David Kikelj

Remote Instructions for Mobile AR

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

SUPERVISOR
Schmalstieg, Dieter, Univ.-Prof. Dipl.-Ing. Dr.techn.
Institute for Computer Graphics and Vision

ADVISOR
Kalkofen, Denis, Dipl.-Ing. Dr.techn.
Institute for Computer Graphics and Vision

Graz, Austria, Mar. 2017

To my family

The important thing is not to stop questioning. Curiosity has its own reason for existing.

Albert Einstein (1879 - 1955)

Abstract

Presenting clear and understandable instructions from a remote location is generally a challenging task due to the vast diversity of operable objects in the modern world. The growing public interest and acceptance in augmented reality and the omnipresence of smart-devices has enormous potential to bring information representation to a new level. Because augmented reality offers an intuitive and understandable way for presenting additional information, and the computational power of smart-devices grew rapidly in the past years, these two combined form a good basis for presenting additional information in everyday life.

In this thesis, smart-devices are used to obtain a textured model of the present environment, which is sent to a remote person, who can add additional information to describe physical interactions. This information consists of animated, geometric primitives, which represent the actual control elements and their motion. The added instructions are then displayed as moving augmented reality objects on the filming persons smart-device.

With this method of describing instructions, it is possible to describe diverse tasks in an intuitive way, like performing car maintenance, or depicting how to play a song on the piano.

Keywords. augmented reality, instruction representation, SLAM, reconstruction, IBR

Kurzfassung

Aufgrund der Vielfalt von zu steuernden Geräten in der heutigen Zeit ist die verständliche Darstellung von Arbeitsschritten aus der Ferne prinzipiell eine herausfordernde Aufgabe. Das wachsende Interesse und die steigende Akzeptanz von Augmented Reality und die allgemeine Verfügbarkeit an Smart Devices birgt enormes Potenzial für die Repräsentation von Anweisungen. Augmented Reality bietet einen gut verständlichen und intuitiven Weg zur Darstellung von zusätzlichen Informationen, und die Rechenleistung von Smart Devices hat in den vergangenen Jahren immense Fortschritte gemacht. Daher bietet es sich an, die Vorzüge von beiden miteinander zu kombinieren, um zusätzliche Informationen im Alltag verfügbar zu machen.

In dieser Arbeit wurden Smart Devices verwendet, um ein Modell von der Umgebung zu erstellen und dieses an eine entfernte Person zu übertragen. Nun können zusätzliche Informationen an das Modell angebracht werden, um Arbeitsabläufe zu beschreiben. Diese Information besteht aus animierten, geometrischen Primitiven, die die tatsächliche Bewegung des zu bedienenden Elements nachahmen. Die hinzugefügten Arbeitsanweisungen werden anschließend als Augmented-Reality-Objekte auf dem Smart Device angezeigt.

Diese Methode, Instruktionen zu vermitteln, macht es möglich, verschiedenste Arbeitsabläufe und Bewegungen wie zum Beispiel einzelne Aufgaben bei der Wartung eines Autos oder die Tastenfolge eines Lieds auf dem Klavier zu beschreiben.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisor Professor Dieter Schmalstieg, who made this work possible, and my advisor Denis Kalkofen, who always found time to guide me when I was lost. Furthermore, I would like to thank my university professors, who provided me with all the essential knowledge for this thesis, and my colleagues for many joyful moments.

Last but not least, I want to thank my parents, my family and friends for supporting me throughout my years of study and through the process of researching and writing this thesis.

Contents

List of Figures

List of Tables

1 Introduction

This section introduces to the work, starting with the reason why this project was planned in the first place in the Motivation part of this section. Followed by the problem analysis, given the goals, which were set in the Motivation part. Here possible challenges or problems which need further consideration before diving into the project are elaborated. The last part of this first section will outline the structure of the remaining work.

1.1 Motivation

The goal was to provide a way to explain tasks on given machines, control boards or other objects which need user interaction. Given that there is an *expert* who possesses the knowledge of how to control the machine, but that this *expert* is not locally available. Therefore, a way was searched for, to simply record a 3D model of the object which needs user interaction. This model then shall be passed to the *expert* who is then in charge of applying elements which describe and clarify the user interaction with the given object.

Now, the simple approach would be to directly send the object with the added information of the *expert* to the person who is in charge of interacting with the machine, so that added information can be replayed to better understand the handling of the machine. A slightly different approach would be to only send the added information from the *expert* back to the local user and display it on the recording device of the local user as an augmented reality overlay. Another usage of this project could be a slightly modified variation of the first approach, resulting in a version where the recorded 3D model and the added information of the *expert* are saved and provided offline. This data could then be provided along with a user manual, since often, user manuals are not detailed enough or are not able to depict certain control movements needed to operate a particular machine.

Figure 1.1 from [33] compares the acquisition, representation and instantiation of different representation techniques. It can be seen that the two 2D representations—image and movies—are relatively easy to acquire and that technologies to represent and instantiate them are wide spread. The same is true for the recording of sound. When a representation of the shape of an object is wanted, the acquisition needs more complex technology like a 3D scanner and furthermore a surface reconstruction from the acquired data is needed. Different data acquisition techniques are discussed in section 2.2.

physical form	acquisition	representation	instantiation
image	document scanner	TIFF image, postscript	printer
movies	videocamera	videotape, laserdisc	television
sound	microphone	compact disc, digital audio tape	speaker
shape	3D scanner + **surface reconstruction**	**concise representation**	NC milling, stereo lithography

Figure 1.1: This table from [33] compares the acquisition, representation and instantiation of different representation techniques.

1.2 Problem analysis

The first problem needed to be addressed is the choice of which technique should be used to acquire data from an object in the real world and, what type of data is needed. It seems important to avoid the need of a complex or expensive hardware to record the object like a 3D scanner, a stereo setup like the Microsoft Kinect which is not commonly available everywhere or other techniques which require a certain device to create a model. Therefore, an overview of recording devices needs to be made to compare the results and the possibility of further processing from the generated data. It should also be considered that perhaps a combination of multiple recording devices is desirable but the constraints on availability, pricing and usability need still to be respected. Thus, a single capturing device seems to be more suited for this approach.

Scanning techniques can also be divided into groups in respect to in which space they operate. A single beam would be an example for a 1D scanning technique, while laser rangefinders, sonar or cameras usually operate two dimensional. An example for 3D scanning techniques would be light detection and ranging (LIDAR) systems. In [33] Hoppe elaborates various methods used in 3D scanning. There are contact based 3D scanners using touching probes which are either mechanical or hand-held touching probes which need to make contact to the object which is being scanned. Even though these scanners

Name	**Pros**	**Cons**
Mechanical Touch probes	accurate	slow; contacts material
Hand-held 3D digitising probes	less accurate; cheaper	slower than mechanical probes; contacts material
Time-of-flight	long distance (km); good for buildings, landscapes, ...	round-trip-timing difficult hence accuracy of distance low (mm)
Triangulation range finders	high accuracy (μm)	limited range (m)
Hand-held laser scanners	speed; precision	high cost of implementation; finding ideal beam geometry; self-interference
Computed tomography (CT)	accurate; internal structure of object	high computational requirement; cost
Passive 3D imaging	low cost (simple camera)	rely on detecting reflected radiation; need further computation to compute 3D model from pictures

Table 1.1: Recording devices to acquire surface data from a real scene.

are very accurate, they are extremely slow and another drawback is that the material of the scanned object must withstand the touching process.

Other scanners, which do not need to *touch* the object directly, are for example time-of-flight 3D scanners which use a laser range finder to reconstruct the surface of an object. Another similar technique uses triangulation. Here a laser is used to probe the object as well, but different to the previous method, a camera is used to detect the laser dot on the surface. Table 1.1 shows an overview of some different 3D scanning techniques and their pros and cons.

Besides the physical device, there also has to be made a choice of what technique should and can be used to process the incoming data into data that is describing the scene in a way which permits further processing. The knowledge that the recorded data will be processed into a model and that this model furthermore needs to be textured, impose several constraints. For the modelling part, the data needs to describe the scene in an accurate manner to obtain a clear model from the recorded scene. The fact, that besides the need of outlining the underlying geometry also images for texturing are needed, constrains the algorithm restrictions even further. Considering that there is no prior information available about the surface or the environment which is to be modelled, imposes another important constraint for the data acquisition algorithm.

After deciding on a suitable capturing routine, the recorded data needs to be processed by an algorithm to obtain a 3D model from an object in the world. These algorithms usually belong to the reconstruction category which offers various approaches to solve this problem. An overview over the available techniques needs to be made to be able to find a suitable algorithm which can create a model from the recorded data under the made restrictions. The available methods should also be considered when choosing an algorithm for processing the incoming data to find suitable interfaces between these two. Since the created model must be transported to another location, the portability of the created model should also be respected.

When a plain model is available the next step is to find a technique of applying the recorded images onto this model to make its appearance more realistic. To apply textures on a given model, texture coordinates come in mind but since they are not available, another way must be found. Thus, having the coordinates of the camera positions from where the pictures of the scene have been made seems to be crucial. This adds yet another constraint to the data recording and processing algorithm.

With the prior in mind, the constraints on the data recording device can be listed relatively accurate by being:

- The selected sensor needs to be commonly available.
- The device should also be in a lower price range.
- The usability of the device should be intuitive and straightforward.
- This adds the portability constraint to simplify its operation.
- For the texturing, it needs to have at least a part which is able to record images.

With the constraints for the recording device known, the next part to think about is how to process the recorded data. The following list contains the constraints for the data processing part of the project which must be able to prepare the incoming data in a suitable way for further processing. These constraints are a bit vaguer and more general, because they are dependent on the choice of the recording device and its output, which can not be clearly defined at this point yet. With the given knowledge, they can be described as follows:

- Needs to be able to process the recorded input data.
- Has to be able to create data which is describing the scene.
- Has to be able to record or approximate camera positions.
- Must not need prior information about the scene.
- The created data needs to be available for further processing.
- The created data should be in a commonly used format to not restrict further processing.

When a suitable solution for the data processing part is selected, the recorded data has to be processed into a representation of the given scene. Usually a 3D model of the filmed scenery is wanted, thus a reconstruction algorithm is needed which is capable to process the given data. There are fewer constraints for the reconstruction algorithm which generates a 3D model of the recorded data. These constraints can be summarised as follows:

- Must have a suitable interface to the data processing output.
- To grant that the generated model is portable, its memory footprint must be kept low.
- The created model should be in a commonly used format to not restrict further processing.

Now, the geometry representing the scene is available, but the visual appearance of it has to be improved further. This task is usually done by applying textures to the created model to make it more realistic. The constraints on the sensor and the data processing part guarantee that images of the filmed object are available, so these have to be processed next. The texturing part is also restricted by some constraints in this scenario. These constraints can be outlined as:

- Needs to demand a model which it textures as input,
- Needs to demand images of the scene, which have to be applied on the model, as input.
- Needs to be able to understand the given camera positions to apply the textures.

These constraints must be respected when selecting solutions for the separate parts of this approach. When methods for the different parts have been found and a textured model is created, the next problem is to find a suitable way to describe the interaction with the scene in an intuitive way. Constraints on this part can be listed as follows:

- The represented interaction needs to be intuitive.
- The information needs to be easy to apply onto the created model,
- and needs to be adjustable.
- The information must be transportable with the rest of the model or separately.

These constraints for each of the different parts of this approach lead to a better understanding of the demands which are made regarding the separate systems. They also roughly outline the desired interaction between the selected methods.

1.3 Organisation of the work

This section gives a brief overview of the structure of this work to inform the reader, where individual topics can be found and what they contain. Additional information about the contents of the following chapters can be found at the beginning of each section, where their topics are outlined.

The current chapter—Introduction—is followed by the chapter 2 Related Work, which is structured into three main parts with various sub-sections. The first part of the Related Work chapter 2.1 is named SLAM and gives an introduction to Simultaneous Localization and Mapping. The roots of SLAM are in robotic research, where navigation in unknown areas is needed. The two core problems in SLAM, which are creating a map of an unknown environment, and the determination of the own position in the created map, are introduced and several techniques approaching these problems are presented. The first sub-section—2.1.1 History—of this chapter presents the historically first approaches to SLAM, followed by the next section, which presents a general definition of the SLAM problem. Chapter 2.1.3 elaborates different ways to classify SLAM. After the first three sections gave an historical background, defined the SLAM problem, and presented ways to classify the different approaches, the next chapter—2.1.4—presents various ways which can be used to approach the SLAM problem. In 2.1.4.1, the fundamentals of the Extended Kalman Filters are presented, followed by the Graph-based Optimization approach presented in 2.1.4.2. Afterwards the works on Particle Filters and Bundle adjustment are presented in 2.1.4.3 and 2.1.4.4 respectively. Finally, section 2.1.4 is concluded with 2.1.4.5, which is a brief introduction to the Mapping part of SLAM, and presents two methods used for proper map representation—topological-maps and grid-maps. After different methods approaching the SLAM problem have been introduced, chapter 2.1.5 talks about sensors which are needed by SLAM systems to obtain information of the environment. The last section of SLAM—2.1.6—introduces in its sub-sections to different open problems of SLAM, which are still being objects of researches. The problems presented here are multiple objects, which is tightly related to data association, or the correspondence problem. This problem is to determine which parts of a given image correspond to which parts in another image. Another open problem are moving objects, which are present in every non-static scene, like pedestrians or cars on sidewalks and roads respectively. The next problem presented is loop-closure, which is the problem of recognising already visited places, which is mandatory for building a consistent map. Finally, the exploration problem, describing the task of finding the best path to create a map efficiently, is mentioned shortly, followed by biological SLAM, which is inspired by the hippocampus of the brain, which performs a sort of SLAM algorithm to help living beings to understand their environment.

Section 2.2 gives an overview to reconstruction techniques, which are used to acquire a 3D model to provide a digital description from an object of the real world. Before various reconstruction methods are presented, section 2.2.1 gives an overview of what a polygon mesh is and what its components are. Then, 3D scanning, which is a popular technique for gathering data of an object, is introduced in 2.2.2 and its sub-sections. The methods presented in the sub-sections are 3D reconstructions from multiple images, where several images from the same scene, but from different angles are used to obtain information. Shape from shading, which uses the fact that the amount of reflected light from an object differs regarding the objects orientation, coordinate measuring machines, which utilize touch probes to sample a physical object, and laser range scanners, which are also used as probes in coordinate measuring machines and use methods like triangulation, interference or time of flight. After the wide spectrum of 3D scanning was introduced, another method called reconstruction from contours, which uses data describing the contour of an object to obtain a reconstruction is presented in 2.2.3. Then surface sketching, which tracks the path of an input device used to draw or sketch a model of the scene is introduced, followed by unorganized input, which poses a more universal approach, where less assumptions and prior knowledge of the scene is needed.

Section 2.3 introduces to Image Based Rendering, which is used to create images of 3D models from novel views. Before presenting various techniques approaching Image Based Rendering, the IBR Continuum is introduced, which categorises the different IBR techniques by the amount of geometry used. The first IBR technique presented is texture mapping 2.3.1, which is a simple two-dimensional approach to image based rendering and refers to the process of applying an image, also called texture, onto a 3D surface of a model. The next presented method is view-dependent texture mapping 2.3.2, where several pictures of the same object from different points of view are being processed to mitigate problems of texture mapping, which are for example the lack of being able to display complex effects like reflections, transparency or highlights. After view-dependent texture mapping, billboards are presented in 2.3.3, which basically relies on rendering the object as either a two-dimensional plane or two planes intersecting each other to form a cross like shape. Afterwards, the two layering techniques, image layering and layer based rendering are presented in 2.3.4. These methods try to describe a scene by creating several planar layers of it, where each layer has its own texture and sometimes transparency information. The method presented next in 2.3.5 is called 3D warping, which exploits given depth information of an image to perform IBR. To perform this method, points in an image are projected to their 3D position on the model followed by projecting the model to the screen space, which results in the output image. The next presented method is called Layered Depth Images 2.3.6, or LDI, which approaches the problems of 3D warping by not only respecting the visible parts of the input images, but also the occluded parts. Afterwards, view interpolation is introduced in 2.3.7, which differs to the previously mentioned techniques, by using implicit instead of explicit geometry. By utilising the information of two input images and the optical flow between them, images of different viewpoints can be created. The next presented method is view morphing 2.3.8, which is also an implicit method and is able to calculate greater camera angles than view interpolation. The next section 2.3.9 introduces light field and lumigraph, which both make use of the plenoptic function to perform IBR, and operate in a similar way. Then, the method named concentric mosaics is introduced in 2.3.10, which also uses the plenoptic function, but constrains the camera motion to planar concentric circles. Finally, image mosaicking techniques are presented in 2.3.11, which demand that multiple images from a scene are being taken which are then mapped onto a cylindrical or spherical representation. To conclude the IBR chapter, the advantages of the presented techniques are discussed in 2.3.12, before moving to the next chapter.

After the related work has been presented, chapter 3 discusses how the problems stated in the problem analysis section 1.2 have been approached and then presents solutions, which have been selected for this work. In section 3.1 the needed tools to approach the problem are elaborated by walking through an example problem situation. While progressing in the example situation, the different approaching problems are addressed and solutions are compiled. The three steps of the given scenario are being walked through in the following sub-sections, starting with the initial scenario, progressing to the data processing scenario, and finally the instruction scenario. After the scenario is concluded, chapter 3.2 introduces the processing pipeline, which contains the needed steps to successfully approach the problems shown in the scenario. The processing pipeline is divided into four steps which are the gathering-step, which deals with the data acquisition needed for the following steps, followed by the modelling step, which is in charge of processing the gathered data into a 3D model. The texturing-step describes the process of applying textures to the created model by using the gathered pictures and finally the interaction-step, where the methods of applying and viewing instructions are handled. These steps are elaborated in the sections 3.2.1 to 3.2.4.

With the processing pipeline present, which is a fundamental guide for the wanted project, chapter 4 goes in depth with the implementation details. Beginning with introducing the SLAM system, which has been added to gather data of the surrounding scene in section 4.1.1 and presenting another system in 4.1.2, followed by the created interface for SLAM systems, which allows to add more SLAM systems to this project. After the interface is described in section 4.1.3, it is shown, how PTAM has been adjusted to match the declared interface of this project and outlined what tasks need to be made when including another SLAM system like LSD-SLAM. To conclude the SLAM part of this chapter, some key-functions of the interface are described in detail. Section 4.2 describes the challenges of adding a reconstruction method to this implementation, followed by describing the chosen reconstruction method in detail. Section 4.2.2 describes why filtering techniques had to be applied and what information and tools PTAM provided regarding the data which can be discarded without further consequences. The last section dealing with filtering describes what filtering methods were used outside of PTAM and how the data was improved by using them. Section 4.3 describes what data was available for image based rendering and which method was used. Furthermore is described, what changes or interfaces needed to be made, to support the chosen IBR implementation in 4.3.1. Section 4.4 is the last part of this chapter and gives an overview of the instruction handling. Section 4.4.1 describes the chosen types and forms of the implemented control elements, and how their appearance and animation was dealt with. The last section 4.4.2 of this chapter introduces two methods of displaying the added information to the user.

Chapter 5 presents the results of this work with various examples of different scenes and situations. The selected scenes feature a motor compartment of a car in section 5.1, which also shows how the implemented user interface can be used to apply and animate a control element. The next example shows the control panels of a cooker in section 5.2 with an applied and animated control element. The following example in section 5.3 displays a magic cube, which shows how this project can handle smaller objects. The results of filming a piano and a keyboard can be found in section 5.4 and 5.5 respectively. The control elements applied in the keyboard example are of different shape and have various animations added, the piano example shows the results of a different type of animation, which allows to guide the user on playing the piano by making the control elements perform jumps from one key to the next. In section 5.6 a fuse box with static control elements can be seen. Section 5.7 discusses problematic scenarios and circumstances, as well as their consequences.

Finally, chapter 6 recaps this thesis by giving a short overview of the fundamental topics of this work. Starting with the underlying scenario of a local and a distant person and a given problem situation. Followed by discussing the applied solutions, and concludes with a résumé of this work. The second part of this chapter discusses possible future extensions and optimisations of this work.

2 Related Work

This section gives the reader a general overview over the principles of the technologies used to plan and implement this project. The first sub section will describe the idea and history of SLAM and then goes a bit deeper into this part and describes the theoretical solutions for this problem. After that, the next part will talk about approaches to reconstruct a model with given information of it and describes the different problems resulting from different input data. The third and last part of this section wants to give a short overview over IBR and discusses the two major different approaches to this problem.

2.1 SLAM

The problem named SLAM—acronym for Simultaneous Localization and Mapping—has its roots in robotic research and tries to make robot navigation in unknown areas possible. There are various approaches to solve the SLAM problem and they find use in all kinds of autonomous 'robots' like self-driving cars, domestic robots like a vacuum cleaner or a lawn mower, autonomous aerial or underwater vehicles. They are also needed in planetary rovers or microscopic machines in the human body. The general scenario behind this problem is that there is a robot whose task it is to explore a region which has no background information available yet. Now the robot has to create a map of its environment and at the same time needs to use this newly generated map to determine its own position within this map. Even though this sounds to be a chicken-and-egg-problem, there are various works, with different solutions, approaching this challenge. These works can be categorised into three different groups. The first approach uses the extended Kalman filter (EKF) to estimate a solution for the SLAM problem. The next tries to describe the SLAM problem as sparse graph constraints and then uses nonlinear optimisation to solve these constraints resulting in the environment map and the location of the robot within it. The third and final approach uses particle filters to calculate results for the SLAM problem. It needs to be said that simultaneous localisation and mapping should not be viewed as a single algorithm but more like a bigger concept. As Riisgard et al. stated in [61], most of the published works in this field of research who introduce novel approaches or algorithms to this problem, focus on developing small parts of SLAM and do not produce entirely new systems to solve SLAM. These small parts of SLAM are landmark extraction (or feature extraction), data association, state estimation, state update or landmark update. This resulted in many different approaches for the small sub problems. Furthermore, different solutions to the sub problems were exchanged and reused to be able to produce better results.

2.1.1 History

The work of Randall C. Smith and Peter Cheeseman in the year 1986 named 'On the Representation and Estimation of Spatial Uncertainty' [74] is definitely one of the first approaches to SLAM. They were working on 'a general method for estimating the nominal relationship and expected error (covariance) between coordinate frames representing the relative locations of objects'([74]). They published another work four years later in 1990 called 'Estimating Uncertain Spatial Relationships in Robotics' [73]. In this work they introduce a representation for spatial data which they call 'stochastic map' and how the building, reading and updating of the map works. In this map, relationships between objects including their uncertainties are stored.

Another early work regarding the SLAM problem was published in the year 1991 by John J. Leonard and Hugh F. Durrant-Whyte. Its name—'Simultaneous Map Building and Localization for an Autonomous Mobile Robot' [42]—pretty much defines the problem called SLAM today. They originally called it SMAL, but the name was changed later to SLAM. They had a robot with multiple sonar sensors and tracked the environmental features available in the initial position to determine the position of the robot.

These three works describe the pioneer work to simultaneous localization and mapping and also introduce the traditional way to approach this problem by using extended Kalman filters. Today there are plenty of solutions which approach the SLAM problem in various ways and which encountered new obstacles during the improvement of their algorithms. Some of these methods will be described in section 2.1.4 and in the sub-sections of 2.1.6, different open problems of current SLAM systems are presented.

2.1.2 Definition of the SLAM Problem

When approaching the SLAM problem, the wanted parameters are:

- An estimation of the pose of an agent to be able to create a map of the environment.
- A map of the environment to be able to determine the position of the agent.

Even though this seems to be a chicken-and-egg problem there are various works approaching and solving it on different ways. The general scenario for the SLAM problem is that there is an agent—also referred to as a robot—who is sent into an unknown area. The robots task is to create a map of this area and determine its own position in that newly generated map. To achieve this, the robot has to be mobile to advance in the unknown area and is equipped with one or more sensors to retrieve data of its vicinity. To write the problem in probabilistic terminology, some terms need to be defined.

Let $\mathbf{X_T}$ be the history of positions where x_0 is the starting position and its successors are incrementally updated and x_T being the terminal time.

$$\mathbf{X_T} = (x_0, ..., x_T) \tag{2.1}$$

The history of observations over time t can be written in the same way as

$$\mathbf{O_T} = (o_0, ..., o_T) \tag{2.2}$$

The history of the motions of the robot containing a series of control inputs which navigate the robot, describes the motion from each point in time to the next point.

$$\mathbf{U_T} = (u_0, ..., u_T) \tag{2.3}$$

It is needed to find a relation between the motion u_t and the position x_t. This is commonly written in a probabilistic distribution

$$\mathbf{P}(x_t|x_t - 1, u_t) \tag{2.4}$$

Similarly the observations o_t can be related to the actual environment m

$$\mathbf{P}(o_t|x_t, m) \tag{2.5}$$

Now the SLAM problem can be phrased as being the problem of retrieving a model of the environment m as well as the history of the locations X_T of the observer or the current position. Figure 2.1 is a graphical description of how the described variables interact with each other over time. Here the two main forms of the SLAM problem can be seen where the first may be called the full SLAM problem where the entire path X_T is desired.

$$\mathbf{P}(X_T, m|U_T, O_T) \tag{2.6}$$

This definition is also known as the 'offline' SLAM problem. The conditions on the right side—the history of observations and the history of the robots motion—are directly observable. The 'offline' problem often is approached in processing the entire data at once and not incrementally. This leads to the second definition, which is the 'online' SLAM problem.

$$\mathbf{P}(x_t, m|U_T, O_T) \tag{2.7}$$

It can be seen that the 'online' problem does not desire to retrieve the entire path of the agent, but only wants to obtain the current position. This presumes that algorithms

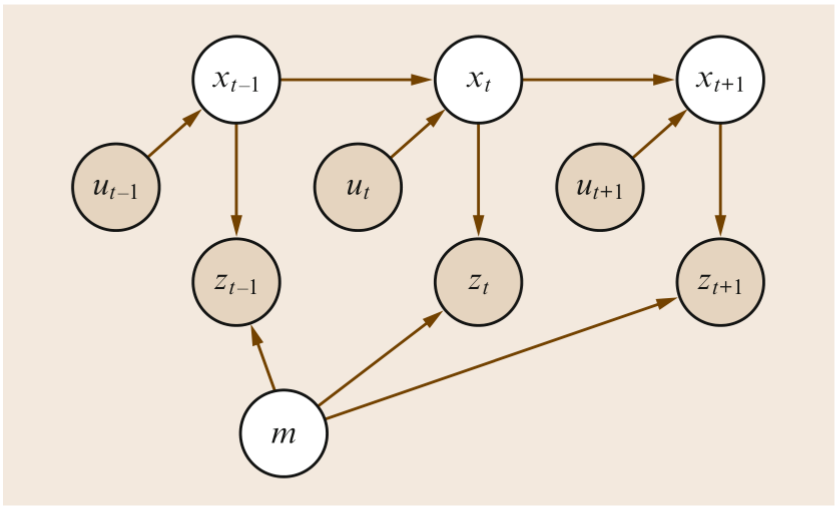


Figure 2.1: Graphic from the 'Springer handbook of robotics'[71], depicting the interaction of the different variables in the SLAM problem

approaching the 'online' SLAM problem need to be incremental.

2.1.3 Classifications

There are several other ways to classify SLAM besides the 'online' and 'offline' approaches. It can be differentiated between the ways the input data is processed. Examples would be the *volumetric SLAM*, that uses high resolution input, which even allows photorealistic reconstructions. The drawback of this approach is that it demands a lot of computational power to process the high dimensional data. Another way to approach the classification is the *feature based SLAM*, which uses very sparse sensor data and therefore the generated map results to be more sparse as well. Another way to distinguish SLAM can be how they interpret the given input. 'Topological SLAM' for example may only connect a set of places via relations with each other without any further knowledge. The other extreme would be a metric approach where metric units describe the distances between places exactly. These two examples can give some insight in how SLAM systems can be classified.

2.1.4 Methods used for SLAM

This section shows the most common approaches to the SLAM problem, which also form the basis for most of the other solution attempts. The 'Springer Handbook of Robotics' [71], categorises them into three main SLAM paradigms. These categories have been adopted and supplemented for the presentation of the different approaches in this work.

2.1.4.1 Extended Kalman Filters

Historically the first paradigm was the extended Kalman filter mentioned in the works of Smith and Cheeseman [73, 74]. Therefore there are many works using this approach, but recently they became less popular due to its computational properties. With the Kalman filter, a solution to the online SLAM problem shown in equation 2.7 can be achieved. In the early works a state vector was used to estimate the location of the agent as well as a set of features retrieved from the environment measurements. The measurements also had a covariance matrix containing the error related to the measurement. After new measurements were available the state vector and covariance matrix were updated via the extended Kalman filter. New observations result in new states in the state vector. Due to the data handling in this approach the size of both, the state vector and the covariance matrix grow quadratically. Figure 2.2 from [79] shows an example of a live SLAM system using the EKF.

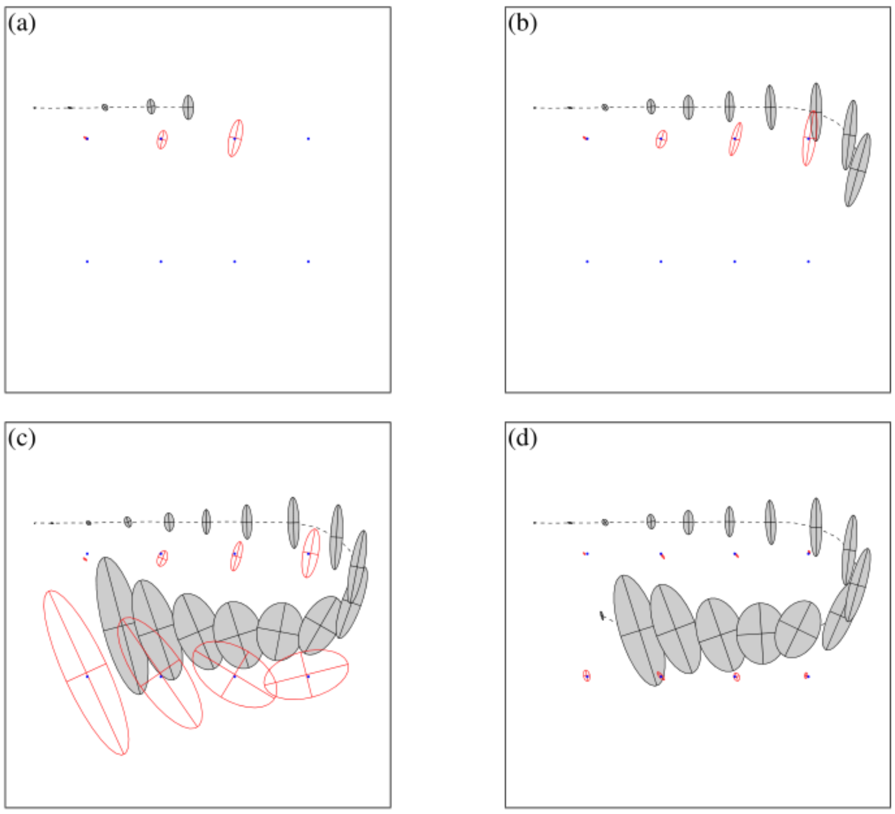


Figure 2.2: This figure from [79] shows an online SLAM problem using EKF. The path of the robot is depicted as the dotted line, and the estimation of the robots position are the shaded ellipses. The small dots represent eight landmarks and the white ellipses their estimated locations. From (a) to (c) the uncertainty about the robots position and the landmarks increases. In (d) the first landmark is detected again resulting in a decrease of the uncertainty.

2.1.4.2 Graph-based Optimization

The graph-based optimization techniques follow a different approach by solving the SLAM problem with nonlinear sparse optimisation. This method was first mentioned in [74], a few years later in 1997 a first working attempt was proposed by Lu, Feng and Evangelos Milios in their work 'Globally Consistent Range Scan Alignment for Environment Mapping' [45]. There also is a lot of more recent work like [40, 52].

When using this method in a general SLAM scenario of a robot in an unknown environment, the observations and the positions of the robot are stored as nodes in a graph. Every corresponding pair of positions x_{t-1}, x_t is connected with a line representing the motion needed to get from x_{t-1} to x_t. There are also lines between the locations x_t and the observations o_t made at that position. These lines represent soft constraints and by relaxing them the path of the robot X_T and the map can be acquired.

