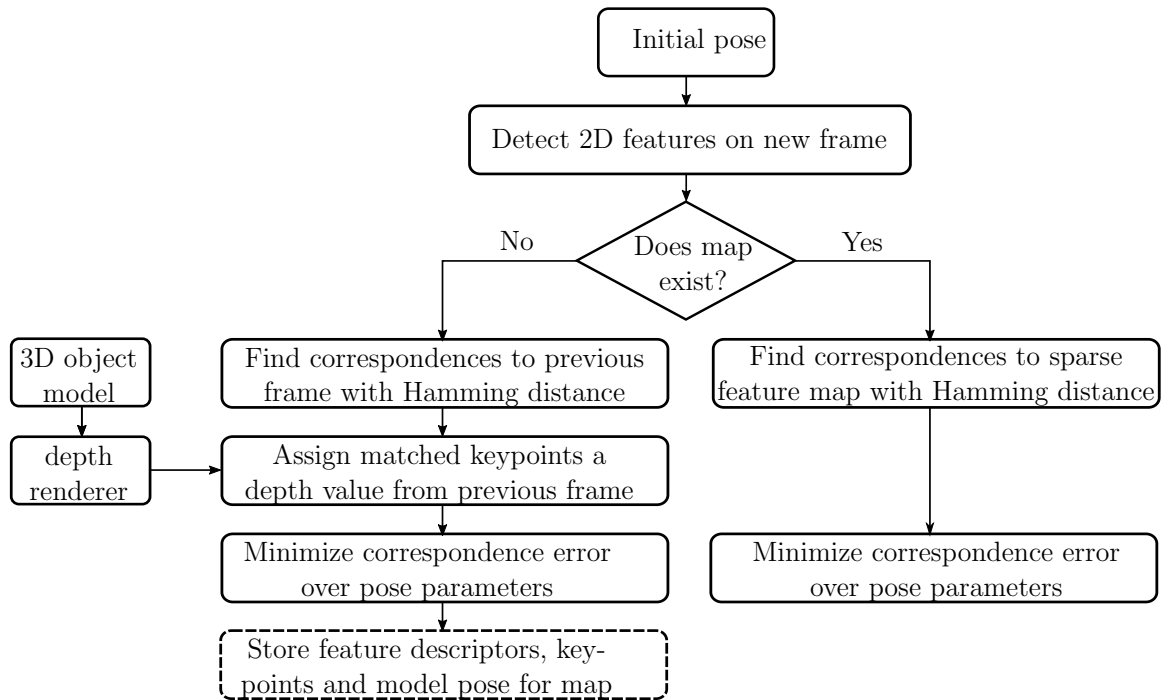


Figure 5.2: Flowchart of the tracker. If a sparse feature map has been previously generated, it will be used as a reference for tracking.

5.3 Integration into the tracker

Using the sparse feature map during tracking naturally extends the existing tracking procedure. Fig. 5.2 shows a flowchart of the complete tracking algorithm. If a sparse feature map has been previously generated, it is used as a reference for tracking. Instead of finding correspondences to the preceding frame and then reconstructing the points, the correspondences can be found directly to the closest view on the map, and the 3D points are directly known. If a map is not available yet, the tracking is done as presented in the previous chapter with an optional step of storing the relevant data to generate a sparse feature map.

Chapter 6

Implementation

Having explained the theoretical background of the algorithm in the previous chapters, we can now provide details on the implementation. The algorithm is programmed in *C++* and relies on the libraries *Eigen* [GJ⁺10], *OpenCV* [Bra00], *GLFW* [GLF19] and *Ceres* [AMO10]. In the following chapter, we first present the implementation measures utilized to improve the computation speed during tracking. These apply to the different methods in the feature-based tracking modality. Then, we explain two different methods used to handle possible occlusions by known or unknown objects during a tracking procedure. Finally, we define the user parameters that influence the tracking performance and give their default values that are used in the evaluation.

6.1 Efficiency optimization

One of the key requirements for this tracker is the real-time pose detection constraint, as presented in Sec. 2.2. For smooth tracking, the detection cycle has to be much faster than 33 ms. The standard implementation for the different components already fulfills the minimum requirement. However, optimizing the run time further enables the use of higher camera frequencies, multiple object tracking, and the implementation of more complex control cycles. This section presents the optimization measures that were applied for the individual steps in the feature-point-based tracking cycle.

Feature detection In the standard form of feature detection, the algorithm scans the entire camera frame for features that are then filtered in later steps according to their position and relevancy. To shorten the detection time and to only get relevant features, we limit the search area. Using the pose estimate from a preceding iteration and an estimate of the object diameter, we project a bounding box of the object into the camera frame and define a region of interest around the object's projection. With this method, the detection is limited to the pixels in the region of interest.

In addition to performing this *crop* operation, we also include a *scale* step. During tracking, the object can be closer or further away from the camera. Because the detection algorithm processes each pixel in the image, it will have higher computation times for close objects that fill many pixels. By scaling the cropped image according to the estimated depth of the object, the region of interest of an object very close to the camera will have a similar size to one of a further object. This operation minimizes the number of pixels to be processed and keeps the computation time constant for different objects and frames.

Feature matching Binary descriptors are traditionally matched with brute force: Each descriptor from one camera frame is matched against all descriptors from the other frame using the Hamming distance, as described in Sec. 4.2. In our tracking application, we have a relatively small motion from frame to frame, because we consider a continuous stream of camera frames during a motion. We thus expect the matched keypoints to be spatially close in the image frame of reference. To optimize the feature matching, we tried using only spatially close points as matching candidates instead of brute force, by only searching within a certain region of a feature for its match. However, determining the set of spatially close points requires some computation time, and the overall method does not provide a significant improvement. Performing the efficient Hamming distance calculations in brute force has comparable computation time to finding spatial comparisons between pixels and computing a decreased number of descriptor matches.

Depth rendering In depth rendering, the computation bottleneck often lies in copying the rendered pixel data from the Graphical Processing Unit (GPU) back to the Central Processing Unit (CPU). To minimize the number of superfluous pixels, we employ the same method used to improve feature detection and define a region of interest around the object that is scaled according to the object's depth. We adjust the depth image dimensions for the renderer and adapt the intrinsic parameters (focal length and principal point) to achieve the desired view and only render the depth values for the defined region of interest.

Point reconstruction The point reconstruction step involves computing the 3D reconstructions of the image feature points using a rendered depth image. By switching the order of computation and only reconstructing the image points that were successfully matched instead of including a reconstruction step over all feature points, the number of operations is significantly reduced.

By considering the computational complexities of the individual tracking steps and optimizing their implementation as presented in this section, the total tracking cycle computation time was decreased by 39%.

6.2 Occlusion handling

One of the tracker requirements presented in Sec. 2.2 is occlusion handling. The tracker should be able to handle partial occlusions to the object while maintaining accuracy. While the general formulation of the tracker is robust to partial occlusions to some extent, the performance can be further enhanced by including a dedicated occlusion handling step. Occlusion handling can be done either explicitly or implicitly. The choice of the method depends on the availability of data and tools. In this section, we will present both methods.

