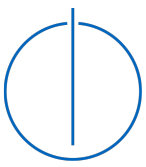
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Optimizing Pens for Use as 3D Raycasting Interactables with Monocular 6-DoF Object Tracking

Min-Shan Luong



DEPARTMENT OF INFORMATICS

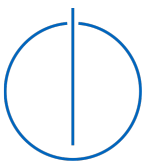
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Optimizing Pens for Use as 3D Raycasting Interactables with Monocular 6-DoF Object Tracking

Optimierung von Stiften für die Nutzung als 3D-Raycasting-Interaktionsgerät mit monokularer 6-DoF Objektverfolgung

Author:	Min-Shan Luong
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Linda Rudolph
Submission Date:	March 15, 2020



I confirm that this Bachelor's Thesis in Informatics: Games Engineering is my own work and I have documented all sources and materials used.

Munich, March 15, 2020 Min-Shan Luong

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor, Linda Rudolph, for her continuous guidance, availability and patience throughout this thesis. I deeply appreciate her contributions of time and ideas to my work.

I am also grateful to Adnane Jadid for being very generous with his time and knowledge, providing both technical support and encouragement.

My sincere thanks goes to Felix Neumeyer for his priceless support, technically and spiritually. He gave me valuable suggestions concerning my work and constantly took care of my well-being.

Last but not least, I would like to thank my family for their continuous support and care throughout my life.

Abstract

In this thesis, we propose an approach to track the pose (6-DoF) of a pen by attaching an object marker specifically designed for this purpose. Our work includes designing an optimal object marker, as well as a 6-DoF tracking system for this object with one stationary webcam. For the design of the object marker, we focused on textured planar models. After calibrating the camera, the object to be tracked is registered in order to use these model information as input for the tracking algorithm. Furthermore, 3D scans of the objects are created to retrieve a digital 3D representation which is used for object registration. Tracking quality is then evaluated by testing the accuracy and speed.

The aforementioned work is part of a more extensive project, with the goal to interact with objects in the augmented environment intuitively by adding extra functionality to an ordinary pen. This pen can then be used similarly to a laser pointer. For high mobility, instead of a stationary camera, this camera is attached to the user while capturing visual input for the tracking.

In dieser Arbeit wird ein Ansatz für das Tracking der Pose (mit 6 Freiheitsgraden) eines Stiftes vorgestellt. Ein speziell für diesen Zweck entworfener Marker wird dazu an einem Stift befestigt. Das Entwerfen eines solchen optimalen Objekt-Markers sowie die Entwicklung eines Algorithmus für das Tracking des Markers mit einer stationären Webcam ist Teil dieser Arbeit. Beim Objekt-Marker-Design lag der Fokus auf texturierte, planare Modelle. Nach dem Kalibrieren der Kamera wird das Objekt, das getrackt werden soll, registriert, da die Objekt-Modell-Informationen für den Tracking-Algorithmus benötigt werden. Desweiteren werden 3D-Scans des texturierten Objekt-Markers erstellt um eine digitale Repräsentation zu erhalten, die während der Objekt-Registrierung genutzt wird. Die Tracking-Qualität wird anschließend hinsichtlich ihrer Genauigkeit und Geschwindigkeit bewertet.

Diese Arbeit legt die Grundbausteine für ein weiterführendes Projekt mit dem Ziel Interaktionen in der Erweiterten Realität durch einen gewöhnlichen Stift zu ermöglichen. Dieser Stift soll ähnlich einem Laser-Pointer bedient werden. Für größere Bewegungsfreiheit sollte die Kamera dann am Nutzer befestigt und nicht stationär sein.

Contents

List of Notations

Notation	Description
FPS	The camera's frame rate.
K	The camera's intrinsic matrix.
R	The camera's rotation matrix.
t	The camera's translation vector.
$(R\|t)$	The camera's extrinsic matrix.
$dist_{coeffs}$	The camera's distortion parameters.
θ	Distance between rotations measured by an angle
tr(M)	Trace of matrix M
M^T	Transpose of matrix M

Abbreviations

Term	Definition
FPS	Frames Per Second
DoF	Degrees of Freedom
PnP	Perspective-n-Point
RANSAC	Random Sample Consensus
F2F	Frame to Frame
OF	Optical Flow
RE	Mean Reprojection Error
GT	Ground Truth

1 Introduction

1.1 Motivation

The use of augmented reality systems in the industry is on the rise. The benefits are certainly given: It can raise efficiency and productivity, as well as facilitate the workflow of employees by augmenting their environment with additional information. Common application areas include assembly, training, and others [1]. For example, the time it takes to look up information about components in manuals can be shortened by displaying the corresponding information directly into the environment whenever the user points at a component. This can be done by using an interaction device. However, workers in the industry usually carry along a large set of tools that they need during work, which means that an additional tool would be likely burdening.

In order to keep the number of additional tools low, we suggest an approach that reuses or extends the functionality of an already existing tool, as for example, a pen. The idea is to interact with objects in the augmented environment intuitively by converting an ordinary pen into an interaction device in order to use it similar to a laser pointer. This can be done by attaching an object marker to the end of a pen. While taking notes on a clipboard, workers or employees can then easily use the end of the pen to point at things in their environment. Multiple stationary cameras installed on the ceiling would probably have a good view on the object marker. However, neither multiple nor stationary cameras are desired in the industry due to lack of space and possible issues with employee monitoring. Hence, as a solution, a single camera for visual input could be either attached to the clipboard or to the users clothes, so that the tracking system offers high mobility with a minimum of camera devices. For easy collaboration, we suggest to use a projector in this setting, so that co-workers can share what they see. This may be helpful to convey and discuss thoughts so that everybody can quickly get the idea.

1.2 Contributions

The scope of this thesis includes creating the first building blocks of the above explained idea, which is to develop a cheap, lightweight interaction device. Therefore, this

thesis proposes a 6-DoF object tracking algorithm for well textured objects. Our main contributions are:

- The use of synthetic data to facilitate object registration
- An image feature based object tracking algorithm using a single stationary camera
- A frame-to-frame tracking algorithm using optical flow
- The design of object marker prototypes that are adjusted to our tracking algorithm, and which can be plugged on a pencil

1.3 Thesis Structure

First, we present related work in chapter 2. Afterwards, we describe our reference project more detailed in chapter 3, which inspired a large part of this thesis. Since our goal is inter alia to design a good object marker, we want to understand the functionality and construction of image features. Therefore, in chapter 4, we analyze a subset of feature descriptors and and detectors, and then point out their advantages and drawbacks. The camera calibration process and its importance is explained in chapter 5. Chapter 6 presents the approach to register the to be tracked object necessary for the object tracking algorithm. This 6-DoF object tracking algorithm and our problem formulation are described in chapter 7. Furthermore, we demonstrate our approach for designing the object marker prototypes in chapter 8. Then, in chapter 9, we evaluate the performance of our tracking and detection algorithm in combination with our designed object marker. Finally, the last chapter 10 summarizes the main points of this thesis and suggests ideas for future work.

2 Related Work

In the project *Augmented Chemical Reactions* [2], an application to display chemical reactions on molecular level was designed. A molecule can react to other molecules depending on the distance and orientation to each other. In order to make interactions in an augmented reality environment intuitive, [2] uses a physical cube marker with a handle. This cube has black and white square markers on every side, except for the one with the handle attached, to enable tracking of its pose. Using *UbiTrack*'s tracking system for markers, the orientation and position of the cube is tracked relative to the camera, making it possible to display a virtual chemical molecule on top of a marker on the camera image. [2]

Most of the existing augmented reality system solutions visualize virtual information on a dedicated screen or projection area which may restrict the user's mobility and, additionally, is less intuitive than having the information directly projected into the real world. Therefore, *WUW - Wear Ur World* [3] proposes the idea of a wearable mobile information interface that augments the user's environment. The user wears a hat with a camera and a projector clipped to it, so that the camera input is similar to what the user sees. The projector then projects virtual information into the real world with which the user can interact via hand gestures. [3]

Vuforia [4] and *ARKit* [5] are proprietary frameworks that allow to recognize and track 3D objects after scanning them with their respective 3D scanning application. The environment can then be augmented by e.g. overlaying manuals, adding virtual 3D content, interactive guidances, and more [6]. Additionally, if available, Vuforia is also able to deal with CAD models as digital representation instead of a 3D scan [1]. While ARKit is limited to mobile iOS devices, Vuforia offers a cross-platform solution for smartphones, PCs and head-mounted displays like *Microsoft HoloLens* [7].

WUW - Wear Ur World describes our future system setting well. It should be mobile and support intuitive user input. However, in this thesis, we want to use an interaction device instead of hand gestures to interact with the augmented environment. Similar to Maier's interaction device to augment molecules, the interaction device proposed in this paper consists of a 3D marker attached to a stick, here a pen. Moreover, instead of a cube marker we use a self-designed object marker that does not require the camera to see the whole marker in order to detect it. In the end, we want to detect and track a 3D object as is the case for *Vuforia* and *ARKit*.

3 Reference Project

Most of this project's tracking algorithm is inspired by [1]. It proposes a way to track the pose (6DoF) of an arbitrary object with texture under challenging conditions including fast motion, occlusions and illumination changes. The tracking is based on 3D high quality scans of that object. Furthermore, [1] has implemented remote live support realized with Mobile Edge Computing (MEC) in order to provide high computational power to less powerful devices like smartphones or tablets. Since this thesis is highly inspired by the tracking procedure of [1], the following part presents an overview about the methods used for object registration, detection and also tracking.

Object Registration To register the object to be tracked, [1] scanned the object by means of phase shifted structured light. One projector and two cameras were used, requiring multiple scans from different perspectives to capture every side of the object. The single scans are then matched and merged afterwards by using the Iterative Closest Point method (ICP). ICP iteratively aligns one point cloud (*source*) to another fixed one (*reference*) by finding the closest point from the *source* to the *reference* point cloud for each point in the *source* point cloud and by minimizing the root mean square error. Thus, we receive one single point cloud representing our scanned object. [1]

The second step is to define *anchor points*. These are points that are statistically highly probable to be found, using a point detector (*Good Features To Track* [8] in [1]) to find the most salient points in the image. Moreover, a discrete 3D accumulator is used to determine the likelihood of points to be detected after reconstructing the 3D position of the found 2D points with help of depth information received from the scanning. The size of the bins are in respect to the size of the object. Afterwards, a Non-Maximum Suppression is run over the collected points in order to limit the amount of points within a specific area (suppression of non-maximum) by choosing the one with the highest probability (maximum). [1]

In addition to that, a certain number of rendered images are saved that act as reference images later for the detection. The ORB features as well as their respective 3D coordinates are saved for the same purpose.

Object Detection This section is divided into two parts called *ORB Initializer and Reinitializer* in [1]. Before tracking can commence, an initial pose has to be determined as

the tracking requires information about the pose in the previous frame. For this reason, ORB Initialization was implemented, where the ORB features of the rendered images saved during object registration are matched with the current frame. Additionally, potential outliers in the set of matches are filtered out by performing a ratio test like it is done for SIFT [9] (see 4). Then, the image with the highest number of remaining matches is taken as reference image to obtain information about the 3D positions to the corresponding 2D keypoints so that the camera pose, and therefore the object pose, can be estimated by solving the Perspective-n-Point (PnP) pose problem combined with RANSAC. To speed up the matching between ORB features, the process is divided among multiple threads run in parallel. In case the ORB Initialization fails, the current frame is skipped and ORB Reinitialization is started for the consecutive frame. [1]

The same problem occurs when the tracking fails to propose a valid pose. This may happen due to insufficient number of matchings e.g. due to motion blur. Hence, in order to provide a previous pose to the tracker, ORB Reinitialization is carried out. The idea of ORB Reinitialization is to retrieve a correct pose within a short time with help of stored keyframes, which means that if there is no keyframe, ORB Initialization has to be executed instead. Those keyframes are obtained in a thread updated less frequently than the main threads. However, certain conditions apply for selecting keyframes. Only those frames are chosen whose ratio of outliers to inliers from the RANSAC algorithm as well as the reprojection error of the tracked features are below particular thresholds. Together with the frame, the corresponding pose, its ORB descriptors including 2D coordinates and their calculated 3D positions are stored. Finally, the matching of the current frame with the latest keyframe is performed to estimate the pose, which is usually comparatively fast due to the high similarity between the two frames resulting in a high number of matches. If the attempt of ORB Reinitialization is not successful, ORB Initialization is started. [1]

Object Tracking Tracking-by-detection and frame-to-frame tracking are both popular approaches for tracking objects in a series of images. On the one hand, tracking-by-detection needs no prior information about the object pose in the last frame since detection is done every frame. On the other hand, the detection process is relatively time consuming. If a faster solution is desired, frame-to-frame tracking offers a good alternative as long as an initial pose is given. Frame-to-frame tracking tracks the movement of objects in consecutive frames while making use of relevant information from previous frames. The implementation in [1] works with frame-to-frame-tracking. If during the object detection process (ORB Initialization and Reinitialization) the previous pose is successfully determined, frame-to-frame-tracking is possible. Otherwise, ORB Reinitialization is attempted. [1]

In case that the pose of the last frame was valid, this frame is rendered off screen and the areas around the registered anchor points are matched with the ones of the current frame via optical flow, namely the *Kanade-Lucas-Tomasi (KLT) tracker* [10] (*Render to Frame* or *R2F*). Likewise, the real image is matched with the current frame (*Frame to Frame* or *F2F*). With the help of the R2F matching, the pose resulting from the F2F matching is corrected in order to avoid drift that unavoidably occurs after some steps with the optical flow method. [1]

If anchor points are lost during the tracking (e.g. due to occlusion), or if they are considered to be an outlier, they are marked as not valid. In this case, their 2D position will be computed through their 3D position and the estimated object pose, that is computed with the remaining anchor points, until the lost points become visible again in case of temporary occlusion. [1]

Since optical flow with Good Features To Track is used, applying an edge enhancing image filter supports the tracking of the feature points. Therefore, [1] proposes a pencil filter which is described further in the next paragraph.