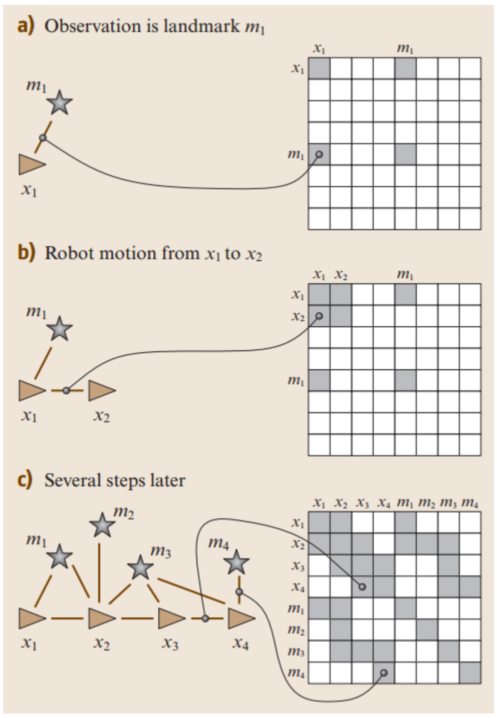


Figure 2.3: Graphic from the 'Springer handbook of robotics'[71], depicting the creation of the graph (*left*) and of the constraints in matrix form (*right*)

Figure 2.3 from [71] shows an example of the construction of the graph and the corresponding matrix. Part *a*) of the figure assumes that at time $t = 1$ the agent senses the landmark m_1, which is depicted by the graph on the left side, connecting the two events—x_1 and m_1. The right side of the figure puts the connection of the two events into matrix form by adding a value between the two elements. In the part *b*) of the figure, the agent moves from its initial position x_1 to x_2 which results in an arc from x_1 to x_2 and in new entries in the shown matrix. The *c*) part of the figure shows a more advanced stage after multiple movements and observations was stated in [71].

2.1.4.3 Particle Filters

The third and last of the three main methods used in SLAM systems is using particle filters. The principles of particle filters can be tracked back to the paper named 'The Monte Carlo Method' [49] from the year 1949. When using particle filters for SLAM each particle represents a guess of the true value of a state. These particles are collected into a set of guesses. These sets are then passed into the particle filters to get a representative sample from the posterior distribution. This is a nonparametric approach for representing the given data and became very popular in mid 1990 due to the advent of efficient microprocessors resulting in many works using it [16, 37, 62]. The problem with the particle filter approach is that the representation of both, the map and the agents path are very big. Systems using particle filters scale exponentially with the dimension of the processed data. Three or four dimensional data is processable but the higher the dimensions go the more time is needed to solve the problem. Montemerlo and Thrun presented their particle filter based algorithm named FastSLAM in 2002 [53]. They were able to use their algorithm on 50.000 landmarks. This represents an environment which far surpasses the capabilities of previous works. Figure 2.4 from [78] compares the mapping of a cyclic environment of the lazy version of FastSLAM with conventional techniques.

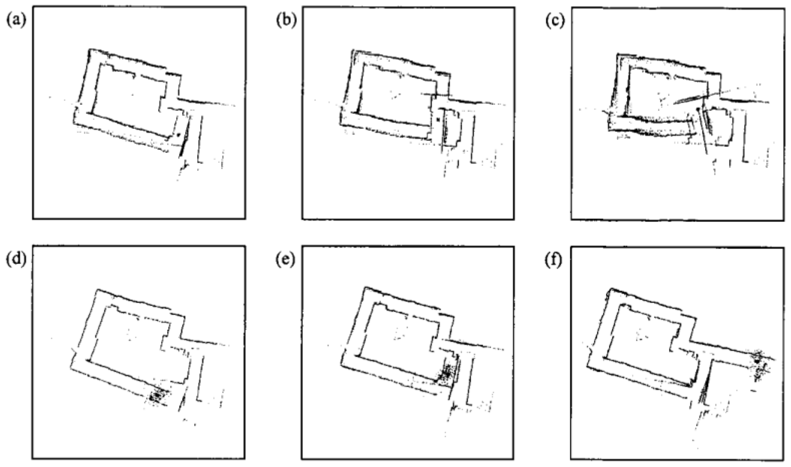


Figure 2.4: This figure from [78] compares the results of conventional techniques (top row) with the lazy version of FastSLAM (bottom row).

2.1.4.4 Bundle adjustment

Another method besides the mentioned statistical approaches is bundle adjustment, which is also a popular method used for SLAM with image data. Bundle adjustment seems to be well suited for the SLAM problem because it tries to refine the 3D coordinates which describe the scene geometry while also trying to calculate the relative motion of the camera providing the data of the environment.

2.1.4.5 Mapping

The task of the mapping part of SLAM is to generate a map of the surrounding scene. Two examples of a proper map representation would be a *topological-map* or a *grid-map*. *Topological-maps* are used to connect different parts of the environment to one coherent map and to be closer to a global consistent map. *Grid-maps* are a different approach, which uses an array of cells for the representation. These cells are being filled during the map creation and the filled cells will be marked as occupied. To simplify the calculations, these cells are usually assumed to be statistically independent. An example of a SLAM system using a topological map is the work of Gummins and Newman [11]. Their system operates by recognising the appearance of places in a probabilistic way.

2.1.5 Sensors

A SLAM system needs at least one sensor which is in charge of recording the environment. Sensors can be distinct not only by their accuracy but also by the way they operate. A very precise sensor would be a laser scanner to perform range measurement but a drawback for it is a reflective surface like glass or water where it cannot operate accurately. A totally different approach for range measuring would be the sonar which is measuring sound instead of light. The results of the sonar are not as good as the results from the laser but a positive aspect of the sonar is that it is much less expensive. The laser operates in a single line of measurement and the sonar can easily cover areas with a width of 30 degrees. Due to the different accuracy of both sensors the retrieved data from them differs in size and therefore in computational power needed to process them. The following list contains a small number of possible sensors for a SLAM system.

- Laser
- Sonar
- Tactile sensors
- Optical 1D to 3D
- WiFi
- Radar
- GPS

There was a strong trend towards optical sensors recently, especially towards cameras due to their relative low price and high availability. It is also common to combine various sensors with each other to compensate for each others drawbacks. An example for combined sensors can be a robot who knows about the distance of its own movement, its speed and its rotation and also has a camera attached to it. Drawbacks of the recently popular vision based sensors are that the processing of the data is computational expensive and that they are error-prone to light changes. Cameras are also used in a stereo setup, which is similar to the eyes of the human, or utilize even more cameras. An open challenge for camera based sensors is that their data contains way more information than a laser or a sonar and it needs to be determined which data is of interest and can be used to improve the SLAM system behind it.

2.1.6 Outline

To find a well suited algorithm for the SLAM problem is still an open research area, not only because of the difficulty to solve it, but also because different SLAM systems are needed for different areas of operations and the choice of the used sensor(s) also will result in different requirements of the wanted SLAM system. Even though there are solution attempts to many of the open SLAM problems they still need to be addressed furthermore to produce better, non-ambiguous results.

2.1.6.1 Multiple Objects

One of the open problems is how to deal with multiple objects. This problem is tightly related to data association or the correspondence problem. The actual task here is to determine which parts of a given image correspond to which parts in another image. The differences between the images can occur due to movement of the camera or movement of the objects in the filmed scene. A recent approach in the year 2011 proposed a novel approach named ‘A Random Finite Set Approach to Bayesian SLAM’ which presents a framework for feature-based SLAM respecting the case of uncertain number of features and uncertain data association [56]. This approach examined the probabilistic foundation of the SLAM problem and used Bayesian filtering with random finite sets to get better results than previous works on this field.

2.1.6.2 Moving Objects

Another problem which modern SLAM has to deal with are moving objects in the recorded scene. Good examples for moving objects are pedestrians or cars. Especially in crowded areas, tracking becomes a hard task. This problem is also related to the data association problem when it comes to motion-modelling of the moving objects. The term SLAMMOT—short for Simultaneous Localization, Mapping and Moving Object Tracking—is used to describe the additional variable of moving objects to the original SLAM problem. Chieh-chih Wang et al. proposed a paper regarding the SLAM problem respecting the moving object tracking addendum with their work 'Simultaneous Localization, Mapping and Moving Object Tracking' [82]. Chung et al. introduced an additional constraint in their work by adding scene prediction to SLAMMOT [9].

2.1.6.3 Loop closure

When an agent, who is performing SLAM, returns to a location it already visited earlier, and of which data has already been added to the generated map, it is wanted, that the agent recognizes, that this location was analyzed before. The problem dealing with recognising already visited places is called loop closure. A common approach to dealing with this problem is to use a separate algorithm to detect similarity of features describing locations.

In 'SLAM-Loop Closing with Visually Salient Features' [58] Newman and Ho showed how visual features can be used to approach the problem of loop closure. They stored image features of visually prominent places in a database including the time when a certain place was visited. With the help of the stored data it becomes possible to recognize previously visited places and when integrated into a SLAM system this knowledge can be used to close loops. Whelan et al. [83] introduced a SLAM system with so called non-rigid map deformations which allow alterations and corrections on the built map when loop closure is performed. Figure 2.5 shows results of this approach.

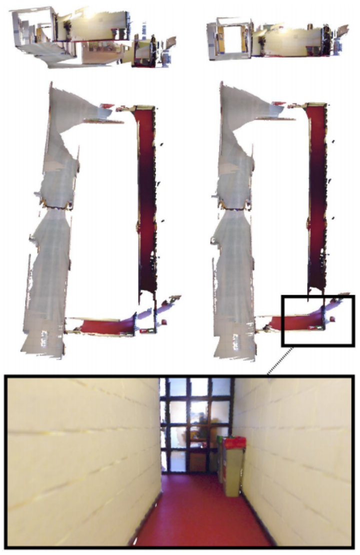


Figure 2.5: Loop closed with the method from [83]. The inset emphasizes the map consistency at the point of loop closure.

2.1.6.4 Exploration

A similar problem is exploration, which describes the task of finding the best path to create a map efficiently. Due to the fact, that the environment is unknown at the start, the initial movement has to have a good pattern to get an approximately understanding of the environment, to further plan the route.

2.1.6.5 Biological SLAM

Another inspiration for SLAM is the hippocampus of the brain, which roughly performs a SLAM algorithm to help animals and humans to understand their environment. RatSLAM used the studies from the rodent hippocampus to implement a SLAM system from the gained information [51].

In the paper ‘The SLAM problem: a survey’ [3], Josep Aulinas described the current state of the SLAM problem as follows: ‘Simultaneous Localization and Mapping (SLAM) also known as Concurrent Mapping and Localization (CML) is one of the fundamental challenges of robotics, dealing with the necessity of building a map of the environment while simultaneously determining the location of the robot within this map’ (Aulinas).

2.2 Reconstruction

This section gives an overview to the technique of acquiring a 3D model to provide a digital description from an object of the real world, also known as surface reconstruction. A 3D model can be understood as a digital representation of a physical object. The ambition of reconstruction can be summarised by assuming that there is a set of information X—commonly points—describing a wanted object U which now shall be processed to create a 3D model.

The 3D reconstruction of a model from the outside world becomes more and more relevant in several distinct application scenarios. Examples for areas where 3D models are wanted are hospitals, other medical facilities or factories, where models can be used to replace or repair damaged parts of a machine. An advantage of 3D models is that they can be distributed very fast over a digital way, for example in an internal network of a hospital. These models can also be stored in a database for later usage, if for example a part of a machine has to be replaced, the model of this part can then be conveniently retrieved from this database. Another advantage of 3D models over physical models, which could be made of clay or other materials, is that the 3D models can not only be viewed and analyzed, but also can be edited and adjusted more easily.

There are different approaches of retrieving information about a given object. Depending on the type and structure of this data, different algorithms have been developed to reconstruct a surface. They can be categorised into the conventional three big categories which are 3D scanning, reconstruction from contours and surface sketching. Besides these three methods which usually make use of prior knowledge of the scene is a fourth group of algorithms which tries to advance this problem more generally with processing unorganized point clouds.

Dependent on the type of data acquisition, different algorithms and techniques are needed to process the data. As mentioned, algorithms tend to exploit specific prior knowledge of the model which has to be reconstructed. This information helps to create more authentic models but also lead to less general approaches and are more error-prone to noisy or unstructured data [2]. There is also work addressing more general approaches, which may not be as accurate as algorithms with enough prior knowledge, but therefore is usable in more general domains [33].

2.2.1 Polygon Mesh

The techniques described in the following sections usually generate a polygon mesh—more commonly called mesh. A mesh is used to describe a model in a digital way. Meshes consist of vertices which are being connected to edges. These edges are then combined to form faces in the form of triangles. By connecting the triangles with each other, polygons can be described. These polygons are then further connected to form surfaces. Thus the underlying data type is just points describing an object, which are then further processed into complex polygon meshes. The components of a mesh are shown in Figure 2.6.

There is a variety of polygon mesh representations like ‘Face-vertex meshes’, ‘winged edge meshes’, ‘half-edge meshes’ or ‘quad-edge meshes’ and many more. These different types of mesh representations have advantages and drawbacks which have been discussed by Smith [72]. Different approaches for acquiring surface data and reconstructing the surface of the model have to select a suitable representation of the data.

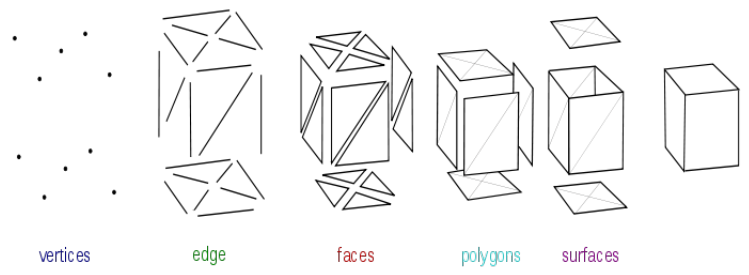


Figure 2.6: Components of a polygon mesh and how they are derived from each other. Source: [84]

2.2.2 3D Scanning

A popular technique for gathering data of an object is 3D scanning. Unlike other devices to capture information of an object, like a camera or a scanner, who create a picture or a video of an object, a 3D scanner delivers 3D information about the recorded object.

There are several methods and technologies with different proposals of how to perform 3D scanning. It can be observed that different approaches of 3D scanning result in different qualities of the results and also need a different amount of human interaction to operate them.

2.2.2.1 3D Reconstruction from multiple images

A computer vision based approach is the registration and matching of landmarks from different views to calculate the shape of an object. This technique can be called '3D reconstruction from multiple images'. An early work on this topic was proposed by Tomasi and Kanade in 1992 [80].

2.2.2.2 Shape from shading

Shape from shading is a different approach which uses the fact that the amount of reflected light from an object differs regarding the objects orientation. This property was originally called photometric stereo and aimed to determine the surface orientation by using several light sources from different angles [85]. This technique was enhanced to a special case called 'shape from shading' where only one image is needed [34].

2.2.2.3 Coordinate measuring machines—CMM

A different method is using touch probes which are being used to sample a physical model. These touch probes are usually mounted on a coordinate measuring machine, short CMM, which usually analyses three orthogonal axes to create three-dimensional coordinates of the model. The scanning via a CMM is usually accurate but very time consuming. First occurrences of a CMM can be dated back to a machine developed by Ferranti, a British company, which appeared at the 'Inernational Machine Tool exhibition' in Paris in 1959 [75]. Not all touch probes do physically touch the object, others use magnetic fields or ultrasound instead.

2.2.2.4 Laser range scanners

Laser range scanners became a popular alternative to previous technologies. They are also used as probes in coordinate measuring machines. Methods used to generate samples with a range laser are triangulation, interference or time of flight. Range laser scanners did rise in popularity because of the dense and accurate data they produce. The typical output consists of a set of data describing the distances between the sensor and the object. There are differences in whether the sensor and the object are static or if one or both of them are moving. When the sensor or the model has been moved, the generated images can be combined to obtain even more complex models. Merging of the range laser scanner results is a challenging task.

2.2.3 Reconstruction from Contours

A slightly different data retrieval approach uses data, which describes the contour of an object to obtain a reconstruction. Contour data is generally being obtained from computerised axial tomography (CAT) scans. The word ‘tomography’ originates from the Greek words ‘tomos’ and ‘graphe’ which mean ‘slice’ and ‘drawing’ respectively. A CAT system produces cross sectional images of the scanned object. Some fields of use for CAT systems are in the medical field, to treat patients, or in engineering, to analyze certain parts of machines amongst others. The result problem of getting a surface from contour data is to calculate a 3D surface from the sliced 2D contour data. There are several works addressing this problem, like the work from Bresler et al. [5] who developed a Bayesian approach, or the works of Fuchs et al. [27] who addressed optimal surface reconstruction from planar contours in 1977. Meyers et al. [50] addressed the surface reconstruction from three dimensional contours. The underlying difficulties from the reconstruction from contours can be traced back to the difficulty of working with branching structures. The works on surface from contours is generally successful, but the represented algorithms are usually designed towards certain predefined input data and do not find universal use like general reconstruction algorithms.

2.2.4 Surface Sketching

Another way to approach the acquisition of a 3D model is to track the path of an input device which is used to draw or sketch the model. Schneider [65] and Eisenman [21] developed techniques to create two dimensional curves. Sachs et al. [64] presented a system to obtain three dimensional curves. This system can be used to design free-form surfaces from the user inputs. Constructing a surface given the recorded points has then to be performed.

2.2.5 Unorganized Input

The previous mentioned approaches usually require knowledge about the data input which is being exploited for the surface reconstruction. Examples would be the given structure information of contour data which is defined as two dimensional contours parallel to each other. Others assume that the structure of an object is known. Another common requirement of reconstruction algorithms is, that they assume that the surface orientations or vertex normal are given or that the input data is noise free. Unlike the previous mentioned methods of data acquisition and surface reconstruction, there are more universal approaches like the one from Hoppe et al. [33]. This work uses several phases where the input is being processed into an accurate model.

2.2.6 Outline

There are many areas where a digital three dimensional model is required or wanted to work with. Industries use to have collections of models of common machine parts which are used for further development. These collections could be replaced by 3D models which are not only more comfortably to store but also easier to manage and redesign. Computer-aided design (CAD) systems are also used to represent models but since they are usually two dimensional draws, they are not as informative as physical models from clay or other substances. Digital models cannot only be used as pattern for future designs but are also used for simulations like aerial flow simulations of cars and planes. With the advent of 3D-Printers, who are capable of printing digital 3D models into physical models, the meaningfulness of supported 3D models rises again. These models can now not only be transmitted very fast over a digital way, but can also be reproduced—e.g. printed—when they are being needed.

2.3 IBR

The goal of Image Based Rendering (IBR) is to create images of 3D models from novel views, just like a viewer would see the real object when walking around or inspecting an object from different views. The challenge is, to use given knowledge about the scene, to calculate novel views with the help of the given information. This given knowledge can be in various forms, like images from the object, a 3D model of the object, knowledge about surface normals and several other types of information. This challenge unites the two different fields of research, computer graphics and computer vision, when trying to use and combine the techniques of both of them.

In the last decades, there has been much improvement on the field of rendering realistic scenes. For example, the bidirectional reflectance distribution (BRDF), subsurface scattering and various illumination modelling techniques [17] were improved and used to render natural scenes in a photorealistic way. The graphic processing hardware made huge improvements as well and the way of processing the data of graphics was further developed as well and now programmable shaders are being used to process the data in a fast and flexible way. Even though the results of sophisticated methods like ray tracing or radiosity are very promising, they are still a bit off from being as realistic as when looking out a window into the real world. A further drawback is, that the more realistic the output becomes, the more calculation time is needed due to its computational complex structure.

When realising, that the problem of generating fully photorealistic views is very challenging and hard to solve, the methodology started to switch from the calculation of light models and other methods to make use of the commonly available imaging hardware to simply take images and use them for further processing. These methods which switched from using geometry information for photorealistic rendering to using images to render the scenes were called 'Image-based modeling and rendering'. There has been done much

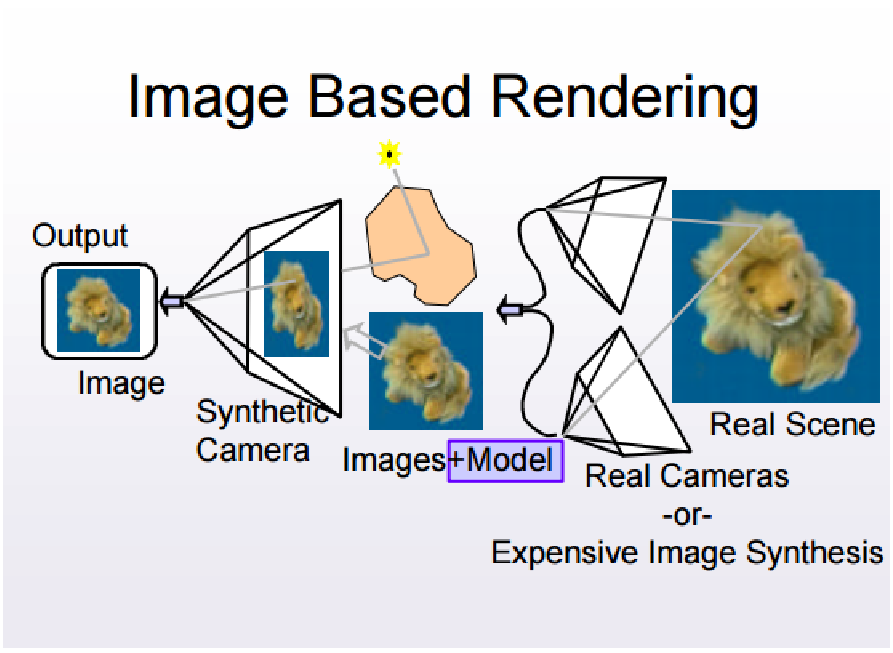


Figure 2.7: The steps needed to get an image from a real scene with IBR. Source: [10]

research on this approach which promised a powerful alternative to the methods used in the past. There are various approaches to image based rendering which were categorised by previous works regarding the amount of geometry being used by them. The resulting space of IBR techniques is also called the IBR continuum presented by many works [36, 69]. The IBR continuum is roughly divided into three categories, where on the one extreme the methods using no geometry for rendering followed by methods using implicit geometry and finally on the other extreme the techniques which need explicit geometry.

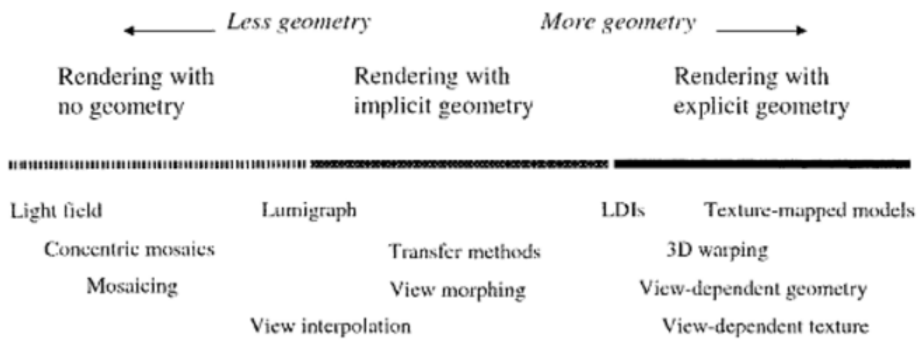


Figure 2.8: IBR continuum. Categorisation of IBR techniques by geometry used. Source [69]

As Figure 2.8 from [69] shows, on the one extreme are algorithms like light field, concentric mosaics or mosaicking, which do not need geometry at all. On the other extreme, techniques like texture-mapped models or 3D warping, which need very accurate geometric models, can be found. In the middle of those extremes are methods which require correspondences between frames. View interpolation for example needs the correspondences to interpolate the optical flow between keyframes or view morphing which needs point correspondences as well to compute new camera matrices. The following sections give an overview over the different approaches to create realistic views of models.

2.3.1 Texture Mapping

A simple two dimensional approach to image based rendering is texture mapping. Texture mapping refers to the process of applying an image, also called texture, onto a 3D surface of a model. The textured model is then mapped from its object space to the screen space. Figure 2.9 from [31] depicts this process in a simple way. Usually the texture coordinates are labeled as (u, v) and the object space coordinates as (x_O, y_O, z_O) and finally the screen space with (x, y). This process makes the resulting object with the applied image on it to appear to be more detailed and complex then the blank model, without significant calculation needed.

Due to the continuously rising processing power, the models became more and more detailed and complex and thus harder to texture via the texture mapping approach with simple texture coordinates. Due to the fact that the geometry is explicitly needed before textures can be applied to it, this technique can be found on the right side of the IBR continuum shown in Figure 2.8.

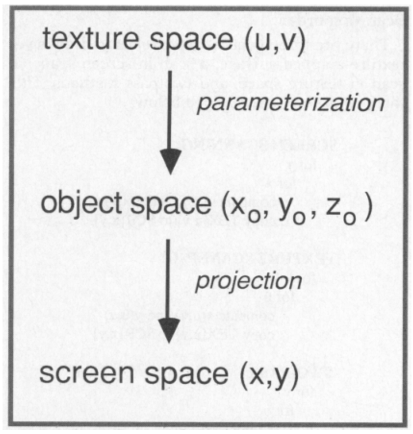


Figure 2.9: Texture Mapping pipeline. Source [31]

2.3.2 View-dependent texture mapping

There are different ways to retrieve 3D models with associated textures. CAD systems can be used to virtually create objects and to apply textures on them. On the other hand

3D scanners are able to scan real world objects to create models of them. Computer vision techniques are then used to map the pictures onto the scanned model. A problem occurring with both of these techniques is, that they lack displaying complex effects like reflections, transparency or highlights. This problem leads to view dependent texture mapping, where several pictures of the same object from different points of view are being processed to mitigate the stated problems. Debevec et al. proposed in [14] an approach, where the images from different views are warped and then composited to obtain new views of them. In later works [13], Debevec et al. proposed methods to improve the resulting views and to reduce the computational cost. Techniques used there are e.g. averaging the input images and applying weights to calculate the appearance from a new point of view of the object. Figure 2.10 illustrates how the view onto a pixel with a virtual camera is calculated with the help of a weighting function. Two actual views of the corresponding pixels are used in this example as input for the weighting function.

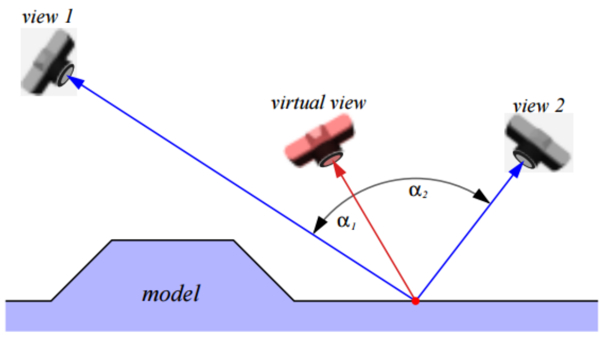


Figure 2.10: This image from [14] shows a pixel in the 'virtual view' derived with a weighting function used in view-dependent texture mapping. Here the pixel in the 'Virtual View' is assigned a weighted average of the corresponding pixels in the actual views 'view 1' and 'view2' ([14]).

2.3.3 Billboards

The basic principle behind billboards is to render the object as either a two dimensional plane or two planes intersecting each other to form a cross like shape. In the case of single planes, they are directed normally to the viewer and thus appear to be real objects when the textures are then being mapped onto those planes. Mayor advantages of billboards are their low memory footprint and the low computational power needed to create and render them. Thus, they have been quite popular in the computer games development area. The results look satisfying when viewed from far, but when the viewer gets closer the drawbacks of billboards are visible because they appear rather flat. When the shape of objects becomes more complex, billboards tend to lead to not satisfying results even when viewed from distance.

To mitigate the problems of billboards, so called billboard clouds were introduced by Décoret et al. in their paper 'Billboard Clouds for Extreme Model Simplification' [15].

They propose to build 3D models from a set of textured planes to simplify it. This so called cloud of several billboards does not need connectivity information like a polygon mesh would and thus there is fewer computational power needed to process them. Another advantage of this approach is, that the memory footprint is lower than it would be for classic model representation.

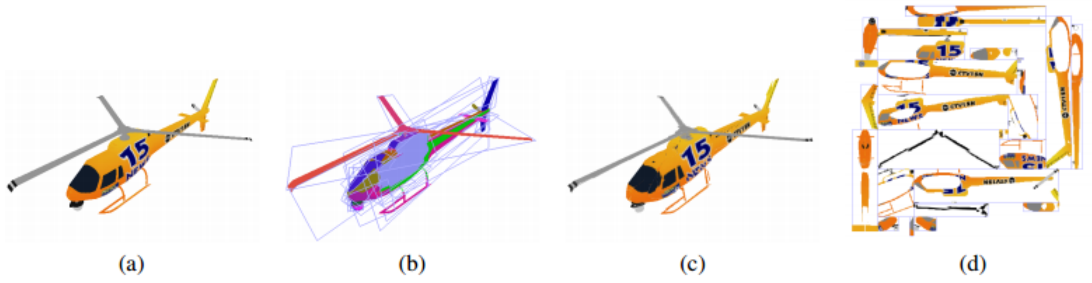


Figure 2.11: Example of a billboard cloud by [15]: (a) shows the original model with over 5,000 polygons, (b) is a false-color rendering with one color per billboard, (c) shows the model generated from 32 billboards, (d) view of the different billboards

2.3.4 Image Layering / Layer based rendering

Some methods, also called layering techniques, try to describe a scene by creating several planar layers of it, where each layer has its own texture and sometimes transparency information as well. This approach, which is utilising layers, is relatively simple to implement on the GPU where just the set of planes has to be rendered and textured. The single layers do not have connectivity information and thus the processing overhead is kept low as well, but the condition of not being connected also brings a disadvantage because the scene is only viewable from a small range of views before this construct reveals its fundamental structure.

There are approaches, like the one from Jeschke et al. [35] to overcome the limited field of view. They proposed, among other things, to only use the layer based approach for objects which are far away from the viewer and to use regular polygons for near objects. There are usually several steps done in layered rendering. First the layers are being rendered, and then the layers are composed in back to front order to create the result. A common method used to combine the layers is the painter's algorithm.

2.3.5 3D Warping

3D warping exploits given depth information of an image. For this technique, the depth values for each point in an image are needed to calculate the appearance of the point in different points of view. This is done by projecting the point in the image to its 3D position on the model and then again projecting the model to the screen space resulting in the output image. Problems occur, if the input images have different sampling then the output image or when the output image tries to view parts of the model which are not covered by the input images and thus resulting in holes. Different input images can be combined with texture splatting to reduce the number of holes. Oliveira et al. [60] proposed to divide the process into two steps, with a first pre-warping step followed by traditional texture mapping. There have also been various other approaches to deal with the problems of 3D warping and how to improve it [47, 54].

2.3.6 Layered Depth Images

Layered Depth Images were proposed by Shade et al. [67]. They approached the problems of 3D warping by not only respecting the visible parts of the input images, but also the parts being occluded by the scene, and therefore not being visible. To obtain the wanted information for the LDI process, they used a series of images with a known camera path to get the stereo, or created virtual environments where the geometry was known. The input then includes depth and color information and a list of where a ray of the pixel intersects with the environment. The LDI technique is an example for the layer based rendering described above.

2.3.7 View Interpolation

The previous described methods needed explicit model information to reach their goals. View interpolation instead falls into the category of implicit geometry in Figure 2.8. This means that detailed information regarding the underlying model is not needed anymore but information about correspondences between input point positions. Projection calculations are then being performed to obtain the needed information. By utilising the information of two input images and the optical flow between them Chen and Williams [8] were able to calculate images of various different points of view. The closer the input images are together the better the results of this approach will be. When the images are farther away from each other, flow fields have to be constructed more strictly to achieve acceptable results.

2.3.8 View Morphing

Every possible view on the line between two optical centres of two basis views can be reconstructed with the view morphing algorithm proposed by Seitz and Dyers [66], given that a visibility constraint is satisfied. View morphing is capable to calculate greater camera angles than view interpolation and an additional prewarping step is capable to align the different image planes while still keeping the original optical centre intact. After the first prewarping step, the morph step, and a final postwarp step are being performed. The three steps suggested by Seitz and Dyers were outlined by them in their work.