6.2.1 Implicit occlusion handling

Implicit occlusion handling requires the availability of depth information, such as from a depth camera. To detect occlusions implicitly, the algorithm compares the difference in expected and actual depth values of the points on the object. The expected depth value can be obtained in the point reconstruction step of the feature-based tracking modality. Each detected feature point is used to reconstruct a model point on the object's surface. Its depth is the expected depth value for this point. The actual depth value is the depth provided by the camera in the coordinates of the feature point. If the depth value of one model point is greater than the measured depth by a certain threshold, then this implies that another object is closer to the camera, and this point is likely occluded. In this case, it is excluded from further computations. This comparison is performed on all pixels within a certain radius to reduce the effect of possible faulty measurements. With this formulation, depth values from a camera are used to detect and handle occlusions.

6.2.2 Explicit occlusion handling

Another method to handle occlusions is explicit occlusion handling. In contrast to the first method, it does not require a depth camera but assumes the potentially occluding objects to be known. This is typically provided when multiple objects are tracked simultaneously. Occlusions are handled in this case by analyzing the estimated depth values of the objects in relation to each other. Model points on an object that are estimated to be further away from the camera than points of a different object occupying the same space are considered occluded and are excluded from the further computation. Like the implicit occlusion handling formulation, this comparison is performed on all pixels within a certain radius. Explicit occlusion is useful for multi-object tracking as mutual occlusions are likely in this case. It provides an alternative to implicit handling if a depth camera is not available.

6.3 Parameter tuning

The formulation of the tracker with the feature-based tracking modality enables setting different parameters that influence the tracking performance. These are thresholds, maximum numbers, or weighing factors that are used within the different methods. In this section, we will briefly explain these parameters and give their default values. The default values are presented in Table 6.1 in a compact form.

Correspondence search

- Number of features: During feature detection, all the points that fulfill the FAST detector requirement are computed, and then, according to the ORB formulation, the n best features are retained. We set this value to 500, meaning that a maximum of 500 feature points are used in each frame. Setting a maximum number of features improves the quality of the detector.
- Scale factor: The ORB detector utilizes a scale pyramid to compute features in different image scales. The scale factor defines the ratio of the scale pyramid levels. We set it to 1.2, which means that each scale level has $(1.2)^2$ fewer pixels than the preceding level. This number provides a good trade-off of covering a certain scale range without having big jumps in the scale pyramid.
- Number of levels: This defines the number of levels in the ORB scale pyramid. The default value is 8 levels.

Correspondence matching

- Lowe's ratio threshold: This threshold is connected to Lowe's ratio test [Low04]. In the original formulation, it was set to 0.7, and we adopt the same value. The distance of a match has to be smaller than 0.7 times the distance to the second closest match.
- Lowe's ratio threshold (map): When tracking with a sparse feature map, the risk of matching with an outlier is greatly minimized, so we increase this threshold to 0.8 to allow for more matches with the map.

Occlusion handling

- Occlusion handling threshold: If the distance between depth values from two measurements is within this threshold, then the point is considered occluded. This value was chosen to be 0.05 m.
- Occlusion handling radius: This radius defines the region around a pixel in which occlusion comparisons are considered. It is set to 9 pixels.

| Tracker step | Parameter | Value |
|-------------------------|------------------------------|--------|
| Correspondence search | Number of features | 500 |
| | Scale factor | 1.2 |
| | Number of levels | 8 |
| Correspondence matching | Lowe's ratio threshold | 0.7 |
| | Lowe's ratio threshold (map) | 0.8 |
| Occlusion handling | Occlusion handling threshold | 0.05 m |
| | Occlusion handling radius | 9 |
| Optimization | Tukey norm constant | 12 |
| | Texture modality weight | 0.4 |
| | Region modality weight | 0.6 |

Table 6.1: Default values for the tracker parameters

Optimization

- Tukey norm constant: The constant c is part of the Tukey norm formulation. For the tracker, we set this value to 12.
- Texture modality weight: The texture modality weight defines how much the gradient and Hessian for the pose estimate from the texture modality influence the update for the next pose computation. We set this value to 0.4.
- Region modality weight: The region modality weight is set to 1 minus the texture modality weight.

Chapter 7

Evaluation

Having now explained the framework and presented the implementation details, in this chapter, we provide a comprehensive evaluation and present the results for the different contributions of this work. In the first section, we evaluate the tracker on the Region-Based Object Tracking (RBOT) dataset. We show that the multi-modality tracker consisting of the region-based tracking modality from [SPS⁺20] and the proposed feature-based modality performs better than the state-of-the-art tracking approaches tested on this dataset. We then perform a similar evaluation on the Yale-CMU-Berkeley (YCB) video dataset. With this, we assess the performance of the feature-based tracking modality when added to the region-based tracker on real-world data. To test the effectiveness of adding a sparse feature map, we perform an evaluation on a self-recorded sequence and present its results in the final section.

7.1 Evaluation on the RBOT dataset

The main part of the evaluation is done on the RBOT dataset. In this section, we show that the combined method of a region and a texture modality outperforms the current state of the art algorithms tested on this dataset. In the following, we describe the design of the dataset and the used error metric, then present the results of our proposed method and the state of the art.

7.1.1 Design

The RBOT dataset [TSSC18] consists of eighteen objects that are shown in Fig. 7.1. Of these objects, five are from the Rigid Pose Tracking dataset [PRDR13], twelve are from the LINE-MOD dataset [HLI⁺12] and one was created for RBOT. These simulated objects are placed on a real-world background that depicts a cluttered desk scene. Two examples are shown in Fig. 7.2.

For each object, the dataset provides four sequences with different qualities. The *regular* sequence shows the object moving along the background scene with a fixed light source. The *dynamic* sequence shows a similar view but with a moving light



Figure 7.1: Overview of the objects used in the RBOT dataset [TSSC18]. The textured objects marked with \blacksquare are from the Rigid Pose Tracking dataset [PRDR13] and the objects marked with \blacklozenge are from the LINE-MOD dataset [HLI⁺12]. The object marked with \star is an addition from RBOT.

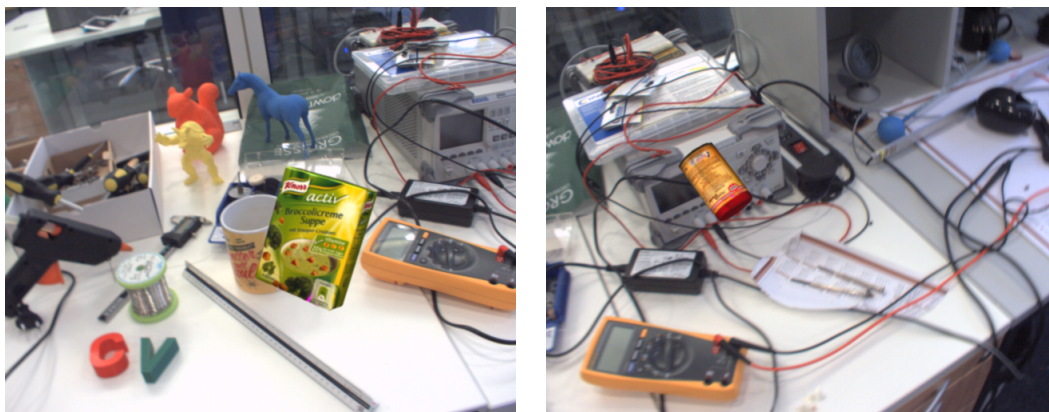


Figure 7.2: Examples from the RBOT dataset [TSSC18]

source. Then, there is a *noisy* sequence, which has artificial Gaussian noise added to it and a dynamic light source. And finally, there is an *occlusion* sequence that includes a second object (the squirrel) with a trajectory that overlaps and occludes the main object. During each sequence, all objects follow the same trajectory. The occluding object has a different trajectory moving around the main object. Each sequence consists of 1001 RGB frames with a resolution of 640×512 px. The dataset provides 3D models for each object, and the ground truth poses for all frames as a rotation matrix and translation vector.