Pencil Filter In [1], a pencil filter is applied on the images to enhance the KLT feature matching. This filter entails the properties of enhancing edges in the image as well as normalizing the image to become more illumination invariant. The two steps of this filter are dilation of the image converted to grayscale, and thresholding the pixels to normalize the pixel values. In a later paper [11], the pseudo code for the filter was published (see algorithm 1).

Algorithm 1 Pencil Filter

```
1:  function PENCIL FILTER(I)                                   ▷ Image I
2:      G = Convert to grayscale(I)
3:      P = Dilate(G, ELLIPSE)                                  ▷ Dilation with ellipse
4:      for y=0 to y=G.rows - 1 do
5:          for x=0 to x=G.columns - 1 do
6:              if (P(y,x) == 0) then
7:                  P(y,x) = 255
8:              else
9:                  P(y,x) = int(G(y,x) * 255)/P(y,x)           ▷ Measuring Pixel Difference
10:             end if
11:         end for
12:     end for
13:     return Pencil Image P
14: end function
```

Dilation is the operation of convolving an image with a kernel that has a particular shape and size. OpenCV [12] provides a function for dilation with a rectangle, cross, or ellipse given a predefined size plus an anchor point for the kernel. While running the kernel over the image, the pixel located at the anchor point position (here the center of the kernel) is replaced by the maximum value of the pixels within the kernel. Thus, bright regions in the image become brighter, and the image generally becomes blurrier. However, it should be mentioned that the blurriness is to be distinguished from Gaussian blur, where the edges become soft, whereas with dilation sharp edges may remain. All in all, dilation makes the image appear brighter, but details in the dark would be still lost. In order to preserve those details, the difference between the pixel values of the grayscale image and the dilated one needs to be calculated. The same is achieved by dividing the values of the grayscale image by the ones the dilated image [11]. The application of the pencil filter can be seen in figure 7.2.

4 Image Features

Image features are typically located at distinctive image areas like edges, corners, peaks, or generally in areas with high intensity changes and unique texture. These features are selected by a feature detector that scans through an image and picks points or patches that satisfy the respective requirements to be considered a feature. Accordingly, there are various algorithms for feature extraction, each focusing on different kinds of features. Often, the feature's appearance is saved additionally to its location. This appearance is described by a feature descriptor, which can be stored in different ways. Some descriptors are more complex and manage to recognize the same patch in another image well during the matching phase, while others may return less accurate results but score higher in regards of computational cost.

In the following, the functionality of a subset of image detectors and descriptors is explained. This includes SIFT (section 4.1), since it is used frequently to define features in images, FAST(section 4.2) as well as BRIEF (section 4.3), and the combination of FAST and BRIEF named ORB (section 4.4), which we are using for this project.

4.1 SIFT

SIFT was introduced in 1999 and stands for *Scale-Invariant Feature Transform* [9]. SIFT features are invariant in location, scale, rotation, small viewpoint changes and illumination changes [13], and therefore, highly accurate. However, this high accuracy comes along with large computational complexity. In the following, the SIFT detector, as well as the creation of a SIFT descriptor is explained.

In the first step, the location of potential keypoint candidates are detected by finding scale-space extrema of the *Difference-of-Gaussian* convolved with the image. The scale-space is defined by a continuous function that applies convolutions with a Gaussian kernel on an image at different scales. Furthermore, it consists of a certain number of octaves with different image resolutions that are reduced by half after each octave. To find extrema in scale-space, [13] has proposed to compute the Difference-of-Gaussian (DoG) as an approximation to the scale-normalized Laplacian-of-Gaussian (LoG) due to its efficiency. A visualization of the DoG is shown in figure 4.1. After applying DoG, the pixel of one image is then compared to its 26 neighboring pixels in the next upper (9 pixels), next lower (9 pixels) and the same scale level (8 pixels) to finally identify a

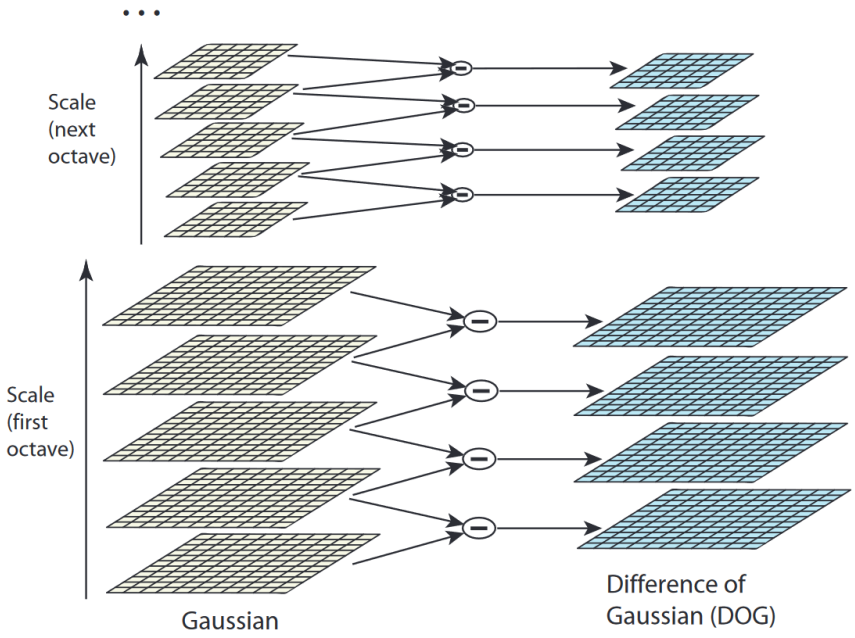


Figure 4.1: The computation of the DoG with images convolved with a Gaussian kernel.

local extremum. With this result, the found keypoint candidates are still prone to some noise. In order to keep them more stable to noise, keypoints with low contrast and poorly determined location along an edge are filtered out. [13]

Now that the keypoints are scale-invariant, the next step is to make them rotation-invariant. Before this, each keypoint is assigned a specific orientation using the following steps. For each sample point within a certain range around the keypoint center, a gradient vector is calculated together with its length and orientation. The computations are done on a Gaussian smoothed image, while the scale of the keypoint determines from which scale this image is taken. Then, those gradients are assigned to one of 36 bins where each bin represents 10 degrees from a total of 360 degrees. The value added to a bin is determined by the magnitude of the gradient vector which is additionally weighted by a Gaussian-weighted circular window (see figure 4.2). In the end, the most dominant direction has the highest peak in the histogram. Therefore, this direction becomes the orientation of the keypoint. In addition to that, other local peaks above 80% of the highest peak also create a separate keypoint with this orientation at the same location and with the same scale. This significantly improves the stability of the matchings. [13]

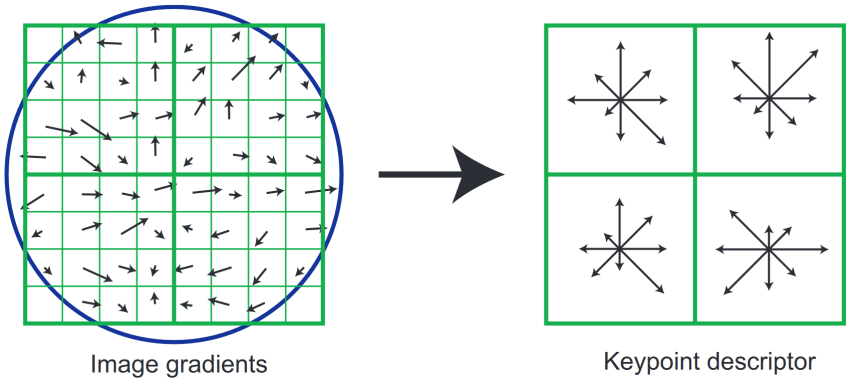


Figure 4.2: Building a descriptor from image gradients, here with a 8x8 window instead of a 16x16 window. The Gaussian-weighted circular window is shown on the left (blue). A gradient area of 4x4 create one oriented histogram, resulting in 4 oriented histograms in total. These 4 oriented histograms together build up one descriptor.

At this point, the keypoints have an assigned orientation but the descriptor to each keypoint is lacking. In the following step, to include orientation-invariance, the gradients within a 16x16 window around the keypoint are computed with respect to the keypoint orientation. We use 4x4 sample regions to build one orientation histogram (see figure 4.2). These window sizes have turned out to be the best choices. In contrast to the previous procedure, the number of bins here is only 8, dividing 360 degrees in these 8 bins. Altogether, this results in the final descriptor consisting of a 4x4x8 vector. After normalization of this vector, the keypoint becomes invariant to affine illumination changes, but not to non-linear illumination changes. [13]

Features are matched by identifying the best match for a keypoint first, while bad matches are filtered out through a ratio test afterwards. The best match is defined by the nearest neighbor of a keypoint. By searching for the shortest Euclidean distance between the respective descriptor and all other descriptors, the nearest neighbor is determined. [13]

SIFT's high accuracy and relatively high robustness against several visual changes makes it popular for the usage in computer vision. Nevertheless, due to its high computational complexity, it is suitable to only a limited extent for real time applications. SIFT is patented and therefore not free to use for commercial purposes.

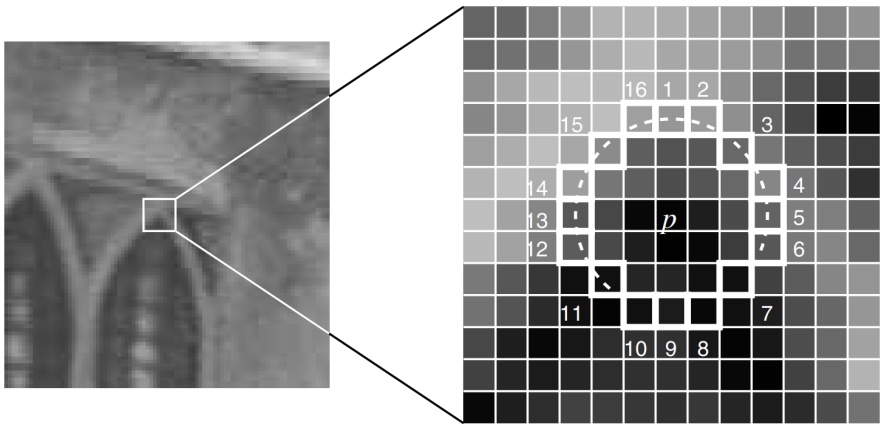


Figure 4.3: Scheme of the basic FAST algorithm. Pixels 1 to 16 build the test circle around p. The dashed arc shows a detected corner.

4.2 FAST

The FAST feature detector [14] is a high-speed corner detector which is suitable for real-time applications. After an improved version of the initial approach was proposed by [14] using machine learning, FAST outperforms several other existing corner detectors.

FAST stands for *Features from Accelerated Segment Test*. The segment test criterion is applied to each of the 16 pixels $p \to x$ with $x \in \{1, \ldots, 16\}$ lying on a circle around the corner candidate p. It checks sequentially whether the intensity of the pixels is either higher or lower than the intensity I_p of p. The check includes a certain threshold t, so that $I_{p \to x}$ is considered to have a higher intensity if $I_{p \to x} > (I_p + t)$ or to have a lower intensity if $I_{p \to x} < (I_p - t)$. As soon as a number of n contiguous pixels with either higher or lower intensity are found, p can be considered a corner and all remaining pixels can be neglected. To accelerate the algorithm, a high-speed test is run to quickly filter out non-corners. This can be done by checking only pixels of the four compass directions (pixels 1, 5, 9 and 13 where the algorithm starts at 1 located at the top of the circle). The ps with less than two of the four pixels having intensities either higher than $I_p + t$ or lower than $I_p - t$ can be considered non-corners, and all other pixels on the circle can skip the test. The whole procedure is called the accelerated segment test that is able to detect corners. The scheme of this basic FAST is shown in figure 4.3. [14]

Nevertheless, the initial algorithm, that is run over the complete image, has several weaknesses: non-optimal efficiency, non-optimal speed and detecting multiple adjacent corners. Hence, an extended version of that algorithm was published in [14], which involves machine learning and non-maximal-suppression.