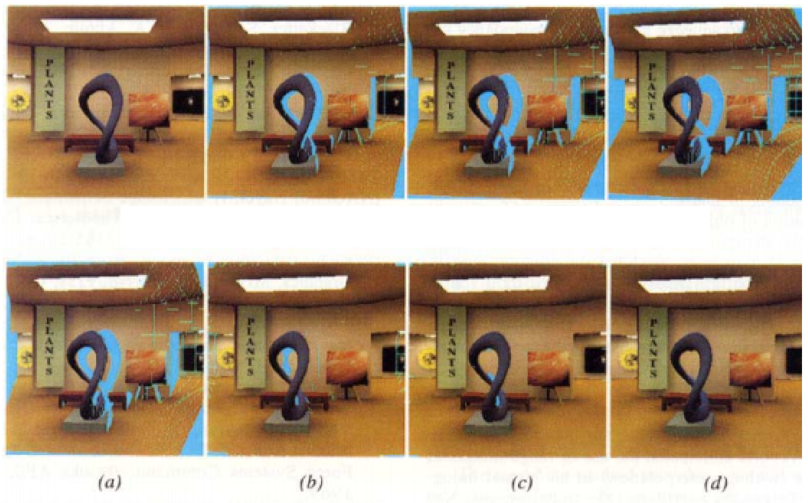


Figure 2.12: This figure from [8] shows in the upper row, the source image from a camera rotated to the right and on the lower row, in image (a) the holes from the source image, in (b) the holes from two source images, in (c) the holes from two closely space images and finally in (d) the holes filled with interpolation.

1. Prewarp: $\hat{I}_0 = \mathbf{H_0^{-1}} I_0$, $\hat{I}_1 = \mathbf{H_1^{-1}} I_1$
2. Morph: Linearly interpolate positions and intensities of corresponding pixels in $\hat{I}_0$ and $\hat{I}_1$ to form $\hat{I}_s$
3. Postwarp: $I_s = \mathbf{H_s} \hat{I}_s$

The shown three-step-algorithm was proposed in the paper written by Seitz and Dyers [66].

2.3.9 Light field & Lumigraph

The following methods do not need a model to calculate novel views from given input images. Thus they are located more to the left in the IBR continuum displayed in figure 2.8. To describe the approaches called light field and Lumigraph, the plenoptic function should be discussed first. The term plenoptic was introduced by Adelson and Bergen in their work 'The Plenoptic Function and the Elements of Early Vision' [1]. It consists of two words, the first being the Greek word *plenus*, which means complete or full, followed by the word *optic*. They chose this terminology because the function desires to describe everything that can be seen. The plenoptic function consists of seven parameters and thus is a 7D function. The first two parameters describe the intensity distribution of spherical coordinates $P(\theta, \phi)$ or of Cartesian coordinates $P(x, y)$ of a planar image. The wavelength λ is added to be able to describe the information given by colored images, resulting in

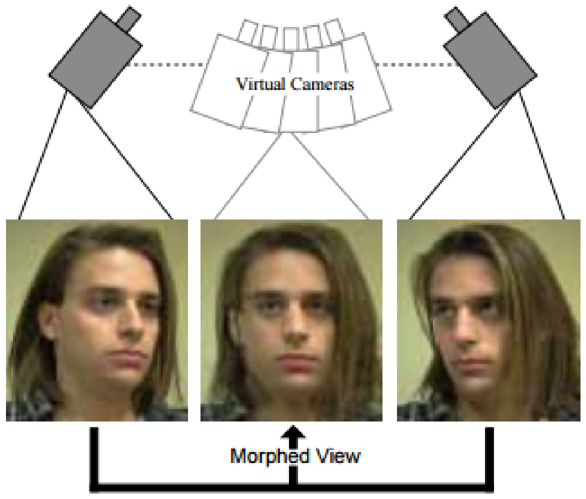


Figure 2.13: A visual example of View Morphing from [66]

$P(\theta, \phi, \lambda)$. The next parameter of the plenoptic function describes the time t which represents the value, when the given samples were recorded extending it to $P(\theta, \phi, \lambda, t)$. And finally, the observable light intensity at every viewing position is addressed as well, leading to $P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$.

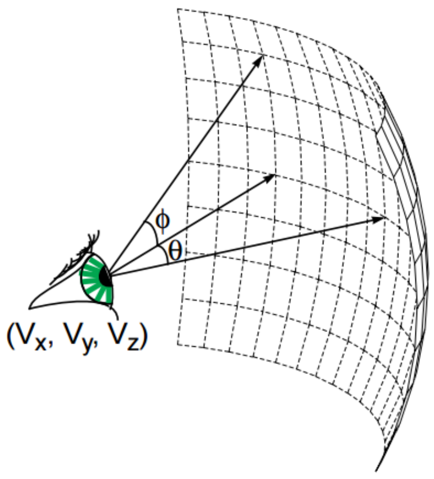


Figure 2.14: The plenoptic function describes all of the image information visible from a particular viewing position [48]

Plenoptic modeling [48] was proposed by McMillan and Bishop who reduced the complexity of the original plenoptic function by not taking time and wavelength into account, resulting in $P(\theta, \phi, V_x, V_y, V_z)$.

The two techniques named 'Light field rendering' [43] and 'The lumigraph' [30] came to the conclusion, that, if they stay in a bounding volume, the 5D plenoptic modeling can be reduced to a four dimensional function $L(s, t, u, v)$. These parameters represent two planes, (s, t) being the first plane, which is intersected by a light ray and then passing through to the second plane (u, v) as shown in Figure 2.15. In the light field approach, the (u, v) plane represents the camera plane and the (s, t) plane the focal plane, where the scene is located. Lumigraph uses the same two planes but calls the camera plane (s, t) and thus the focal plane (u, v).

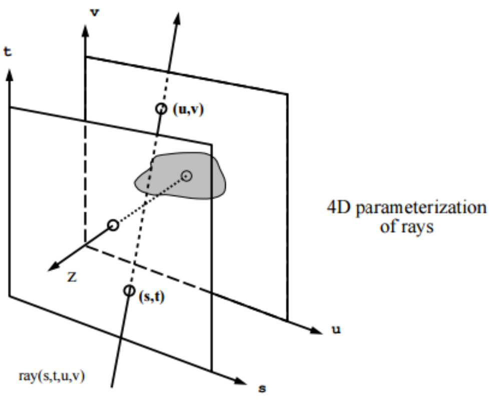


Figure 2.15: This graphic from [30] represents the bounding volume used by 'Lumigraph' and 'Light field rendering' with a ray intersecting the two planes.

The light field method from Levoy and Hanrahan assumes that the scene surface is close to the focal plane and thus rays coming from the camera plane and passing through the same point of the focal plane are treated as samples of the same point of the object from different viewpoints. The assumption, that the scene is close to the focal plane leads to a blurred appearance the farther the surface is away from the focal plane, after the samples have been interpolated. To reduce this problem, the bounding volume is usually the convex hull of the object. Figure 2.16 depicts two example visualisations of a light field. The similar approach, which was followed by Lumigraph, reduces the problem of blurred images by assuming an approximated 3D surface of the object for interpolation.

2.3.10 Concentric Mosaics

Another approach based on the plenoptic function is called 'Concentric Mosaics', introduced by Shum and He [70]. They constrained the camera motion to planar concentric circles which they realized by using a rig where the camera was rotating around a fixed point. Using this technique they are able to record the scene in less than ten minutes. This method reduces the plenoptic function to three dimensions which are the rotation angle, the radius and the vertical elevation. A positive effect of using only three dimensions of

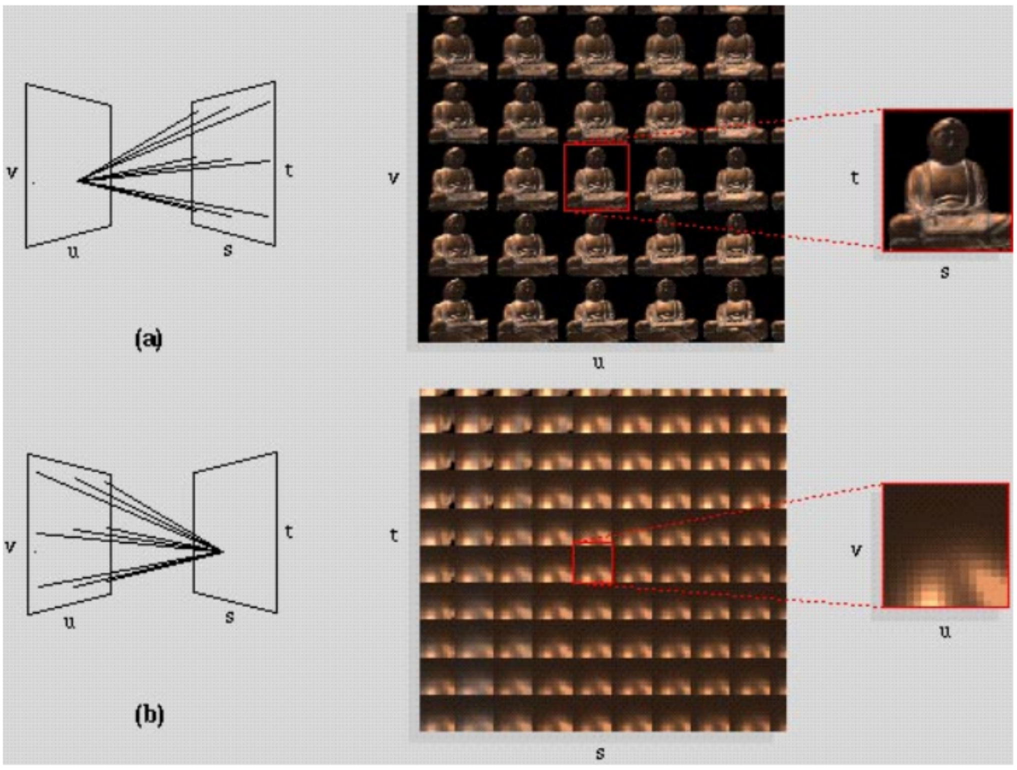


Figure 2.16: Two visualisations of a light field. (a) Each image in the array represents the rays arriving at one point on the *uv* plane from all points on the *st* plane, as shown at left. (b) Each image represents the rays leaving one point on the *st* plane bound for all points on the *uv* plane. The images in (a) are off-axis (i.e. sheared) perspective views of the scene, while the images in (b) look like reflectance maps. The latter occurs because the object has been placed astride the focal plane, making sets of rays leaving points on the focal plane similar in character to sets of rays leaving points on the object. Image and caption originate from [43]

the plenoptic function instead of four, like light fields and Lumigraph, is that the mosaics need much smaller space to save the needed data. Concentric mosaics are a set of mosaics constructed by slit images taken at different positions in each circle. The results arranged by concentric mosaics can be viewed in a circle shaped field around the object, with novel views being generated by combining the captured rays. To avoid vertical distortion, depth correction has to be performed. The rich user experience presented by Concentric Mosaics combined with the relative ease of recording the scene while still using much less data than the 4D approaches made Concentric Mosaics a popular technique.

2.3.11 Image mosaicking

The principle behind image mosaic techniques is that multiple images from a scene are being taken, and then mapped onto a cylindrical or spherical representation. To combine these mapped images into a complete image showing the scene, simple translations are needed. A normal mosaic representation generally allows the panning of and zooming on the image. By mapping it onto a cylindrical shape, rotating around the object allows a 360° viewing. There are several techniques falling into this category, like the one from Chen [7] or from Mann and Picard [46]. Some systems use the fact, that when images have been taken from relatively close positions from each other, small strips from the images—so called slit images—are needed to construct a panoramic mosaic. Others rely on omnidirectional cameras or fish-eye cameras to create panoramas [57, 86]. There is constant improvement on the field of image mosaicking and new further developed systems appear regularly [6, 77]. Figure 2.17 and Figure 2.18 from [6] depict image mosaicking with SIFT and RANSAC, and image mosaicking with automatic straightening respectively.

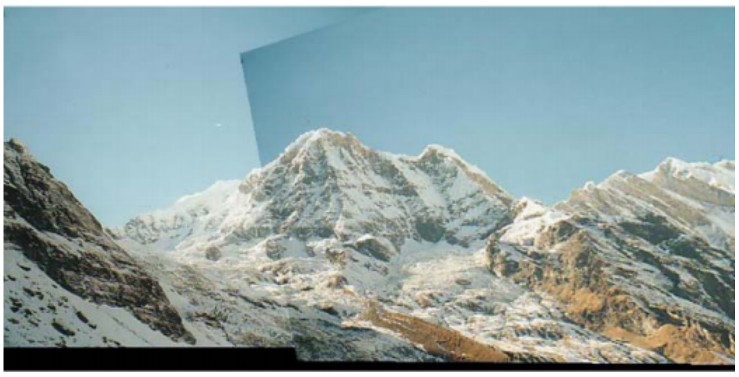


Figure 2.17: Image mosaicking with SIFT and RANSAC from [6]

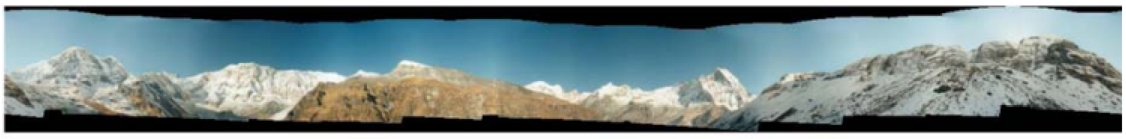


Figure 2.18: Image mosaicking with automatic straightening from [6]

2.3.12 Outline

Compared to classical rendering, Image Based Rendering methods have significant advantages. The biggest is probably due to the fact that in IBR images are used and thus the computational complexity is not as high and real time approaches are more realistic. Also the scene complexity does not directly increase the rendering complexity.

The above sections showed that there are many different methods to approach Image Based Rendering. Some systems rely on only a small sample of images from a scene, while others need huge sets of images with only small camera movement between them. This also results in a different data size needed for different methods. Some methods rely on accurate underlying geometry, while others roughly estimate them to create novel views. The different solutions can also be categorised by the dimensionality in which they approach the problem. The dimensions go from 2D to 5D starting with 2D cylindrical or spherical panoramas, followed by concentric mosaics which use 3 dimensions of the plenoptic function and 4D approaches like Light field and Lumigraph. The plenoptic modeling respects five dimensions of the plenoptic function.

Even though there has been a lot of research on this domain, there are still problems which remain unsolved or only partially solved like reflections or transparency. But given the advantages in contrast to classical rendering, Image Based Rendering seems to be a good direction for photorealistic rendering.

3 Method

The following sections will discuss how the problems stated in the problem analysis—section 1.2—have been approached and will furthermore present the solutions, which have been selected for this work. To elaborate the needed tools to approach the problems, the first section walks through an example problem situation. While processing through the scenario, the different approaching problems are addressed and solutions are compiled. After the scenario is worked through, the following section will give a detailed overview of the entire process elaborated in the example. The approach is divided into different parts, which deal with the inherent problem of these certain parts. The different parts are then compiled into a processing pipeline by putting them together to form the solutions from each sub-problem into one approach for the problem.

3.1 Scenario

To find solutions to the problems and constraints presented in the problem analysis—section 1.2—a clearer description and understanding of the given scenario is needed. The following three sections will elaborate a scenario to get a picture of what kind of problem situation is to be expected and how to approach it. The first section gives an overview over the problem in the scenario and then advances to approach the problem of how to get information from the environment and how to deal with the obtained data. Then, the second section makes use of the data retrieved by further processing it to make it possible to solve the underlying problem. Finally, the third section concludes the scenario and summarises the results.

3.1.1 Initial scenario

This setup consists of two persons, with the first being mobile in a local environment and the second being stationary. The first person is now faced with a problem situation, which cannot be solved without further information from the outside. The second person is in possession of the needed knowledge to revoke the given problem, but because only the first person is locally available, a way has to be found with the given instruments to make the local scenery available to the distant person.

This scenario leads to several questions. The first being, what tools would usually be available to the first person. It can be assumed that this person has no access to sophisticated recording devices like laser scanners or advanced stereo setups which may also have depth sensors and would be well suited for recording the present scenery. It is also supposed that the local person has no knowledge about how to operate advanced technology. These assumptions reduce the number of qualified sensors drastically. Some devices that are usually available to a normal person today are a smartphone or a tablet, a smartwatch and maybe some other wearable devices. While devices like the smartwatch usually do not have any sensors which could be used, smartphones and tablets tend to be in possession of some, and it appears that the available sensors for these devices will grow in quality as well as in quantity in the near future. While some sensors like the front and the back cameras of phones and tablets are obvious, there is also a vast diversity of others, like an accelerometer, a proximity sensor, or a GPS sensor. Another sensor which finds more and more popularity, but is not built in yet to the vast majority of devices, is the depth sensor. A collection of sensors, which are commonly available in modern smartphones or tablets, is shown in 'List of Sensors' in A. While the given sensors can answer the question of which instruments are available to acquire information about the environment, it has yet to be thought about in which way the information should be processed to pose a clear description of the environment. Even though smart devices possess a diversity of sensors, most of them are not very suitable to create a clear understandable copy of a local scenery. The most intuitive sensors for humans are definitely the built-in cameras.

A naïve approach utilising the cameras would be to simply take a picture of the scene and send it to the second person which can then identify the situation and give guidance. This approach comes with some problems inherent to two dimensional depictions. The first and most critical problem is that even though the image does clearly represent the given scene, there is only one point of view available and thus no spatial information can be extracted. This problem leads to the next, which is that the distant person who is in charge of understanding the given situation to solve the problem does only get a limited amount of information from a single picture. Even if the problem can be understood by viewing the image, there still is the problem of how the instructions, describing how to solve the nuisance, can be delivered to the local person.

Another approach using the cameras and avoiding some of the problems from the first naïve approach would be to create a series of pictures or a video of the local scenery. This would lessen some of the problems in the first attempt, but some other problems can be observed. While recording a video does not pose a problem to smartphones, and the distant person is definitely able to understand the problem more clearly, than if only given a single picture, the transportation of the video poses a problem, which is not to be underestimated. Even though state of the art video compression algorithms try to reduce the size of videos to a practical size, the data transmission still needs a considerably large amount of time. Another drawback of this approach is that the person, which is filming the scene, is assumed not to be an expert on the domain, where maintenance is needed. Thus, this person does not know which parts of the area are of interest and important to film and which parts can be considered irrelevant. When the local person has now to create a video of the problematic vicinity, the record will consist of several unneeded scenes, which will consume time from the distant expert, where no relevant conclusions can be made to solve the problem.

Using a live video stream between the local and the distant person can reduce this problem, when the distant person can give movement instructions do direct the local person to film needed locations. To establish a live stream and guarantee high video quality, an appropriate infrastructure is generally needed which is not assumed to be available to the local person. Even if the transmission problem could be solved in a satisfactory way, the information of how to solve the problem would still be needed. The smartphone would of course offer an acoustic way for explaining the solution process over a telephone conversation, but it can be supposed that the solution to the problem is not trivial enough to solve via this way. The local person would furthermore be limited in the movements due to filming the scenery with the smartphone or a handheld device to establish the material for the live stream. Given that constraint, performing physical repair routines will thus be a more complicated task.

Optical head-mounted displays like the google glass come to mind when physical restrictions are a problem, because these devices usually come with a built-in camera and are capable of displaying information without needing much interaction. Common Wi-Fi standards are usually supported as well by these devices which would make data transmission possible. Even though head-mounted displays appear to be a suitable solution to the given scenario, availability of these devices is not granted and thus they are not further considered here.

When analysing the given situation in a more general way it can be said that there is an agent located in a location which is unknown, meaning no prior information is available to the structure of the scene. Additionally, the agent is in possession of one or several sensors, which can be used to obtain information about the environment. The task of the local agent is now to utilize the available technology to create a map of the agents environment. The imposed restrictions are that, as mentioned, the composition of the environment is unknown, and furthermore the location of the agent within the scenery is unknown as well. Creating a map of a given environment and simultaneously determining the position within the created map is exactly the fundamental problem of SLAM. The information obtained by simultaneous localization and mapping techniques can be viewed from several different points of views which has been a severe restriction from the discussed approaches above. They either had only a single view available, or the recording of several views limited the local person and also confronted the distant person with unneeded information. To ensure that a SLAM system is suitable for the given task the constraints listed in the problem analysis—section 1.2—need to be fulfilled.

The first constraint to the system was that it needs to be able to process the recorded input data. Because the different implementations all rely on processing the given sensor data, it can be said that this condition is fulfilled. The fact, that SLAM is used to create a map to describe the environment of an agent, satisfies the second condition inherently. Because the second task of SLAM, besides creating an environment map, is to determine the position of the agent, which is implicitly the position of the sensor, the next condition, which demands the camera position, is also granted by SLAM. The next constraint—to not need prior information about the scene—is also fulfilled by the definition of SLAM inherently and can be taken as granted. The last two constraints, which demand availability of the created data for further processing and a commonly usable format of the data have to be solved by the implementation of the SLAM system. Thus these two constraints have to be respected when selecting a certain implementation. Further information about simultaneous localization and mapping can be found in section 2.1.

By this method of elimination, SLAM was selected to serve as a technique to obtain the needed information about the scene. This leads to the next scenario, which deals with the constraints regarding the representation of the obtained data.

3.1.2 Data processing scenario

By now, the local person is using a smart device like a smartphone or tablet and an implementation of a SLAM system to record the scene and to obtain data describing the scene in a fundamental way. Even though the created data describes the scene, it still needs to be further processed to be clearly understandable for humans. The created data consists of camera positions describing from where the scene was filmed. Because a camera was chosen to serve as sensor, image data from the camera positions is available as well. Furthermore, there is some kind of map available, which is describing the environment created by the SLAM system.

Now, a representation of the scene is desired, which gives the distant person the freedom to move around in the scene in order to understand the given problem and to be able to give instructions accordingly to the local person. Other constraints to the chosen representation are that it needs to be feasible to transport the created data. A three-dimensional model of the local scene seems to appropriately fulfill these demands. It is possible to view a 3D model from different directions and commonly used file formats, which can describe such a 3D model, do need far less disk space than a video. This reduces the effort of the data transmission significantly. The underlying data of a 3D model description usually has information of how points sampled from the scene have to be connected to form adjacent surfaces which then result in a model. Thus, when using a 3D model to describe the scene, points describing the scene seem to be mandatory. This additional constraint—the need of points describing the scene—resulting from the selected representation, has also to be respected from the selected data processing part, which in this case is SLAM. The created map data produced by SLAM systems can in most implementations either be processed into a point cloud representation or already is a point cloud. Thus, SLAM systems still appear to be a viable choice.

A simple 3D model of a scene may be able to describe the geometry well but additional information of the scene besides its structure is still missing. To add more details and make the model appear more realistic, textures are usually applied. Section 2.3 gives an overview over a special texturing approach named image based rendering, which went away from calculating complex light models like radiosity, and moved towards using real images to calculate novel textured views of an object. Needing pictures of the scene is thus an obvious constraint which has already been satisfied by choosing a camera as sensor and also the camera positions can be retrieved from the selected SLAM system which serves as data processing unit. As described in section 2.3 there is a spectrum of IBR systems which can basically be divided by their different need of input information. Starting on the one side with approaches needing explicit geometry, followed by methods which need implicit knowledge about the model and finally reaching the other side of the continuum containing approaches which do not need any geometry information. Because a 3D model was selected as representation for the scene the selected IBR system will be positioned more on the one end of the IBR continuum containing methods requesting explicit knowledge about the underlying geometry. Knowing about the general type of the image based rendering system just leaves the constraints open which deal with the available type and format of the input data. The data which will be passed to the selected IBR system consists of the 3D model representing the scene, images taken from the scene which have to be applied to the passed model and the camera positions describing the view from where the images have been recorded. Any IBR system which can process these types of data can thus be used to create a textured model from the scene.

3.1.3 Instruction scenario

The local person of the initial scenario is now able to create a textured 3D model of the problematic scene. After the creation of the model, it needs to be transmitted to the remote expert. Due to the relative low memory footprint of the model, the transportation does not need a sophisticated infrastructure anymore like it would be needed when establishing a video stream.

The distant person who is able to solve the nuisance the local person is confronted with can now analyze the situation to identify the problem and find a way to dissolve it. The created 3D model provided now allows both persons to inspect the scene independently from each other which was not possible in the approach where a livestream has been established. The two recording approaches which took an image or a video from the scene respectively were not able to provide this amount of freedom either.

Now the distant person in the scenario has all the needed information to find a solution to the problem. The next challenge is how to supply the local person with the needed instructions to dissolve the problem. A naïve approach would be to utilize the smartphone of the local person again and make a phone call to describe the steps, which lead to the solution, in an auditive way. This may lead to satisfying results when the underlying tasks to solve the problem are simple to describe and to understand in words. Because that cannot be taken for granted, another approach is wanted.

When taking the first approach of capturing the scene with a single image in account, a possible solution to describe tasks to the local person would have been to draw arrows or similar forms to depict the sort of activity needed to be performed. Due to the discussed drawbacks of information from a single image, the descriptions would lack detail as well. This very naïve approach with 2D images can be mapped to the current situation of the scenario where a 3D model is available. Following this chain of thought leads to adding information to the created model. The resulting problem is to find a useful way to describe the added information. A relevant constraint on the instruction description is definitely, that it needs to be intuitively understandable. This constraint can be fulfilled by avoiding complex descriptions and keeping the added information as simple as possible. Other constraints stated in the problem analysis (section 1.2) were, that the information, describing the tasks needed to solve the problem, needs to be easy to apply on the model and furthermore needs to be adjustable. The constraint demanding easy apply routines also results from the given scenario implicitly. The scenario shows that an expert is available but has not enough time to travel to the local person to help solve the nuisance together. The constraint demanding a low memory footprint to make a fast transmission of the data possible also shows that there is a hidden time constraint as well. Thus, it is reasonable that the process of applying the needed information has to be kept simple. The other constraint which desires the information to be adjustable is reasoned from the assumption that the initial information might not conclude in a solution of the problem and thus needs further adjustments.

Adding geometry to the created model seems to satisfy the stated constraints. The geometry can be kept simple when only using geometric primitives but can also describe more complex shapes if needed. When providing a list of available primitives, the process of adding them can be kept simple as well. With the knowledge about the model, the primitives can be applied onto it to be clearly placed onto the geometry of the model instead of floating somewhere around it. After primitives have been added it is also possible via transformations to move or adjust them, which satisfies the other constraint. The memory footprint can still be kept low because primitives do not require a complex description of their structure.

Because primitives alone are not very well suited to describe the process of a manual task, there is still the need of additional information to describe a mechanical task. By adding simple motion in the form of transformations to the added primitives, the physical exercises which are needed to operate typical control elements can be described. Common physical tasks which usually need to be performed when maintaining a machine, a control board or other user interfaces can be of different types, defined by the nature of the given

Control Element	Needed Operation	Needed transformation
button	push	translation in z axis
lever	pull, push, tilt	rotation around fixed point
knob	spin	rotation around centre
door	open/close	rotation around fixed axis
sliding door/gate	push	translation along one axis (or several)
door handle	tilt	rotation around fixed point
unlock with key	spin (the key)	rotation
socket/plug	plug in/out	translation

Table 3.1: List of common control elements and the needed transformation to describe their inherent operation.

control elements. Interfaces which are omnipresent in environments created by humans are buttons, levers, knobs, doors, locks and various others. Each of these so-called control elements require different interactions to operate them. Table 3.1, in this section, gives an overview over commonly used control elements, their associated type of operation and the transformation needed to be performed when using them. The needed transformations for all kinds of different control elements can be reduced to two, which are the translation and the rotation. Further can be observed that most objects only need either a rotation or a translation and only sometimes a combination of both is needed.

Performing these two transformations on a geometric representation of the underlying control element is feasible under the given constraints. To create a motion describing both transformations some more information is needed. For a translation, the starting point and the destination point of the translation is required. Alternatively to the destination, a direction and a given length would be sufficient as well to perform the translation. For a rotation, the starting point and orientation is needed as well as the rotation direction and duration. Alternatively, the starting position and orientation and the destination orientation could be used as well. This observation leads to a common need for both transformations which can both be described by their starting position and orientation and the corresponding destination position and orientation. The memory footprint of this data is low and thus suited for this approach. To perform the motion over time, an adequate interpolation between start and destination is needed.

The final challenge is now to define what needs to be sent back to the local person of the scenario so that the nuisance can be dealt with. A simple solution would be to transmit the whole model with the applied control element information. Because a low memory footprint was one of the constraints this is definitely a possible solution. A more elegant approach would be to save the information regarding the added control primitives in a separate file and only sent this information to the local person. Because the local person is already in possession of the model and the only information needed is the newly generated description of the added geometry this approach exists as well. A positive aspect of this second method is that the amount of data needed to be transmitted would be greatly reduced. The first approach would need a transmission of the model M and another transmission of the model M including the added geometry G resulting in $2M+G$. The second approach would obviously only need a single model transfer and a geometry transfer ending up with $M+G$. Because the data describing the control element geometry can be neglected compared to the size of the data from the textured model this reduces the transfer load by half. The time needed from the local person to load the received control element geometry can also be neglected due to the simple geometry description needed for primitives.

Another possible solution to this question would also be to ignore the created model on the side of the local person, and display the added control elements as an augmented reality overlay over the video of the person in charge to handle the problem. Since the local person runs a SLAM system to record the desired scenery, the current camera pose is available to him at any time. The control elements added from the distant person have their inherent position regarding the created model attached. Next, the coordinate systems of the created 3D model and the live scenery visible on the video of the local person need to be fitted so that they match each other. Now the added control elements can be displayed on top of the video present to the local person, while the distant person can see and adjust them on the model visible on the remote side. This simplifies the task for the local person, who is not an expert and may not be able to fully understand the created 3D model. The local person can now move freely around the scenery and will be able to see the added instructions from any desired point of view. To make sure that the local person can see the added control elements correctly, a live video stream could be established between the two parties, so that the remote person can see the same as the local person. Since a video stream poses a high amount of data load, it could either be a sampled down version of the original video to reduce the data, or the video stream could be made optional so that it is only available on demand.

Now there are two possible solutions to show the instructions to the local person. The first one is to show the 3D model to the local person and add the control elements to this model, and the other solution is to hide the model from the non-expert and show the control elements directly on the video as an augmented reality overlay. Advantages of the first solution would be, that the 3D model can be saved and used at any time as an addition to a normal user manual, with the drawback that the local person who views the recorded model is a bit more challenged to understand the model and the corresponding scenery present to him. The second approach, where the control elements are shown as an augmented reality overlay to the local person is definitely less challenging to the local person but it assumes, that there is a remote person available at any time who is able to understand the given problem and can add control elements to help solve the problem. Both of the discussed solutions for presenting the generated information of the distant person to the local person pose viable possibilities. The offline version has the advantage of being available at any time in contrast to the live version, which relies on a permanently available distant person. On the other hand, the online version is arguably more intuitive to understand because the added information is directly presented on top of the live video present to the local user, while the offline version has to be interpreted correctly to be able to utilize the given information.

3.1.4 Scenario outline

By successfully filming a local scene and using a SLAM system to create a point cloud which is then further processed into a 3D model and then by applying textures with the help of an IBR technique, the local person in the scenario is able to create and deliver the needed information of the local scene to the distant person. After receiving the textured 3D model, the distant person has now the possibility to add geometry in form of primitives to the scene and animate the added geometry to describe tasks which need to be performed on the scene. The information created by the distant person can then be sent back to the local person and being viewed to understand the mechanical task to solve the given situation. The detailed elaboration of the given scenario leads to several solutions to the problems described in section 1.2 and lead to a better understanding of the needed techniques.

3.2 Processing pipeline

The last section elaborated the different tools which are needed to solve the problem of creating a suited representation to describe a given environment and how to issue instructions with the help of suited representations. The different techniques and their interaction to approach the given task need to be further discussed to form a process which is capable to utilize and combine the tools. Combining the selected tools and techniques into a new process results in a processing pipeline with several steps. The individual parts of the pipeline represent the different methods which were chosen. The pipeline does furthermore describe how the output of one part is created from the input and how the resulting data is delivered to the next step. These steps are divided into the gathering part where information about the environment is acquired and prepared for further processing. The modelling part which creates a geometric description of the environment and the texturing part which is adding additional information to the created model by adding textures. And finally, the interaction part which deals with adding and animating geometry to describe the interactions with the environment.

3.2.1 Gathering

The first part of the pipeline has to deal with the data acquisition. Because SLAM characterises the problem of obtaining information about the environment and the position within the environment, it was chosen to serve for the data acquisition part. Furthermore was decided, that a monocular camera which is commonly available in smart devices will be the sensor of choice. The first part of the problem thus needs to implement a SLAM system which is capable of solving the SLAM problem with a monocular camera. The output data of this step needs to contain a point cloud describing the environment and images from the scene including the position from where they have been taken.