To evaluate our method, we use the same evaluation metric that was proposed in [TSSC18] for this dataset and followed in the remaining publications [TSSC18, SPS⁺20, 2]. For each frame during tracking at time t_k , the translation error is computed as

$$e_{\mathcal{T}}(t_k) = \|\mathbf{T}(t_k) - \mathbf{T}_{gt}(t_k)\|_2, \quad (7.1)$$

the distance between the the estimated translation $\mathbf{T}(t_k)$ and ground truth $\mathbf{T}_{gt}(t_k)$. The rotation error is computed as

$$e_{\mathbf{R}}(t_k) = \cos^{-1} \left(\frac{\text{trace}(\mathbf{R}^T(t_k)\mathbf{R}_{gt}(t_k)) - 1}{2} \right), \quad (7.2)$$

with the estimated rotation $\mathbf{R}^T(t_k)$ and ground truth $\mathbf{R}_{gt}(t_k)$. A pose estimation is considered successful, if $e_{\mathbf{T}}(t_k)$ is below 5 cm and $e_{\mathbf{R}}(t_k)$ is below 5° .

In each tracking sequence, the tracker is initialized with the ground truth pose of the first frame. If the estimated pose does not fulfill the above error requirements during tracking, the tracking is considered lost and is reset with the ground truth pose of that frame. The percentage of successful pose estimations in one sequence constitutes its success rate. For tracking the *occlusion* sequence, two cases are defined. In unmodeled occlusion, the occluding object is not considered separately, and the tracker is configured the same way as for the other sequences. In the case of modeled occlusion, the main object and the occluding object are tracked to possibly achieve a higher success rate.

7.1.2 Results

We evaluate our combined tracker on the complete RBOT dataset and compare the results to the current state of the art approaches. The method in [TSSC18] is a region-based method from the creators of the RBOT dataset. The methods [ZZZ⁺20] and [SPS⁺20] are also region-based. The latter is the method integrated into our combined tracker. We further include the results from [HZZSQ20], [LZXQ21] and [SZZ⁺21], which use a combined method of region and edge information, and finally [LWZ21], which is based on region information and feature descriptor fields. The results are shown in Table 7.1. The table shows the success rates of our proposed method on all sequences, along with the success rates of the state of the art methods. Our proposed method outperforms the state of the art in most cases by a large margin and achieves the highest average for all sequences.

To further underline the added value of the texture-based modality, we color the textured objects *Baking soda*, *Broccoli soup*, *Cube*, *Glue*, and *Koala candy* in red and compute a separate average for these objects. The use of texture information considerably increases the success rate when compared to the other methods. The average success rate over these objects is around 10% higher than the second-best method for most sequences. This improvement is especially significant for objects with rotational ambiguities, such as *Baking soda* and the *Koala candy*. For these objects, the improvement in success rate reaches 18%.

The evaluation was conducted on a laptop with an Intel i7-8500Y dual-core processor running at 1.6 GHz. The average execution time for all sequences is 21 ms, when tracking a single object. The modeled occlusion evaluation runs at 65 ms, because it requires tracking two objects. We evaluated [SPS⁺20] on the same laptop and measured runtimes of 6.2 ms for the single-object sequences and 33 ms for the modeled

occlusion sequences. The authors in [ZZZ⁺20], [SZZ⁺21] and [LZXQ21] performed their evaluation on a quad-core processor and reported runtimes between 32 ms and 50 ms. The method [LWZ21] achieves a runtime of 10 ms on a six-core processor.

7.2 Evaluation on the YCB video dataset

While our proposed tracker shows significant results on the semi-synthetic RBOT dataset, here, we also perform an evaluation on real-world data from the YCB video dataset and compare the performance of our proposed algorithm with the region-based state of the art approach in [SPS⁺20]. We first introduce the dataset and describe the design of the evaluation, then present the results.


| | ID | Name |
|--|-----|-------------------|
|  | 002 | Master chef can |
| | 003 | Cracker box |
| | 004 | Sugar box |
| | 005 | Tomato soup can |
| | 006 | Mustard bottle |
| | 007 | Tuna fish can |
| | 008 | Pudding box |
| | 009 | Gelatin box |
| | 010 | Potted meat can |
| | 011 | Banana |
| | 019 | Pitcher base |
| | 021 | Bleach cleanser |
| | 024 | Bowl |
| | 025 | Mug |
| | 035 | Power drill |
| | 036 | Wood block |
| | 037 | Scissors |
| | 040 | Large marker |
| | 051 | Large clamp |
| | 052 | Extra large clamp |
| | 061 | Foam brick |

Figure 7.3: Objects used in the YCB video dataset [XSNF17]

Table 7.2: Object names

7.2.1 Design

The YCB video dataset was published in [XSNF17]. It uses 21 objects of the YCB dataset [CSW⁺15] shown in Fig. 7.3 and includes real-world tracking sequences of these objects along with their 3D models. The videos were taken with an RGBD camera with a resolution of 630×480 at 30 frames per second (fps). The dataset provides RGB images as well as depth images of 92 video sequences depicting different assortments of the object and recorded under different conditions (lighting,

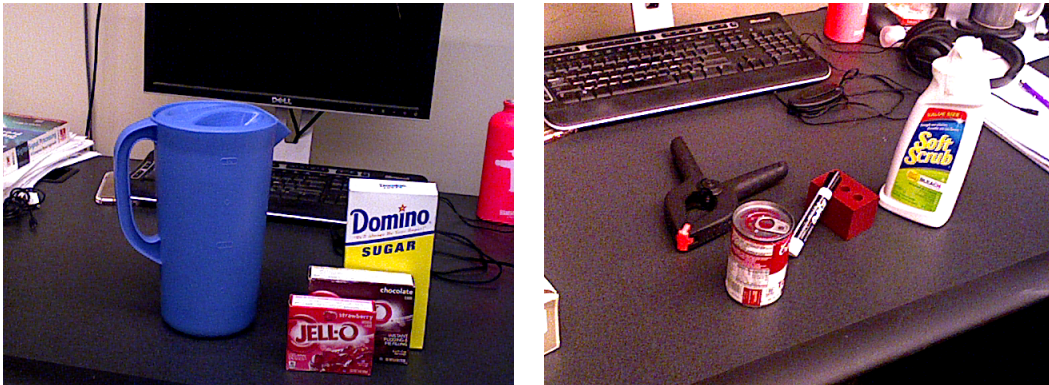


Figure 7.4: Examples from the YCB dataset [XSNF17]

noise, occlusion, ...). It was originally developed for deep learning approaches and its creators define 80 sequences for training and 12 for testing. Two examples from the dataset are shown in Fig. 7.4.