The outcome of the machine learning extension is a trained decision tree which, for a given n, is able to correctly redetect corners in images that are different from the set of training images. First, all training images are scanned for corners using the segment test criterion explained above. Additionally, the 16 pixels $p \rightarrow x$ can be assigned to three states: darker pixels with $I_{p \rightarrow x} \leq I_p - t$, similar pixels with $I_p - t < I_{p \rightarrow x} < I_p + t$, and brighter pixels with $I_p + t \leq I_{p \rightarrow x}$. For each fixed x, and for all $p \in P$, P being all pixels in the set of training images, p is assigned to one of the above states. This creates subsets P_d, P_s, P_b with respectively darker, similar and brighter pixels. Consequently, there are separate P_d, P_s, P_b for each x. K_p is a variable of type boolean that is *true* if p is a corner and $false$ if it is not. With the help of the entropy of K for set P and subsets P_d, P_s, P_b, the x with the highest information gain about whether p is a corner can be determined. Then, recursively for each of the subsets P_d, P_s, P_b of the previously selected x with highest information gain, the new x with highest information gain is determined similarly. The recursion stops when the entropy K of a subset is zero. This process creates a decision tree that can then serve as a corner detector. [14]

Now being able to detect corners, these corners still should be filtered to receive one distinct corner instead of multiple adjacent ones. By means of a score function V, non-maximal-suppression is applied, where the corners with a lower V value are removed and only the one with the maximum V value remains [14]. In [14], V is defined as follows:

$$V = max \left(\sum_{x \in S_{bright}} |I_{p \rightarrow x} - I_p| - t, \sum_{x \in S_{dark}} |I_p - I_{p \rightarrow x}| - t \right) \tag{4.1}$$

where

$$S_{bright} = \{x | I_{p \rightarrow x} \geq I_p + t\} \tag{4.2}$$

$$S_{dark} = \{x | I_{p \rightarrow x} \leq I_p - t\} \tag{4.3}$$

Compared to SIFT, FAST is less accurate but it uses less than 7% of the available processing time whereas SIFT took 300% of it for feature detection [14]. Moreover, it is challenging for FAST to deal with high levels of noise. Another drawback is that FAST does not consider rotation and is thus rotation-variant [15]. Additionally, scale-invariance is given only to a limited extent, thus it has lower scale-invariance than other detectors [16].

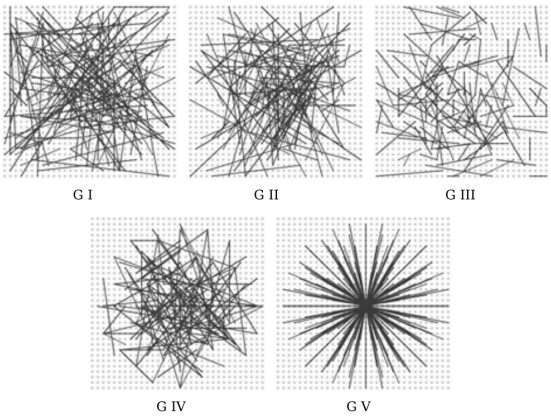


Figure 4.4: Different approaches of the test locations evaluated in [17]. One straight black line shows one pixel pair.

4.3 BRIEF

In contrast to most feature descriptors, Binary Robust Independent Elementary Features (BRIEF) uses binary strings for the representation of features [17]. A string size of around 128 to 256 bits is often sufficient for very good matching results according to [17]. Feature matching is done by computing and comparing the Hamming distance between descriptors [17]. The Hamming distance is determined by summing up the number of 1's in the bit string after applying an XOR operation on the bit strings to be compared. In this section, an insight of the construction of these bit strings is given.

Each descriptor contains data about an image patch of size $S \times S$. This patch is smoothed in order to be more robust to noise, since it compares single pixel values within the patch in a test. After comparing the intensity of a pixel pair $(p(x), p(y))$, either a 1 (when $(p(x) < p(y))$) or a 0 (otherwise) is added to the bit string. In the original paper [17], a patch size of 9×9 pixels was chosen as well as the level of Gaussian smoothing for the patch which is 2. [17]

With the general idea of BRIEF presented above, it is still necessary to choose $p(x), p(y)$ for each pixel pair. After testing different approaches for the test locations, the structure *G II* shown in figure 4.4 yielded the best results. The pixel pairs are distributed with an isotropic Gaussian distribution around the keypoint center. [17]

All in all, BRIEF uses a very compact and lightweight descriptor structure that realizes fast matchings. Nonetheless, neither orientation nor scale are directly respected in the descriptor calculation [16, 15]. Therefore, BRIEF's rotation- and scale-invariance is very limited.

4.4 ORB

As the name depicts, Oriented FAST and Rotated BRIEF (ORB) [18] is based on FAST and BRIEF (see 4.2, 4.3). It uses oriented FAST (oFAST) as feature detector and rotated BRIEF (rBRIEF) as feature descriptor. Both are extensions of their respective original version which were modified such that ORB becomes rotation-invariant as well as to a certain degree scale-invariant.

In the original paper, FAST-9 is used with radius 9 for the circle around the potential corner. To achieve a certain degree of scale-invariance, a scale pyramid for an image is created and the FAST detector is applied to each scale of the pyramid. As soon as the FAST algorithm has detected sufficient corners, edge-like corners that are low ranked according the Harris corner measure are filtered out. The Harris corner measure provides information about the cornerness of a keypoint. Thus, only a subset of keypoints with the highest cornerness remain. Moreover, the intensity centroid [19] is computed with with moments in order to improve the corner's rotation-invariance. Here, the direction of the vector from the corner center to the centroid location indicates the orientation of the keypoint. [18]

In the next step, to each detected keypoint, a BRIEF descriptor is constructed. By means of the previously calculated keypoint orientation, the image patch defined by its BRIEF descriptor is reoriented (steered) so that it matches the keypoint orientation. Furthermore, a look-up table is created which contains different rotated versions of each descriptor discretisized to $2\pi/30$ (12 degrees) increments. Moreover, an additional step is necessary since steered BRIEF leads to a loss of variance compared to the original BRIEF, but a high variance makes the feature more discriminative. Therefore, steered BRIEF is extended by a greedy algorithm to increase the features' variance and to diminish correlation among binary tests. [18]

For feature matching, ORB also computes and compares the Hamming distance of the descriptors. However, since rBRIEF uses binary strings as descriptor representation, multi-probe Locality Sensitive Hashing [20] is used to speed up the nearest neighbor search. [18]

ORB offers a good trade-off between performance (with regard to matching accuracy and various invariance) and computational costs. Its performance is comparable to SIFT, except for scale changes [15]. Image pyramids are not sufficient to offer proper scale-invariance [18]. However, since ORB is significantly faster than SIFT, we use ORB for our real-time application.

5 Camera Calibration

If the position and orientation of the camera is known in 3D world space, it is possible to do a perspective projection of the 3D space captured by the camera. For that projection, the pinhole camera model is used to describe the relation between objects in 3D space and 2D scene, which is inter alia dependent on the camera properties. For further information about the pinhole camera model, see chapter 7. Since camera properties differ from camera to camera, it is necessary to identify the individual properties for the used camera first before continuing to work with the captured images. This is done once for a camera by performing camera calibration. The properties relevant for the tracking algorithm as well as the camera calibration progress of our webcam are explained in the following.

Intrinsic Parameters Intrinsic parameters define the geometric properties of the camera. Those parameters can be presented in matrix form as following intrinsic matrix K, which is parameterized by [21] (taken from [22]):

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.1}$$

$$= \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}} \tag{5.2}$$

In terms of the pinhole camera model, the camera intrinsic matrix transforms 3D camera coordinates to 2D homogeneous image coordinates. The intrinsic matrix consists of five parameters, namely focal length f_y and f_y considering the pixel size in x and y direction, axis skew s, and coordinate (x_0, y_0) of the principal point given in image coordinates which are determined during the camera calibration process. The focal length F, usually given in millimeters, indicates the distance from the image plane to the optical center of the camera (center of projection). Depending on the image sensor characteristics, image pixels may be not square other than assumed [23]. In case the

pixels' height and width are not equal in length, the focal length F needs to be divided by the pixel width p_x and height p_y as shown in following formulas:

$$f_x = F / p_x \tag{5.3}$$
$$f_y = F / p_y \tag{5.4}$$

If the pixel are not rectangular, the skew parameter s indicates how much the pixels are skewed in x direction. Last but not least, the principal point (x_0, y_0) is located at the intersection of the ray from the optical center to the image plane while perpendicular to this plane.

Distortion Parameters Lens distortion is not included in the ideal pinhole camera model since it does not have a lens unlike real cameras. Therefore, distortion parameters should be identified during the calibration to represent the camera more accurately. Generally, there are two kinds of image distortions to distinguish. One is the tangential distortion that occurs when the camera lens is not perfectly parallel to the camera sensor. This kind of distortion usually has a relatively low impact on the image so that it is often neglected [24]. In contrast, radial distortion occurs due to the natural property of a spherical lens and should not be neglected due to its high impact. The lens causes straight lines in the images to be bend in opposition to how it looks in reality [24]. Using the distortion parameters

$$dist_{coeffs} = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{pmatrix} \tag{5.5}$$

where k_1, k_2, k_3 express the radial distortion and p_1, p_2 the tangential distortion, it is possible to undistort images by recalculating the new image coordinate of each pixel. Together with the images, the intrinsic matrix needs to be adjusted to the undistorted images. Changes in the image resolution do not affect the distortion parameters[12].

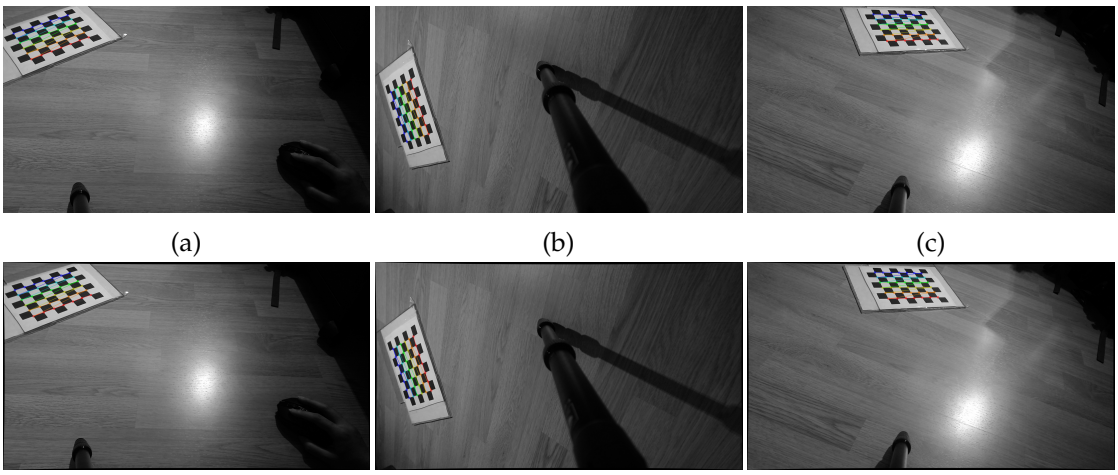
Reprojection Error The reprojection error measures the distance between detected keypoints in an image and the corresponding points in 3D projected onto the image plane with the estimated projection matrix. The error is measured in pixels and should be ideally as low as possible (e.g. a value below 1 is considered good) since it indicates the accuracy of the estimated projection matrix. During camera calibration, it is important to keep the value low since we are going to work with this camera matrix in the further steps of camera pose estimation. Nevertheless, keeping the mean reprojection error (RE) low is not the main goal during calibration, but rather receiving an intrinsic matrix and distortion parameters that reproduce an image as close as possible to the real appearance of our world. For example, taking only photos of the

calibration board lying in the center of the image may yield lower RE but less accurate distortion parameters since the edges are not covered leading to less well undistorted images.

Calibration Procedure Initially, we used the *Logitech QuickCam S5500* webcam to obtain visual input. However, we decided to change the camera to the *Logitech HD Pro C920* later on due to lack of information about the built-in sensor, which was needed to render images in the game engine *Unity* [25] with realistic camera properties.

Throughout the whole calibration procedure as well as the use for tracking, auto-focus needs to be deactivated which means that the focus should be fix since it influences the focal length. During calibration process, a pre-defined calibration pattern with known sizes is tried to be found in a series of captured images. Therefore, we use a solid calibration board to keep the errors due to uneven surfaces low. *OpenCV* [12] offers an implementation of a camera calibration algorithm based on [26] and [27]. Using the camera calibration code (see appendix C) provided by OpenCV for C++ [12] it is possible to choose between chessboard pattern, asymmetric circles grid or symmetric circles grid. We have tried the asymmetric circles grid first since it may yield better results than the other calibration patterns. However, in the end we switched to using OpenCV for Python [12] and calibrated with the chessboard pattern because in our case the total RE was considerably lower ($\sim$8.5 with asymmetric circles vs. $\sim$0.2188381264343743 with chessboard for Logitech QuickCam S5500 calibration). We have taken photos that show the pattern from different tilts and positions covering the whole image area while rotating the board as much as possible to the extent that it was still detected. A small subset of our calibration images including their undistorted version for comparison is shown in figure 5.1.