3.2.2 Modelling

After the data has been acquired, it needs to be further processed. The next step in this processing pipeline is the creation of a model from the previously obtained data. This will be done via a reconstruction algorithm which is capable of creating a mesh representing the underlying geometry. Because the created model has only geometric information and no further description of the appearance of the environment another step, which is discussed in the next section, is needed.

3.2.3 Texturing

Image based rendering was selected to add appearance information to the model. It was further elaborated on the basis of the given information and the IBR continuum that the selected IBR approach will require explicit model information. The selected IBR system will now receive the created mesh and the captured keyframes along with their capturing position as input. The result of the IBR system is now a textured 3D model representing the composition of the captured scene in the first step.

3.2.4 Interaction

To describe certain interactions with the scene, it was decided to add geometry in the form of primitives to the scene. The next system needs now to be able to view the newly created textured 3D model and provide the possibility of adding the geometric primitives. Furthermore, it is necessary for the system to provide an option to animate the added geometry. After the geometric primitives have been added to the created 3D model of the scene, the last step of the processing pipeline is the representation of the created information. Therefore, either the created 3D model with applied control elements or an augmented reality overlay containing the control elements comes in mind. Figure 3.1 shows the steps needed to create a textured model with applied control elements from a filmed scene.

3.3 Outline

This chapter introduced an example problem situation in section 3.1 with two persons needing to communicate about a given nuisance. Then the assumed problem situation was approached by trying to find a solution for it and its emerging sub-problems, while still

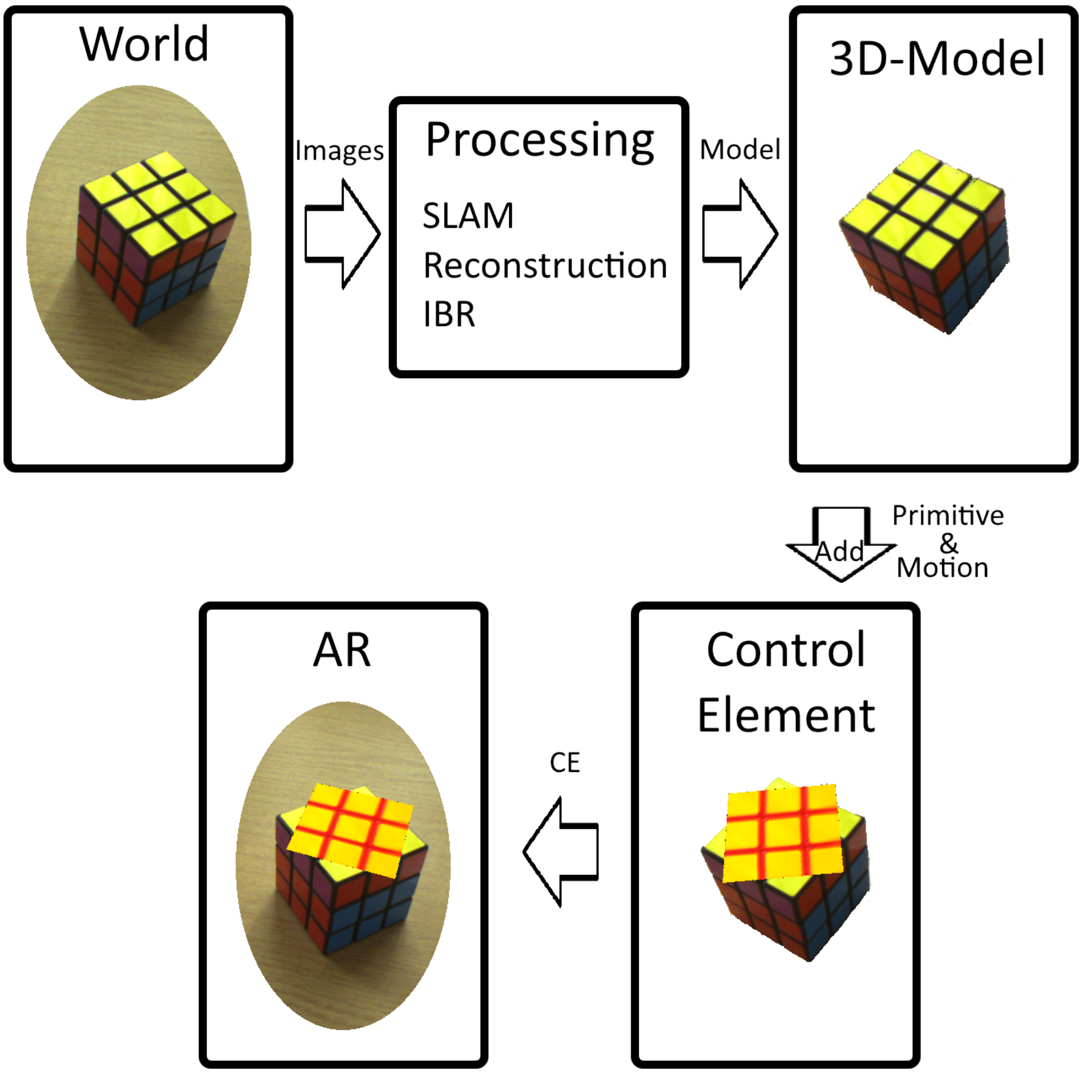


Figure 3.1: Processing pipeline, showing the steps needed to get a textured model with applied control elements from a filmed scene.

respecting the given constraints from section 1.2. The result of this successive approach was the processing pipeline presented in section 3.2.

The next section (4) will discuss how the implementation of the processing pipeline was approached, and which techniques and implementations have been chosen to fulfill the given constraints.

4 Implementation

In the previous section, a processing pipeline was developed by advancing a general example problem. The created advance to the problem was then verified on the basis of a concrete problem. The next step is now to implement the processing pipeline which has been established. This chapter explains in the following sections, which concrete solutions for each step of the pipeline have been chosen and how the different parts have been connected with each other.

4.1 SLAM

Since there are many constraints to the SLAM system and it is hard to satisfy all of them, due to the fact that every system has its limitations, it makes sense to provide a framework which allows changing the SLAM system to be able to try out and evaluate the results of the different systems.

Another problem seems to be, that SLAM systems are not inherently built to use their results to create a model, but more to get a vast understanding of the surrounding area. This can result in problems with unwanted data, because not always the entire surrounding area will be of interest. Another question is how to deal with outliers which are possibly created by the SLAM system. These thoughts strengthened the decision to create an implementation which allows changing between different SLAM systems and also allows adding more systems in the near future. The following sections will discuss the SLAM system which has been embedded in the created implementation and why it has been chosen. After the system has been introduced, it will be discussed how it has been connected to the whole implementation to make further processing with the acquired data possible. Besides the main SLAM system, another SLAM implementation is discussed, and how it would be possible to integrate this system to the project by utilising the given framework structure. The two systems are PTAM—'Parallel tracking and mapping'—and LSD SLAM—'Large-Scale Direct Monocular SLAM'.

4.1.1 PTAM

The paper presenting PTAM which is short for 'Parallel tracking and mapping' was introduced by Klein and Murray in 2007 [39]. The complete title of the paper is 'Parallel tracking and mapping for Small AR Workspaces' which already points into the direction in which fields their implementation is commonly suited to be used—small workspaces. This implementation has been interesting for several reasons. The first was because it is '. . . a system specifically to track a hand-held camera in a small AR workspace' ([39]). This proposition describes the scenario from section 3.1 very accurately, where a person which is present at a scene is in need of acquiring data of the environment with a smart-device. Another reason to use PTAM as the first SLAM system in this implementation was because it was constructed to '. . . split tracking and mapping into two separate tasks, processed . . . on a dual core computer' ([39]). The majority of modern smartphones or tablets do possess a multicore CPU and thus the implicit constraint of the scenario is not violated by choosing PTAM to acquire the needed data of the environment.

The goals of PTAM are to provide a system which is not only fast but also accurate and possesses a robust camera tracking and is able to refine and expand the created map. All of these goals have been approached by PTAM under certain constraints. The first of these constraints is that there is no prior knowledge about the scene except that the system is aimed to be more suited for small workspaces than huge environments. Besides this constraint which descends from the general SLAM problem, the others are that a monocular camera is to be used as a sensor and thus no sophisticated sensors which can acquire depth information or obtain other useful information about the scene can be used. A constraint, which is imposed to the environment, is that it has to be a mostly static and relatively small scene. Another condition for the resulting PTAM system was that it has to be able to process the needed data on the CPU and does not need an explicit GPU for calculating the output data. Klein and Murray developed a method which can be summarised by the following list from the original PTAM paper [39].

- Tracking and Mapping are separated, and run in two parallel threads.
- Mapping is based on keyframes, which are processed using batch techniques (Bundle adjustment).
- The map is densely initialised from a stereo pair (5-point algorithm).
- New points are initialised with an epipolar search.
- Large numbers (thousands) of points are mapped.

The choice of separating tracking and mapping was made after studying the two other SLAM systems which use a monocular camera and were available at that time. These were MonoSLAM [12] by Davison et al. and the works of Eade and Dummond [18, 19] including 'Scalable Monocular SLAM'. Both of these available systems operated on an incremental approach and linked tracking and mapping together resulting in systems which updated the current position and the created map jointly at every frame. This does not represent a problem because the two systems were implemented to be used for a robot which typically provides additional information like the moving direction and can be driven at low speed. A hand-held system cannot rely on these conditions and thus data association errors become a problem. This was the motivation to separate tracking and mapping into two processes decoupling the tracking from the map making procedure. Due to the common availability of multicore processors, the two separated tasks were implemented in two threads where the tracking thread had no longer to wait for the mapping procedure and the mapping thread did not have to process every single frame anymore. Instead only a smaller number of more relevant key frames are being processed. Due to the lowered amount of frames needed to be processed, it was possible to replace the incremental mapping by the computationally more expensive bundle adjustment. Several works, which successfully applied bundle adjustment in real-time visual odometry and tracking [25, 55, 59], inspired Klein and Murray. From these inspirations they kept and adapted the five-point initialisation procedure and used bundle adjustment for local optimisation, but went to creating a long-term map instead of discarding older information. As the fourth point in the list above states, an epipolar search replaces long 2D feature tracks from previous works to find new features in the environment. This method results in the three core components of PTAM. They are the map which is responsible for storing the acquired data, the tracking part in the one thread responsible for gathering new information and the mapping thread which initializes and maintains the map.

The map was built to store M point features and N keyframes. A world coordinate frame W contains the points where each point represents a planar textured patch in the world. The points do not store the patch information locally, but do have a reference to the originating keyframe, which is usually the first frame where the points have been recorded. The N keyframes are the frames captured by the monocular camera and are stored in the map as well. Furthermore is an image pyramid of grey value images from the frames stored in the map. The pyramid consists of four levels starting with the resolution of 640x480 as the highest resolution, which is bisected three times to create the other levels, resulting with 80x60 as the smallest.

The task of the tracking system is to process the images delivered by the camera. The incoming frames are used to calculate and update the pose of the camera position respective to the created map. This task of updating the camera position was described with a two-stage tracking procedure by Klein and Murray [39] in six points:

1. A new frame is acquired from the camera, and a prior pose estimate is generated from a motion model.

2. Map points are projected into the image according to the frame's prior pose estimate.

3. A small number (50) of the coarsest-scale features are searched for in the image.

4. The camera pose is updated from these coarse matches.

5. A larger number (1000) of points is re-projected and searched for in the image.

6. A final pose estimate for the frame is computed from all the matches found.

With the help of their coarse-to-fine camera pose approximation, they are able to update the current camera pose incrementally. To obtain the features needed in the above steps, FAST corner detection [81] is used and a fixed range search is performed to find the point from the previous frame in the current frame.

The task of the mapping system is to build a map consisting of three dimensional points with the given input images provided by the tracking thread. Figure 4.1 illustrates the steps performed by the mapping routine.

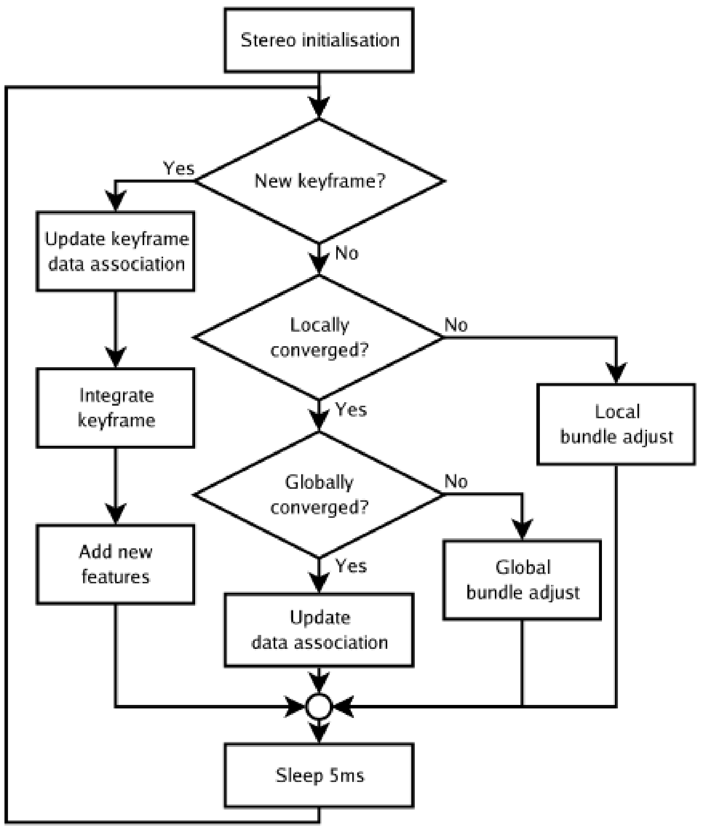


Figure 4.1: Flow diagram of the asynchronous mapping thread of PTAM. Source of the figure: [39]

The mapping is, as shown in Figure 4.1, divided into two tasks, the first is to create an initial map and the second to refine and expand the existing map. For the map initialisation, the five-point stereo technique [76] was used. For this method to work, it is implied that certain user interaction is needed to perform the initialisation task. At first an initial frame needs to be registered, followed by a translation with a possible rotation of the camera to a second frame. The path of the features is tracked from the first to the second frame, creating correspondences between the two frames which can be further used by the five-point algorithm and RANSAC to obtain an initial camera pose and the initial map.

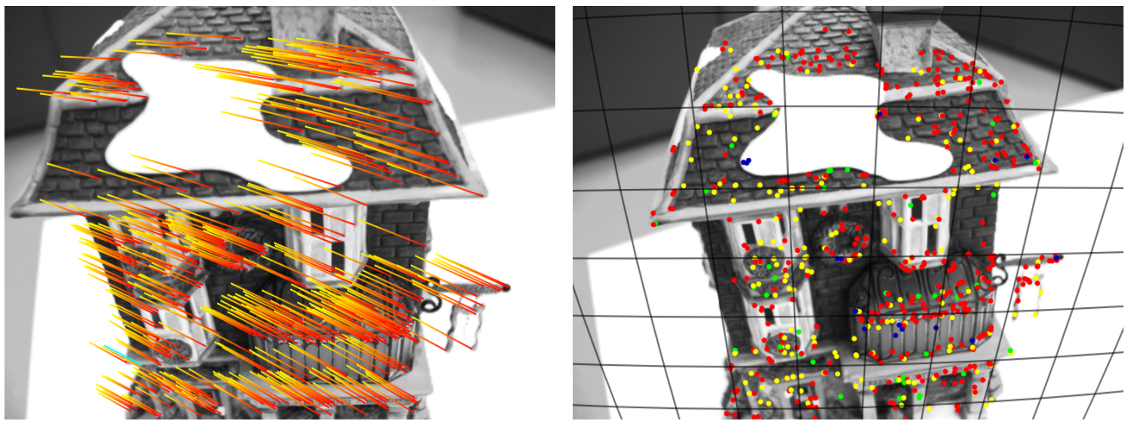


Figure 4.2: The picture on the left displays the user-guided initialisation routine of PTAM. The lines show the movement of the detected features from the first picture to the current camera pose. The picture on the right shows the features found for tracking, which are drawn onto the filmed scene while tracking. The different colours of the dots represent the four different layers of the generated image pyramid.

In the left part of Figure 4.2 the above described initialisation task is shown. To record the image, the original PTAM code was adjusted to work on Microsoft Windows systems and using Qt for the graphical user interface. The lines depict the movement of the features found in the first recorded picture, starting from their red side and advancing along the performed camera movement. When the initial camera movement is executed too fast or the distance between the starting frame and the final frame is too long, the tracking of the initial features will fail, resulting in the lines breaking. In that case it can be tried to finish the initialisation with the remaining features, or to restart the process. The right part of Figure 4.2 displays an example scene filmed with PTAM. The coloured dots represent the features used for tracking in the different pyramid layers. Figure 4.3 shows the resulting point cloud of the previously filmed example and the camera positions from where the object has been filmed. The last example of PTAM is shown in Figure 4.4. The picture on the top shows the filmed scene, which was a keyboard and a cube on top of it—marked with the red box. The bottom picture displays the resulting point cloud. It can be seen that even though various outliers are present, the fundamental shape of the keyboard and the cube can be determined.

For further refinement of the initial map bundle adjustment is used. A new keyframe is added to the map, when the following four conditions are met:

- A candidate is available.
- The tracking is not lost.
- A certain time passed between the insertion of the last keyframe.
- A certain distance lies between the previous and the new keyframe.

The incoming new keyframe has already undergone a fast inspection from the tracking system which needed features to estimate the new camera pose. Due to the limited time for each frame, it is assumed that the tracking system was not able to find all features in the given frame and thus another investigation is performed from the mapping thread to add more features. The FAST corners calculated from the tracking system for each pyramid level are then further processed with non-maximal suppression and thresholding based on Shi-Tomasi [68] to confine the total set of features to the most suited ones. The remaining points now need to have a depth information to be able to become map points. The depth is acquired with the help of triangulation with another frame.

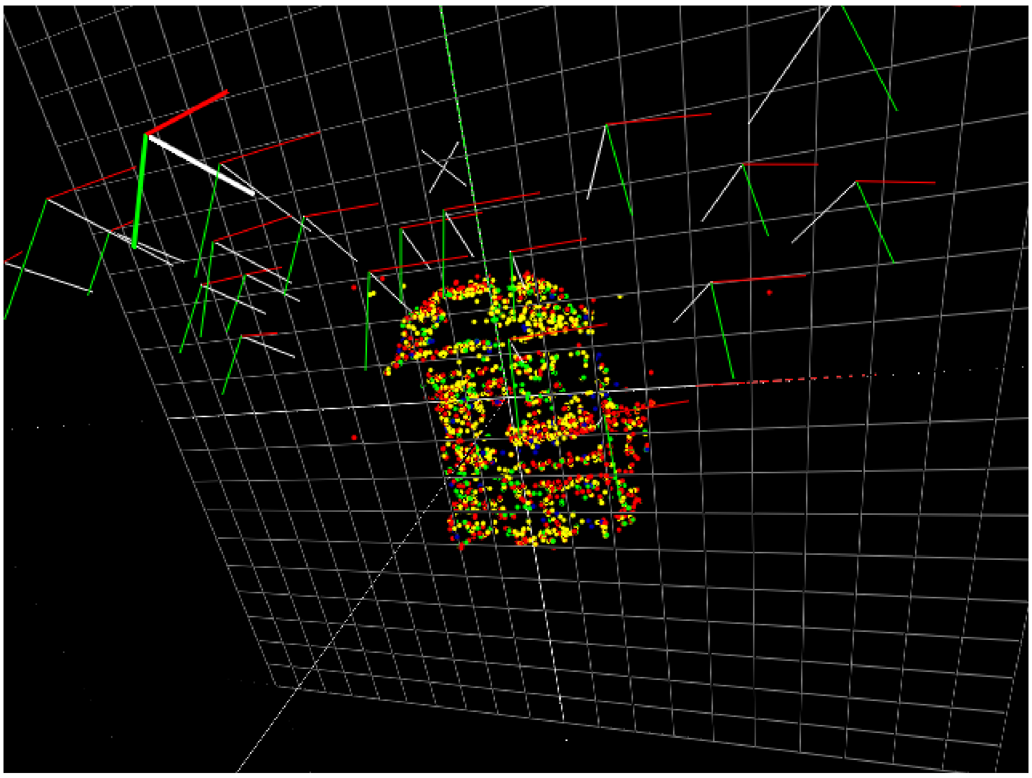


Figure 4.3: This figure shows the point cloud created from the scene initialized and shown in the two parts of Figure 4.2. The camera positions used to obtain the data can be seen as well.

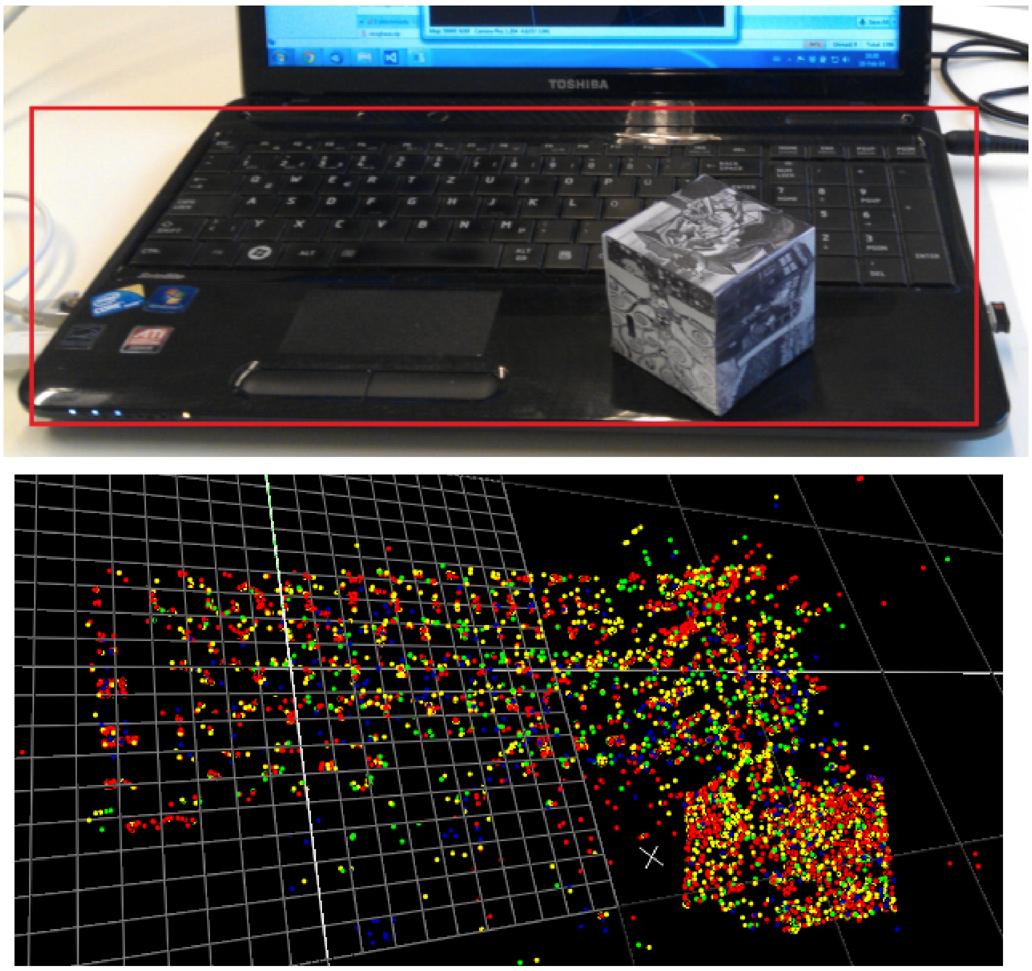


Figure 4.4: The picture on the top shows the filmed scene, and the bottom picture shows the point cloud recorded by PTAM

Bundle adjustment is performed locally and globally to maintain map fidelity. Local bundle adjustment is done if the camera guided from the user is in the so called exploration mode meaning that new areas of the scene are discovered. The resulting amount of frames is thus be first adjusted locally. Global bundle adjustment is done when there are not too many new frames and the thread has time to refine the entire map.

When there is no more need for further bundle adjustment because it already converged, and the mapping thread has no other tasks to perform, the already added keyframes are re-visited and searched for previously undetected features. This step is called 'update data associations' in Figure 4.1. By performing tracking and mapping to update the camera pose and the map in separate threads as illustrated above, PTAM achieves a relatively robust camera tracking for a monocular setup and also is able to create sparse but representative point clouds of the filmed environment. Thus PTAM seems to be a solid SLAM system choice for this project. Before discussing how PTAM is linked with the rest of the project and how the data generated by PTAM was retrieved for further processing, another SLAM system implementation is presented in the next section.

4.1.2 LSD-SLAM

Another interesting SLAM system for this project was proposed seven years after PTAM in the year 2014 in the paper 'LSD-SLAM: Large-Scale Direct Monocular SLAM' by Engel, Schöps and Cremers [23]. In contrast to PTAM, which is explicitly designed for small workspaces, the full title of LSD-SLAM already induces that this system allows the creation of large scale maps. LSD-SLAM was implemented for different sensor setups, which are a stereo setup, an omnidirectional approach, and even a mobile implementation. The last of these was presented on android devices. The mobile implementation is of interest for this work, because a monocular camera of a smartphone or a tablet can be used. The authors claim that their system inherits highly accurate pose estimation and that the 3D environment can be reconstructed as pose-graph of the recorded keyframes and their associated semi-dense depth maps. The techniques making this system possible are 'two key novelties: (1) a novel directed tracking method which operates on sim(3), thereby explicitly detecting scale-drift, and (2) an elegant probabilistic solution to include the effect of noisy depth values into tracking' ([23]). To get a clearer picture of the goals from LSD-SLAM, the difference between feature based SLAM and direct SLAM needs to be elaborated.

Feature based SLAM systems, like the previously explained PTAM, are based on extracting features from incoming frames, then matching the found features in successive images. With the found matches, the camera motion and the filmed geometry is then retrieved. Drawbacks of this standard method are that they rely on detecting features and break if features are not found anymore, which could happen under different lighting, and that they need a certain matching threshold to find correspondences between the found features. They are furthermore limited to a certain feature type and only by using a higher dimensional feature space this problem could be avoided with the drawback of massive performance loss.

Direct methods approach the problem of extracting features and motion by directly estimating the wanted information from intensity values of the image. Instead of considering the distance between features, like feature based methods approach the problem, the intensity gradient magnitude and the direction is used. Thus, direct methods consider all available information instead of only a subset of the information in form of certain features.

Prior works to LSD-SLAM who utilized direct methods used to track the camera motion only locally and were not able to build large consistent maps [24, 26]. The creation of large consistent maps addresses one of the mentioned open problems in SLAM (section

2.1.6) which is loop closure. A particular problem of scaled sensors—like depth sensors or stereo setups—is that they are not flexible regarding change in scale, which can lead to erroneous large scale maps. Changes in scale occur for example when switching from a close indoor scene to a vast outdoor environment. Monocular systems have the inherent problem that the scale of the world cannot be determined, which is also known as scale-ambiguity. This makes the use of monocular cameras for SLAM challenging but offers the flexibility to switch between environments with different scales.

The goals of LSD-SLAM are to provide a framework capable of creating consistent large-scale maps of an environment and to deal with the occurring scale drift to maintain map fidelity. The constraints for this system are basically the same as the ones for PTAM. These are that there is no prior knowledge available to the system, and a monocular camera is to be used to perform the SLAM algorithm. Like in PTAM, the underlying geometry of the filmed scene has to be constant, but due to the goal of creating maps of large-scale environments, pedestrians and similar moving objects are allowed to occur. And finally, the resulting system has to be able to operate in real time on a CPU.

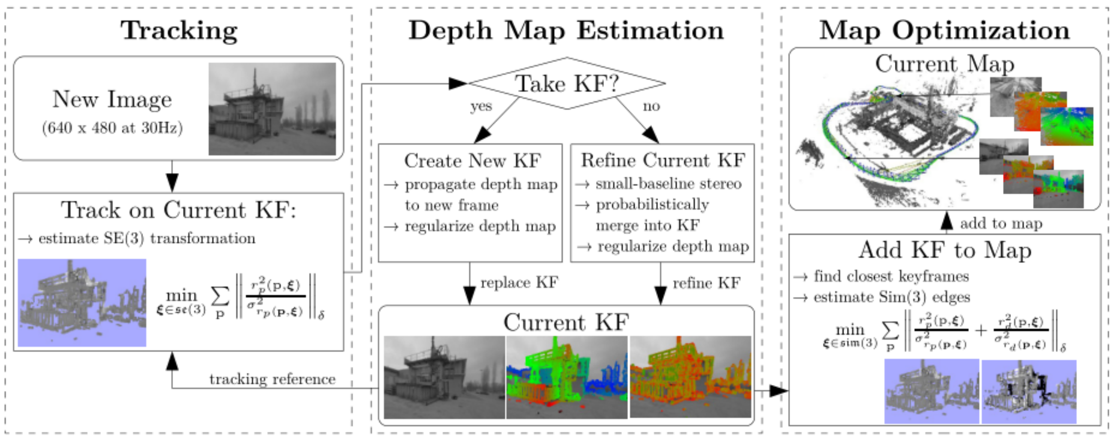

Tracking
New Image
(640 x 480 at 30Hz)
Track on Current KF:
→ estimate SE(3) transformation
$\min_{\xi \in se(3)} \sum_p \left\| \frac{r_p^2(p,\xi)}{\sigma^2_{r_p(p,\xi)}} \right\|_\delta$
tracking reference
Depth Map Estimation
Take KF?
yes
no
Create New KF
→ propagate depth map to new frame
→ regularize depth map
Refine Current KF
→ small-baseline stereo
→ probabilistically merge into KF
→ regularize depth map
replace KF
refine KF
Current KF
Map Optimization
Current Map
add to map
Add KF to Map
→ find closest keyframes
→ estimate Sim(3) edges
$\min_{\xi \in sim(3)} \sum_p \left\| \frac{r_p^2(p,\xi)}{\sigma^2_{r_p(p,\xi)}} + \frac{r_d^2(p,\xi)}{\sigma^2_{r_d(p,\xi)}} \right\|_\delta$

Figure 4.5: The interaction of the separate tasks performed by LSD-SLAM. (source: [23])

The method developed to implement LSD-SLAM is described by three components—tracking, depth map estimation, and map optimisation—shown in Figure 4.5. The first important step, not shown in the figure of the method overview 4.5, is the initialisation of the system. In contrast to PTAM, which needed certain user interactions to perform the stereo initialisation, LSD-SLAM initializes the depth map of the first keyframe with random values and a large variance. With enough camera translation in the first seconds a suitable configuration is found. With the following found keyframes, the propagation converges, resulting in a proper depth map. This method of initialisation requires the user only to move the camera around the scene instead of locking certain keyframes manually.

Before describing the three stages of the LSD-SLAM method, the representation of the maintained map is of importance. This map consists of a pose-graph of the collected keyframes. Each keyframe holds a greyscale image of the scene, an inverse of the depth map mentioned in the initialisation step, and the variance of the inverse depth map. The authors of LSD-SLAM especially mentioned that 'the depth map and the variance are only defined for a subset of pixels containing all image regions in the vicinity of sufficiently large intensity gradient, hence semi-dense'([23]). The keyframe and their corresponding information form the nodes of the pose-graph, while the edges describe the relative alignment of the connected keyframes as similarity transform and the corresponding covariance matrix. The similarity transform describes the translation, rotation and scaling of the keyframes.

After the structure of the map has been roughly outlined, the three stages of the method are to be treated. In the tracking stage, an already processed keyframe is used to compute the relative pose of the new frame. This is done by 'minimizing the variance-normalized photometric error' ([23]). The chosen formulation is able to deal with varying noise on the estimated depth maps. It is important to address this issue in monocular SLAM systems, because the noise differs relatively to the duration of how long a pixel has been visible.