We use the 12 test sequences for this evaluation and compare our proposed method of the combined tracker with the state of the art region-based tracker from [SPS⁺20]. As the original publication only includes an RBOT evaluation, we conducted the evaluation on the YCB video dataset ourselves using the provided code. We ran both trackers for all objects in each of the tracking sequences. To quantify the results, we use the same error metric and success rate calculation used for the RBOT dataset and explained in Sec. 7.1.1. Like the RBOT dataset evaluation, a reset to the ground truth pose is made if the tracking is considered unsuccessful. The success rate is the percentage of successful pose estimations throughout a sequence.

7.2.2 Results

The results from the evaluation on the YCB video dataset are shown in Table 7.3. Each object appears in a subset of the test sequences. The table shows the average success rate for each object across the sequences. The reference between object ID and name is clarified in Table 7.2. The results show that the proposed combined tracker outperforms the region-based tracker from [SPS⁺20] for all objects by a substantial margin. The combined tracker achieves a success rate 20% higher than the performance of [SPS⁺20]. Especially in the presence of occlusion between objects, utilizing more information by combining different modalities increases the tracker's robustness. The tracker [SPS⁺20] achieves an average runtime of 5.2 ms and the combined tracker runs in 23 ms on the YCB dataset.

| Method | 002 | 003 | 004 | 005 | 006 | 007 | 008 | 009 | 010 | 011 | 019 |
|----------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| [SPS+20] | 76.2 | 72.2 | 82.9 | 76.2 | 77.3 | 76.9 | 76.4 | 77.0 | 75.3 | 75.5 | 76.6 |
| Proposed | 93.4 | 95.7 | 96.7 | 94.3 | 94.2 | 92.9 | 92.2 | 92.5 | 92.3 | 91.8 | 92.1 |
| Method | 021 | 024 | 025 | 035 | 036 | 037 | 040 | 051 | 052 | 061 | Avg. |
| [SPS+20] | 73.6 | 74.5 | 75.6 | 76.9 | 76.5 | 75.1 | 74.6 | 74.6 | 74.8 | 75.1 | 72.3 |
| Proposed | 89.1 | 89.6 | 89.8 | 90.5 | 90.5 | 90.1 | 87.5 | 87.5 | 87.7 | 87.8 | 91.3 |

Table 7.3: Tracking success rates on the YCB video dataset for all objects. The numbers 002-061 are IDs for each object in the dataset.

7.3 Evaluation on a recorded sequence

In the previous sections, we provided an extensive evaluation of the texture-based modality in the combined tracker. In this section, we evaluate the sparse feature map presented in Chapter 5 and provide comparisons to the combined tracker and the region-based tracker of [SPS+20]. For this purpose, we have created a self-recorded sequence and defined an evaluation metric to quantify the performance regarding a ground truth. We first explain the design in more detail and then present the results.

7.3.1 Design

For the evaluation, we chose three objects of different shapes: Pringles, Calculator, and Chocolate. They can be seen in the example in Fig. 7.6. To track them, we use a *RealSense D435* camera [Int19]. It is an RGBD camera, but we only use the RGB outcome for our tracker. We use the camera with a frame rate of 30 Hz and a resolution of 640×480 . We first place the three objects on defined positions and record a sequence of them while moving the camera, as depicted in Fig. 7.6 showing frames 1, 160 and 371. The objects remain static during the recording.

To perform the evaluation and quantify the results, we first have to obtain a ground truth reference of the poses. The poses of the objects in relation to the camera in the different time steps are unknown, as we do not have access to the camera poses during its motion. However, as the objects remain fixed, their relative poses to each other are unchanged. We use this information to obtain a ground truth reference. We measure the relative distances between all three objects according to their placement, as shown in Fig. 7.5, and obtain the relative rotation from their geometry. Thus, we have three ground-truth poses for each pair of objects. To initialize the tracker, we experimentally obtain the object poses in the first frame and refine them with the tracker.

During tracking, we track all three objects simultaneously and estimate their poses as usual. Then, we compute the relative translation and rotation parameter for each pair as

$$\mathbf{T}_{12} = \mathbf{T}_2 - \mathbf{T}_1, \quad (7.3)$$

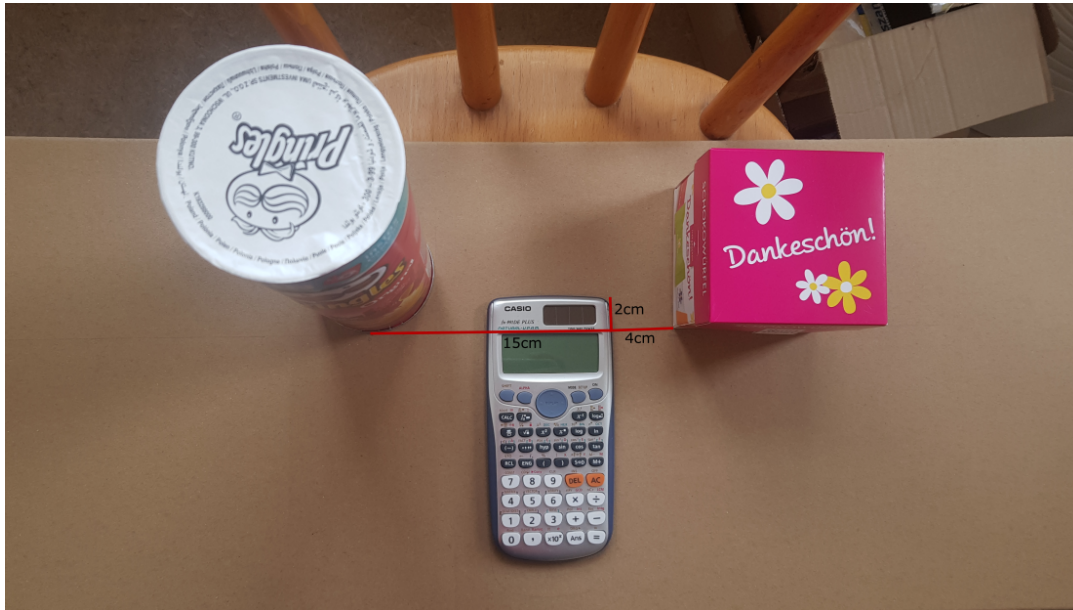


Figure 7.5: Setup of used objects and their ground truth distances



Figure 7.6: Examples from the self recorded sequence

$$\mathbf{R}_{12} = \mathbf{R}_1^T \mathbf{R}_2 \quad (7.4)$$

and compare these against the measured ground truth values. The translation error between the estimation and ground truth is calculated as

$$e_{\mathbf{T}}(t_k) = \|\mathbf{T}_{12}(t_k)\|_2 - \|\mathbf{T}_{12gt}(t_k)\|_2, \quad (7.5)$$

The rotation error is calculated with the metric presented in Sec. 7.1.1.