By providing perspective foreshortening the focal length can be identified more accurately. Moreover, we considered the lighting condition and provided images with even illumination on the board. Following the advices from [28] and [29], we have received following data:



(a) (b) (c)

(d) (e) (f)

Figure 5.1: Sample images used in our camera calibration process. Distorted images (a)-(c) and undistorted images (d)-(f). The found pattern grid is overlayed on the images (multicolored). Note that the undistorted images are slightly squeezed which is particularly visible along the image edges.

Logitech QuickCam S5500:

Intrinsic Matrix:

$$\begin{pmatrix} 556.18715438 & 0 & 299.44241873 \\ 0 & 515.35959909 & 208.76060998 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.6}$$

Distortion Coefficients:

$$\begin{pmatrix} -0.03845472 & 0.14343245 & -0.00129602 & 0.00158407 & -0.20751661 \end{pmatrix} \tag{5.7}$$

RE: 0.2188381264343743

Logitech HD Pro C920:

Intrinsic Matrix:

$$\begin{pmatrix} 1407.89952 & 0 & 939.552924 \\ 0 & 1405.37747 & 555.658058 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.8}$$

Distortion Coefficients:

$$\begin{pmatrix} 0.01898021 & -0.12506012 & 0.00024567 & 0.00166058 & 0.07576625 \end{pmatrix} \tag{5.9}$$

RE: 0.5750416686950641
Focal Length (mm): 3.71685472195223
The focal length is calculated using (manually set) Sensor Size (mm):

$$\begin{pmatrix} 5.0688, & 3.3792 \end{pmatrix} \tag{5.10}$$

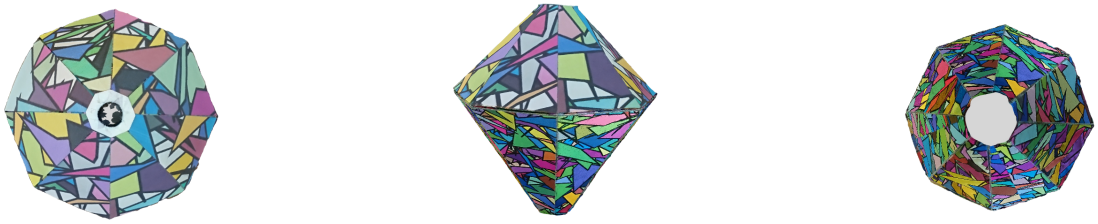
6 Object Registration

Our main objective is to track a predefined object marker. However, before an object can be tracked, it is necessary to define the appearance of that object so that the tracking algorithm knows what it should look for. Hence, we use ORB features to obtain knowledge about the object's appearance. While doing so, the data used to extract features can be either real or synthetic. Our algorithms use synthetic data in order to facilitate the registration procedure.

Based on our objective, this chapter describes the method of reconstructing an arbitrary object digitally in 3D (section 6.1), using a physical camera in Unity [25] to render the object marker with real camera properties (section 6.2), and saving 2D and 3D positions of selected objects to the respective rendered images (section 6.3.2). Moreover, this chapter includes the approach of building an ORB feature model that contains the 2D keypoint location of an ORB feature, along with its descriptor and its corresponding 3D coordinate in world space (6.3). The ORB model needs to be created before the actual tracking procedure (chapter 7) since it stores the 2D-3D correspondences necessary for solving the Perspective-n-Point (PnP) problem (see 7.1).

6.1 Model Reconstruction via 3D Scan

To receive a digital representation of the object to be tracked, we tried two different approaches. For scanning the first prototype of the object marker we used *Structure Sensor (ST01)* (see appendix A) by *Occipital*. It is a device with built-in depth sensor that uses knowledge about depth to recover 3D positions of the corresponding 2D points in images. For all subsequent prototypes, we changed to the photogrammetry software *ReCap Photo* offered by Autodesk Inc., which comes along with the download of *ReCap Pro* [30], to produce 3D models. The only input required by ReCap Photo is a set of photos of the object marker from different perspectives. Here, undistorting these photos is not necessary since ReCap Photo automatically calculates lens distortion and other camera properties. Then, these photos are uploaded to the cloud where the model is computed. After the computation is completed, the generated model can be downloaded and edited directly in the ReCap Photo program. The outcome is a textured and triangulated model with a large number of vertices that can be decimated



(a) top view (b) side view (c) bottom view

Figure 6.1: 3D scanned object marker shown from different views.

to a lower number if needed. The result of one of our scans (see appendix E) is shown in figure 6.1.

An example of how the images used for 3D model reconstruction can look like is shown in 6.2. Although using a professional camera is recommended, we used a smartphone camera to take photos of the object marker since the result was sufficient for our purposes. Since photogrammetry works by tracking the pixels' movement from one image to another and by matching features, it is important to ensure that enough visual features and details can be found on the image [31]. This can be achieved via a texture-rich object or via additional texture given to the background. If a high-quality 3D model is desired, it is recommended to follow the provided guideline [31].

Figure 6.2: One of the photos that were used for the 3D model reconstruction in ReCap Photo.

6.2 Synthetic Data Creation in Unity

We worked with Unity version 2019.2.17f1 [25] to create synthetic data for the object registration. In the following sections, the steps from camera setup to the rendered images, along with a list of image coordinates of the plain object's mesh for each image, are described. The Unity project can be found in the appendix D.

6.2.1 Physical Camera Setup

Unity offers the possibility to render images with given real camera properties. It assumes no distortion which means that the output images resulting from the rendering do not need to be undistorted afterwards. The values we used for the camera component are shown in figure 6.3.

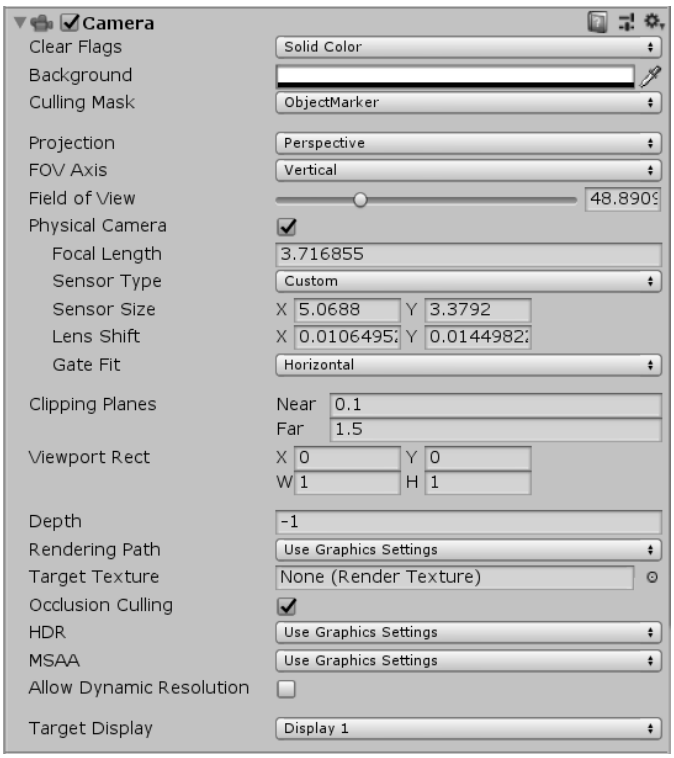
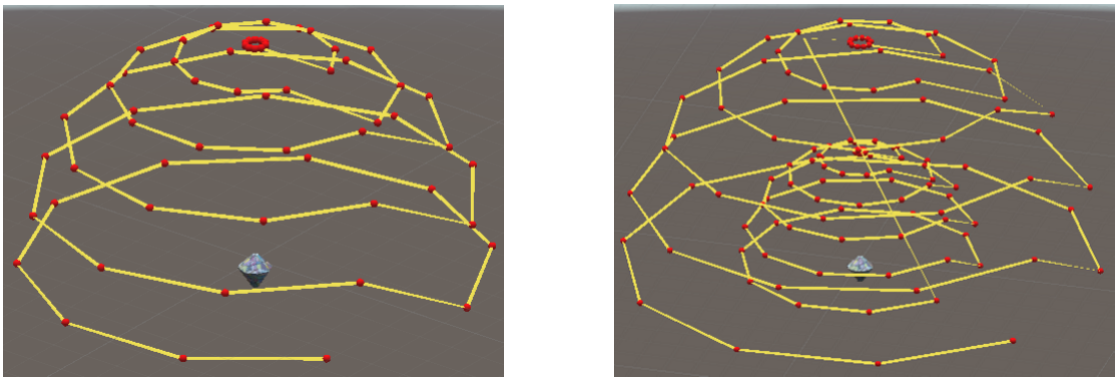


Figure 6.3: The camera component in the Unity inspector showing all values needed to emulate a real camera (without distortion). The lens shift is calculated using the principal point coordinates from the intrinsic matrix determined in chapter 5 and the code from [32]. The sensor size and focal length have the same values as in chapter 5.

6.2.2 Image Rendering

As soon as the physical camera set-up is complete, it is prepared to take images around the 3D scanned object. For this purpose, we have implemented a program that moves the camera along a hemisphere and adjusts its rotation to always look at the object while saving renderings regularly. The track of the camera movement can be seen in figure 6.4. Furthermore, we tested another approach where the camera's renderings are generated from two different distances between the camera and the object, as shown in figure 6.4, so that we have more diverse data for the ORB model (e.g. 24146 keypoints instead of 19191 keypoints). This approach is tested with the intention to support ORB's scale invariance since this property is only partially given by implementing image pyramids. We used the code from [33] to generate and save the images.



(a) Renderings from one fixed distance

(b) Renderings from two different distances

Figure 6.4: Track of the camera movement visualized by means of lines (yellow) and spheres (red) for (a) one hemisphere and (b) two hemispheres of scale 1 and 0.5 around the object marker located in the center.

6.2.3 Image Coordinates of Mesh Vertices

Together with the images rendered from different camera positions, the image coordinates of the (plain object's) mesh vertices are stored. This is done by several calculations in Unity. In order to get from 3D local space to 2D clip space, we apply several transformation matrices, namely the *model matrix* (local to world space), *view matrix* (world to view space) and *projection matrix* (view to clip space) (see equation 6.1). This is followed by a normalization of the clip space coordinates to range $[-1, 1]$ (*viewport transform*) to get to screen space coordinates. Finally, these coordinates in screen space are converted and saved as OpenCV's image coordinates.

$$\underbrace{x_{clip}}_{\text{Clip Coordinates}} = \underbrace{M_{projection}}_{\text{Projection Matrix}} \times \underbrace{M_{view}}_{\text{View Matrix}} \times \underbrace{M_{model}}_{\text{Model Matrix}} \times \underbrace{X_{local}}_{\text{Local Coordinates}} \tag{6.1}$$

Although we are rendering the scanned 3D object, for the coordinate conversion, we take the 3D mesh vertices of the plain object model aligned with the scanned model. This is because we are interested in where those 3D coordinates are located in the images for the registration with known image coordinates (see subsection 6.3.2).

6.3 ORB Feature Model

Different methods exist to build a feature model. In this section, two approaches are presented to obtain the desired ORB model. Both require a series of registration images that show the object in different poses, as well as the mesh of the plain object without texture (see figure 8.1) as PLY file that contains the coordinates of the object's vertices and its triangles. Those registration images can be either captured with the webcam itself (real data) or rendered using camera properties that are as similar as possible to the ones of the real camera (synthetic data).

6.3.1 Click Registration

OpenCV offers a complete sample code that allows to register a planar object by clicking on the vertices of the object mesh shown in an registration image in sequence of how they are listed in the PLY file [12]. The idea is that each click would give the program a 2D image coordinate corresponding to the respective vertex. Consequently, the PnP problem can be solved iteratively to estimate the camera pose for the current image if enough 2D-3D correspondences are available. The minimum is 3 correspondences, which can yield up to 4 solutions, but at least 4 are necessary to diminish pose ambiguity (see section 7.1).

In the next step, by running an ORB feature detector, feature locations and descriptors are extracted from the image. However, the locations in 3D space are yet unknown. Given the pose of the object from previous 2D-3D point registration, these can be identified using the object mesh and the ray-triangle intersection algorithm from [34].

The whole procedure is done for all images in the set of registration images, which results in the final ORB model after merging the outputs together. This registration method does not require a textured 3D model and is suitable for simple meshes with few vertices. However, it can become inconvenient especially if used for more complex

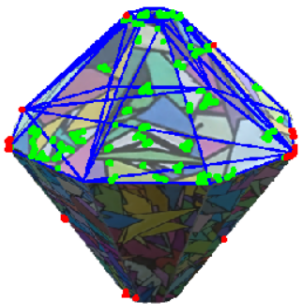


Figure 6.5: Registration image with overlaid mesh (blue), keypoint inliers (green) and keypoint outliers (red)

meshes. It should be noted that this registration method works only for textured planar objects.