The next stage called 'depth map estimation' can be separated into two sub-tasks, which are performed either when a new $keyframe$ is needed or the current $keyframe$ is to be refined. The terminology of LSD-SLAM describes every incoming image as $frame$ and those frames which are selected to become a part of the map are called $keyframes$. It is furthermore important to mention, that each keyframe is scaled in a way that the mean inverse of its depth becomes one. The relative distance from the active $keyframe$ to the current $frame$ is measured, and when it exceeds a certain threshold, the current frame will be selected to become the new $keyframe$. If this case occurs, the depth map of the current $frame$ has to be initialized. Therefore the points from the current $keyframe$ are projected to the current $frame$. Afterwards, spatial regularisation and outlier removal is performed and finally the depth map is scaled in such a way that its mean inverse is one. After the current frame has been fully transformed into a $keyframe$, the current $keyframe$ is replaced by it. If there is no need for a new $keyframe$ yet, the current $frame$ is used to refine the current $keyframe$. For this purpose, baseline stereo comparisons are performed to refine the depth of pixel areas and to add new pixels as described in [41]. Before adding a new $keyframe$ the possible scale drift is respected and treated. With the help of the similarity function mentioned above the scale drift can be detected. After the new $keyframe$ is added, it is checked, if a loop closure is possible with the already existing $keyframes$.

The optimisation of the map is permanently performed in the background while the graph is being built up. Therefore, pose graph optimisation techniques provided by [26] have been employed. LSD-SLAM is an innovative approach on monocular SLAM presenting a novel attempt on direct methods and presenting a solution to approach loop closure by detecting scale drift. The evolution of monocular SLAM in recent years can clearly be seen, when comparing the results from PTAM and LSD-SLAM. A semi dense point cloud can be sampled from the used depth maps of each keyframe, which provides a detailed description of the environment.

The following section describes how the interface for PTAM was implemented to provide the functionality of this system to the rest of the processing pipeline. Furthermore, it is outlined how the implemented interface can be utilized to integrate LSD-SLAM into the given framework. While outlining a possible integration of this SLAM system, it is also mentioned where special attention has to be paid to make sure that this innovative system can deliver meaningful input when trying to add it to this project.

4.1.3 SLAM System Interface

A constraint mentioned in the method section 3.1 is, that the data created by the used SLAM system needs to be accessible e.g. via a given interface. The data needs to consist of the recorded keyframes including the position from where they have been captured and the point cloud sampled from the given environment. This constraint is crucial, because the data must be passed down the pipeline performing the reconstruction and texturing. Because the SLAM system has to become a part of the whole project, an interface between the project and possible SLAM systems has to be created. The following part of this section will go into details about what requirements the interface has to fulfill for each of the two systems and how the wanted data from the two SLAM implementations can be acquired.

4.1.3.1 Integration of PTAM

An implementation of PTAM [39] is available from the author's page [38] and is licensed under GPL. A list containing the additionally needed program libraries to successfully compile PTAM can be found in the appendix A. To obtain all of the needed data from PTAM for this work, some changes in the structure of PTAM needed to be made. Before being able to access the data, the data needs to be created, and this data creation from PTAM is initialized by a stereo algorithm, which requires user interaction. This interaction was originally tightly connected to the inherent user interface of PTAM and needed to be reworked to offer a function, which was accessible from the outside to initialize this SLAM system. After the scene had been recorded by PTAM, keyframes which are containing images from the different views of the filmed scene and the corresponding camera positions from where these keyframes have been recorded are stored in a so called *map*, which is in charge to manage all the created data. Besides the keyframes, a collection of points is stored in the *map* as well. Each point stored possesses a list of keyframes in which the point has been seen. A problem of the recorded images from the keyframes has been, that only greyscale images have been stored for each keyframe for further processing, and colour images were not available. Because the texturing part of this project is in need of colour images, which are to be applied to the created model, parts of the original PTAM code had to be changed to be able to provide colour images in the map as well. Finally, the processed keyframes and the points with their reference keyframes are stored in the map. Slight modifications needed to be made to allow the access from the outside to this data.

4.1.3.2 Theoretical integration of LSD-SLAM

A proper integration of LSD-SLAM [23] into this project would go beyond the scope of this work, because there are several non-trivial tasks to respect. To give a prove of concept of the created interface for SLAM systems in this project, it will be discussed in the following how the integration of a SLAM system like LSD-SLAM can be approached. An implementation of LSD-SLAM is available on the computer vision group's webpage [22]. The code available on their GitHub does only support ROS-based systems and is tested only on Ubuntu systems. To integrate LSD-SLAM into this project, it would be necessary to replace the ROS parts and add a suitable camera interface. Because the available implementation of LSD-SLAM processes greyscale images for each keyframe, some adjustments would be needed, similarly to the changes in PTAM, to be able to get coloured images for each keyframe.

Another obstacle when wanting to implement LSD-SLAM into this project would be the available data structure, which does not conform entirely with the wanted structure. The data of LSD-SLAM consists of a set of $keyframes$, and each $keyframe$ has a depth map from which a point cloud can be calculated. Then, the created point clouds have to be merged to build one complete cloud. A result of this is, that each point in the merged point cloud has only one associated keyframe in which it had been seen. The algorithms used for the reconstruction and the image based rendering require more than one reference frame and thus a workaround has to be found. In the current implementation, the data has to consist of keyframes including their associated camera coordinates and a set of points which have been seen from this position. Each point has to contain a list of keyframes from where it has been recorded. To acquire more views from where the points have been seen, the underlying principle of using frames to refine the $keyframes$ can be exploited. In LSD-SLAM, each $frame$ has an inherent camera position, further is known, that the $frames$ between each pair of $keyframes$ were used to refine the corresponding $keyframes$. This observation can be used to take several of the $frames$, which were obtained before and after each $keyframe$, as further reference camera positions for the calculated points. While these new camera positions for each point might not be totally accurate in all cases, they should be a good approximation.

Another difficulty which needs to be approached arises from the fact that the $keyframes$ tend to overlap each other partially, which results in multiple occurrences of the same point in the merged point cloud. Because of the use of a monocular camera, the depth approximation for each $keyframe$ tends to be slightly different. Therefore, the multiple occurrences of the same point do not have identical coordinates. This means, that a well thought merging algorithm should be implemented to get rid of the ambiguities arising from combining the separate point clouds. This problem seems not to affect the large-scale example environment maps, which are also available on the author's webpage. But figures 4.6 and 4.7 show how this slight depth approximation error looks like in two of their examples. When viewing the recorded point clouds from farther away, this inaccuracy does not pose a problem, but it could become a problem when trying to mesh the created point cloud into a 3D model.

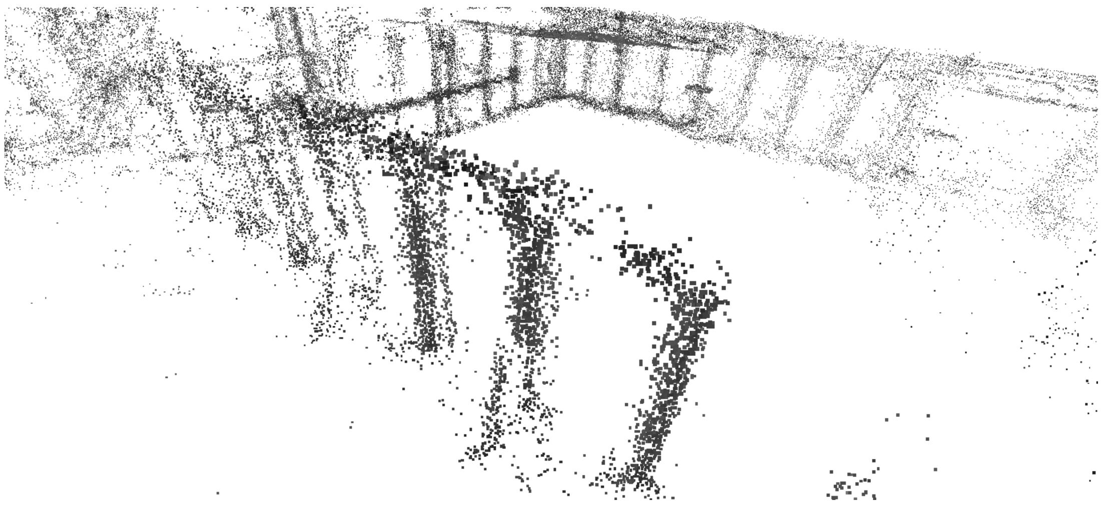


Figure 4.6: When zooming into an example point cloud from [22], points with erroneous depth can be seen near the pillars.

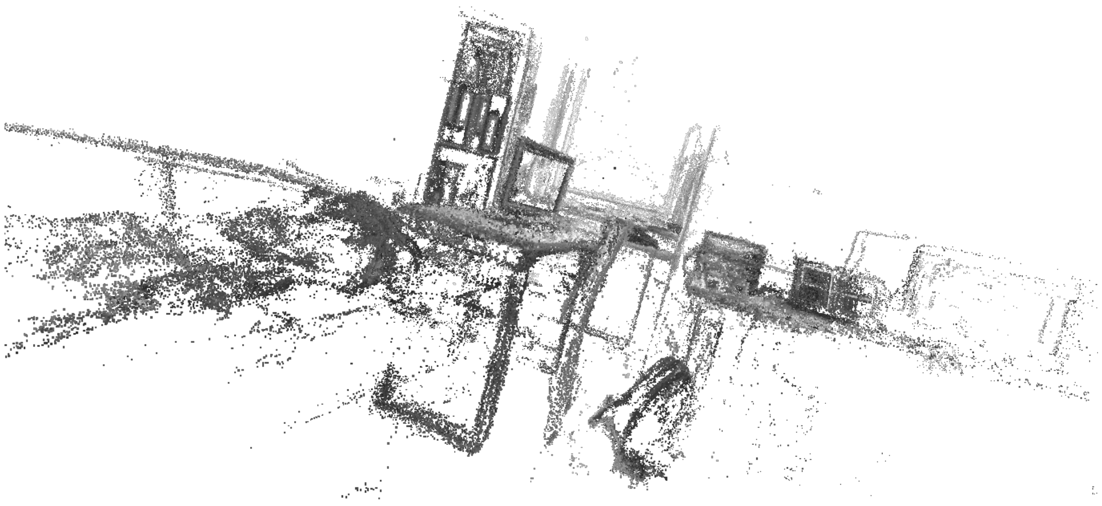


Figure 4.7: Depth estimation errors like in Figure 4.6 can be found on other examples from [22] as well.

4.1.3.3 Interface

PTAM has been modified as described above to provide the wanted data describing a filmed scene. With the data being available from the SLAM system, an interface was needed to access the data generated from the system and to save the data for further use in this project. To be able to add further SLAM systems like LSD-SLAM in the future, and not to be bound to PTAM alone, a wrapper class providing a fixed interface to control SLAM systems was implemented. This interface provides several functions to operate the corresponding SLAM system. The two most relevant are the *track()* function and the *saveToMap()* function. In the *track()* function the main loop of the selected SLAM system is executed, which usually performs the tracking and provides the data for performing the inherent SLAM algorithm. The *saveToMap()* function is usually called after the recording process of the selected SLAM system has generated enough information about the scene and the created data is ready to be used for further processing. This function takes care of extracting the keyframes—with the camera positions from where they have been recorded—and the points—with the list of keyframes from where they have been observed—from the memory of the SLAM in order to save them into the data collection of this project. After the data has been extracted successfully, the SLAM system can be shut down or used to create more data of the scene. Because PTAM needs a special initialisation method to start the tracking, the wrapper class has an *input()* function to send signals to the connected SLAM system if needed. With the help of these signals, the five-point stereo algorithm used by PTAM can be performed similarly to the original PTAM user interface. Another important function is used to obtain the camera parameters from the SLAM system, which are needed for further processing and displaying of the created data. These intrinsic camera parameters are usually used by SLAM systems to mitigate the inherent distortion of a camera. PTAM has a built in camera calibration tool to acquire these parameters. In more general cases the intrinsic camera parameters can be obtained by tools like the ‘Camera Calibration Toolbox for Matlab’ [4]. A table containing a collection of the available functions in the SLAM system wrapper class and their functionality can be found in appendix A.

The implemented wrapper class makes it possible to control the SLAM system in charge and to obtain the data created from it. It is also possible to add further SLAM systems to the project, which are not bound to any kind of sensors and only need to be able to create keyframes and point clouds during their data processing phase.

4.2 Reconstruction

After the data describing the scene has been recorded via a SLAM system, the acquired data needs to be further processed to obtain the wanted textured 3D model of the scene. The next step is to process the gained data into a polygon mesh describing the recorded environment. For this purpose, a reconstruction method as described in section 2.2 is needed. There are several given constraints for the reconstruction algorithm. One originates in the format of the data, another one results from the composition of the obtained points and the relative inaccuracy regarding the camera positions and the recorded points. The cause of this inaccuracy arises from the estimated depth values from monocular SLAM approaches. At this moment the available data consists of a number of keyframes, which have been recorded from the scene. These keyframes contain the camera coordinates from where each keyframe has been recorded and a colour image, which has been taken from the respective camera position. The available data consists furthermore of several points, which describe a rough outline of the given environment. These points possess a 3D coordinate in the scene and a list of keyframes in which the point has been seen. With the help of the keyframe references for each point, camera position coordinates—describing from which positions the point has been seen—are available to each point as well. The created point clouds from PTAM are sparse, this means that the obtained point cloud is not a detailed description of the recorded scene. This rather vague description of the scene cannot be compared to the resulting point clouds of laser scanners or other more sophisticated scanning techniques—see section 2.2. The camera positions for each of the recorded keyframes is—as mentioned above—only an approximation, due to the lack of knowledge about the depth of the scene. Another difficulty hidden within the recorded point cloud is that there is the possibility that several of the points in the point cloud are in fact outliers, which do not add information to the scene at all, but should rather be treated as noise to not corrupt the created polygon mesh. Because outliers pose a severe threat to most reconstruction algorithms and thus lead to meshes, which do not describe the scene accordingly, some filter techniques have been used to mitigate this problem.

The used filtering approaches are further described in section 4.2.2. Another constraint to the reconstruction method, not discussed yet, is that it needs to have a low processing footprint due to the fact that it needs to be able to operate on a smart device and thus does not have vast amounts of processing power available. With all of these constraints in mind, a suitable candidate reconstruction algorithm has been chosen. It was not possible to find a method, which is perfectly suited to all of the constraints, but most of them could be satisfied. The chosen approach will be described in the following section.

4.2.1 Incremental Surface Extraction from Sparse Structure-from-Motion Point Clouds

The reconstruction method which has been chosen was introduced by Hoppe et al. [32] in their paper named ‘Incremental Surface Extraction from Sparse Structure-from-Motion Point Clouds’. The term ‘Structure from Motion’ or short SfM describes a general problem in computer vision with the goal of estimating 3D structures from 2D images. Usually the same scenario as in SLAM (section 2.1) can be assumed as basis, which is a mobile agent who is acquiring information about the present environment. In the SfM case, the sensor input is restricted to a series of two dimensional images. With the help of the sequence of images, a structure of the environment has to be observed. Thus, the point cloud created by one of the SLAM systems can also be called a 'Structure from Motion' point cloud. The chosen paper explicitly deals with the problem of extracting a shape described from a SfM point cloud and hence seems to be well suited for this approach. The other part of the title also deals with the incremental creation of a surface, which is not mandatory for this project but can be used for further model creation. The incremental approach could be added to the scenario described in section 3.1 as another step of refining the initially created model, which may be needed if the first model does not contain enough information, or more data was recorded to refine the model.

The reconstruction method introduced by Hoppe et al. uses a 3D Delaunay triangulation on the given 3D points. The idea of the algorithm is then to determine whether each of the tetrahedrons is in an occupied space or in free space. These spaces are determined with the help of their visibility information given from the camera poses. After the tetrahedrons have been assigned to one of the two spaces, the underlying surface can be extracted by finding the boundaries between occupied and free space. This is a common approach of methods which try to reconstruct a surface from a SfM point cloud. Hoppe et al. proposed a novel energy function to extract the surface as accurate as comparable algorithms, but with far less computational effort. The formulation of their energy function is also adaptive to an updated or changing point cloud and therefore supports an incrementally increasing number of points. The newly introduced energy function uses the ideas behind the truncated signed distance function (TSDF) which is able to extract surfaces from dense point clouds. The TSDF connects a 3D point X and the camera position from where it has been seen with a ray and assigns to each voxel passed by this ray the probability of whether it is free or occupied space. To achieve good results with this method, a dense point cloud and therefore a huge number of points is needed. By having the voxels divided into visible and occupied space, the actual surface can be extracted as the crossing from one space to the other. Because the newly introduced energy function by Hoppe et al. needs to be able to extract the surface of a sparse point cloud, adjustments to the principle ideas of the TSDF had to be made. The idea behind the energy function is to advance the surface extraction as a binary labeling problem. With the help of the 3D Delaunay triangulation of the point cloud, a set of tetrahedrons were created and these tetrahedrons now need to be labeled as either free- or occupied space. The visibility information available—consisting of the rays between the camera positions and the extracted 3D points, which form the point cloud—is being used to assign the appropriate space type to each tetrahedron. Similarly to the TSDF, a tetrahedron has a high probability to be free space, if it is in front of a point X and on the other hand the probability of the tetrahedron to be occupied space is high, if it lies behind a point X. Another assumption is that neighboured tetrahedrons have a high probability to belong to the same space with the exception that this assumption is weakened if a tetrahedron is close to a point X. With this in mind, the problem was formulated as a pairwise random field. The random variables of the random field are the tetrahedrons V and the binary labels L, which result in the maximum a posteriori solution being looked for by using the visibility information R. This resulted in the pairwise energy function 4.1 with $\mathcal{N}_i$ being a set of the four neighbouring tetrahedrons of tetrahedron V_i and $\mathcal{R}_i$ being a subset of $\mathcal{R}$ which consists of all rays connected to the vertices that span V_i.

$$E(\mathcal{L}) = \sum_i (E_u(V_i, \mathcal{R}_i) + \sum_{j \in \mathcal{N}_i} E_b(V_i, V_j, \mathcal{R}_i)) \tag{4.1}$$

The unary costs $E_u(V_i, \mathcal{R}_i)$ use the idea behind the assumption of the TSDF where a tetrahedron is assumed to be free space if many rays pass through it, and occupied space if the tetrahedron lies in the extends of many rays. The unary costs for labeling V_i as occupied space are set to $n_f \alpha_{free}$ where n_f is the number of rays passing through and to on the opposite site when labelling as free space, the costs are set to $n_0 \alpha_{occ}$ with n_0 being the number for rays in front of the tetrahedron V_i. The example in part (a) of Figure 4.8 depicts the unary costs where only the light green ray passes through and contributes to n_f being 1, and n_0 being 3 because V_i is in extend of the three green rays. The red rays are not used to calculate the unary costs.

The pairwise terms describe the assumption that it is unlikely for neighboured tetrahedrons V_i, V_j to have different labels assigned to each other with the exception of pairs which have a ray R_k passing through the triangle connecting them. This results in the term $E_b(V_i, V_j, \mathcal{R}_i)$. Depending on whether a ray R_k intersects the two connecting triangles of two tetrahedrons V_i, V_j or not, $E_b(V_i, V_j, \mathcal{R}_i)$ is set to β_{vis} or β_{init} respectively. Part (b) of Figure 4.8 depicts an example of the pairwise costs.

Part (c) of Figure 4.8 shows a graph representation of the energy function. Here the costs for labeling V_1 as free are $3 \times \alpha_{free}$ because V_1 is passed by three rays. V_2 lies in the extend of these three rays and thus is connected to the occupied node and has the costs $3\alpha_{occ}$. The neighboured tetrahedrons get the weight β_{init} except the edges neighboured to V_1, which get β_{vis} assigned to them.

A global optimal labeling situation for the surface extraction problem can now be found with this formulation of the random field by using standard graph cuts. The developed energy function has a good accuracy and lower computational power than prior works as the results from this approach showed. With the help of the introduced energy function the underlying surface can be extracted. The information needed by this approach consists of a point cloud and the camera positions from where each point has been seen. Exactly this information is available from PTAM, which is the SLAM system of choice in this project. Because a sparse point cloud is supported by this approach, it seems to be well suited.

4.2.2 Filtering in PTAM

The point clouds created by PTAM had a vast number of outliers inherent to them, thus a filtering and refinement step seemed to be mandatory to be able to create approximated

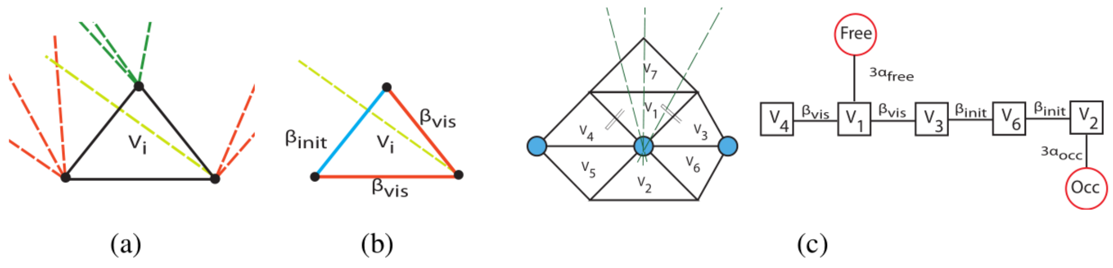


Figure 4.8: Graphical explanation of the (a) unary term, (b) the pairwise term and (c) of the graph representation of the pairwise energy function. Source of the figure: [32]

models of the filmed scene. Before advancing the filtering of the received point clouds from the SLAM system wrapper class, the concrete implementation of the used SLAM System has been investigated more detailed.

In the PTAM implementation, a built-in inlier- and outlier-count can be found for each point in the generated map. This data is used by PTAM to reject outliers on its own. Dependent on how often a certain point has been classified as an inlier or an outlier, the point is permanently removed from the generated map by PTAM. Because PTAM is cautious in removing points from its map, the map points and the generated inlier- and outlier-values of the points have been saved into a separate data structure for further processing. By manually adjusting the accepted ratio between the inlier- and outlier-count, acceptable results could be created. The picture on the top in Figure 4.9 shows the resulting point cloud of a textured cube filmed with PTAM before any outlier removal has been performed. The bottom picture of this figure shows the same point cloud after a certain threshold has been applied to the ratio between the build-in inlier-count and outlier-count.

Another observation made from the built-in outlier handling routine of PTAM was, that most points which have been classified as outliers, were removed from the map within a certain time after they had been added to the map. Figure 4.10 shows in its two pictures, which contain datasets of point clouds with different size, that most rejected points have been removed within the first 20 seconds after they had been added to the map. This observation can be used to only pass points of the created map from PTAM which have been added to the map for at least 20 seconds. Points below this threshold are very likely to be classified as outliers and thus it seems to be better to only use points for further processing which have passed this first outlier-test performed within the PTAM runtime.

4.2.3 Filtering outside of PTAM

For further map filtering, some built in algorithms of the 'Point Cloud Library' (PCL) version 1.7.2 have been used [63]. This filtering process is optional and the filtering can be done manually by the user after the point cloud has been recorded with the provided

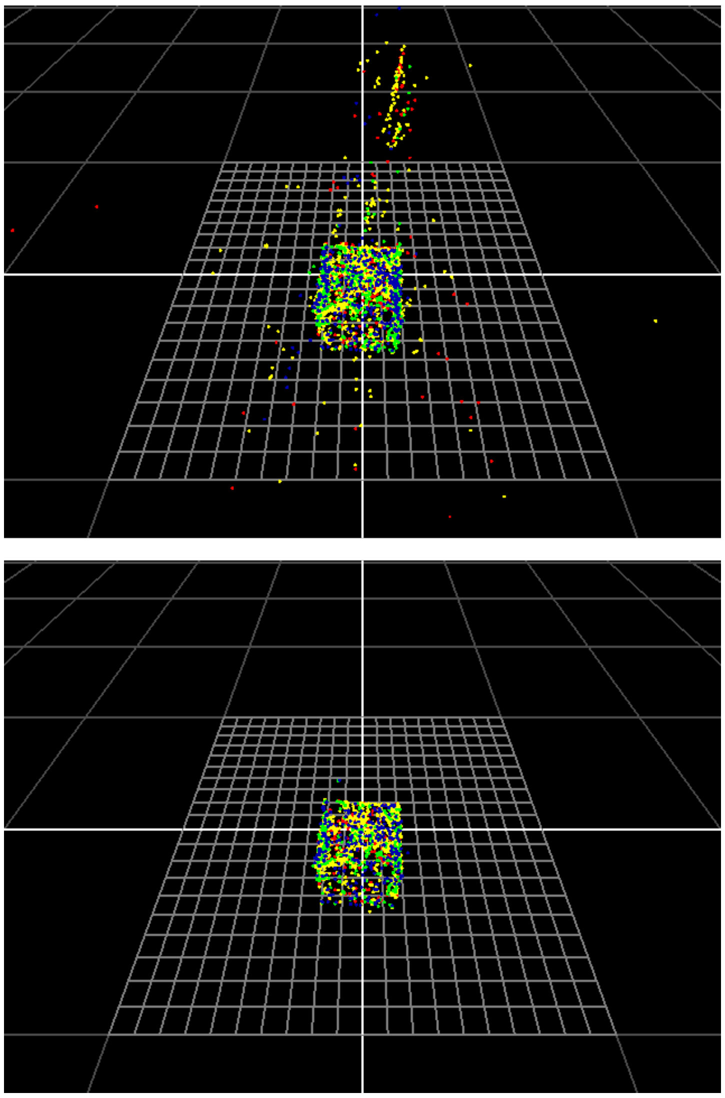
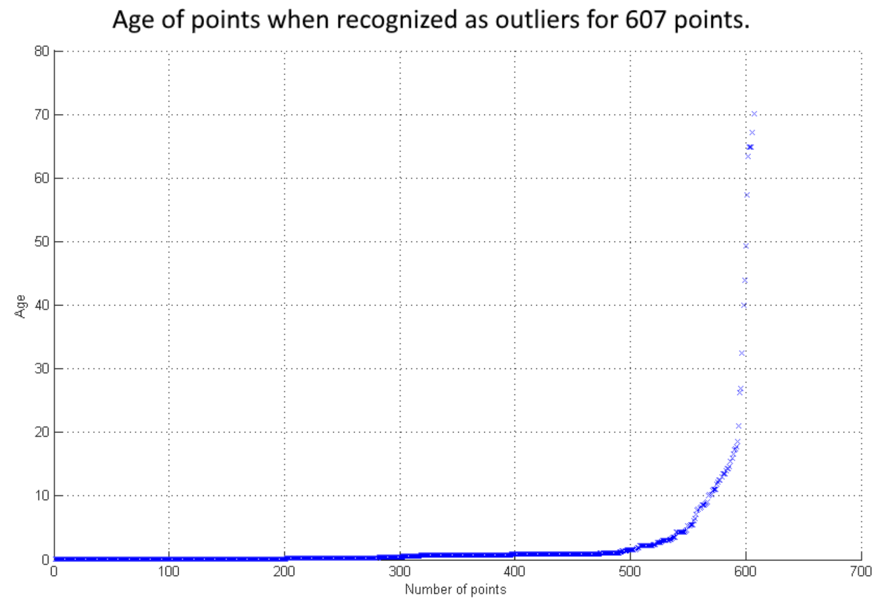


Figure 4.9: The picture on the top shows the resulting point cloud from PTAM when filming a textured cube. The picture on the bottom shows the same point cloud after removing all points exceeding a certain outlier threshold.

SLAM system. The graphical user interface of the implementation provides in total three filtering approaches to address problems observed in the generated point clouds.

The first filter used is the *statistical-outlier-removal* filter from the PCL. This filter performs a statistical analysis on the neighbourhood of each point and removes those points, which do not meet a certain criterion. This implementation calculates the distribution of distances from a point to its neighbours. For each point the mean distance to k neighbours is calculated. With a local distance mean for each point, the global distance mean is calculated. All points, which have a distance longer than d times the standard deviation



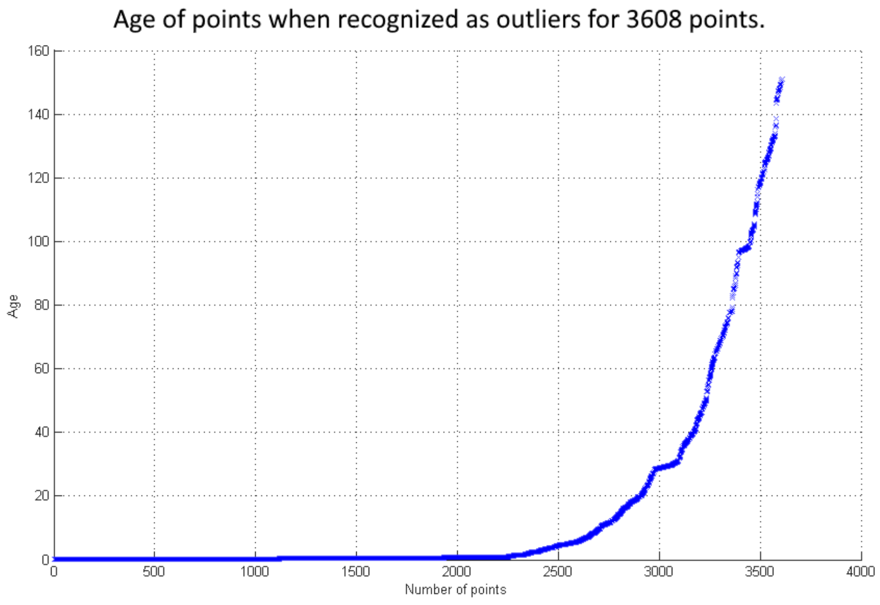


Figure 4.10: The picture on the top represents data of a point cloud with 607 points, the bottom picture a point cloud with 3608 points. The x-axis depicts the number of points being removed and the y-axis shows the age of the points in seconds when they have been identified as outliers.

are marked as outliers and will be removed from the point cloud. The two variables k and d, which represent the number of neighbours, whose distances are wanted to be analyzed for each point, and the standard deviation multiplier, can be modified over the provided interface. Figure 4.11 shows on the left side a point cloud before and after applying the statistical outlier filter to it and on the right side a graph of the k-nearest neighbours for each point before the filtering in red and after the filtering in green.

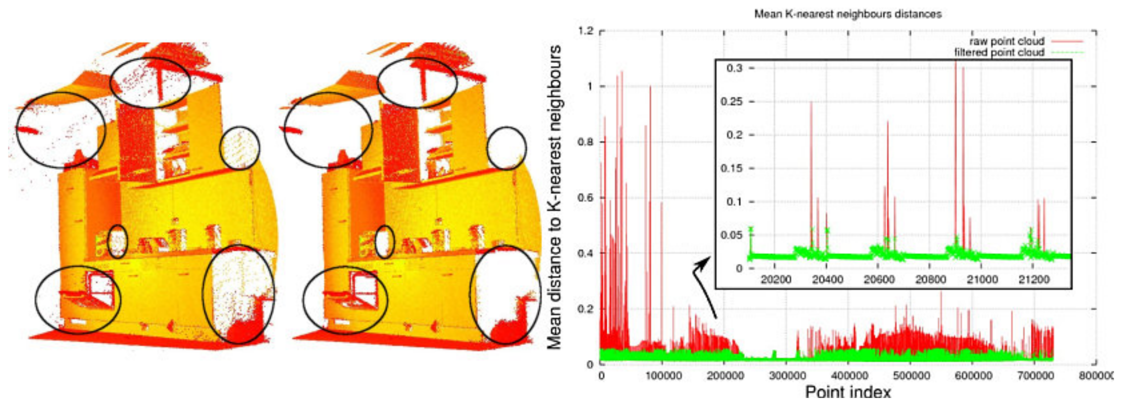


Figure 4.11: This figure from the PCL [63] documentation shows in the left part a point cloud before and after filtering with the statistical outlier filter and the statistics in graph form on the right part.

The second filtering approach used from the PCL is a *Moving Least Squares* (MLS) surface reconstruction method. It directly addresses the problem of noisy data, which originates from uncertain depth value approximations in monocular SLAM systems. Noisy data can be smoothed and resampled with the help of this algorithm and thus shows improvements to the point clouds generated by PTAM. By using higher order polynomial interpolations between the surrounding points, this algorithm tries to resample the given point cloud. With the provided user interface, a sphere radius r can be passed, which is used to determine the k-nearest neighbours for the fitting process. Figure 4.12 shows on the right side the estimated surface normals of a point cloud, which is the result of combining two data-sets. The arrows on the bottom left part of the figure clearly depict the problem inherent to this combined point cloud. The right part of the figure shows the results after the *Moving Least Squares* algorithm has been applied to the point cloud.