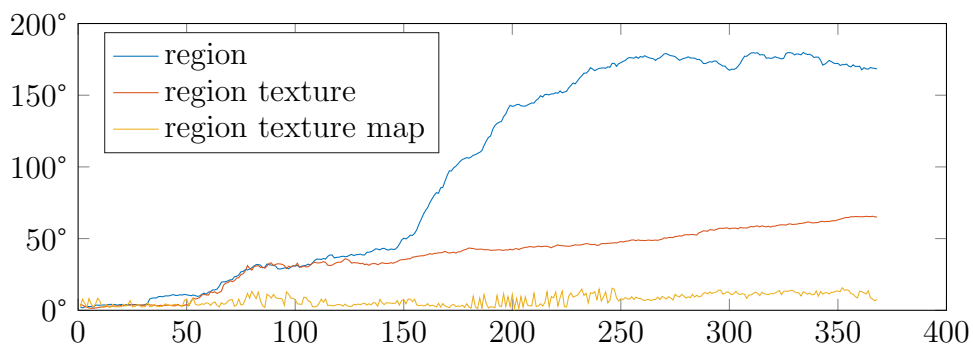
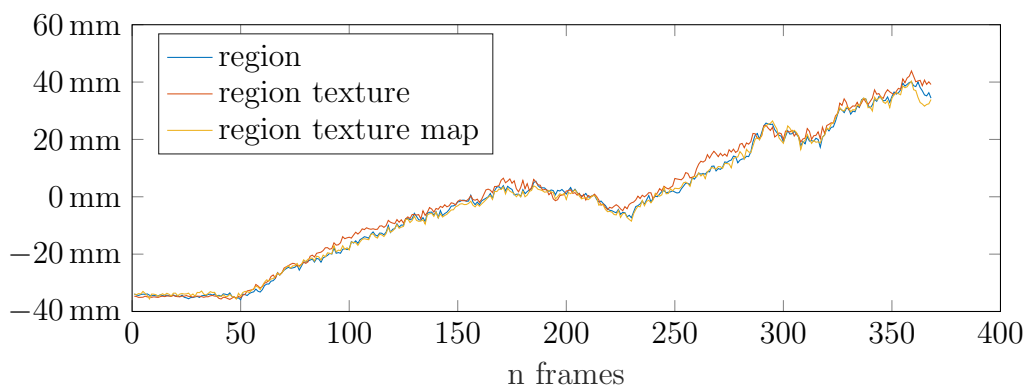
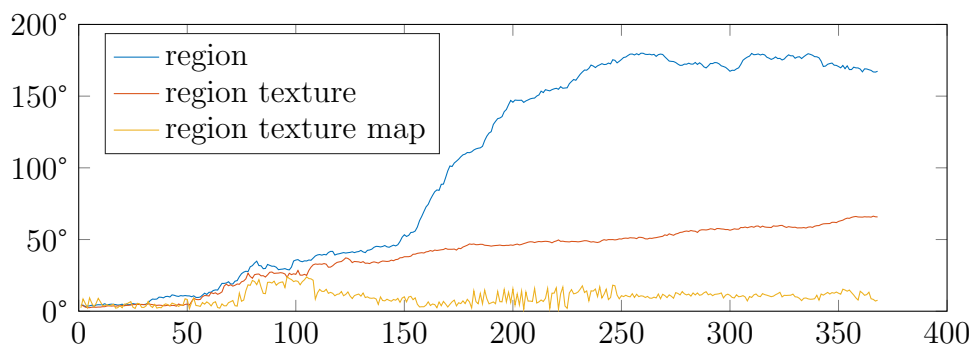
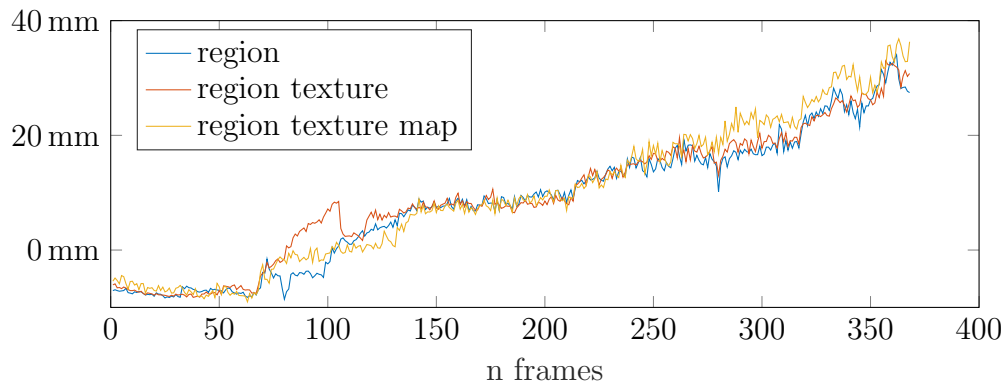
We evaluate the tracking performance on this sequence of the region-based tracker of [SPS+20], the combined tracker and the combined tracker using a sparse feature map. To obtain such a map, we take one recording for each of the objects that shows it from almost all sides and generate the map according to the algorithm described in Chapter 5. During the tracking sequence, the texture modality utilizes this map as previously described.

7.3.2 Results

In this section, we present the results on the recorded sequence for the combined tracker of region and texture modalities and the tracker when additionally utilizing a map. We also show the performance of the region-based tracker [SPS⁺20] on its own as a reference. The results are shown in Fig. 7.6. We plot the relative rotation and translation error of each object pair throughout the tracking sequence. In the errors related to the *Pringles* object, the region-based tracker suffers from a high rotational drift. This is due to the rotationally symmetrical characteristics of the *Pringles* object, which results in pose ambiguities when only the silhouette of the object is used. Adding the texture-based modality to the tracker significantly limits this drift because texture information is utilized to infer the object’s orientation. The yellow line shows the error result when the texture-based modality relies on a sparse feature map. The drift is further minimized because using the map allows correcting the pose in any time step regardless of the preceding estimation.

The plot showing the relative error between the *Calculator* and *Chocolate* objects does not show rotational drift for any method. The high errors around the 100th frame result from texture ambiguities in the *Chocolate* object. The same text written on the side and top of the object leads to false matches and an incorrect pose estimation. However, the tracker recovers, once more information is available.

For this experiment, we also measure the execution times. Tracking the three objects with the region-based tracker from [SPS⁺20] requires 79 ms on the laptop specified in Sec. 7.1.2. Running the combined tracker has a runtime of 168 ms for the three objects. Using the sparse feature map reduces this runtime to 140 ms.

Rotation error Pringles-Calculator**Translation error Pringles-Calculator****Rotation error Pringles-Chocolate****Translation error Pringles-Chocolate**