6.3.2 Registration with Known Image Coordinates

Our object registration with known image coordinates builds on the above described click registration (see subsection 6.3.1). However, instead of manually registering an object by clicking its vertices, the process can be automated by rendering a scanned object (see subsection 6.2.2) and using the image coordinates of the object's vertices (see subsection 6.2.3). Specifically, we export the corresponding 2D image coordinates of the mesh's vertices from Unity using a simple CSV file. This can then be imported to OpenCV in order to specify the 2D-3D correspondences for pose estimation of the camera. After finding ORB features via ray-triangle intersection, the object mesh is rendered onto the image together with the found keypoints (see figure 6.5). Since this registration method is more user-friendly, it should be preferred when dealing with more complex meshes. In the end, this is also the registration method we opted for.

The registration with known image coordinates is implemented as workaround for the initial idea of taking known camera poses from Unity. This approach allows to skip all steps up to the camera pose estimation in the approach presented in subsection 6.3.1, so that only feature extraction and ray-triangle intersection are left to be done. However, due to issues with coordinate system conversions, we decided to use the aforementioned workaround with known image coordinates.

7 6-DoF Object Tracking

Our goal is to track the six degrees of freedom (6-DoF) of an object marker, whose appearance is described by the information stored in the ORB model (see 6). This means that we want to determine the pose of this object appearing in different image frames. This chapter presents our proposed 6-DoF object tracking pipeline. We first clarify the problem statement and afterwards describe the tracking and detection procedure in detail. Throughout the whole tracking and detection project, we used OpenCV library version 4.1.1 [12]. The code can be found in the appendix C.

7.1 Problem Formulation

The camera pinhole model we use for the projection of 3D points to an image plane can be described by the following equation [35]:

$$\underbrace{\begin{pmatrix} u \\ v \\ w \end{pmatrix}}_{x_{img}} = \underbrace{\begin{pmatrix} f_x & s & x_0 \\ 0 & f_x & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K} \times \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{[R|t]} \times \underbrace{\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}_{X_{world}} \tag{7.1}$$

The homogeneous point in 3D (x_{img}) as well as in 2D (X_{world}) are assumed to be known (e.g. through feature matching). Same applies for the intrinsic matrix K which can be determined through camera calibration (chapter 5). Consequently, the problem we want to address is about identifying the extrinsic matrix $(R|t)$ that satisfies the above equation 7.1 best. $(R|t)$ is composed of the rotation matrix R and the translation vector t. It is the matrix that transforms a point X_{world} in world space to camera coordinates. Then, using the intrinsic matrix, this point is back-projected onto the image plane where we receive x_{img}.

Perspective-n-Point Pose Problem Solving the Perspective-n-point (PnP) pose problem is one approach to estimating $(R|t)$, which represents the camera pose. For the PnP approach, 2D-3D correspondences are needed to be able to solve the following equation [35]:

$$0 = h_i^2 + h_j^2 - 2h_i h_j \cos\theta_{ij} - d_{ij}^2 \tag{7.2}$$

where:

$i \neq j$

h_i, h_j are the distances from camera center C to each X_i and X_j respectively

d_{ij} is the distance between two points X_i and X_j in world space

θ is the angle between X_i and X_j at camera center C

In general, each point correspondence adds one constraint to the yet unknown camera-point distances h_i, h_j. Therefore, the minimal number of 2D-3D point correspondences required to receive a finite number of solutions is 3 [36]. However, in some cases it would yield up to 4 possible solutions. In order to remove this ambiguity, a fourth point correspondence is required [37]. Nonetheless, low resolution images and image noise can still lead to ambiguous poses [38]. Particularly coplanarity of the points is often problematic for finding a unique pose.

In our case, we use an iterative approach of solving the PnP problem from the OpenCV library [12]. The implementation in OpenCV determines a pose that minimizes the reprojection error by applying the Levenberg–Marquardt algorithm [39, 40].

Random Sample Consensus Random Sample Consensus (RANSAC) [41] is an iterative method that can make the PnP approach more robust to outliers. Outliers lead to wrong poses whereas inliers lead to correct ones. Hence, instead of trying to fit a model to a complete dataset, RANSAC fits a model to a random subset of points. This subset is chosen such that the number of inliers is the largest and thus the number of outliers the lowest. The size of a subset s as well as a threshold t_{ransac} that limits the maximum reprojection error has to be predefined. For a certain sample size s, the probability e that a point is an outlier, and a desired probability p of having at least one random sample that is free of outliers, it is possible to calculate the number of iterations N that is needed to achieve this p. This is done via the following formula [41]:

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s)} \tag{7.3}$$

7.2 Algorithm Outline

In the beginning, the intrinsic matrix of our webcam (see chapter 5) and other parameters are set and used until the program terminates. After the set-up is complete, the camera begins to capture frames that are passed to the tracking and detection algorithm described below in section 7.3 and 7.4. Since our optical flow (OF) tracking algorithm requires a set of keypoints to be tracked as well as knowledge about the pose in the preceding frame, the detection process is always triggered for the very first input frame. For detection, the object mesh along with its ORB model, which is created during the registration (chapter 6), are loaded. If the detection is successful, the OF tracking algorithm handles subsequent frames. Nonetheless, the set of keypoints passed from the detection algorithm may either get lost during the OF tracking and thus the number of tracked points decreases, or the points' drift becomes too large over time which leads to an inaccurate pose. In these cases, the detection is run again to retrieve an updated estimation of the pose. If the detection fails, it is triggered again for the next frame. Our general tracking and detection algorithm is shown in figure 7.1.

Apart from the general tracking and detection algorithm, a camera input handler captures frames asynchronously on another thread, while a certain number of frames (in our case 30) is cached. In case of a successful detection, which takes a considerably long time, these cached frames can be processed by an OF tracking method. Without caching, the OF tracking method could fail immediately due to very large movements between consecutive frames. This being said, our OF tracking processes all cached frames sequentially. Depending on its performance, the OF tracking, which is usually multiple times faster than the detection, eventually catches up with the latest frame so that the OF tracking may work in real-time. The code of the camera input handler is taken from [42]. It should be noted, that the frame rate is limited by the choice of the camera. In our case, the frame rate is 30 frames per second (FPS).

7.3 Detection

The aim of object detection is to find a specific known object in images that possibly contain that object. Information about the appearance of that object is provided by object registration (see chapter 6). In our algorithm, we do not only try to implicitly determine the position, but also the rotation of our object marker by estimating the camera pose with 6-DoF (translation and rotation in 3D space). This gives us the initial pose necessary for the OF tracking later. Moreover, another task of the detection algorithm is to select and pass sufficient keypoints to the OF tracking algorithm. In the

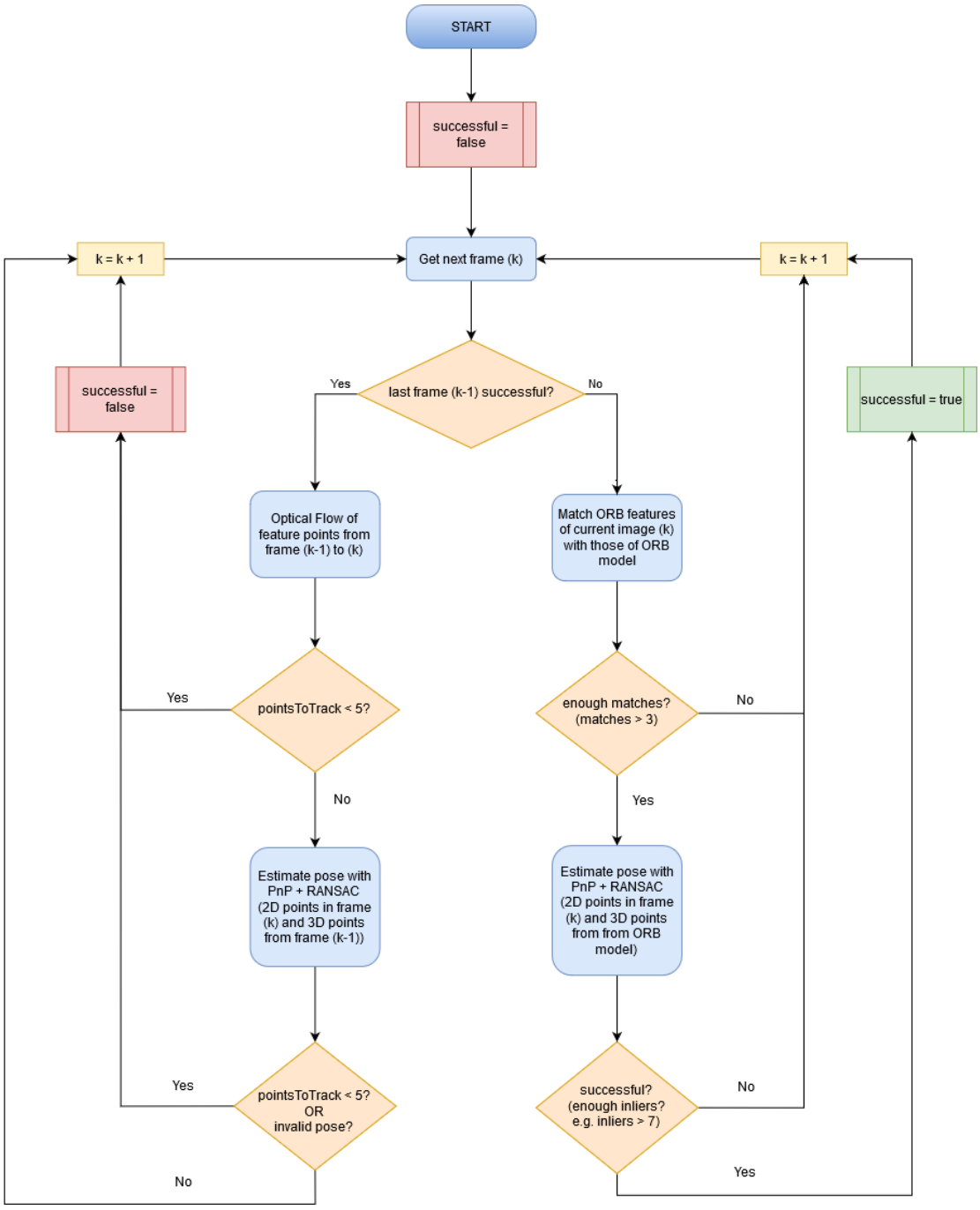


Figure 7.1: Flowchart diagram of the 6-DoF tracking algorithm

following, the detection algorithm is explained, which is a slightly modified version of OpenCV's sample code [12].

Algorithm First, from the input image we extract image features, more precisely their descriptors and keypoint locations in 2D. Their descriptors are then matched with the ones of the stored ORB model, so that we can identify the 3D correspondences of the found 2D keypoints. Thus, we can use those 2D-3D correspondences to estimate the camera pose by solving the PnP problem iteratively so that it minimizes the reprojection error [12]. This is complemented by the RANSAC algorithm to discard potential wrong matches. However, if less than 4 matches are found, detection has to be run again since the PnP problem cannot be solved uniquely. Same happens if the number of inliers $points_{in}$ after applying PnP and RANSAC falls below the threshold $points_{min}$, with $points_{min} \geq 4$. We chose $points_{min}$ to be 8 to ensure better results since the detected pose is assumed to be correct for the further steps in our OF tracking algorithm (see section 7.4). The detection is considered successful if $points_{in}$ is higher than $points_{min}$. In that case, OF tracking is started for the following frame.

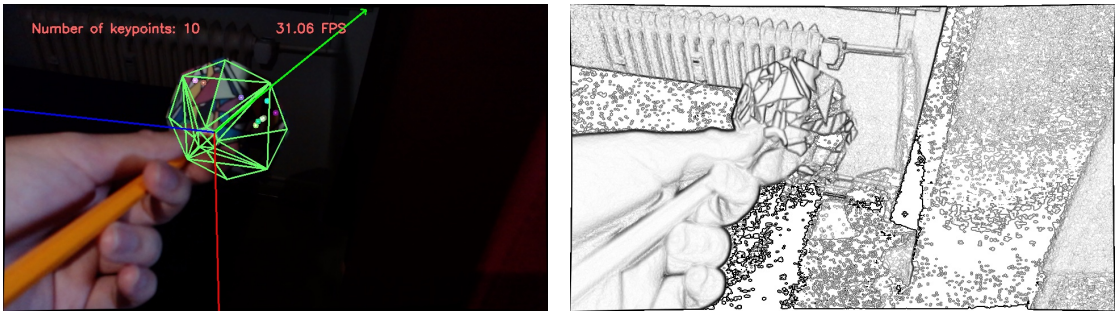
7.4 Frame-to-Frame tracking using Optical Flow

Given that our detection provides sufficient 2D-3D correspondences after feature matching, it is possible to solve the PnP problem in order to estimate the camera pose thus the object pose. The idea is, now that we have point correspondences, to only track the movement of those 2D points in 2D space while their corresponding 3D points are retained. Since the matching process is a highly costly step during detection, the frame-to-frame tracking becomes several times faster by skipping that step. Therefore, we use optical flow (OF) to track the displacement of given points in the images.