The last used filtering technique provided in this implementation is called *VoxelGrid Filter*. This filter is used to downsample a given point cloud. Even though the created point clouds from PTAM are already relatively sparse, and thus a downsampling algorithm does seem to be controversial, it turned out that by downsampling the point clouds, bumpy and uneven surfaces could be smoothed out, resulting in a better representation of the filmed scene. The algorithm provided by the PCL creates a 3D voxel grid over the given point cloud and then iterates over each created box and approximates the centroid of all

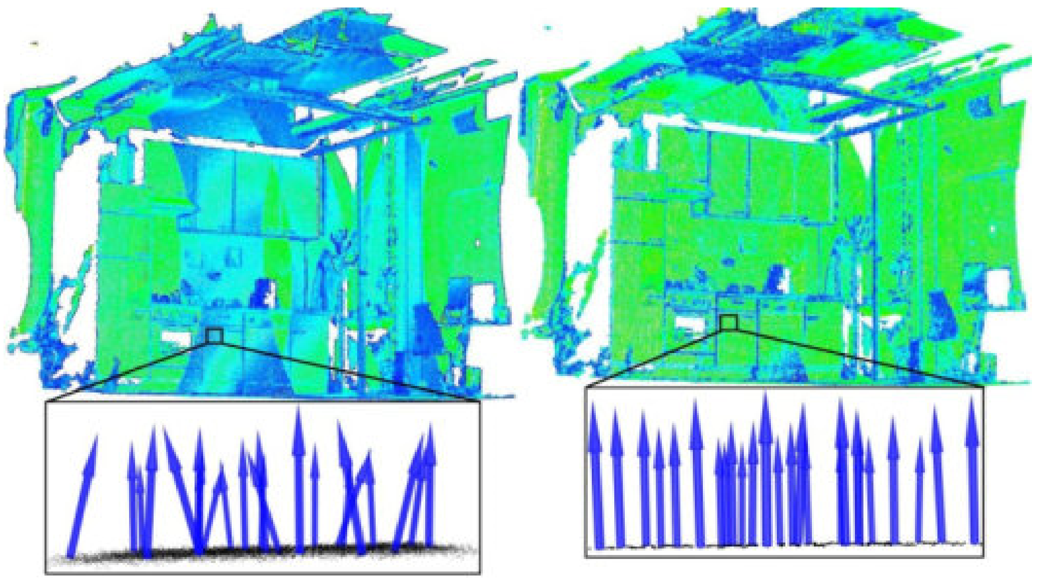


Figure 4.12: This figure from the PCL [63] documentation shows in the left part a point cloud before, and on the right side after smoothing with their MLS approach.

vertices in each box of the grid. With the provided GUI a size s can be passed, which represents the side length of the cubic grid elements. Figure 4.13 shows on the left side a high density point cloud and on the right side the same point cloud downsampled with the *VoxelGrid filter* from the PCL. It can be seen, that even though the example point cloud becomes sparser, the shape of the presented object is still preserved. This is crucial, because the 3D model created from the resulting points should be able to clearly represent the filmed scene.

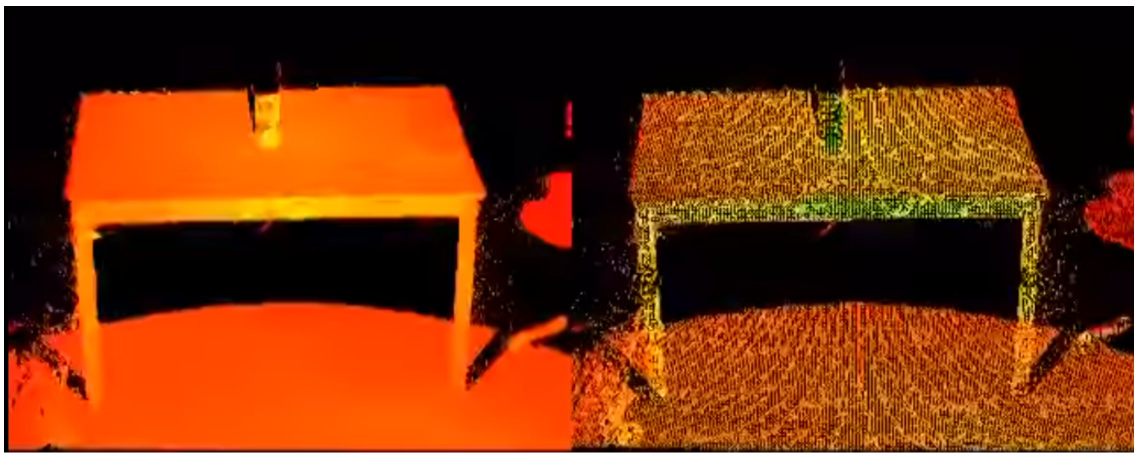


Figure 4.13: This figure from the PCL [63] documentation shows on the left side a dense point cloud and on the right side the same point cloud downsampled with the PCL $VoxelGridfilter$

The user interface provides pre-set values for each filter, which have been tested for the point clouds created from the supported SLAM system, but can be adjusted for further refining. The combination of these filtering techniques is able to get rid of most of the problematic artefacts of the created point clouds, but there is still space for improvements. Outliers still create problems during the reconstruction process.

4.3 Image Based Rendering

At this point, the gathered data from the chosen SLAM system has been polished with the help of some filtering methods of the PCL and a 3D model has been created by the chosen voxel carving algorithm—Incremental Surface Extraction from Sparse SfM Point Clouds. The next step is to apply textures onto the model to let it appear in a more realistic way. In section 2.3 several IBR methods have been introduced, which are capable to solve this task. The IBR continuum—shown in Figure 2.8—helps to narrow down the possible solutions. The spectrum divides the different approaches by their needed input. To recap, the continuum starts on the one side with approaches needing explicit geometry, going to methods requiring implicit knowledge, and finally solutions, which do not need any geometric information. Because a 3D model is already available at this point, the IBR system must be in the group, which requires explicit knowledge about the underlying geometry. This already satisfies the first constraint regarding the texturing approach listed in section 1.2. The other two constraints demand a certain input data for the IBR system. The available data consists besides of the already mentioned 3D model, of images taken from the model and the camera positions from where the images have been taken. Because the existing 3D model was created from a sparse point cloud, it is not rich in details. The colour-images are available in the common RGB format and the camera positions are, as described above, only an approximation, because monocular SLAM systems are not entirely able to deliver highly accurate positions.

With these constraints in mind, an image based rendering approach was chosen, which is able to process the given input. Again, the chosen solution does not strictly fulfil all constraints, but most of them could be satisfied. The chosen approach is described in more detail in the following section.

4.3.1 Filtered Blending

The texturing method, which has been chosen was introduced by Eisemann et al. in 2007 in their paper named ‘Filtered Blending: A new, minimal Reconstruction Filter for Ghosting-Free Projective Texturing with Multiple Images’ [20]. They proposed a GPU-based method, which is capable to deal with imprecise 3D geometry and a small number of available images. This already shows that their approach fits the outlined requirements above. Especially the fact that the 3D model, which is needed as input, does not have to be extremely precise, eliminates one of the core problems of the model created from the

SLAM point cloud. That already a small number of images is enough for this IBR method is a positive aspect as well, since it reduces the amount of data sent over the network.

In their work, they presented at first some research on when and how ghosting effects appear. They came to a definition of ghosting, which generally holds for IBR methods. Picture 4.14 from [20] depicts this definition. Here L represents an actual point in the scene and C_1 and C_v are two cameras viewing the point. The point L is projected onto two different points F_0 and F_1 on the approximated surface S. The conclusion from Eisemann et al. was, that if the distance from F_0 to F_1 is larger than half a pixel in the input images, the ghosting effect occurs. It is also pointed out, that their definition can be reduced to the one proposed by Lin et al. in [44].

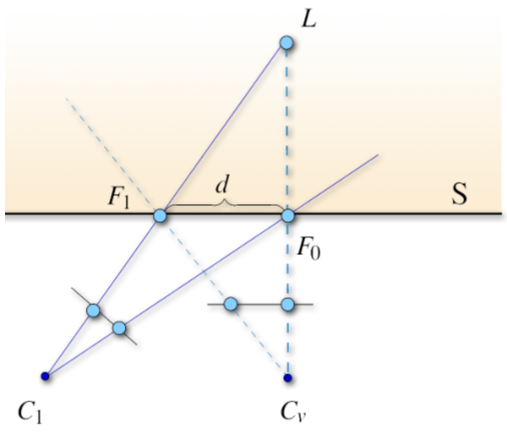


Figure 4.14: Ghosting in projective texture mapping, view-dependent texture mapping and light field rendering from [20]

To illustrate their approach, they used a second example shown in Figure 4.15. Here they explain that the scene point L lies on the viewing ray of C_v to L_{p_0}. Furthermore is known, that L can only lie in the boundaries of the maximum depth uncertainty of the surface. Now, the colour value observed from camera C_1 has to lie somewhere on the line between the projected texture coordinates $T_{L_{p1}}$ and $T_{L_{p2}}$, which they call the line of disparity. The resulting uncertainty problem was now solved in a resampling process. After moving to frequency domain, they concluded from their prior statement about ghosting, where ghosting is prevented, when the projected disparity is less than $\frac{1}{2}$ texel, that they have to remove all frequencies higher than $\frac{1}{2d} \times \omega_t$. Where ω_t is the highest representable frequency in the texture function t, and t is given by its resolution and the Nyquist Theorem. After considering appropriate sampling positions and performing an anisotropically resampling of the texture function, they are able to avoid ghosting. Because they use the current viewport of the virtual camera in their calculations, the results will be detailed, when the virtual camera comes closer to one of the input cameras, since viewer frequencies from the input image will be cut off. If both cameras have the same view, all details from the image is present because nothing will be cut off. The authors of the paper implemented the proposed method with strong GPU usage and were able to render a test data set in real-time while moving the virtual point of view.

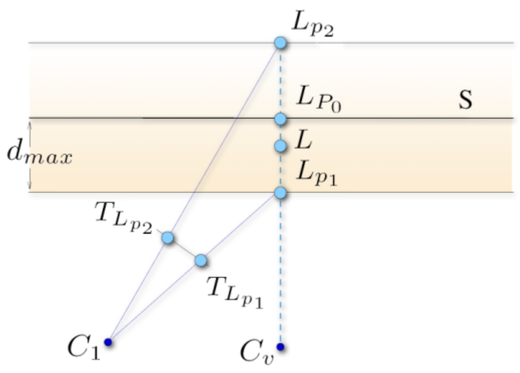


Figure 4.15: Scene point estimtion from [20]

Figure 4.16 compares the results of the Filtered Blending approach to Linear interpolation and Band-limited reconstruction. Strong ghosting can be determined in the Linear interpolation approach in the second and third row while the Band-limited approach removes ghosting, the results become blurry. The results of the presented Filtered Blending approach show that aliasing was completely removed and that the edges are very sharp.

Before it was possible to integrate this IBR method into the processing pipeline of this project, some changes had to be made. The original implementation of Filtered Blending does not support incremental updates of the textured scene. Therefore, a new interface to the Filtered Blending framework was developed and an update function which is able to integrate new information to the current scene was implemented. Now the initially created 3D model is passed to the Filtered Blending routine along with the recorded images of the scene and their camera positions. When an updated 3D model of the scene or images from new camera positions are available, the added update function of the IBR approach is called and the new data is seamlessly integrated. With the help of the update function, the transmitted data can be kept low and the created textured scene is constantly growing.

4.4 Instruction Handling

Now, the textured 3D model is available, and the next and final part of the processing pipeline is to add instructions, which depict how the shown object can be operated with. In section 3.1.3 it was elaborated, that geometric primitives provide enough information, if they are transformed over time to depict the needed operation. It was shown in Table 3.1 that all common interactions with the environment can be reduced to a rotation, a translation or a combination of both of them. Gauglitz et al. followed a very similar approach in [28], where they decided to use drawn dots and arrows on the given 3D model to describe

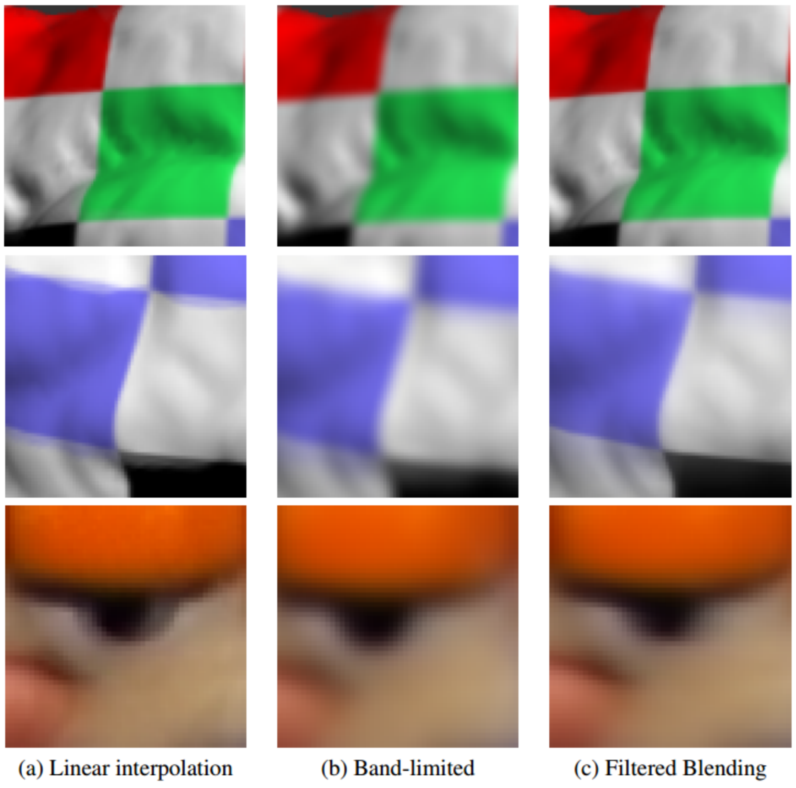


Figure 4.16: The first two rows show projective texture mapping of the Stanford Bunny. Results for a real-world data set of Garfield are shown in the third row. Picture from [20]

certain tasks. While this approach leads to satisfying results, a clearer representation of the needed user interaction is wanted for this work. Thus, further investigations regarding common user interactions with the environment were made, resulting in the introduction of control elements.

4.4.1 Control Elements

A result of these observations was, that a few different shapes of primitives are enough to represent most common control elements in the real-world. These are a circle-like element for the interactions where the user has to rotate something like a door knob or a volume controller and a rectangular element, which can represent door-like interfaces. Of course there are always some interfaces which can not be depicted by those two forms. Thus, it was decided to implement the two most common control elements, which are the circle and the rectangle, and a third one which is called the free-form-element, where the shape can be defined by the user. With the knowledge of the different forms, the next step to think about was, how to describe a motion with a geometric primitive instead of using arrows. Since the chosen forms are already close to the forms of the real-world control elements, it seems appropriate to animate the primitives in a way the control element has to be moved. While the movement of a door, which is a simple rotation along one axis, can be easily imitated by a rectangle, it is not possible to see whether a perfect, untextured circle rotates or stands still. This observation lead to some adjustments of the newly introduced control elements and their appearance. First, an octagon-shaped form was implemented besides the circle, to be able to clearly determine a rotation and secondly, it was decided that the added primitives need to be textured, so that the connection between the control elements and the real-world can be noticed immediately. Figure 4.17 shows example control elements without textures.

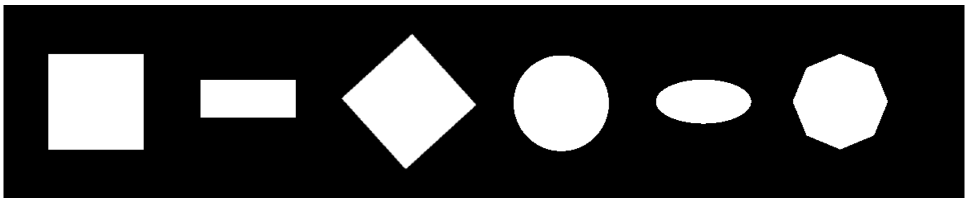


Figure 4.17: This figure shows example control elements without textures applied to them.

After the appearance of the introduced virtual control elements was known, the animation part of them still had to be thought about. The wanted transformations, rotation and translation, can be performed with simple matrix multiplications and did not pose a problem. To replay the transformations over time, some more thoughts about the general scenario had to be made. The conclusion was, that the user adding the control elements to the created 3D model has to define a starting position of the wanted animation, then has to perform the wanted transformation with the control element and finally has to define the end position of the animation. While the motion is performed by the user, each rotation and translation has to be registered to be able to properly replay it on demand.

To correctly reproduce the recorded motion, the rotation and the translation had to be handled in a slightly different way. The new model matrix M' after a translation is calculated as shown in formula 4.2, where T is the matrix describing the wanted translation and M is the current model matrix. When updating the position of p several times in sequence with multiple translations it can be observed that the point p after the first iterative update can be described as shown in formula 4.3, with p' being the new point p and T_1 symbols the first translation matrix used. The next performed translation can therefore be described as shown in 4.4 and finally after n iterations this process results in the equation shown in 4.5. Thus, the single matrices describing the total translation can be multiplied by each other iteratively, which then properly displays a matrix containing the total translation performed.

$$M' = T \times M \tag{4.2}$$

$$p' = T_1 \times M \times p \tag{4.3}$$

$$p'' = T_2 \times p' = T_2 \times T_1 \times M \times p \tag{4.4}$$

$$p(n) = T_n \times \ldots \times T_1 \times M \times p \tag{4.5}$$

When performing a rotation, a point r describing the centre of the rotation is needed as well as another point p, which has to be rotated. Now the new model matrix after a rotation can be described as shown in equation 4.6. Where $T_r = T \times (M \times r)$. The iterative update of several rotations in a row is derived as shown in formula 4.7 for the first iteration, where p' is the point p after the first rotation is applied. Since the model matrix M is initially the identity matrix I, this can be written as depicted in 4.8 and simplified to formula 4.9. The second iteration can now be written similarly as formula 4.10 shows, which equals formula 4.11. Because the point of rotation r remains the same, $T_1 = T_2$ and thus equation 4.12 holds, which can be simplified to formula 4.13. This finally results, after n iterations to equation 4.14, where $R_n \times \ldots \times R_1$ represents R_{total}, the total rotation performed. With the matrices describing the total translation and total rotation present, the next step is to apply these over time to a given control element. This is again done in slightly different ways for the translation and the rotation part. For the translation, a linear interpolation between the start and the end position was implemented. To perform the rotation with the given data, the matrices describing start and end position have been converted to quaternions and then a spherical linear interpolation was performed between the two of them.

$$M' = T_r \times R \times T_r^{-1} \times M \tag{4.6}$$

$$p' = T_1 \times R_1 \times T_1^{-1} \times M \times p \tag{4.7}$$

$$T_1 \times R_1 \times T_1^{-1} \times I \times p \tag{4.8}$$

$$T_1 \times R_1 \times T_1^{-1} \times p \tag{4.9}$$

$$p'' = T_2 \times R_2 \times T_2^{-1} \times p' \tag{4.10}$$

$$T_2 \times R_2 \times T_2^{-1} \times T_1 \times R_1 \times T_1^{-1} \times M \times p \tag{4.11}$$

$$p'' = T_1 \times R_2 \times T_1^{-1} \times T_1 \times R_1 \times T_1^{-1} \times M \times p \tag{4.12}$$

$$p'' = T_1 \times R_2 \times R_1 \times T_1^{-1} \times M \times p \tag{4.13}$$

$$p(n) = T_1 \times R_n \times \ldots \times R_1 \times T_1^{-1} \times M \times p \tag{4.14}$$

With being able to record and replay the motion of the control elements, the last task is to apply a texture onto the elements. Instead of using generic templates to represent the surface of the control elements, it was decided that the actual texture of the object possesses more relevant information. Meaning, that for example a door is better described by a picture of that particular door, instead of just a template displaying a generic example door. Another advantage of using the actual texture of a given object is that no templates have to be prepared and chosen by the user, which simplifies the process of adding control elements to the scene. To obtain the texture of the 3D object for a newly added control element, the scene was first viewed normally to the control element to avoid any further distortions of the texture. From this view, the part which is being represented by the control element was cut out and saved as texture for the said control element. To be able to distinguish the rendered scene and the control element representing one part of the scene, the RGB values of the copied texture were manipulated to have a clear contrast to the shown model. This contrast was achieved by amplifying the red colour channel and reducing the intensity of the other two channels. Figure 4.18 shows the view of the server program with a control element applied to the filmed object.

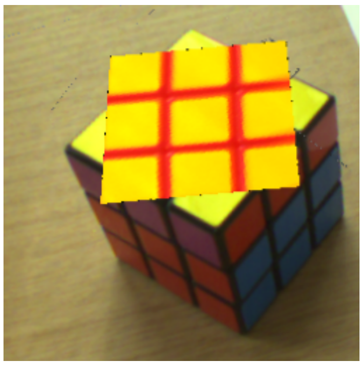


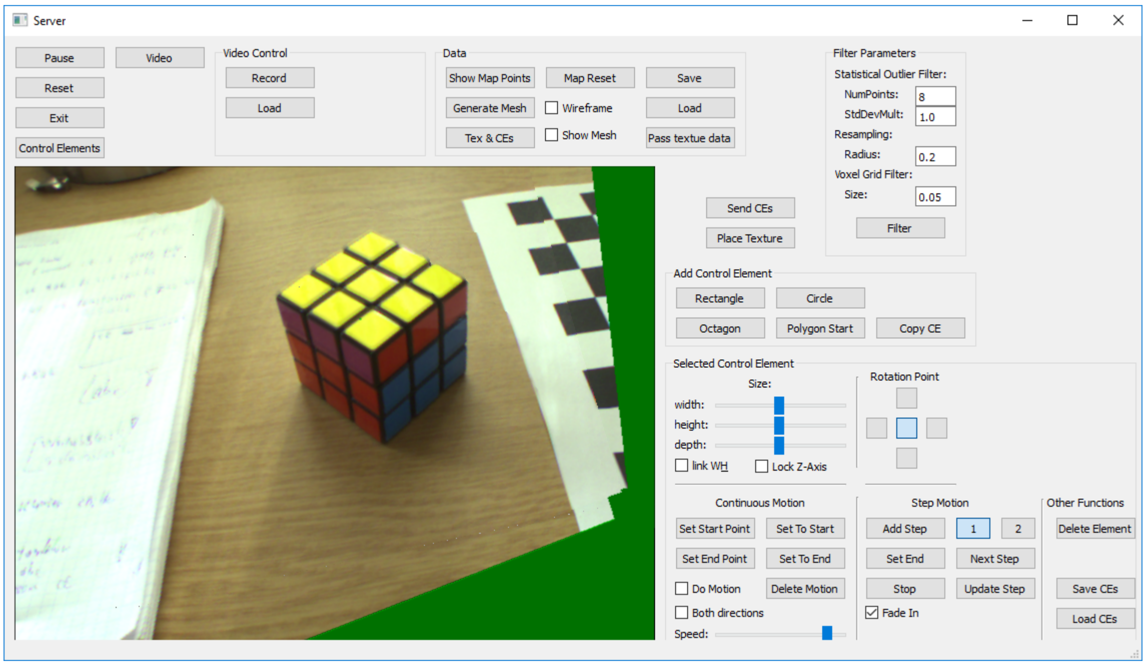
Figure 4.18: This figure shows a rotating example control element applied onto the 3D model of a magic cube.

With the newly introduced concept of control elements, it has to be decided how these are presented to the users. The possibilities of displaying the information and how these were implemented are discussed in the following section.

4.4.2 Displaying Instructions

At this point, the final control elements are available and have to be displayed to the users. As discussed in section 3.1.3, there are two possible ways which come in mind and will be described in more detail. Before discussing the two ways of representation, it has to be known that the project is divided into two parts—client and server—which represent the interface of the remote and local person in the example from section 3.1. The remote person will from now on be referred to as the server and the local person as the client.

The naïve approach would be not to differ between the two users, where the client which is receiving instructions from the server sees the same information. This results in the control elements being added to the 3D object and the textured object including the control elements is shown to both users in the same way. An advantage here is definitely that both users see the exact same results and the chance of misinterpretations of one user is relatively small. The model could also be saved for offline usage. On the other hand, this solution requires a better understanding of the given scene from both persons, since it is not always trivial to interpret a generated 3D model. It was mentioned further in section 3.1.3 that a description of certain tasks with an offline model can be saved and provided as an addition to a user manual. Thus, this approach was implemented by giving the client the possibility to either see the video, which is currently being filmed, or the created 3D model of the scene including the control elements added by the remote person. The server only sees the 3D model and has an interface, which provides the tools to add control elements and send them to the client. Figure 4.19 shows the user interface of the server, which contains a 3D model on the left side, and the controls for adding and modifying control elements on the right side. The green areas around the model are the outer boundaries of the 3D model.



NumPoints: 8
StdDevMult: 1.0
Radius: 0.2
Size: 0.05
☐ Wireframe
☐ Show Mesh
☐ link WH
☐ Lock Z-Axis
☐ Do Motion
☐ Both directions
☑ Fade In

Figure 4.19: This figure displays the view of the server program with a loaded 3D model and a control element applied to it.

The second approach to displaying the created data to the client and the server is to differ between what both parties can see. Here, the view of the server remains the same, but the view of the client was simplified. The client can no longer see the created 3D model and is only provided with the added control elements, which are shown as augmented reality overlay on the displayed video. To implement this approach, the control elements were no longer rendered in respect to the view matrix of the 3D model, but with the current view matrix describing the camera position filming the scene. Figure 4.20 shows a quadratic control element applied onto the top side of a magic cube. The textures of the magic cube have been applied to the control element, and a red tone was added to make the element distinguishable from the actual cube. Furthermore, a horizontal clockwise rotation was assigned to the element to indicate how the top part of the magic cube has to be moved to solve the riddle. Here the advantages of this approach can be seen. The created 3D model, which only shows a part of the real scene and has relatively sparse information is removed and instead the total environment can be seen through the video, which is currently being recorded by the client. This way, the client is not only able to view the control elements directly on the environment and is no longer challenged with the abstract model, but is also not limited by the relatively small boundaries of the model and can now move freely. More examples of applied control elements can be found in chapter 5, where different objects and scenes are shown.

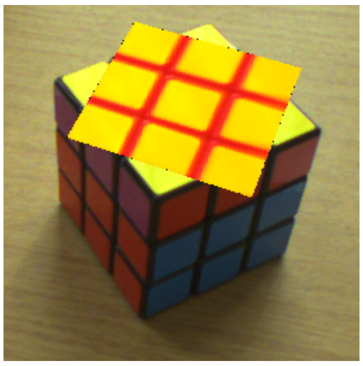


Figure 4.20: This figure shows a rotating example control element applied onto a magic cube as an AR overlay.

5 Results

This section presents the achieved results from the implementation of the project described above. All recordings were made with a uEye UI-2210SE-C camera shown in Figure 5.1. This camera has a resolution of 640 x 480 and a Bayer RGB color. It is connected over USB 2.0 and supports a framerate up to 75fps. Before filming, the camera was calibrated with both, the 'Camera Calibration Toolbox for Matlab' and the built-in camera calibration from PTAM. There were no noticeable differences between those two calibration techniques. The shown examples were filmed in- and outdoor to cover both of these lighting scenarios. It was also tried to film examples, which were presented in the works of Gauglitz et al. [28, 29] to be able to compare their system to the one presented here in a more meaningful way. In example 5.1 the views of both, the client and the server application will be shown and described. The strengths and weaknesses of each of the scenarios will be discussed, which depend not only on the surface and the complexity of the object, but also on outside factors like lighting, which is the most important one. After presenting the strengths and weaknesses of the system with several examples, it was tried to find scenarios, where the presented system breaks and is not able to deliver acceptable results in section 5.7. These factors for failure of the system are again dependent on either the complexity of the filmed scene or the given lighting conditions.

Figure 5.1: This figure shows a uEye UI-2210SE-C, the camera used for recording.

5.1 Car motor compartment

The first example was filmed outdoor and shows a part of the motor compartment of a car, where several liquids can be changed. This object was also used by the works of Gauglitz et al. [28, 29], because it is a really good outdoor example. Cars are ubiquitous in our modern society and thus are available to almost everyone and also the maintenance, shown in the examples, which is performed in the motor compartment can be performed by anyone. Another positive aspect of the motor compartment of a car is, that it is not viewable from all sides, and thus the 3D model which has to be created while observing the object must not be viewable from all sides, but only from the top and the sides, which simplifies the creation of the 3D model and results in a better representation. Objects, like the motor compartment of a car, where not all sides of it can be viewed will further be referenced as 2.5D objects. It can be observed that almost all user interfaces are 2.5D objects and only in few cases all sides of the interface possess controls. While the motor compartment of a car is well suited, a bicycle poses significant problems due to its delicate, non-solid body. The pictures discussed in the following have all been made in bright sunlight which simplifies the work for the tracking part due to the well-lit objects, but the strong sunlight also creates a lot of shadows while filming which has to be avoided to not confuse the tracking with changing and moving shadows.

With the rough outline of the setup being explained, the results of it have to be discussed next. The first picture shown in Figure 5.2 simply shows the filmed motor compartment of two different cars. The shown arrows were added to indicate which control units are going to be described in the following by the presented project. These are the interfaces, where the windscreen wash, the coolant and the oil can be refilled. All three of them can be opened and closed by a rotation of the cover.

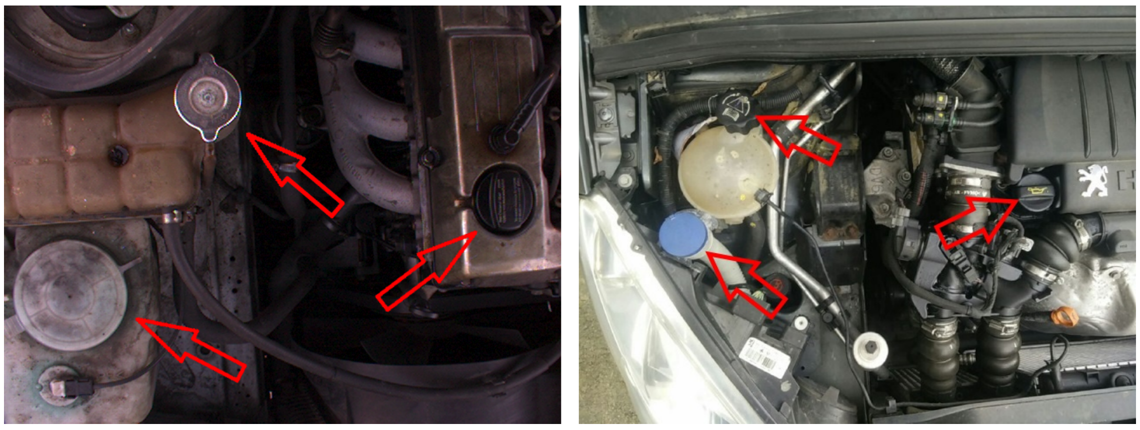


Figure 5.2: This picture depicts the interfaces of two different cars. The ones for refilling the windscreen wash, the coolant and the oil are being marked with arrows in both cars.

Figure 5.3 shows the view, which can be seen in the client part of the project, while recording the wanted part of the car. The first thing that can be noted, are the coloured dots, representing the features found by PTAM during the tracking. These are usually not shown to the user, but are enabled in this particular image to create a better understanding of the kind of features used by the tracking algorithm. It can also be seen how few points are used during the tracking and hence can be deferred how sparse the created point cloud is. Besides the coloured points, which are projected as augmented reality overlay onto the scene by PTAM, the interface of the client program can be seen. The interface contains options for controlling the operating SLAM system, e.g. saving and loading data and other controls for debugging, like displaying or hiding the coloured dots.