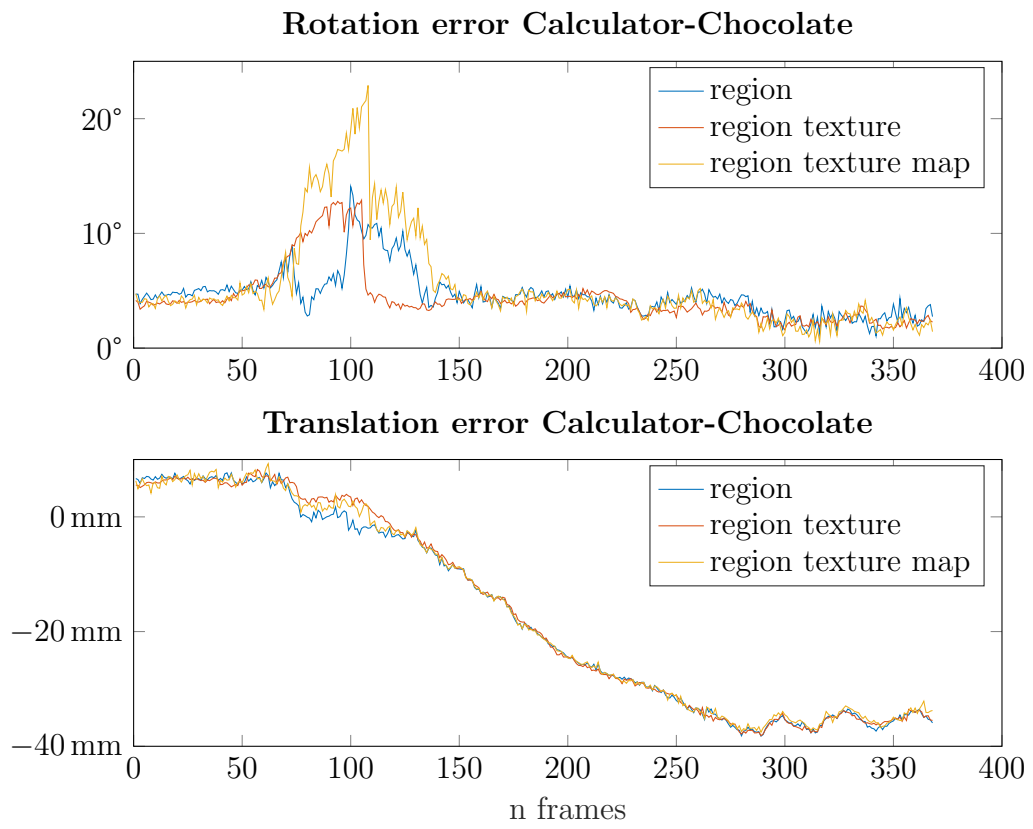


Figure 7.6: Rotation and translation error of the relative poses

Chapter 8

Discussion

In this thesis, we first developed a feature-based tracking modality that utilizes texture information and can be added to other modalities based on different approaches. The results show that the use of the texture modality significantly improves the tracking performance. Especially, the combination of a region modality and a texture modality, which was considered in this work, results in a strong tracker as both modalities utilize different types of information. While both methods provide estimations on the translation and rotation components of the pose, the region modality considers the silhouette of the object and maintains a robust pose estimate. The texture modality evaluates the texture information on the object surface to refine the estimation and provide additional value to the translation and rotation components. The result is a more accurate estimation with reduced drift.

The evaluation on the RBOT dataset shows that this combined tracker beats all of the state of the art approaches and provides the best success rates for this dataset. The evaluation on the YCB video dataset illustrates that the performance is also significantly improved on real-world data when comparing the combined tracker with a purely region-based tracker. This effect is further demonstrated in the evaluation of the *Pringles* object in the self-recorded sequence. By only relying on silhouette information, it is not possible to estimate the orientation of the object along the rotationally symmetric axis. These examples show the importance of considering texture information during tracking if the tracked objects provide such information. In addition, the texture modality does not present a disadvantage when tracking untextured objects. The evaluation on the RBOT datasets still shows superior results for such objects. Because of the gradient and Hessian formulation and the Gauss-Newton optimization, the texture and region modalities will not significantly affect the pose estimation if they do not have a strong certainty.

After establishing the impact of the texture-based modality, we developed the formulation of the sparse feature map to further improve the performance of the texture-based modality and reduce the execution time. Because both the RBOT and the YCB video datasets do not provide sequences that would be suitable for map generation, we designed the self-recorded sequence to demonstrate the value of the sparse

feature map. The results show that it decreases the error of the combined tracker further. The evaluation on the self-recorded sequence is not comprehensive, but it gives a good indication that the sparse feature map is an advantageous addition to the texture-based modality.

The presented sparse feature map formulation provides a superior automated alternative to traditional offline methods that utilize Structure from Motion (SfM) algorithms. These methods require human supervision to segment the keyframes and tune the scale of the object model, thus requiring a lot of manual work for each new object. Our formulation extracts the needed information automatically using a geodesic grid and calculated pose estimates during a regular tracking sequence. After clustering and optimizing the data, the scale and orientation of the sparse feature map are then refined in an ICP step, eliminating the need for human input. The evaluation of the complete framework has shown that it provides accurate tracking results. To conclude this chapter, we would like to discuss the performance of the algorithm with regards to the requirements of in-hand manipulation presented in Sec. 2.2. The first requirement is efficiency. The measured execution time on the dual-core laptop is around 21 ms. Although this measurement was not performed on a powerful processor, it still fulfills the minimum requirement and utilizes the full camera frequency. Running the algorithm on the processor of the robot system is expected to provide shorter execution times.

A further requirement is the ability to handle occlusions. This was tested on the occluded sequence from the RBOT dataset. The performance for the *unmodeled occlusion* sequence achieves the highest success rate when compared to the state of the art, showing inherent robustness of the region and texture modalities. Including explicit occlusion handling further improved the success rate in the *modeled occlusion* sequence. Lastly, considering the results in *dynamic light* shows that the framework is only slightly affected by illumination variance. These factors indicate that the presented algorithm fulfills the requirements for in-hand manipulation and is suitable for deployment on the robot system *David*.

Chapter 9

Conclusion and outlook

With this thesis, we provide a framework for a multi-modality tracker that extends a region-based method with a texture-based approach. The texture-based method can track directly from frame to frame or can rely on a sparse feature map. The data collection method, the generation, and use of the feature map during tracking were developed as part of this work. We have presented the methodology for this algorithm and conducted a threefold evaluation to assess the method's performance in different scenarios. In this final chapter, we would like to draw some conclusions from the results of this work, discuss some limitations, and give an outlook on possible future research.

The goal of a versatile tracker is to have a good performance on a broad range of objects. Each object has different properties that might pose challenges or advantages for specific tracking methods. To exploit the strengths of each method and have a versatile tracker, different sets of information have to be utilized. This work demonstrated that combining region and texture information achieves superior performance than a single type of information.

A map is often used to minimize drift in texture-based tracking approaches, which can be a textured 3D model from a 3D scan or a sparse map from a Structure from Motion algorithm. Both options require a special process and manual supervision for each new object. However, providing the framework of a sparse feature map that can be generated from regular tracking information simplifies the process and can be used without specialized equipment.

Another simplification in our proposed tracker is that it runs on a single CPU and only requires an RGB camera. This makes it easily accessible for use in different applications run on simple processors and handheld devices without additional financial cost.

One of the limitations of using a texture-based tracker is that the tracked objects have to be sufficiently textured. Depending on the application and used objects, this might be a strong requirement. This limitation further motivates the concept of a multi-modality tracker. One limitation in the map formulation is that the tracking sequence for the map should be created under *similar* conditions as the

remaining tracking instances. This is because the created descriptors can be biased towards the map sequence, which might negatively affect the tracking performance if a new sequence differs to a big extent. Lastly, a limitation that exists for different object tracking methods, is that the object motion from frame to frame needs to be limited to some extent. An object motion that is too fast might lead to inaccurate estimations or even tracking loss.