We used the OpenCV library implementation of the iterative Lucas-Kanade method with pyramids [43]. Operating on grayscale images, it calculates a sparse optical flow of the ORB features passed from the detection stage by computing displacement vectors of given keypoints from one to another image. The original Lucas-Kanade method is designed for only small motions meaning that it fails for large ones. Hence, [43] proposes an extension of that method by introducing image pyramids that is able to handle large motions. Moreover, there are two assumptions about the tracked pixels:

- the pixel intensities of the observed area do not or only slightly change between consecutive frames, and
- neighbouring pixels have similar motion. [12]

Consequently, abrupt illumination changes would influence the OF tracking and eventually lead to failure. In order to make the OF tracking illumination invariant, the pencil filter presented in chapter 3 from [11] is implemented. The effect of the pencil filter can be seen in figure 7.2.



(a) original image (b) image with applied pencil filter

Figure 7.2: Effect of the pencil filter in comparison with the original image. Edges are still visible in shadowed areas after applying the pencil filter.

Moreover, after each estimation, we check whether the estimated pose is likely to be valid. In case of invalid poses, detection is performed on the next frame instead of continuing OF tracking. A further description of that pose validity check can be found further on.

Algorithm Our OF tracking algorithm starts with tracking keypoints of the previous frame via optical flow. If not less than 5 keypoints remain after applying optical flow, the camera pose is estimated by solving the PnP problem iteratively followed by RANSAC. Here. RANSAC rejects outlier points that were possibly tracked incorrectly to maintain a likely correct pose. Then we check, whether too many points were lost after the outlier elimination. We set the threshold to 5, so that at least 5 points are needed in order to run the OF tracking again for the subsequent frame. Otherwise, we need our detection to provide sufficient keypoints again. Furthermore, we perform a pose validity check described in the following paragraph. This starts the detection algorithm as soon as the pose is considered not valid.

Pose Validity Check To determine invalid poses, we implemented two tests based on the assumption that neither sudden large changes in translation nor rotation are usually intended during the OF tracking. Invalid poses may appear, for example, when there is pose ambiguity such that the minimization of the reprojection error yields multiple possible poses with very similar reprojection errors. Thus, one test checks

the translation of the object by comparing its mean translated distance from the last n frames to the latest translation distance, while the distance is defined by the Euclidean distance. If the difference between those two values is larger than a certain threshold, the pose is discarded and detection is started.

Same applies to large rotational changes. The other test calculates the distance between rotations of the rotation matrices $R_{current}$ and R_{last} of the current and last frame measured by an angle θ. This is done by using the following formula taken from [44]:

$$\theta = \arccos\left(\frac{\mathrm{tr}(R_{diff}) - 1}{2}\right) \tag{7.4}$$

with trace of R_{diff}

$$\mathrm{tr}(R_{diff}) \tag{7.5}$$

and difference rotation matrix

$$R_{diff} = R_{current} R_{last}^{T} \tag{7.6}$$

and transpose matrix of R_{last}

$$R_{last}^{T} \tag{7.7}$$

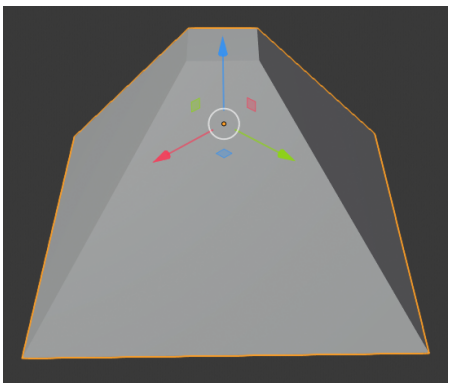
Again, if the rotational change is higher than a predefined threshold, the pose is considered invalid thus discarded.

8 Design of Object Marker

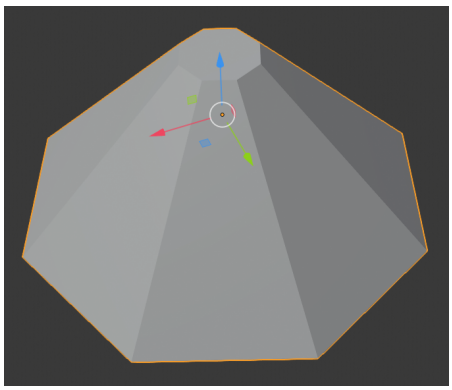
The future application should consist of a camera attached to the user and an object marker that is attached to the end of a pen, so that it can function as an augmented reality raycasting interactable in a similar way to a laser pointer. Generally speaking, it is possible to detect and track any arbitrary object. However, for previously stated purposes, some objects with certain shape and texture may work better than others. Therefore, we tested different designs for our object marker which are presented in this chapter.

General Approach To create a model of the object marker, we used Blender version 2.81 [45]. The resulting models (see appendix E) of the object part relevant to the tracking is shown in figure 8.1. For each prototype, while it is in use, we modeled the object so that there would be a relatively large area of the object marker visible to the camera that is attached to the user. Furthermore, we added an elongated slot to this object in such a way that it can be plugged onto a pencil. The object model is then printed by a 3D printer, which is in our case *Ultimaker 3* (see appendix B). We prepared the object model for 3D printing using the program *Ultimaker Cura* version 4.4.1 [46]. Before printing, the 3D models should be checked for non-manifold geometry which is a common cause for issues during 3D printing. Finally, we pasted the object marker up with printed texture that is supposed to be good to track (see figure 8.2).

Shape The first prototype is an upside down square pyramid. The detection functions well and the tracking is stable as long as at least two sides of the pyramid are clearly visible to the camera. Here, clearly visible means having good lighting without strong reflections and only low levels of blurriness. Otherwise, all points matched during the detection stage lie in the same plane. As it is mentioned in section 7.1, our PnP approach cannot deal with coplanarity well, partially leading to multiple possible solutions for the pose. Hence, our second design is a pyramid with octagonal base that has more faces to decrease the cases of having coplanar points. Additionally, for enhanced usability, the base is formed to be more pointy. This part is not tracked and serves mainly as enhanced visual feedback to the user. We call it *diamond marker* to distinguish it from the first prototype, the *pyramid marker*. The idea is that, since the



(a) Pyramid Model



(b) Octagonal Pyramid Model

Figure 8.1: Plain models of the object marker without texture. Their mesh is used during object registration (chapter 6), as well as in the tracking and detection procedure (chapter 7).

diamond marker has more faces that can be seen by the camera, it may decrease the cases of having coplanar points in one image. Both shapes are shown in figure 8.2.

Texture [47] The texture of our object marker is downloaded from a website, that offers an augmented reality marker generator [47]. To determine which texture size is suitable for our camera, we used both coarse and fine textures for comparison. However, the fine texture's details are not sufficiently captured by our camera, resulting in a low number of matches during the detection. On the other hand, the ORB feature detector struggles to find good features on very coarse textures. This is because ORB uses a corner detector, and coarse textures may contain large areas without corners. Therefore, a good trade-off between the texture sizes is necessary with respect to the expected distance between marker and camera. In the end, we opted for the texture shown in figure 8.2. Note that choice of paper and printer may also affect how well the texture can be seen under some lighting conditions: laser printers produce fairly reflective texture surfaces that can be problematic when a light source is specularly reflected to the camera. We would advise to rather use ink-jet printers.

(a) Diamond with fine texture; too small for good feature matching

(b) Pyramid with fine texture; too small for good feature matching

(c) Diamond with good texture size;

(d) Pyramid with coarse texture; the green quadrangles are too large and do not offer good contrast for corner detection

Figure 8.2: Object marker prototypes.

9 Evaluation

This chapter presents the results from our evaluation. We evaluate the detection quality as well as the optical flow (OF) tracking quality by means of FPS measurements and accuracy testing. Furthermore, the limitations and problems of our approach are pointed out.

9.1 Detection Quality

The result of the detection affects the OF tracking and thus should be as accurate as possible. Providing even illumination of the object marker enhances the detection performance. Therefore, during the accuracy tests, we ensured good lighting conditions for our object marker.

Dataset We used a tape measure to test the detection of our object marker at different distances ranging from 20 cm to 50 cm (horizontal distance from camera to object marker). To obtain the ground truth, we measured the vertical distance $d_y = t_y^c - t_y^o$ of the camera to the origin of the object marker, where t_y^c and t_y^o are the respective vertical distance between the camera and the ground plane, and the distance between the object marker origin and the ground plane. The ground plane is the surface on which the marker is standing. Then, since the horizontal distance d_z from the camera to the object marker is known through the tape measure, we calculate the ground truth of the direct distances from the object marker to the camera as $d = \sqrt{d_y^2 + d_z^2}$ by using the Pythagorean theorem. This ground truth distance is compared to the object marker-to-camera distance $\hat{d}$ according to the detection result. Finally, the detection error $E_{detection}$ is the deviation between these two distances:

$$E_{detection}(d, \hat{d}) = \|d - \hat{d}\|_1 \tag{9.1}$$

The resulting mean absolute errors

$$MAE_{detection} = \frac{1}{n} \sum_{i=1}^{n} \|d_i - \hat{d}_i\|_1 \tag{9.2}$$

from $n = 10$ samples are visualized in figure 9.2 and figure 9.1. Additionally, we count the number of needed attempts for a successful detection. Here, we consider a detection to be successful if the OF tracking does not fall back to the detection stage within one second.

Results The best results with lowest mean error are achieved at 20 cm to 40 cm horizontal distance according to figure 9.1. This could be because on the one hand the texture is captured with more details when the object lies closer to the camera. On the other hand, the focus is set to infinity, which means that objects at medium to long distances should be displayed the sharpest. However, if the object is too far away that the image quality is not sufficient to portray details, the error increases and so does the amount of necessary detection attempts. The reason for the larger number of needed attempts at closer distance could be, that the object is out of focus, or because the object registration images are taken from a farther distance (60 cm). This impacts the feature matching insofar as the features with similar scales are more likely to be correctly matched. As mentioned in section 4.4, ORB is only partially scale-invariant, so it is supposed to perform best at similar scales. Therefore, the lower amount of detection attempts needed at a medium distance could be a result of a similar registration distance, at which the registration images were taken.

As shown in figure 9.2, our data has some outliers with far above average error and number of detection attempts. Nevertheless, the general trend in figure 9.1 seems to persist despite excluding the outliers.

9.2 Optical Flow Tracking Quality

In contrast to the detection, our OF tracking is mostly illumination invariant thanks to the pencil filter (see algorithm 1). The OF tracking can cope with uneven and changing illumination well, particularly with regard to shadows (see figure 7.2). During the OF tracking evaluation we assume to have obtained correct information from the detection.

Dataset To measure the quality of the OF tracking, we evaluated the translation separately from the rotation values. The setup is shown in figure 9.3.

Concerning the translation, we placed the object marker on a start position and moved it gradually towards the camera, away from the camera and, at a certain distance, sidewards. The back-to-front movement began from 50cm, where the detection is still able to achieve enough matches, and stops at 20cm distance (here, horizontal distance from the camera to the object). On the other hand, moving from the front to the back was done between 20cm and 80cm distance. For the sidewards movement, we chose a

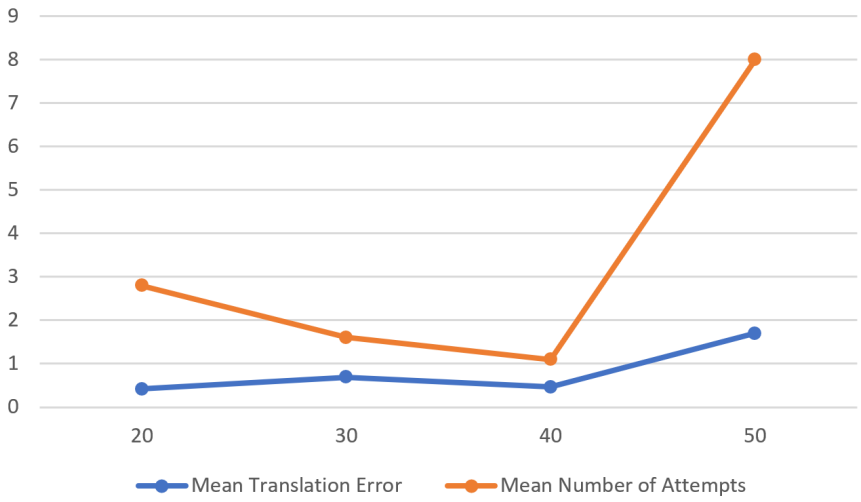


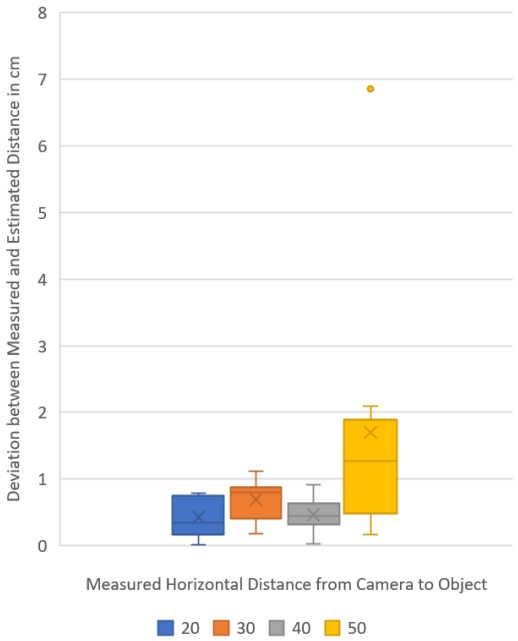
Figure 9.1: Mean translation errors between the ground truth of the object marker-camera distance d and object marker-camera distance $\hat{d}$ (see equation 9.2) according to the detection's pose estimation (blue). The diagram also displays the mean number of detection attempts needed for a successful detection (orange).