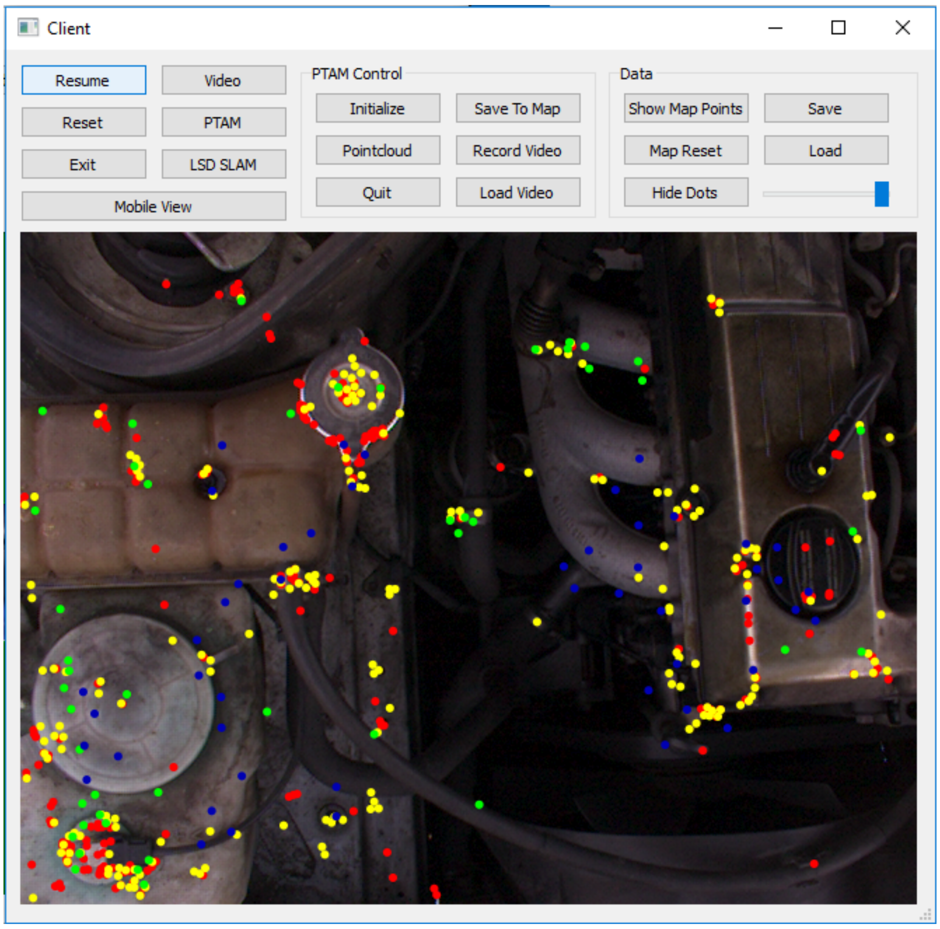
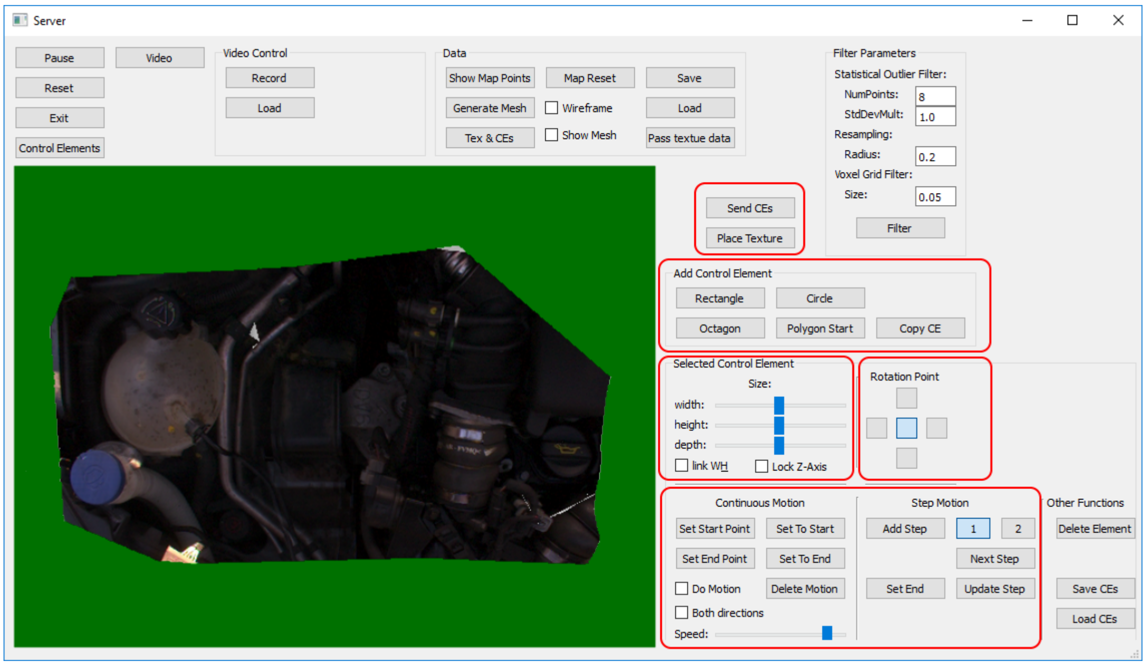


Figure 5.3: This figure shows the view of the client part of the project while recording an object. The coloured dots are features, detected and used by PTAM for tracking.

The next picture shown in Figure 5.4 shows the view of the server part of this project, after the received data has been processed into a 3D model. The green parts seen around the model are areas where no suitable data was present at this point yet and thus contains no further information. The view present to the user of the server part of this project can also freely rotate and translate the model and zoom in and out. To simplify the work of the user of the server part, a live video stream is present in a separate window, which can not be seen in this picture, and thus a clear understanding of the given scene should be possible, in the case when the created 3D model is not sufficient enough to understand the scene. Now the operator of the server can either add control elements to the shown model and send them back to the client, or wait until more data is recorded by the client and the shown model becomes bigger and more understandable. Various options are present in the user interface to add and modify a control element. In this figure, the important sections for interaction with an element are grouped and marked by red rounded rectangles. The second box from the top contains the 'Add Control Element' box, where the different available shapes for control elements can be chosen, or an already existing element can be copied. Below, two boxes can be found, where the left one is used to change the width and height of the selected element, and the right one offers interfaces to choose between five available rotation points. The box at the bottom contains the options for recording and applying a motion to the currently selected control element. The two types of motion are the 'Continuous Motion' and the 'Step Motion'. The first is used to create a smooth transition from a start point to an end point. The second motion type is used to record several control element positions, which the control element will jump to successively. A different time can be set to each step, which defines how long the element will remain on the same position. Finally, the box at the top contains two buttons, which allow to texture the selected control element and to send all control elements with their applied textures and motions to the client program. When control elements are applied and afterwards new data is being received from the client, the model will be updated and the already applied control elements will be positioned accordingly on the new, updated model.

The process of how to apply control elements to a 3D model which was created from the received data on the server part of the project is elaborated in Figure 5.5. Here the model shown in the previous figure was used to guide through the routine of adding control elements. In the top left picture of this figure, the button for adding a circle shaped control element is being used to select the type of element to add, then the user clicked onto the wanted part of the model, resulting in an initially blue circle being projected onto the geometry. As explained in Section 4.4.1, an octagon shaped form was implemented besides a clean circle, because a rotation can more clearly be observed in the form of an octagon instead of a circle, but because the texture of the oil refill lid is distinct enough, the circle was chosen in this case. In the top right picture of this figure, two interactions can be observed. The first performed action was, that the 'Place Texture' button was clicked, resulting in the texture of the 3D model being projected onto the previously blue control element. It has to be noted, that the colour of the red colour channel is intensified to create a contrast to the real-world object. This simple change creates a texture which is highlighted and still be recognizable as the original object. The second action, which has been performed in this picture, was to lock the z-Axis of the control element, so that it can no longer be changed. This action was performed in preparation for the next part of



NumPoints: 8
StdDevMult: 1.0
Radius: 0.2
Size: 0.05

Figure 5.4: This picture shows the view of the server part of the project after enough data was received to create an initial 3D model. The marked regions in the user interface show the controls needed to add, modify and send control elements.

the picture. In the first picture of the second row, it can be observed that at first, the ‘Set Start Point’ button was pressed, signaling the program, that every transformation from this point on has to be recorded to be able to properly reproduce it when replaying the motion. After the start button had been pressed, the control element was rotated around its centre, while the z-Axis remained locked, so that the entire rotation was performed at the chosen height. While this example needed a rotation at the centre of the control element which is the standard rotation point, the rotation point can be altered with the help of the user interface. In the ‘Rotation Point’ menu of the server program, the rotation point of the control element can be altered. This part of the user interface allows to set the top, bottom, left, right and of course the centre of the control element as centre of the rotation. The ‘Set End Position’ button has to be pressed, when the end of the transformation has been reached. After pressing this button, the control element will automatically move repetitively from the set start point to the endpoint. As the second picture in the second row shows, more than one control element can be added and modified at once. Here another circular element was added, textured and a motion was applied as well. The two pictures in the third row of this figure show the process of altering the size of a control element. It is possible to change the width and height of the added element separately or to adjust them together. Here, the option to scale the element equally in both directions has been chosen to maintain the circular form of the element. Finally, the 'Send CEs' button has to be used, which allows the server to send all added control elements with their textures and recorded transformations to the client, where they will be shown as augmented reality overlays over the video shown on the client program.

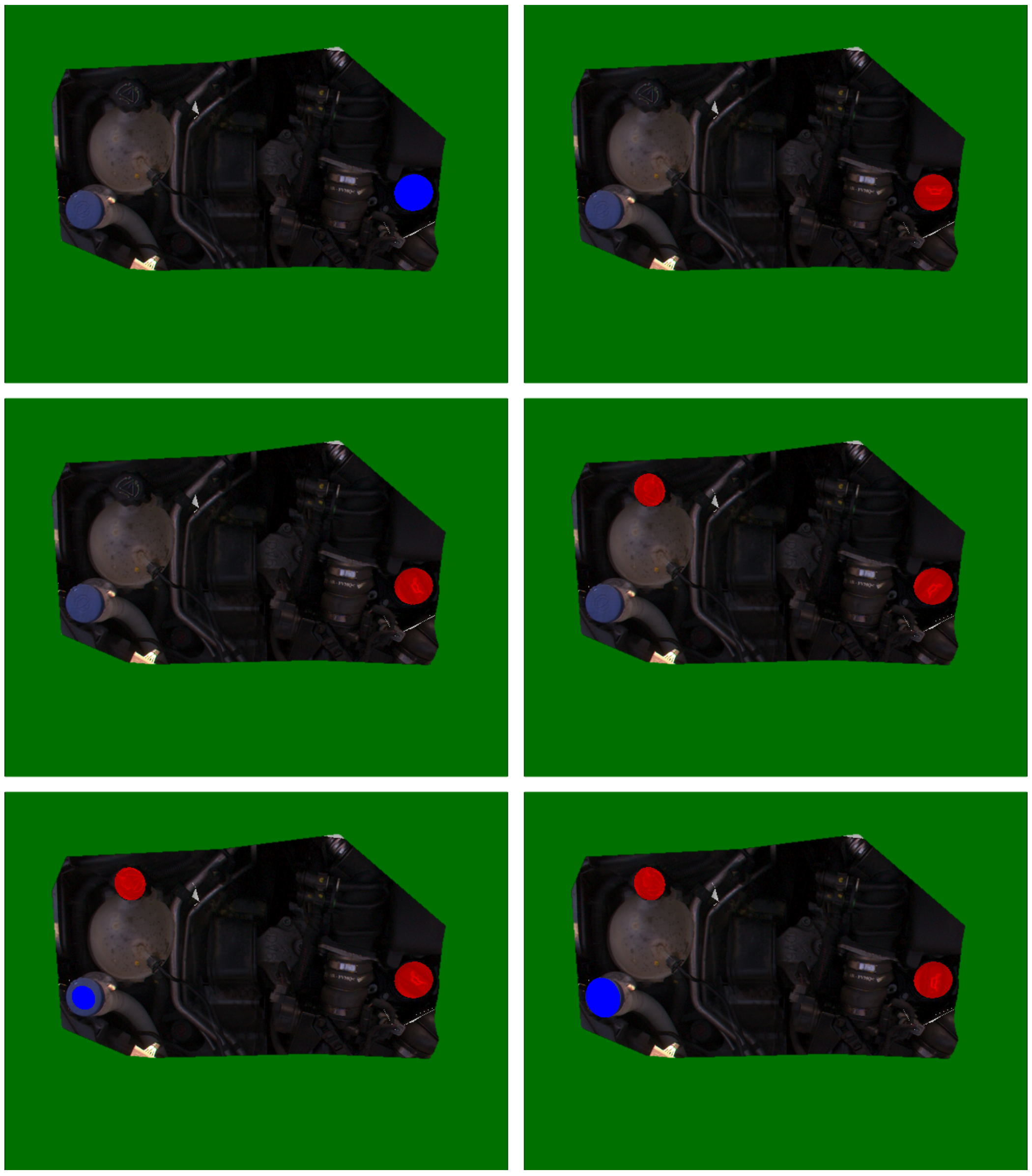


Figure 5.5: This figure shows in its six pictures how a control element can be manipulated with the server program. The first two pictures show how a control element can be added and how a texture can be applied to it. The following two pictures show how a motion can be recorded and set to an element, and finally the last two pictures show how several control elements can be added and how their shape can be changed.

Figure 5.6 shows the view of the client program during filming, after control elements from the server have been received. The three added elements can be seen in the marked regions of this picture. All three control elements were crafted a bit smaller than the three original refill caps on purpose to not entirely occlude them with the added augmented reality overlay. While the two elements on the top can be seen in a strong red tone, the element on the bottom left has a slightly brighter red tone. This is the result of increasing the intensity of the red colour channel, but not altering the other channels. This way, the original colour is slightly preserved and results in the brighter appearance of the bottom left element.

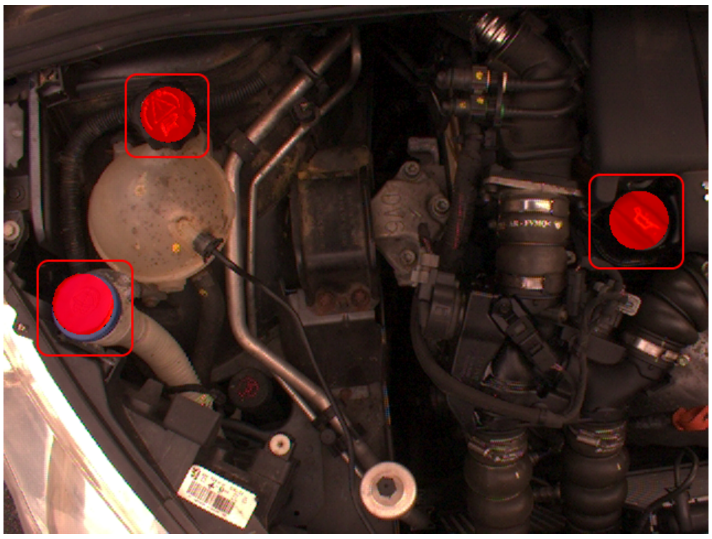


Figure 5.6: This picture shows three added control elements as augmented reality overlay on the live video of the client.

In Figure 5.7 six pictures of the motor compartment from the first car with added control elements can be seen from different viewing angles. It can be observed, that the added control elements stick to their positions regardless of the point of view. This will remain true unless the used tracking algorithm breaks. The first row of pictures in this figure shows three rotating control elements with textures applied to them. In the second row of pictures the camera changes its point of view to only show two of the added control elements. Figure 5.8 shows another car which has different refill lids and textures. It can be seen in the first row of pictures, that the colour tone of the applied textures varies with the different appearance of the refill lids. The two control elements on the left have a clearly brighter appearance than the one on the right, because the two refill lids on the left have a bright silver colour and the one on the right is dark black. This way of respecting the original colour of the covered object allows to more clearly distinguish the different elements from each other. In the second row of pictures in this figure only one rotating control element is displayed from different camera angles. It can be observed again, that regardless of the point of view, the applied object stays statically in the place where it has been applied to.

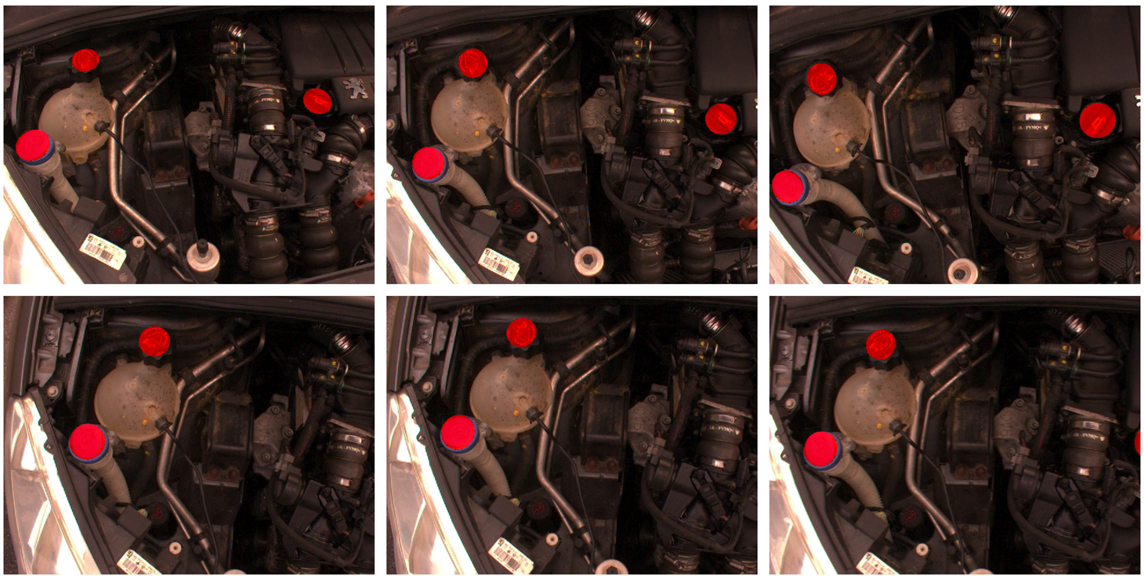


Figure 5.7: This figure shows six separate pictures of the motor compartment with added control elements from different views.

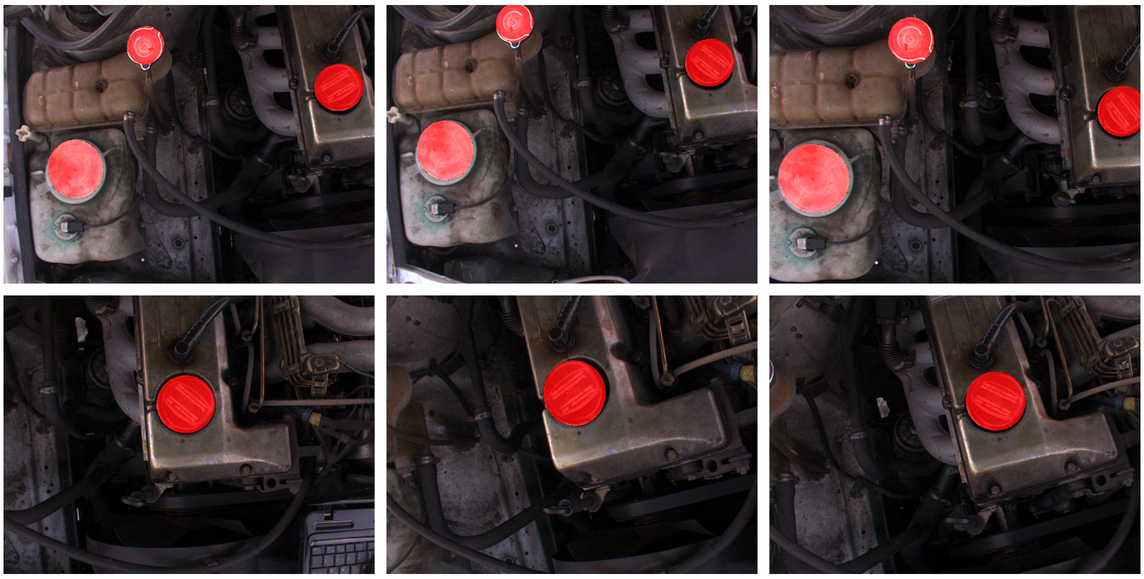


Figure 5.8: The first row shows three rotating control elements from different camera angles. The texture colour of the elements varies, because the original colour of the underlying objects is respected. It can be seen in the second row, that the control element stays in the applied position, regardless of the camera movement.

Figure 5.9 from [28] shows the results of Gauglitz et al. when filming a motor compartment with their solution as well. The first picture shows the viewpoint from where the region was originally marked, the other four show different depth interpretations of the drawing from a different point of view. They also decided to draw segments, which are occluded by the object, semi-transparent. While this approach is definitely more helpful for finding particular regions of the motor compartment, it is more difficult to depict particular movements and interactions.

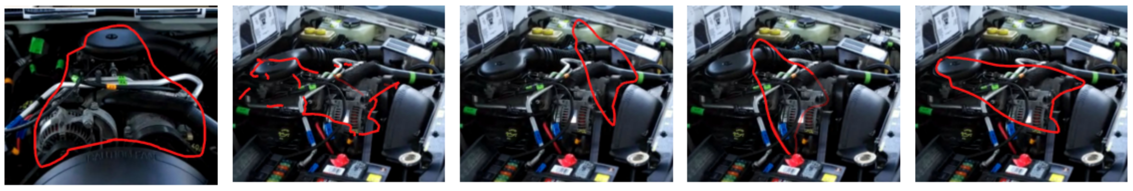


Figure 5.9: This figure from [28] shows the results from Gauglitz et al. with different depth interpretations.

5.2 Cooker control panel

The object of the next recording was also inspired by the works of Gauglitz et al. and depicts the control panel of a cooker. Figure 5.10 from [28] shows an example cooker with its control elements, which was used by them as test object. Like in the previous example from their paper shown in the previous section 5.1, the first of the five images is shown from the point of view from where the drawings have been made, the others show the same drawing from a different view with different depth interpretations.

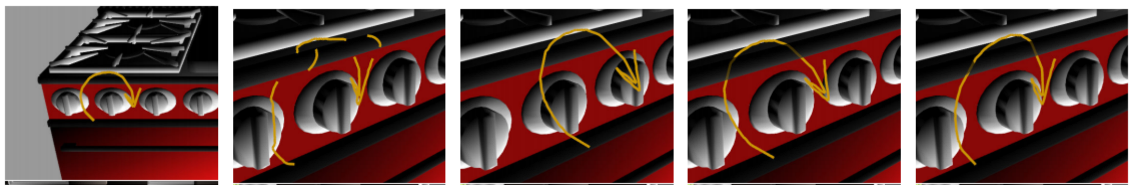


Figure 5.10: This figure from [28] shows the results from Gauglitz et al. of a cooker control panel with different depth interpretations.

While Gauglitz et al. decided to use an example 3D model of a cooker with a control panel, it was decided to directly film a real cooker for this work to compare the two results. A challenge here was, that the control panels of a cooker are usually very smooth and reflective. This poses a challenge to the used SLAM system, which uses image features for tracking and creating the map of the filmed scene. Figure 5.11 depicts the results of the work presented here. The four images shown in this figure view the rotating control

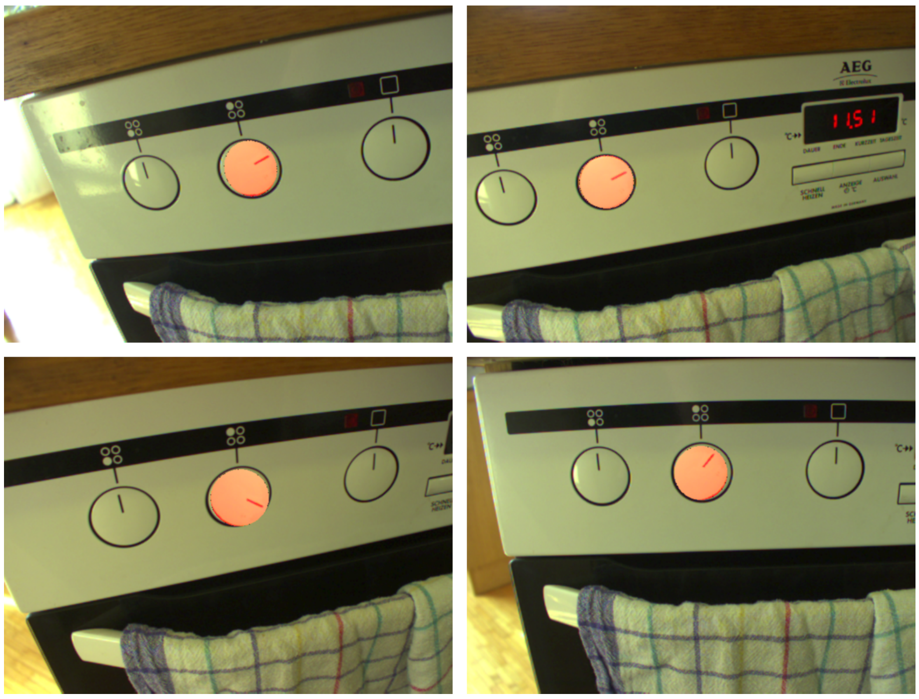


Figure 5.11: This figure shows the results of this work with the control elements of a cooker as model from different camera angles.

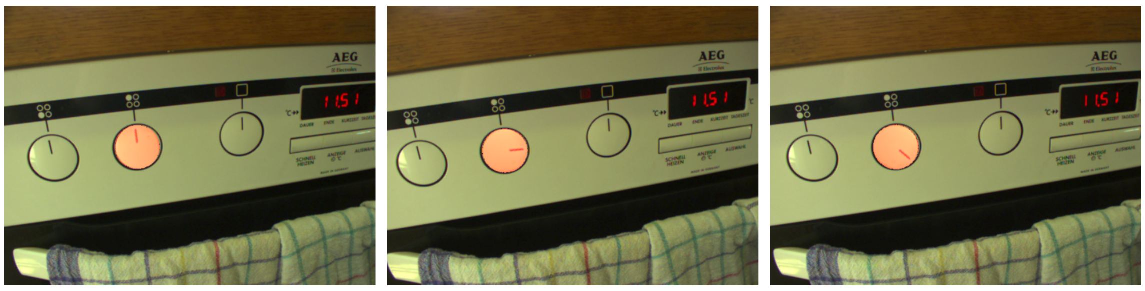


Figure 5.12: This figure depicts the progress of the applied, revolving control element on one of the temperature regulators.

element applied to one of the temperature regulators of the cooker from different camera angles. It can be seen, that the added control element maintains its location relatively static on the real knob, but some slight movements can be detected in the first and last picture. Figure 5.12 shows the same scene, and depicts in its three pictures the revolving control element in three different positions. For this motion a rotation around the centre of the control element was chosen, with the extra property of only rotating from the start point to the end point instead of rotating back and forth. When the element is about to reach its end position, the motion is slowed down a bit, and freezes for a short duration before starting the movement again from its initial position.

The results presented in the following sections are no longer compared to the works of Gauglitz et al. and are presented to show different fields where this project can be used to give instructions.

5.3 Magic Cube

The following example features a magic cube, which was filmed close to a window, under good lighting conditions. Even though the surface of the cube is slightly reflective which can result in a problematic tracking, meaningful results could be achieved. Besides the reflective surface, the sparse number of features present on the surface of the cube also posed to be a problem. Thus, the cube was not filmed in isolation, but feature-rich objects were added to the scene before filming.

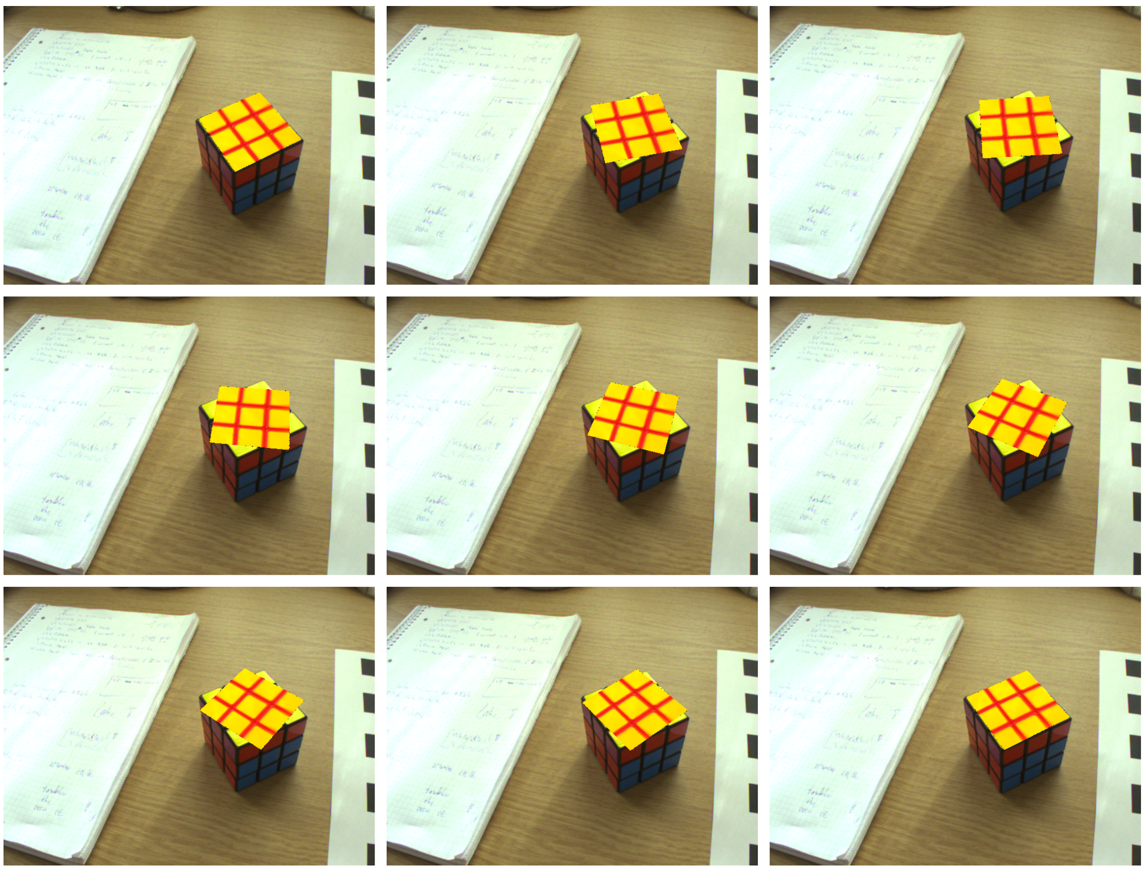


Figure 5.13: This figure shows in its nine chronologically sorted pictures the movement of the applied control element, which describes the motion needed to solve the magic cube.

This little trick allowed the tracking to find enough features besides the ones from the actual object—the cube—which resulted in a stable tracking. Thanks to this, the applied control element remained stable on its assigned position, even under jerky movements and vast transformations of the camera position. Even after intentionally breaking the tracking, by moving fast and far away of the actual scene, the tracking was able to relocate its position after moving the camera back to the scene with the cube. The results of this example can be seen in Figure 5.13 which shows nine separate pictures which are sorted chronologically. On the server side of the project, a rectangular control element was added and a clockwise rotation about its centre was applied. This rotation describes the needed action to solve the riddle of the unfinished magic cube.

5.4 Piano

In this section, a piano was used as the object of interest and a different way to represent the added control elements was used as well. Unlike the keyboard presented in the following section, the piano does not have a meaningful texture and the different keys can not be distinguished as clearly. Due to these circumstances, the gained point cloud of this model was considerably sparser and the created 3D model not as significant. Nevertheless, the created 3D model and the tracking were feasible enough to use this work for filming the piano. Instead of trying to imitate the gentle rotation of the piano keys themselves to depict which key has to be pressed, a different approach has been chosen. Instead of letting the control element perform a smooth motion from the start- to the end-point or simply using the elements to statically point towards a certain key, the control elements move to different locations in a pre-defined sequence. When adding this new type of motion to a control element, every single step of the element has to be defined. After a sequence for the control elements has been recorded, they can be sent to the client program where they will be displayed as an augmented reality overlay as usual. Figure 5.14 displays a simple melody of the well known children's song 'All my little ducklings' guided by the added control element. While this approach works well for an indefinite number of simultaneously pressed keys and amount of positions, a problem occurs when the same key has to be pressed twice in a row. In this case, the added control element would only stay on its current position without indicating that a second press on the same location is needed. Thus, a fade-in animation was added to each position of the control element. This means, that every time the control element performs a step, it will be enlarged slightly before it returns to its original size. This pulsating animation is able to clearly indicate when another press has to be performed.