For future research, this formulation has the potential of integrating more modalities, such as edge-based methods, to handle cases with geometries within an object's silhouette. Additionally, depth information can be included as a depth-based modality by using a depth camera. The formulations of both the region and texture modalities are in RGB image space, leading to slightly inaccurate estimations along the z-Axis of the camera frame of reference. Adding depth information is expected to improve the tracking performance.

Moreover, as a future goal, this framework of a multi-modality tracker can be extended with an object detection method to initialize the tracker and re-detect the object in cases of tracking loss. Finally, the presented formulation was designed to be fused with kinematic information from a robot with an extended Kalman filter to further refine the pose estimation and can be run on the *DLR* robot system *David*. As a future experiment, it would be interesting to evaluate the tracker on the robot system *David* with the complete control cycle of in-hand manipulation.

List of Figures

| | | |
|-----|--|----|
| 1.1 | The <i>DLR</i> robot system <i>David</i> [GAB ⁺ 11] | 6 |
| 4.1 | The segment test of the FAST detector. The arc shows 12 consecutive pixels brighter than the interest point p . [RD06] | 21 |
| 4.2 | Example of feature matching from this work for the can object between two consecutive frames of the YCB dataset [XSNF17] | 22 |
| 4.3 | Tukey norm for $c = 3$ compared to $\frac{x^2}{2}$ | 23 |
| 5.1 | Generating a geodesic grid by recursive bisection [LR05] | 28 |
| 5.2 | Flowchart of the tracker. If a sparse feature map has been previously generated, it will be used as a reference for tracking. | 33 |
| 7.1 | Overview of the objects used in the RBOT dataset [TSSC18]. The textured objects marked with \blacksquare are from the Rigid Pose Tracking dataset [PRDR13] and the objects marked with \blacklozenge are from the LINE-MOD dataset [HLI ⁺ 12]. The object marked with \star is an addition from RBOT. | 42 |
| 7.2 | Examples from the RBOT dataset [TSSC18] | 42 |
| 7.3 | Objects used in the YCB video dataset [XSNF17] | 45 |
| 7.4 | Examples from the YCB dataset [XSNF17] | 46 |
| 7.5 | Setup of used objects and their ground truth distances | 48 |
| 7.6 | Examples from the self recorded sequence | 48 |
| 7.6 | Rotation and translation error of the relative poses | 51 |

List of Tables

| | | |
|-----|---|----|
| 6.1 | Default values for the tracker parameters | 39 |
| 7.1 | Tracking success rates on the RBOT dataset in percent of our proposed method and the state of the art. Best results are marked bold, and second-best values are underlined. Textured objects are colored red and Avg. t is the average of these objects. The value Avg. is the average success rate over all objects. | 44 |
| 7.2 | Object names | 45 |
| 7.3 | Tracking success rates on the YCB video dataset for all objects. The numbers 002-061 are IDs for each object in the dataset. | 47 |

Acronyms and Notations

2D Two Dimensional

3D Three Dimensional

BRIEF Binary Robust Independent Elementary Features

BRISK Binary Robust Invariant Scalable Keypoints

CPU Central Processing Unit

FAST Features from Accelerated Segment Test

fps frames per second

DLR German Aerospace Center

DoF Degrees of Freedom

GPU Graphical Processing Unit

ICP Iterative Closest Point

LINE LINEarizing the memory

LINE-MOD Multimodal-LINE

ORB Oriented FAST and Rotated BRIEF

P3P Perspective-3-Point

PnP Perspective-n-Point

RANSAC Random Sample Consensus

RBOT Region-Based Object Tracking

RGBD Red Green Blue Depth

RMC Robotics and Mechatronics Center

SfM Structure from Motion

SIFT Scale Invariant Feature Transform

SLAM Simultaneous Localization And Mapping

SURF Speeded Up Robust Features

YCB Yale-CMU-Berkeley

Bibliography

- [AFS⁺11] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building rome in a day. *Communications of the ACM*, 2011.
- [AMO10] S. Agarwal, K. Mierle, and Others. Ceres solver. <http://ceres-solver.org>, 2010. Online, accessed 24-August-2021.
- [BHB07] M. Brunato, H. H. Hoos, and R. Battiti. On effectively finding maximal quasi-cliques in graphs. *Learning and Intelligent Optimization, Second International Conference, LION*, 2007.
- [Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [BTVG06] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [bug]
- [CER⁺21] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, JosÃ© M. M. Montiel, and Juan D. Tardós. Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam. *IEEE Transactions on Robotics*, pages 1–17, 2021. doi:10.1109/TR0.2021.3075644.
- [CP90] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 1990.
- [CSW⁺15] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. *International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.
- [DMX⁺21] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox. Poserbpf: A rao-blackwellized particle filter for 6-d object pose tracking. *IEEE Transactions on Robotics*, pages 1–15, 2021. doi:10.1109/TR0.2021.3056043.

- [DRMS07] A. J. Davison, I. D Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. In *IEEE Transactions on Pattern Analysis And Machine Intelligence*, volume 29, 2007.
- [FB81] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981.
- [GAB⁺11] M. Grebenstein, A. Albu-Schäffer, T. Bahls, M. Chalon, O. Eiberger, W. Friedl, R. Gruber, S. Haddadin, U. Hagn, R. Haslinger, H. Höppner, S. Jörg, M. Nickl, A. Nothhelfer, F. Petit, J. Reill, N. Seitz, T. Wimböck, S. Wolf, T. Wüsthoff, and G. Hirzinger. The dlr hand arm system. In *IEEE International Conference on Robotics and Automation*, pages 3175–3182, 2011.
- [GC15] O. Guclu and A. B. Can. A comparison of feature detectors and descriptors in rgb-d slam methods. In M. Kamel and A. Campilho, editors, *Image Analysis and Recognition*, pages 297–305, Cham, 2015. Springer International Publishing.
- [GJ⁺10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. <http://eigen.tuxfamily.org>, 2010. Online, accessed 24-August-2021.
- [GLF19] GLFW. <https://www.glfw.org>, 2019. Online, accessed 24-August-2021.
- [Gru41] J. A. Grunert. Das pothenotische problem in erweiterter gestalt nebst über seine anwendungen in geodäsie. *Grunerts Archiv für Mathematik und Physik*, 1841.
- [Har93] R. Hartley. Euclidean reconstruction from uncalibrated views. *Applications of Invariance in Computer Vision*, 1993.
- [HLI⁺12] S. Hinterstoisser, V. Lepetit, S. Ilic, s. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. *Proceedings of the 12th international conference on Computer Vision*, 2012.
- [HZSQ20] H. Huang, F. Zhong, Y. Sun, and X. Qin. An occlusion-aware edge-based method for monocular 3d object tracking using edge confidence. *Computer Graphics Forum*, 39:399–409, 2020. doi:10.1109/TIP.2020.2973512.
- [Int19] Intel. <https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf>, 2019. Online, accessed 09-September-2021.