distance of 40cm. The translation data was collected in 10cm steps. Based on the start position p_s, the travelled distance of our estimation $\Delta\hat{d}_i = \|p_s - p_i\|$, where p_i is the i-th position measured, is compared with the expected travelled distance d_i to determine the object marker's translation error, which we define as:

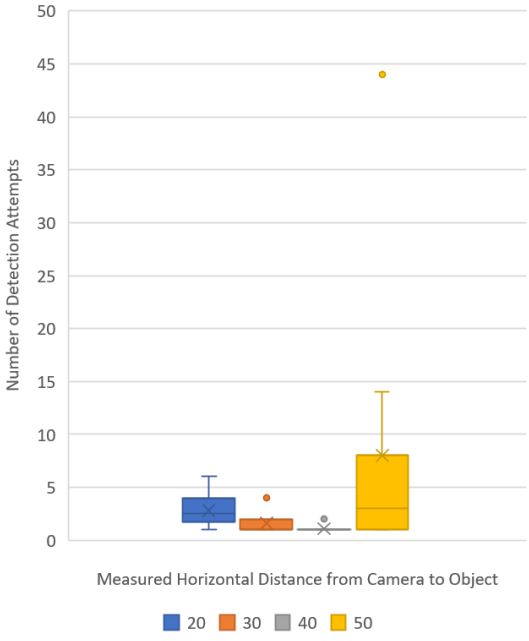
$$E_{tracking} = \|\Delta d_i - \Delta\hat{d}_i\|_1 \tag{9.3}$$

The graph in Figure 9.4 shows the mean translation errors. Note that only data of successful tracking runs were considered here, although the tracking sometimes failed prematurely.

For evaluating the rotation, we placed the marker on a turntable and rotated it in steps of 10 degrees, comparing the actual rotation with the estimated rotation by using the formula 7.4 from the pose validation check. The object marker was placed 40cm away (horizontal distance) from the camera. Similarly to the translation, the rotation difference is measured relatively to the start orientation. The data of one evaluation run (or one sample) is collected until the tracking fails and detection is started. The collected data is shown in figure 9.5.



(a) Deviation between measured and estimated distances in cm

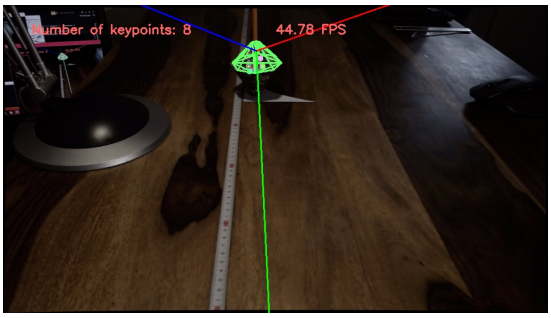


(b) Number of detection attempts needed for a successful detection

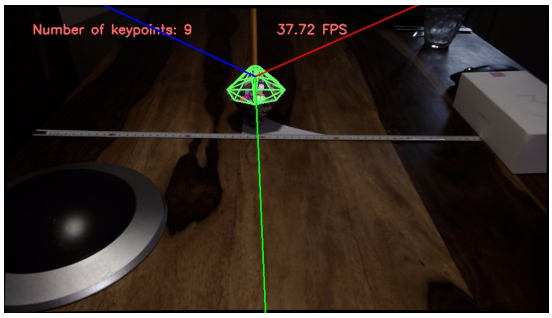
Figure 9.2: Boxplots of the detection data showing their distribution. Outliers exist in our data but they do not influence the general trend in figure 9.1.

Results Figure 9.4 shows that with increasing translation or rotation of the object marker, a larger error can be observed. However, there are some deviations between the different moving directions. While the error during the front-to-back movement increases faster, the error of the back-to-front movement rises more slowly, and the error of the sidewards movement grows linearly. Therefore, moving the object marker from the back to the front seems to be least challenging. Moving the object to the front reveals more texture details, so that the OF tracking can act more reliably. The reverse applies for moving towards the opposite direction. Finally, staying at approximately the same distance, yields a similarly even rise of error. Nevertheless, the number of samples contributing to the statistics is rather low. More samples should be taken if an in-depth comparison between the translation-directions is of interest.

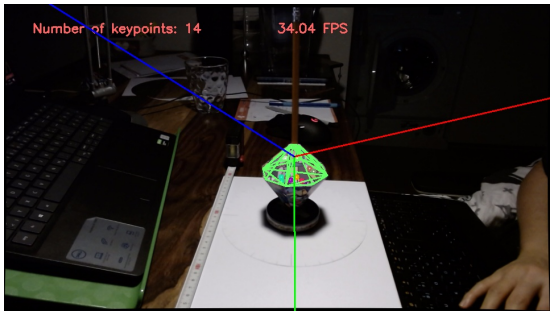
As for the rotation (see figure 9.5), the error stays around 1 to 3 degrees during the first 80 degrees of rotation, but increases drastically after that. At the same time, less samples remain the higher the rotation, meaning that the tracking was aborted from that point on during most of our tracking attempts. We observed that the remaining



(a) towards and away from the camera



(b) sidewards



(c) rotation

Figure 9.3: Setup for the evaluation of the optical flow tracking. To measure the translated distance, we used a tape measure lying along the translation axis. For measuring the rotation, we used a self-made turntable

keypoints after a 90 degrees rotation were located at the transition edge between object marker and background. Concluding thereof, it might be the case that not the keypoints on the texture are tracked anymore but instead the transition edge to the background.

9.3 Speed

The speed is measured by Frames Per Second (FPS). With a plain background and only the marker in the image, the detection runs at around 5 FPS, while the OF tracking is limited by the camera's frame rate of 30 FPS.

On the other hand, a highly cluttered background cuts the detection speed down to approximately 1.9 FPS. Our OF tracking is mostly unaffected, except for the first processed frames since the OF tracking works off all cached frames before running at real-time. The longer it takes for a successful detection, the more frames are stored in the cache, leading to a higher time delay at the start of our OF tracking. In our case, the

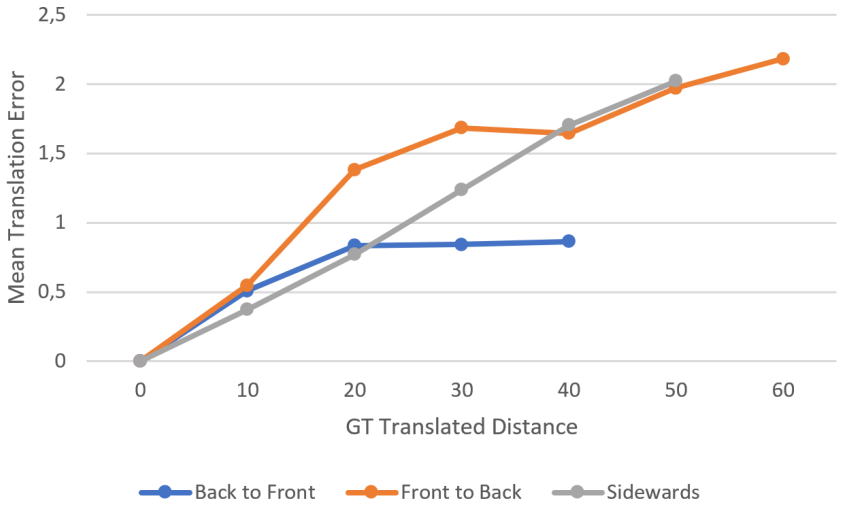


Figure 9.4: Diagram showing the mean translation errors between expected traveled distance (ground truth GT) and the traveled distance according to the OF tracking. The different translation directions are marked with blue (5 samples), orange (5 samples) and gray (3 samples).

OF tracking starts off with approximately 50 FPS and therefore catches up to real-time in about one second. The cache size should be chosen so that no frames are lost during a slow detection. We cache up to one second of old frames, which should be enough since the detection runs at approximately 1.9 FPS.

Rendering from two different distances for the object registration (see 6.2.2) leads to a higher number of matchings in the detection stage. However, due to the larger number of feature points (e.g. 5000 additional keypoints) that have to be checked during matching, the FPS during detection drops approximately by half.

9.4 Limitations and Problems

Texture Our texture used for the object marker is still not optimal. The feature points identified by the ORB detector are located very closely to each other. However, having evenly distributed keypoints would likely enhance the pose estimation (e.g. lower pose ambiguity) and raise the probability of obtaining keypoint matches from different angles. Moreover, at a large distance, some faces of the diamond marker seem to perform suboptimally at the detection. In this case, a low number of matches is found, such that a successful pose estimation is barely possible.

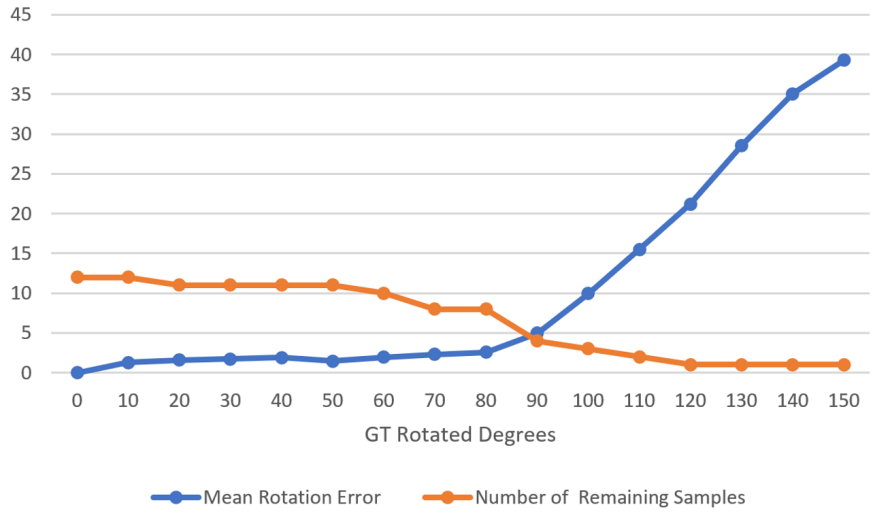


Figure 9.5: Diagram showing the mean rotation errors between expected rotated degrees (ground truth GT) and rotated degrees according to the OF tracking (blue), and the number of remaining sample data starting at 12 (orange). The position of the object marker was fixed during the data collection.

Usability The object size is relatively large compared to a common pen, which limits the usability. Moreover, the usage is limited to pens with designated diameter (in our case approximately 0.7 to 0.85 cm in diameter), but it can be customized before 3D printing. Flexibility can be enhanced by further engineering, for example by adding a clamping mechanism to support a wider variety of pens.

Detection There is room for improvement in our detection algorithm. It lacks efficiency to run at a reasonable speed. In its current state, it is suitable to only a limited extent in real-time applications. The duration of the detection until a valid pose is found can become very long, depending on the available compute power, number of feature points contained in the ORB model and in the image background, and other factors that might influence the matching performance. Furthermore, a successful detection at a distance larger than 55cm is difficult to achieve. In general, the detection has issues dealing with uneven lighting conditions, as well as overly bright illumination, and specular reflections. Hence, the brightness or exposure time of the camera should be adjusted to appropriate values. Like most visual marker based tracking systems, high levels of motion blur (e.g. due to fast movements) become problematic for our detection algorithm. The detection sometimes yield ambiguous poses that are not

detected immediately. However, obtaining correct results from the detection is a crucial precondition for proper subsequent tracking with optical flow.

Optical Flow Tracking Similar to the detection, our optical flow tracking also has troubles dealing with motion blur, especially in dark environments which lead to longer exposure times of the camera and to increased noise in the image. A certain illumination-invariance is given for the optical flow tracking by the use of a pencil filter presented in our reference paper [1]. Although the pencil filter helps to handle uneven illumination, strong specular reflections are still problematic since the texture disappears on the image. Moreover, our OF tracking fails if initially tracked keypoints become covered by any other object. Keypoints can then be carried along by the obstructing object. The same happens sometimes with keypoints located at the border of the object marker to the background. If there are strong edges in the background, they could also become part of the tracked window. This can lead to invalid poses as well, which are identified by our pose validity test. Occlusion during the OF tracking triggers a redetection, which is not quick enough to offer a smooth user experience.