Figure 5.15 depicts the same positions of the control element shown in the previous figure but this time always when the control element is initially placed to a position with its size increased. For that reason, the elements in this figure are slightly larger than in the previous one and are even slightly bigger than the original keys, but are still able to clearly indicate the wanted key. Right after their initial, scaled up appearance, they start to shrink back to their original size as shown in the previous figure. With this technique

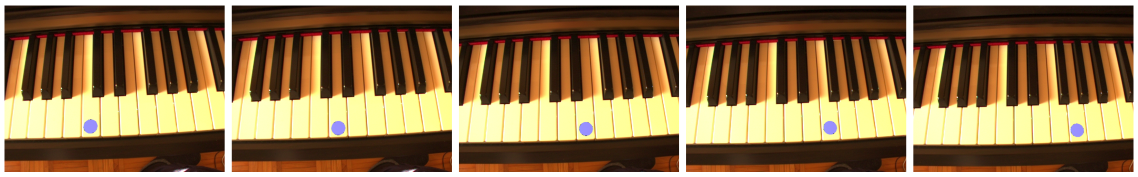


Figure 5.14: This picture shows how the control elements can be used to guide through the simple melody of the well known children's song "All my little ducklings".

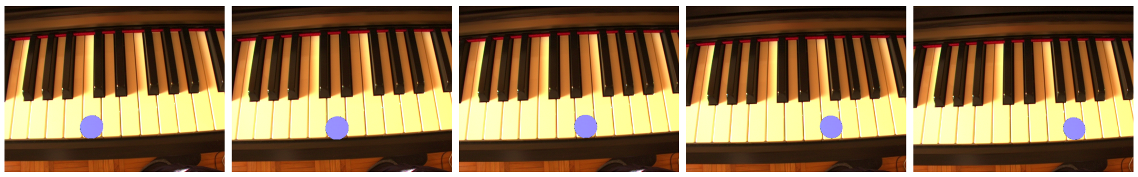


Figure 5.15: The initial size of the control elements from the previous figure during the fading-in animation.

to highlight a control element, it is possible to indicate whether the same button has to be held down or has to be pressed multiple times in a row. Figure 5.16 displays the same scene from a different camera angle in the top row and the bottom row shows how someone can play a song on the piano by following the instructions given by the control elements. It can be seen, that the tracking is robust enough to deal with the inserted and moving hand as well as the keys which are slightly moved when being pressed down by the player.

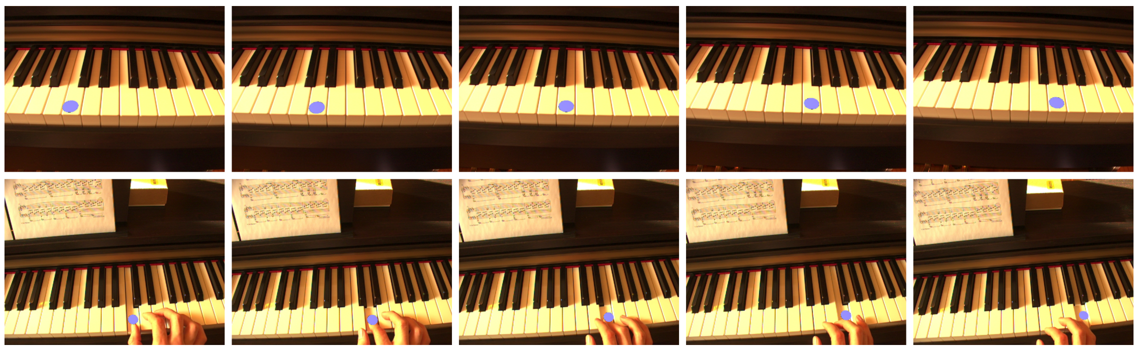


Figure 5.16: The top row of pictures shows how someone sitting in front of the piano would see the control elements. The second row shows someone playing the melody on the piano with the help of the control elements.

5.5 Keyboard

The results shown in this section feature a keyboard. Even though this example is quite similar to the previously shown piano examples, the results of the two of them are different. The keyboard was filmed under similar lighting conditions but due to the different structure and appearance of the keyboard in comparison to the piano, the number and diversity of the features detected by the SLAM system was greater for the keyboard.

The keyboard shown in figure 5.17 has a rich texture due to the clearly distinguishable keys and the letters printed onto them, thus the recorded point cloud contains enough information to create a proper 3D model and the achieved results are meaningful. The highlighted control elements in this figure have different transformations assigned to them. The rectangular control element on the left performs a translation along the x-axis, the circular one in the middle performs a rotation around its centre, which can be a bit difficult to determine on the static pictures but can clearly be seen on the moving video of the client. Alternatively, the octagon form could be used to highlight the rotation even more. Finally, the circular control element on the right performs a translation as well, but this time along the z-axis to indicate that this button has to be pushed. Due to this translation along the z-axis it appears that the added control element is not properly placed on the three separate images of this figure. This perception is created by the still standing pictures and is not present, when viewed on the live video shown on the client program. Alternatively, a control element can also be created without any motion assigned to it, to simply indicate where physical interaction is possible. The following section 5.6 will present this kind of control elements.

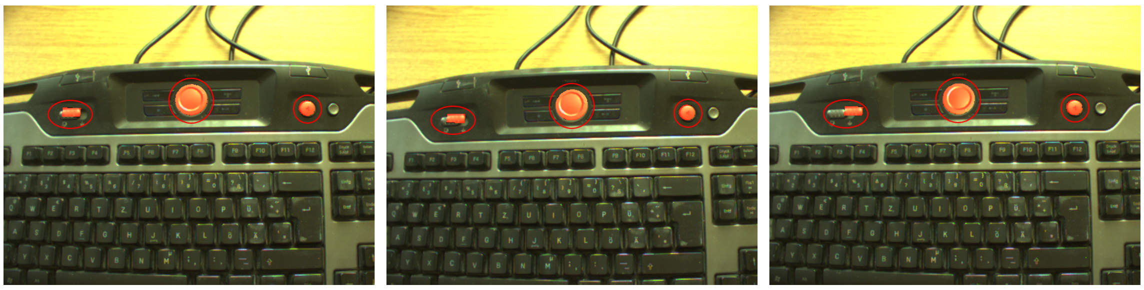


Figure 5.17: This figure shows a filmed keyboard with applied control elements. The marked control elements perform different transformations. The left one has a horizontal translation, the one in the middle a rotation around the centre and the one on the right a translation along the z-axis.

5.6 Fuse box

For this example, a fuse box was filmed in mediocre lighting conditions. While the surface of the object is similar to the one of the cooker control panel shown in a previous section, the SLAM system was able to find enough features so that a proper 3D model could be created. Here, the added control elements were used to indicate different fuses, which have to be operated. In contrast to the previous examples, no motion was added to the individual elements. While the needed motion to operate the single fuses could be done by a slight, horizontal rotation around the centre to simulate the needed tilt, the used 2D control elements are not perfectly suited to properly depict this motion. Respecting the nature of this movement, the control elements were added accordingly and sent to the client. In the first picture of figure 5.18, the created 3D model can be seen without any added control elements. In the following five pictures, the view of the client program with the control elements added as augmented reality overlays on the video stream can be seen from different camera positions. Even though the elements maintain a stable position on their respective fuses, the edges of the real fuses can be seen slightly when the scene is viewed from an acute angle, depending on the angle. This happens, because the created 3D model of the fuse box also contains the slight bumps for each fuse. Thus, the control elements depicting the fuses are pinned on top of the respective bumps, which results in the described behaviour of being able to see the real fuses under certain angles. Another reason for this behaviour is of course the tracking which is not always totally accurate in this scenario, which adds a slight drift to the control elements as well. Nevertheless, it can be seen, that the added control elements are still almost in the correct position with only a slight variation to their original placement.

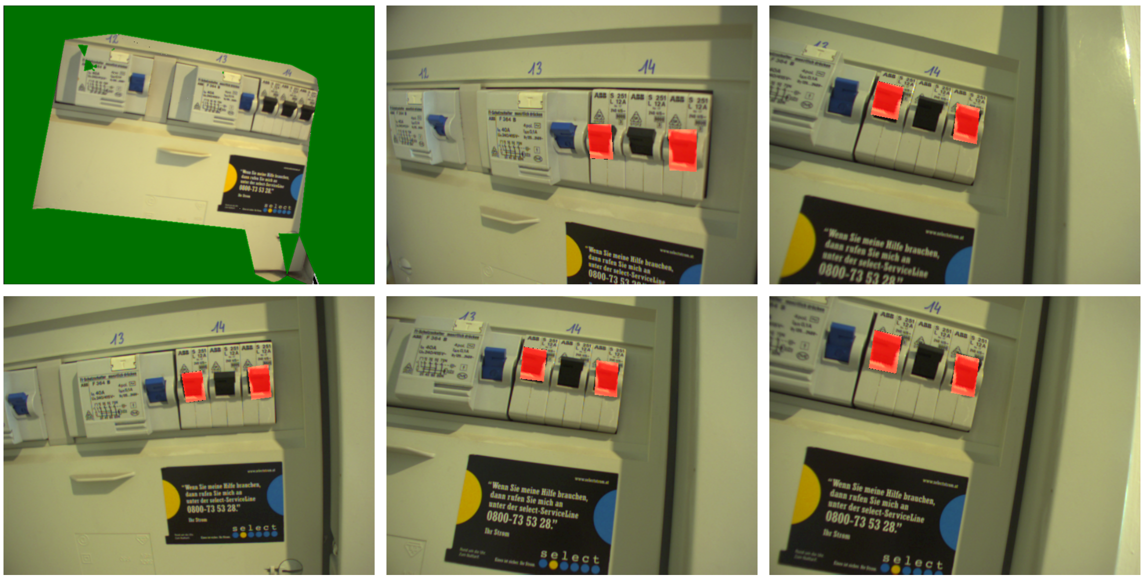


Figure 5.18: The first picture of this figure shows the created 3D model of a fuse box without applied control elements. The remaining pictures show the object and the added control elements from different camera angles.

5.7 Problematic Scenarios

The previous sections showed several scenarios where this project performed fine, resulting in good examples. In this section, problematic scenarios and circumstances will be discussed, as well as their consequences. The main reason for a bad performance are mediocre or bad lighting conditions. These usually result in an unstable tracking which has several consequences. With unstable tracking, the recorded point cloud is not well suited to use as a foundation for reconstruction, which leads to a bad created 3D model and the control elements applied on the model will not be placed correctly when being viewed as augmented reality overlay. Furthermore, due to the unstable tracking, the control elements tend to float around their assigned location instead of staying statically on the correct place. Thus, good lighting conditions are crucial to eliminate one of the biggest problems of the tracking algorithm. When talking about *good lighting*, not only the correct luminance, but also a steady, non-changing light is favored. Another important factor for good tracking are objects which possess enough potential features for the tracking algorithm. When having permanently repeating features or too sparse features, the tracking tends to get lost. While good tracking conditions are wanted to ensure good-looking results, other factors have to be respected when using features like the control elements. Not all forms of control elements are suited for every case, thus they have to be chosen thoughtfully. The examples shown in the following will introduce to some problematic cases, where some of the before mentioned problems occurred.

Figure 5.19 of the cooker control panel featured in section 5.2 shows the consequences of bad tracking- and lighting-conditions. The pictures have been recorded in one process and are timely sorted from top-left to bottom-right. In this scenario, octagon-shaped control elements have been used instead of the previously used circle. It can be seen in the first picture, that the applied control element is not correctly placed due to the very bright and reflective surface. The next two pictures show that even several minutes of collecting new tracking features are not entirely enough to ensure proper tracking. In the fourth picture, the lighting of the filmed scene was changed, resulting in serious tracking issues in the fifth picture. After the lighting conditions were restored in the last picture, the tracking was able to recover to its previous state. It can be seen on the basis of this example, how bad lighting conditions affect the results of this project.

The next example shows the importance of a well-suited control element choice. In figure 5.20 the piano from the example in 5.4 is filmed, but this time rectangular control elements were selected. Furthermore, instead of selecting the fade-in animation presented previously, a slight rotation was applied to the elements. While this rotation could possibly imitate the actual movement of the piano keys, its motion is too small to be recognized. While the elements shown in the first two pictures used to at least indicate which keys have to be pressed, the tracking failed in the third picture, resulting in an incorrect placement. This example showed that not only the shape, but also the applied motion of the control element is important to ensure proper instructions.

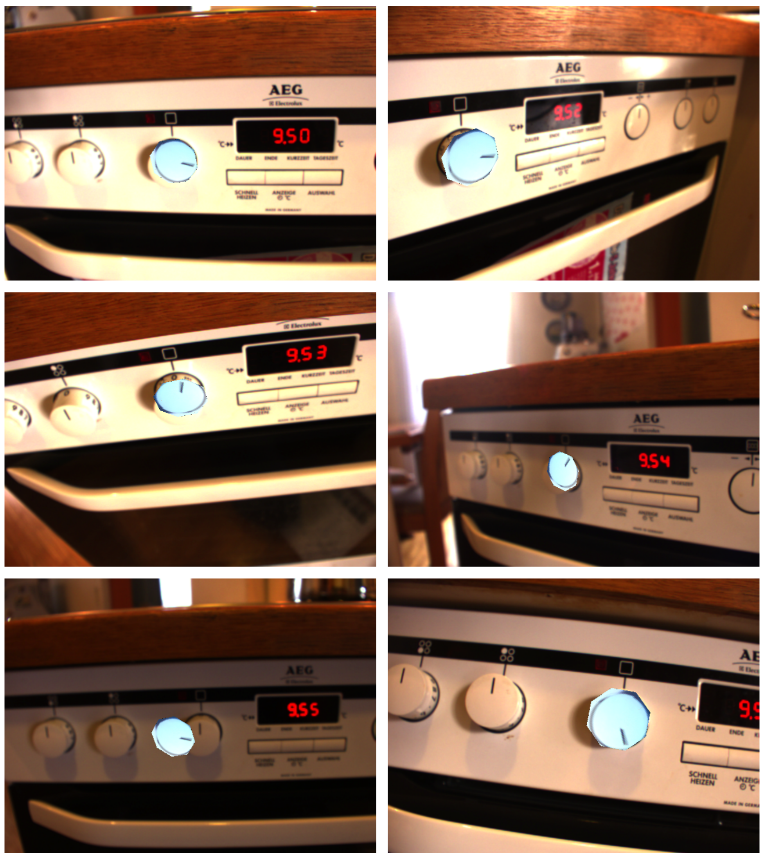


Figure 5.19: This series of pictures shows the poor results of bad lighting conditions. While the tracking had already problems due to the very bright and reflecting surface in the first three pictures, it totally failed when the lighting conditions were changed.

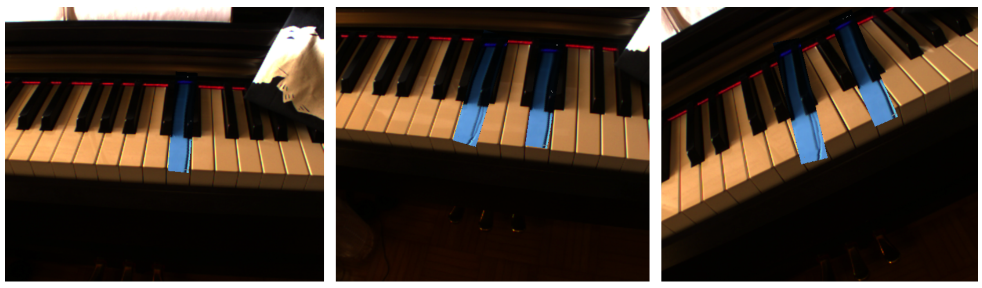


Figure 5.20: This series of pictures shows the results of a different control element shape and animation type. Instead of the circular element which represents the finger position on the piano, a rectangular element was chosen, and instead of the fade-in animation, a rotation representing the actual movement of the piano keys was selected. The last picture shows the result after the tracking became inaccurate due to bad lighting conditions.

6 Conclusion and Future Work

In this thesis the problems of tracking and modelling a scene to be able to give remote instructions, regarding the filmed scene, were addressed along with the problem of instruction representation. These problems were approached by using only a monocular camera as sensor, instead of more complex sensors, to maintain a low cost for the needed equipment. One of the main contributions of this work is the novel and intuitive way for the instruction representation, which was addressed in chapter 4.4.

The assumed situation consists of a local person being confronted with a problem which requires physical interaction on a present machine, and a distant expert who has the knowledge to solve the given problem. This scenario is described in more detail in chapter 3.1. On the assumption of this problem a processing pipeline was developed, as described in chapter 3.2, to be able to address the individual approaching sub-problems. Furthermore, a framework was created to be able to swap and try different solutions for the given SLAM problem and the reconstruction. After the first steps of the pipeline, which deal with gathering data from the given scene and creating a model of it, have been dealt with, the main focus was on creating and presenting instructions.

To describe tasks for the given environment, two dimensional primitives like circles or rectangles were chosen, along with the abilities to apply textures and animate them. As discussed in chapter 3.1.3, most tasks can be described by simple motions consisting of rotations and translations, which can be assigned to the provided control elements.

While it would be possible to add the instructions directly onto the video without a 3D model being involved, the created model allows the user in charge of adding the instructions to freely navigate in the scene, without being dependent on the current camera position. The presence of the model gives both involved persons more freedom and flexibility and does not demand previous knowledge. For the representation of the created instructions, two different options were provided by the created framework. The first representation applies the added instructions as augmented reality overlays to the live video from the user of the program. This representation allows the user to get a clear understanding of the given instructions which are applied on the real objects in the scene.

The second representation utilizes the created model instead of the video to display the added instructions. The advantage of this approach is that the instructions along with the created model can be stored for later usage without being dependent on a remote person to apply instructions to the recorded scene when needed.

Finally, it was shown along with several examples, how the created framework can be used to apply instructions to filmed scenes and that different tasks require a different instruction representation. While the shape of the added geometry is a big factor for a proper representation of the described instruction, another important parameter is the animation of the added control element. Another contribution of this work are the two different types of motion, which deal with continuous movements and step-wise motion. These two types of motion allow to describe various interactions like opening a door and opening a bottle cap with the continuous motion, or can describe the consecutive order of pressing keys on a piano. With the various available types of instruction representation, it is possible to approach different situations and to find a suitable way for giving instructions to the local person.

6.1 Future Work

While the used monocular SLAM system completes its task in a satisfying way, a tracking algorithm as well as the feature selection to obtain points describing the filmed object can be designed to fulfill this task in a better and more purposeful way. The given interfaces of this project allow to change these sub-modules in an elegant way to be able to be more flexible in the choice of algorithms. Thus, an interesting and impactful addition to this work in the near future would be to develop a SLAM system which specifically satisfies the constraints for this project. One advantage would be, that this results into a more stable and reliable tracking, which is crucial to guarantee, that the added augmented reality features stay in the correct position. Another advantage of a customized SLAM system would be, that the obtained data, which consists of feature points describing the scene and camera positions, can be chosen more specifically to permit more meaningful reconstructions of the scene. The augmented reality representation of the added control elements as well as the offline presentation of the elements on the model would benefit from a specifically designed SLAM system.

The currently used reconstruction algorithm is able to create proper 3D models of the received data, and it would definitely benefit from the before mentioned tailored SLAM system. However, a specially designed reconstruction algorithm which should be developed alongside the new SLAM system, could produce even better results. When implementing a specialized SLAM and reconstruction system, the used IBR system could be respected to not need a new image based renderer. However, this module could easily be replaced in the current project as well. It can be seen, that all three core components—SLAM, reconstruction and IBR—could be replaced in the near future to improve the usability and the results of this project. The currently implemented structure allows to change these modules relatively easy to compare different solutions.

Another task, which could improve the visualisation of the added control elements, is to add further shapes and different animation techniques. Especially the group of 3D control elements would be a good addition to the existing two dimensional forms. Objects like levers could be represented in a better way by a three-dimensional object instead of using only the 2D primitives. A problem arising with 3D objects is how to texture them. One solution to this problem would be to simply use pre-set textures and colours for the 3D elements, a more elegant way would be to use the actual textures of the object they represent, like the 2D control elements do. This requires a model of the object with detailed textures, which would be the result of the beforementioned, tailored SLAM system. Other improvements to the control elements which could be done in the future are to implement a system which can automatically detect potential control elements in the filmed scene and suggests them to the user. While this would require research on object recognition to find elements, it would greatly improve the user experience. To be sure which improvements to the provided control elements would truly enhance the usability, some user-studies should be made beforehand. Besides adding new control element shapes, different visualisation techniques like the one presented in the works of Gauglitz [28, 29] could be implemented to replace the currently used control elements. This is another part which should be addressed in future user studies.

The way the created instructions are presented to the user can be improved by using more novel approaches. Instead of using the screen of a smart device, like a tablet or a phone, a head-mounted display could be used to let the user move more freely in the given environment. Besides the visualisation of the added information, the currently used monocular camera could be replaced by utilising more advanced systems like stereo setups, laser-range scanners, or others mentioned in section 2.2.2. While this would soften the constraint which demands a commonly available hardware, it would definitely improve the results of the SLAM and reconstruction systems, which leads to better overall results. By softening other constraints listed in 1.2, even more improvements to the different parts of this project could be achieved.

Even though, the current implementation of this projects provides appealing results, as can be seen in section 5, the steadily improving algorithms and technologies open more and more ways to improve the presented work in this thesis in different ways. The presented flexible framework is a foundation, which has the ability to grow with the permanently advancing possibilities in these fields of research.

A Lists

List of Sensors

- Accelerometer
- Gyroscope
- Magnetometer
- Barometer
- Proximity sensor
- Light sensor
- Microphones
- GPS
- WiFi
- Bluetooth
- NFC
- Cellular (tri-lateration)
- Front camera
- Back camera
- Touch screen
- Temperature
- Humidity

Functions of the wrapper class for SLAM systems and their usage

Function name	Usage
void track()	Runs the tracking of the SLAM system
void draw()	Draws the created points
void input(int input)	Sends input flags to the SLAM system
void mouse(int button, float x, float y)	Sends mouse movement and button presses
void saveToMap()	Saves the created data
std::vector<float>getCamPose()	Gives the current approximated camera position
std::vector<float>getCameraParams()	Gives the intrinsic camera parameters

List of used program libraries for compiling and running the PTAM project [39]

- gvars-3.0
- TooN
- Libcvd
- Glew-1.10.0
- pthreads (x86)
- blas (x86)
- lapack (x86)
- opencv (for the modified video source support)
- opengl

Bibliography

[1] Adelson, E. H. and Bergen, J. R. (1991). *The plenoptic function and the elements of early vision.* Vision and Modeling Group, Media Laboratory, Massachusetts Institute of Technology. (page 31)

[2] Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266. ACM. (page 22)

[3] Aulinas, J., Petillot, Y. R., Salvi, J., and Lladó, X. (2008). The slam problem: a survey. In *CCIA*, pages 363–371. Citeseer. (page 21)

[4] Bouguet, J.-Y. (2015). Camera calibration toolbox for matlab. [Online; accessed 13-January-2016]. (page 62)

[5] Bresler, Y., Fessler, J., Macovski, A., et al. (1989). A bayesian approach to reconstruction from incomplete projections of a multiple object 3d domain. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(8):840–858. (page 24)

[6] Brown, M. and Lowe, D. G. (2007). Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73. (page 35)

[7] Chen, S. E. (1995). Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 29–38. ACM. (page 35)

[8] Chen, S. E. and Williams, L. (1993). View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288. ACM. (page 30, 31)

[9] Chung, S.-Y. and Huang, H.-P. (2010). Slammot-sp: simultaneous slammot and scene prediction. *Advanced Robotics*, 24(7):979–1002. (page 20)

[10] Cohen, M. (1999). Course on image-based, modeling, rendering, and lighting. (page 26)

[11] Cummins, M. and Newman, P. (2008). Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665. (page 18)

[12] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067. (page 50)

[13] Debevec, P., Yu, Y., and Borshukov, G. (1998). *Efficient view-dependent image-based rendering with projective texture-mapping.* Springer. (page 28)

[14] Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM. (page 28)

[15] Décoret, X., Durand, F., Sillion, F. X., and Dorsey, J. (2003). Billboard clouds for extreme model simplification. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 689–696. ACM. (page 28, 29)

[16] Doucet, A. (1998). On sequential simulation-based methods for bayesian filtering. (page 17)

[17] Dutre, P., Bekaert, P., and Bala, K. (2006). *Advanced global illumination.* CRC Press. (page 25)

[18] Eade, E. and Drummond, T. (2006a). Edge landmarks in monocular slam. In *BMVC*, pages 7–16. (page 50)

[19] Eade, E. and Drummond, T. (2006b). Scalable monocular slam. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 469–476. IEEE. (page 50)

[20] Eisemann, M. and Magnor, M. A. (2007). Filtered blending: A new, minimal reconstruction filter for ghosting-free projective texturing with multiple images. In *VMV*, pages 119–126. (page 71, 72, 73, 74)

[21] Eisenman, J. A. (1988). *Graphical editing of composite bezier curves.* (page 24)

[22] Engel, J. (2014). Lsd-slam: Large-scale direct monocular slam. [Online; accessed 13-January-2016]. (page 60, 61)

[23] Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer. (page 56, 57, 58, 60)

[24] Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1449–1456. IEEE. (page 56)

[25] Engels, C., Stewénius, H., and Nistér, D. (2006). Bundle adjustment rules. *Photogrammetric computer vision*, 2:124–131. (page 51)

[26] Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE. (page 56, 58)

[27] Fuchs, H., Kedem, Z. M., and Uselton, S. P. (1977). Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702. (page 24)

[28] Gauglitz, S., Nuernberger, B., Turk, M., and Höllerer, T. (2014a). In touch with the remote world: remote collaboration with augmented reality drawings and virtual navigation. In *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology*, pages 197–205. ACM. (page 73, 81, 82, 89, 101)

[29] Gauglitz, S., Nuernberger, B., Turk, M., and Höllerer, T. (2014b). World-stabilized annotations and virtual scene navigation for remote collaboration. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 449–459. ACM. (page 81, 82, 101)

[30] Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM. (page 33)

[31] Heckbert, P. S. (1986). Survey of texture mapping. *Computer Graphics and Applications, IEEE*, 6(11):56–67. (page 27)

[32] Hoppe, C., Klopschitz, M., Donoser, M., and Bischof, H. (2013). Incremental surface extraction from sparse structure-from-motion point clouds. In *BMVC*, pages 94–1. (page 63, 66)

[33] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). *Surface reconstruction from unorganized points*, volume 26. ACM. (page 1, 2, 22, 24)

[34] Horn, B. K. (1989). *Obtaining shape from shading information.* MIT press. (page 23)

[35] Jeschke, S., Wimmer, M., and Schumann, H. (2002). Layered environment-map impostors for arbitrary scenes. In *Graphics Interface*, pages 1–8. (page 29)

[36] Kang, S. B., Li, Y., Tong, X., and Shum, H.-Y. (2006). Image-based rendering. *Foundations and Trends® in Computer Graphics and Vision*, 2(3):173–258. (page 26)

[37] Kitagawa, G. (1996). Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25. (page 17)

[38] Klein, G. (2008). Parallel tracking and mapping for small ar workspaces - source code. [Online; accessed 13-January-2016]. (page 59)

[39] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE. (page 50, 51, 52, 59, 104)

[40] Konolige, K. (2004). Large-scale map-making. In *Proceedings of the National Conference on Artificial Intelligence*, pages 457–463. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. (page 16)

[41] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE. (page 58)

[42] Leonard, J. J. and Durrant-Whyte, H. F. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee. (page 12)

[43] Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM. (page 33, 34)

[44] Lin, Z. and Shum, H.-Y. (2004). A geometric analysis of light field rendering. *International Journal of Computer Vision*, 58(2):121–138. (page 72)

[45] Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349. (page 16)

[46] Mann, S. and Picard, R. W. (1994). Virtual bellows: Constructing high quality stills from video. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 1, pages 363–367. IEEE. (page 35)

[47] Mark, W. R., McMillan, L., and Bishop, G. (1997). Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff. ACM. (page 30)

[48] McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46. ACM. (page 32)

[49] Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, 44(247):335–341. (page 17)

[50] Meyers, D., Skinner, S., and Sloan, K. (1992). Surfaces from contours. *ACM Transactions On Graphics (TOG)*, 11(3):228–258. (page 24)

[51] Milford, M. J., Wyeth, G. F., and Rasser, D. (2004). Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE. (page 21)

[52] Montemerlo, M. and Thrun, S. (2006). Large-scale robotic 3-d mapping of urban structures. In *Experimental Robotics IX*, pages 141–150. Springer. (page 16)

[53] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598. (page 17)

[54] Mori, Y., Fukushima, N., Yendo, T., Fujii, T., and Tanimoto, M. (2009). View generation with 3d warping using depth information for ftv. *Signal Processing: Image Communication*, 24(1):65–72. (page 30)

[55] Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., and Sayd, P. (2006). Real time localization and 3d reconstruction. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 363–370. IEEE. (page 51)

[56] Mullane, J., Vo, B.-N., Adams, M. D., and Vo, B.-T. (2011). A random-finite-set approach to bayesian slam. *Robotics, IEEE Transactions on*, 27(2):268–282. (page 19)

[57] Nayar, S. K. (1997). Catadioptric omnidirectional camera. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 482–488. IEEE. (page 35)

[58] Newman, P. and Ho, K. (2005). Slam-loop closing with visually salient features. In *proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 635–642. IEEE. (page 20)

[59] Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652. IEEE. (page 51)

[60] Oliveira, M. M., Bishop, G., and McAllister, D. (2000). Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368. ACM Press/Addison-Wesley Publishing Co. (page 30)

[61] Riisgaard, S. and Blas, M. R. (2003). Slam for dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*, 22(1-127):126. (page 11)

[62] Rubin, D. B. et al. (1988). Using the sir algorithm to simulate posterior distributions. *Bayesian statistics*, 3(1):395–402. (page 17)

[63] Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China. (page 66, 69, 70)

[64] Sachs, E., Roberts, A., and Stoops, D. (1991). 3-draw: A tool for designing 3d shapes. (page 24)

[65] Schneider, P. and DeRose, A. D. (1998). An interactive curve design system based on the automatic fitting of hand-sketched curves. *Department of Computer Science, University of Washington.* (page 24)

[66] Seitz, S. M. and Dyer, C. R. (1996). Toward image-based scene representation using view morphing. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 1, pages 84–89. IEEE. (page 30, 31, 32)

[67] Shade, J., Gortler, S., He, L.-w., and Szeliski, R. (1998). Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM. (page 30)

[68] Shi, J. and Tomasi, C. (1994). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE. (page 54)

[69] Shum, H.-Y., Chan, S.-C., and Kang, S. B. (2008). *Image-based rendering*. Springer Science & Business Media. (page 26)

[70] Shum, H.-Y. and He, L.-W. (1999). Rendering with concentric mosaics. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 299–306. ACM Press/Addison-Wesley Publishing Co. (page 33)

[71] Siciliano, B. and Khatib, O. (2008). *Springer handbook of robotics*. Springer Science & Business Media. (page 14, 16, 17)

[72] Smith, C. (2006). *On vertex-vertex systems and their use in geometric and biological modelling*. University of Calgary. (page 22)

[73] Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer. (page 12, 15)

[74] Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68. (page 12, 15, 16)

[75] Srl, C. I. (2015). Coordinate measuring machine history, global supplier of coordinate measuring machines. [Online; accessed 13-January-2016]. (page 23)

[76] Stewenius, H., Engels, C., and Nistér, D. (2006). Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294. (page 52)

[77] Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and environment maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 251–258. ACM Press/Addison-Wesley Publishing Co. (page 35)

[78] Thrun, S. (2002a). Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc. (page 17)

[79] Thrun, S. (2002b). Probabilistic robotics. *Communications of the ACM*, 45(3):52–57. (page 15)

[80] Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154. (page 23)

[81] Trajković, M. and Hedley, M. (1998). Fast corner detection. *Image and vision computing*, 16(2):75–87. (page 52)

[82] Wang, C.-C., Thorpe, C., Thrun, S., Hebert, M., and Durrant-Whyte, H. (2007). Simultaneous localization, mapping and moving object tracking. *The International Journal of Robotics Research*, 26(9):889–916. (page 20)

[83] Whelan, T., Kaess, M., Leonard, J. J., and McDonald, J. (2013). Deformation-based loop closure for large scale dense rgb-d slam. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 548–555. IEEE. (page 20)

[84] Wikipedia (2015). Polygon mesh — wikipedia, the free encyclopedia. [Online; accessed 13-January-2016]. (page 22)

[85] Woodham, R. J. (1980). Photometric method for determining surface orientation from multiple images. *Optical engineering*, 19(1):191139–191139. (page 23)

[86] Xiong, Y. and Turkowski, K. (1997). Creating image-based vr using a self-calibrating fisheye lens. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 237–243. IEEE. (page 35)