- [KSC13] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106, 2013.
- [KTIN17] W. Kehl, F. Tombari, S. Ilic, and N. Navab. Real-time 3d model tracking in color and depth on a single cpu core. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 465–473, 2017.
- [LCS11] S. Leutenegger, M. Chli, and R. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *IEEE International Conference on Computer Vision*, 2011.
- [LD17] S. Li and Lee. D. Rgb-d slam in dynamic environments using static point weighting. *IEEE Robotics and Automation Letters (RA-L)*, 2(4):2263–2270, 2017.
- [LDW91] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991. doi:10.1109/70.88147.
- [LF05] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends in Computer Graphics and Vision*, 2005.
- [LH81] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 1981.
- [LKL15] S. Li, S. Koo, and D. Lee. Real-time and model-free object tracking using particle filter with joint color-spatial descriptor. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6079–6085, 2015.
- [Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, 1999.
- [Low04] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- [LR05] W. H. Lipscomb and T. Ringler. An incremental remapping transport scheme on a spherical geodesic grid. *Monthly Weather Review*, 2005.
- [Lu18] X. X. Lu. A review of solutions for perspective-n-point problem in camera pose estimation. *Journal of Physics Conference Series*, 2018.
- [LWZ21] F. Liu, Z. Wei, and G. Zhang. An off-board vision system for relative attitude measurement of aircraft. *IEEE Transactions on Industrial Electronics*, pages 1–1, 2021. doi:10.1109/TIE.2021.3075889.

- [LZ13] M. Lourakis and X. Zabulis. Model-based pose estimation for rigid objects. In *International conference on Computer Vision Systems*, 2013.
- [LZXQ21] J. C. Li, F. Zhong, S. H. Xu, and X. Y. Qin. 3d object tracking with adaptively weighted local bundles. *Journal of Computer Science and Technology*, 36:555–571, 2021. doi:10.1007/s11390-021-1272-5.
- [MAMT15] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.
- [MAT17] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. doi:10.1109/TR0.2017.2705103.
- [Mic21] Microsoft. Azure kinect dk hardware specifications. <https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification>, 2021. Online, accessed 19-April-2021.
- [MKK⁺20] I. Maroungkas, P. Koutras, N. Kardaris, G. Retsinas, G. Chalvatzaki, and P. Maragos. How to track your dragon: A multi-attentional framework for real-time rgb-d 6-dof object pose tracking. *Computer Vision and Pattern Recognition*, 2020.
- [MTKW02] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.
- [NIH⁺11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.
- [NNB04] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.
- [PCSAS18] M. Pfanne, M. Chalon, F. Stulp, and A. Albu-Schäffer. Fusing joint measurements and visual features for in-hand object pose estimation. *IEEE Robotics and Automation Letters*, 2018.
- [PRDR13] K. Pauwels, L. Rubio, J. Díaz, and E. Ros. Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2347–2354, 2013.

- [Pri12] V. Prisacariu. Pwp3d: Real-time segmentation and tracking of 3d objects. *International Journal of Computer Vision*, 2012.
- [PX94] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 1994.
- [RD06] E. Rosten and T. Drummond. Machine learning for high speed corner detection. *9th European Conference on Computer Vision*, 1:430–443, 2006.
- [RPK⁺17] C. Y. Ren, V. A. Prisacariu, O. Kähler, I. D. Reid, and D. W. Murray. Real-time tracking of single and multiple objects from depth-colour imagery using 3d signed distance functions. *International Journal of Computer Vision*, 124(1):80–95, 2017.
- [RRKB11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surfs. In *IEEE International Conference on Computer Vision, ICCV 2011*, 2011.
- [SAM68] R. Sadourny, A. Arakawa, and Y. Mintz. Integration of the nondivergent barotropic vorticity equation with an icosahedral hexagonal grid for the sphere. *Monthly Weather Review*, 1968.
- [SB20] I. Suarez and J. M. Buenaposada. Beblid: Boosted efficient binary local image descriptor. *Pattern Recognition Letters*, 2020.
- [SDS09] M. Sarkis, K. Diepold, and A. Schwing. Enhancing the motion estimate in bundle adjustment using projective newton-type optimization on the manifold. In K. S. Niel and D. Fofi, editors, *Image Processing: Machine Vision Applications II*, volume 7251, pages 125 – 136. International Society for Optics and Photonics, SPIE, 2009.
- [SK08] B. Siciliano and O. Khatib. *Springer Handbook of Robotics*, pages 10–18. Springer, Berlin, Heidelberg, 1 edition, 2008.
- [SL04] I. Skrypnik and D. Lowe. Scene modelling, recognition and tracking with invariant image features. *IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2004.
- [SLK14] Koo. S., D. Lee, and D. S. Kwon. Incremental object learning and robust tracking of multiple objects from rgb-d point set data. *Journal of Visual Communication and Image Representation*, 25(1):108–121, 2014.
- [SLK16] Koo. S., D. Lee, and D. S. Kwon. Fast visual odometry using intensity assisted iterative closest point. *IEEE Robotics and Automation Letters (RA-L)*, 1(2):992–999, 2016.

- [SPHI14] B. Seo, H. Park, J. Hinterstoisser, and S. S. Ilic. Optimal local searching for fast and robust textureless 3d object tracking in highly cluttered backgrounds. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):99–110, 2014.
- [SPS16] K. Sarkar, A. Pagani, and D. Stricker. Feature-augmented trained models for 6dof object recognition and camera calibration. *International Conference on Computer Vision Theory and Applications*, 2016.
- [SPS⁺20] M. Stoiber, M. Pfanne, K. H. Strobl, R. Triebel, and A. Albu-Schäffer. A sparse gaussian approach to region-based 6dof object tracking. *ACCV: Asian Conference on Computer Vision*, 2020.
- [SSS06] N. Snavely, S. M. Seitz., and R. Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Transactions on Graphics*, 2006.
- [SZZ⁺21] X. Sun, J. Zhou, W. Zhang, Z. Wang, and Q. Yu. Robust monocular pose tracking of less-distinct objects based on contour-part model. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2021. doi:10.1109/TCSVT.2021.3053696.
- [TK92] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 1992.
- [TMHF00] B. Triggs, P. Mclauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Vision Algorithms: Theory and Practice*, pages 298–372, 2000.
- [TSSC18] H. Tjaden, U. Schwanecke, E. Schomer, and D. Cremers. A region-based gauss-newton approach to real-time monocular multiple object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.
- [VLF04a] L. Vacchetti, V. Lepetit, and P. Fua. Combining edge and texture information for real-time accurate 3d camera tracking. *IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2004.
- [VLF04b] L. Vacchetti, V. Lepetit, and P. Fua. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.
- [WLM⁺15] T. Whelan, S. Leutenegger, R. S. Moreno, B. Glocker, and A. Davison. Elasticfusion: Dense slam without a pose graph. *Robotics, Science and Systems*, 2015.

- [WMRB20] B. Wen, C. Mitash, B. Ren, and K. E. Bekris. se(3)-tracknet: Data-driven 6d pose tracking by calibrating image residuals in synthetic domains. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 10367–10373, 2020.
- [WRM⁺10] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 2010.
- [WXZ⁺20] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese. Densfusion: 6d object pose estimation by iterative dense fusion. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3338–3347, 2020.
- [XSNF17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.
- [YJS06] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), 2006.
- [ZZZ⁺20] L. Zhong, X. Zhao, Y. Zhang, S. Zhang, and L. Zhang. Occlusion-aware region-based 3d pose tracking of objects with temporally consistent polar-based local partitioning. *IEEE Transactions on Image Processing*, 29:5065–5078, 2020. doi:10.1109/TIP.2020.2973512.

License

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of this license, visit <http://creativecommons.org> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.