10 Conclusion

10.1 Summary

In this paper, we presented an approach to track a textured object marker with 6-DoF using a single RGB camera. We designed two object markers (4-sided and 8-sided pyramid) and tested different textures in order to get better detection and tracking results. Pose estimation issues due to coplanarity can be reduced by adding more faces to the object marker. The object marker is shaped in such a way that mainly one side is visible to the camera since the overall idea is aimed to transform a pen into an interaction device with a camera attached to the user. Moreover, its texture is designed to offer sufficient distinctive feature points necessary for feature matching. These feature points are supposed to be clearly recognizable within a preferably wide spacial range during application up to one arm length away from the camera.

To register our object marker, we use synthetic data obtained through its 3D scan. The synthetic data is created by rendering the 3D scan in Unity from various poses. This facilitates the object registration procedure as the pose of the rendered 3D scan is known in Unity. While one fixed camera provides input images, our object tracking algorithm estimates the camera pose during the detection stage, and subsequently tracks the textured object in 2D using optical flow. In case of successful pose estimation, the object mesh is rendered onto the screen during tracking to visualize the estimated pose. To obtain keypoints necessary for pose estimation, our detection algorithm performs feature matching between features detected in a live camera feed and features from the registered object model. Furthermore, we implemented tests to identify likely wrong poses during the OF tracking. Implausible pose changes (e.g. flipped axes) between two consecutive frames are detected by our filter so that appropriate measures can be taken. Moreover, our OF tracking runs at up to 50 FPS on a modern laptop CPU (Intel i5 9300H) which enables smooth usage in real-time.

10.2 Future Work

Object Registration The images rendered for object registration could be taken from more evenly distributed positions. Moreover the pose from Unity should be converted

to OpenCV coordinate space instead of using screen positions (see 6.3.2) to directly use the 3D pose of the rendering camera.

Design of Object Marker Future research is needed to test more complexly shaped object markers. This may help preventing pose ambiguity.

Use of Threading to Improve Tracking Threading offers the possibility to process information simultaneously in order to speed computations up. We suggest to integrate threading for the feature matching task during detection. Moreover, future work could include stabilizing our tracking with optical flow. We propose to run a detection on a separate thread, which matches features only within in a small window around the tracked object while optical flow tracking is running simultaneously. This may compensate the loss of keypoints during the optical flow tracking. For this to be a reasonable option, the detection speed may have to be increased. However, running detection only within a small area where keypoints were lost may be already sufficient to achieve an appropriate speed. As our reference paper [1] demonstrates, the additional information derived from the synthetic data creation can be used to enhance the robustness of the tracking with regard to partial occlusions and drift of keypoints during optical flow tracking. This could be also realized in future work.

Primary Project Objective Finally, in order to complete our primary project objective, future work could contain optimizing the integration of a mobile camera attached to the user, as well as visualizing the pointing direction of the pen (or object marker). SLAM could be used on the camera feed to position itself and the object marker relative to world space.

List of Figures

Bibliography

[1] J. Rambach, A. Pagani, M. Schneider, O. Artemenko, and D. Stricker. "6DoF Object Tracking based on 3D Scans for Augmented Reality Remote Live Support". In: *Computers* 7.1 (2018), p. 6. DOI: 10.3390/computers7010006.

[2] P. Maier. "Augmented Chemical Reactions - Research on 3D Selection and Confirmation Methods". Dissertation. Munich, Germany: Technische Universität München, 2014.

[3] P. Mistry, P. Maes, and L. Chang. "WUW - wear Ur world". In: *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems - CHI EA '09*. Ed. by D. R. Olsen, R. B. Arthur, K. Hinckley, M. R. Morris, S. Hudson, and S. Greenberg. New York, New York, USA: ACM Press, 2009, p. 4111. ISBN: 9781605582474.

[4] PTC Inc. *Vuforia*. URL: https://www.ptc.com/en/products/augmented-reality/vuforia (visited on 03/14/2020).

[5] Apple. *ARKit*. URL: https://developer.apple.com/augmented-reality/arkit/ (visited on 03/14/2020).

[6] Vuforia. *Object Recognition*. Ed. by PTC Inc. URL: https://library.vuforia.com/content/vuforia-library/en/articles/Training/Object-Recognition.html (visited on 03/14/2020).

[7] F. Neumann and T. Kuzyna. "Objekterkennung in Augmented Reality Frameworks. Möglichkeiten und Grenzen von AR-Technologien bei der Suche nach dem passenden Ersatzteil im Außeneinsatz". In: *Grenzen in Zeiten technologischer und sozialer Disruption*. Ed. by S. Molthagen-Schnöring. Beiträge und Positionen der HTW Berlin. Berlin: Berliner Wissenschafts-Verlag, 2019. ISBN: 978-3-8305-3957-5.

[8] J. Shi and Tomasi. "Good features to track". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*. Ed. by IEEE. IEEE Comput. Soc. Press, 15.06.1993 - 17.06.1993, pp. 593–600. ISBN: 0-8186-5825-8. DOI: 10.1109/CVPR.1994.323794.

[9] D. G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Ed. by IEEE. IEEE, 20.09.1999 - 27.09.1999, 1150–1157 vol.2. ISBN: 0-7695-0164-8. DOI: 10.1109/ICCV.1999.790410.

[10] B. D. Lucas and T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. Ed. by IJCAI. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1981, pp. 674–679.

[11] J. Rambach, C. Deng, A. Pagani, and D. Stricker. "Learning 6DoF Object Poses from Synthetic Single Channel Images". In: *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. Ed. by IEEE. IEEE, 16.10.2018 - 20.10.2018, pp. 164–169. ISBN: 978-1-5386-7592-2. DOI: 10.1109/ISMAR-Adjunct.2018.00058.

[12] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[13] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94.

[14] E. Rosten and T. Drummond. "Machine Learning for High-Speed Corner Detection". In: *Computer vision - ECCV 2006*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Vol. 3951. Lecture Notes in Computer Science. Berlin: Springer, 2006, pp. 430–443. ISBN: 978-3-540-33832-1. DOI: 10.1007/11744023{\textunderscore}34.

[15] E. Karami, S. Prasad, and M. Shehata. *Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images*. Oct. 7, 2017. URL: http://arxiv.org/pdf/1710.02726v1.

[16] M. Riegler. "A study and comparison of feature matching". master's thesis. 2015. URL: https://elib.dlr.de/100301/1/Riegler_MA.pdf.

[17] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. "BRIEF: Binary Robust Independent Elementary Features". In: *Computer vision - ECCV 2010*. Ed. by K. Daniilidis, P. Maragos, and N. Paragios. Vol. 6314. Lecture Notes in Computer Science. Berlin: Springer, 2010, pp. 778–792. ISBN: 978-3-642-15560-4. DOI: 10.1007/978-3-642-15561-1{\textunderscore}56.

[18] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. Ed. by IEEE. IEEE, 6.11.2011 - 13.11.2011, pp. 2564–2571. ISBN: 978-1-4577-1102-2. DOI: 10.1109/ICCV.2011.6126544.

[19] P. L. Rosin. "Measuring Corner Properties". In: *Computer Vision and Image Understanding* 73.2 (1999), pp. 291–307. ISSN: 10773142. DOI: 10.1006/cviu.1998.0719.

[20] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. "Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search". In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. Ed. by VLDB Endowment. VLDB '07. VLDB Endowment, 2007, pp. 950–961. ISBN: 9781595936493.

[21] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. 2. ed., reprinted. Cambridge: Cambridge University Press, 2004. ISBN: 0521540518. URL: http://www.loc.gov/catdir/description/cam051/2004557531.html.

[22] K. Simek. *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix*. Ed. by Sightations. 2013. URL: http://ksimek.github.io/2013/08/13/intrinsic/.

[23] Y. Morvan. "Acquisition, compression and rendering of depth and texture for multi-view video". Dissertation. Eindhoven: Technische Universiteit Eindhoven, 2009. DOI: 10.6100/IR641964. URL: https://pure.tue.nl/ws/portalfiles/portal/3198275/200910785.pdf (visited on 02/01/2020).

[24] MATLAB. *What Is Camera Calibration? (R2019b)*. Ed. by The MathWorks Inc. Natick, Massachusetts, 2019. URL: https://de.mathworks.com/help/vision/ug/camera-calibration.html (visited on 02/01/2020).

[25] Unity Technologies. *Unity*. URL: https://unity.com/.

[26] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. ISSN: 01628828. DOI: 10.1109/34.888718.

[27] J.-Y. Bouguet. *Camera Calibration Toolbox for Matlab*. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/.

[28] F. Callari. *How to verify the correctness of calibration of a webcam?* Ed. by Stack Overflow. 2012. URL: https://stackoverflow.com/questions/12794876/how-to-verify-the-correctness-of-calibration-of-a-webcam (visited on 02/02/2020).

[29] J. W. *Calibration Best Practices*. Ed. by Calib.io. Odense, Denmark, 2018. URL: https://calib.io/blogs/knowledge-base/calibration-best-practices (visited on 02/02/2020).

[30] Autodesk Inc. *ReCap Pro: ReCap Photo*. URL: https://www.autodesk.de/products/recap/overview.

[31] M. Goehring. *Tips for taking pictures for good photogrammetry*. Ed. by Autodesk Inc. 2018. URL: https://knowledge.autodesk.com/support/recap/learn-explore/caas/simplecontent/content/building%E2%80%94data-preparation%E2%80%94taking-the-right-pictures-for-good-photogrammetry.html.

[32] junggusWns. *How to use OpenCV camera calibration to set Physical Camera parameters?* Ed. by Unity Technologies. 2019. URL: https://forum.unity.com/threads/how-to-use-opencv-camera-calibration-to-set-physical-camera-parameters.704120/ (visited on 03/01/2020).

[33] Code Monkey. *How to take a Screenshot in Unity*. Ed. by YouTube. 2018. URL: https://www.youtube.com/watch?v=lT-SRLKUe5k (visited on 03/02/2020).

[34] T. Möller and B. Trumbore. "Fast, Minimum Storage Ray-Triangle Intersection". In: *Journal of Graphics Tools* 2.1 (1997), pp. 21–28. ISSN: 1086-7651. DOI: 10.1080/10867651.1997.10487468.

[35] Y. Wu and Z. Hu. "PnP Problem Revisited". In: *Journal of Mathematical Imaging and Vision* 24.1 (2006), pp. 131–141. ISSN: 0924-9907. DOI: 10.1007/s10851-005-3617-z.

[36] X. X. Lu. "A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation". In: *Journal of Physics: Conference Series* 1087 (2018), p. 052009. ISSN: 1742-6588. DOI: 10.1088/1742-6596/1087/5/052009.

[37] S. Finsterwalder and W. Scheufele. *Das Rückwärtseinschneiden im Raum*. München, 1903. URL: http://publikationen.badw.de/de/003394486.

[38] P.-C. Wu, Y.-H. Tsai, and S.-Y. Chien. "Stable pose tracking from a planar target with an analytical motion model in real-time applications". In: *2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP)*. Ed. by IEEE. IEEE, 22.09.2014 - 24.09.2014, pp. 1–6. ISBN: 978-1-4799-5896-2. DOI: 10.1109/MMSP.2014.6958793.

[39] K. Levenberg. "A method for the solution of certain non-linear problems in least squares". In: *Quarterly of Applied Mathematics* 2.2 (1944), pp. 164–168. ISSN: 0033-569X. DOI: 10.1090/qam/10666.

[40] D. W. Marquardt. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". In: *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441. ISSN: 0368-4245. DOI: 10.1137/0111030.

[41] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395. ISSN: 00010782. DOI: 10.1145/358669.358692.

[42] Kriticals. *Webcam Reader: Opencv Webcam Reader with Threading*. Ed. by GitLab. 2020. URL: https://gitlab.com/Kriticals/webcam-reader.

[43] J.-Y. Bouguet. *Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker Description of the algorithm*. Ed. by Intel Corporation. 2001. URL: https://pdfs.semanticscholar.org/aa97/2b40c0f8e20b07e02d1fd320bc7ebadfdfc7.pdf.

[44] B. Belousov. *Distance between rotations*. 2016. URL: http://www.boris-belousov.net/2016/12/01/quat-dist/.

[45] Blender Online Community. *Blender*. Blender Institute, Amsterdam. URL: http://www.blender.org.

[46] Ultimaker. *Ultimaker Cura*. URL: https://ultimaker.com/software/ultimaker-cura.

[47] Brosvision. *AUGMENTED REALITY MARKER GENERATOR*. Ed. by Brosvision. URL: https://www.brosvision.com/ar-marker-generator/.

Appendix

The following contents are available on the GitLab repository `https://gitlab.lrz.de/IN-FAR/Thesis-Projects/ba-minshan_loung-trackable_pen`:

A Structure Sensor Product Specification Sheet

The product specification sheet of Structure Sensor (ST01).

B Ultimaker 3 Product Specification Sheet

The product specification sheet of Ultimaker 3.

C Source Code Visual Studio Solution

The Visual Studio 2015 Solution with source code in C++.

D Source Code Unity Project

The Unity project including source code in C#.

E 3D Models

The 3D scanned models of the object markers as well as the Blender models.

F LaTeX

The LaTeX source code and PDF file.