FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

TrACTOr & StARboard

Tracking and Haptic Interaction for Learning in AR

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Christoph Schindler, BSc
Matrikelnummer 00828417

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.Doz. Mag. Dr. Hannes Kaufmann
Mitwirkung: Dipl.-Ing. Mag. Emanuel Vonach, Bakk.

Wien, 29. August 2018

Christoph Schindler	Hannes Kaufmann

Erklärung zur Verfassung der Arbeit

Christoph Schindler, BSc
Lobstraße 25, 3121 Karlstetten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. August 2018

Christoph Schindler

Danksagung

An dieser Stelle möchte ich mich bei meinen Eltern Gerald und Ilse, meinem Bruder Stefan, meiner Freundin Claudia und meinen Freunden bedanken, die mir immer sehr geduldig mit Rat und moralischer Unterstützung beigestanden sind.

Des Weiteren möchte ich mich bei meinem Betreuer, Mag. Dr. Hannes Kaufmann, für die Betreuung meiner Diplomarbeit bedanken. Mein weiterer Dank gilt Dipl.-Ing. Mag. Bakk. Emanuel Vonach, der mich während der gesamten Arbeit sehr gut unterstützt und geduldig beraten hat.

Außerdem möchte ich mich bei allen freiwilligen Test-Personen bedanken, die sich bereiterklärt haben, an der Benutzerstudie teilzunehmen.

Kurzfassung

In unserer heutigen computergesteuerten Welt gewinnt das Thema der Mensch-Maschine-Interaktion bei der Entwicklung von effizienten und flexiblen Interfaces immer mehr an Bedeutung. Ein Feld mit immer rasanter wachsender Bedeutung ist Augmented Reality, wobei die physische Welt mit virtuellen Inhalten erweitert wird. Die derzeitigen Interaktionstechniken und -prinzipien sind nicht ausreichend, um das gewaltige Potential dieser Technologien vollständig ausschöpfen zu können.
Besonders haptisches Feedback während der Benutzerinteraktion ist ein wichtiges und komplexes Thema. Mit dem ACTO-System haben Vonach u.a. [VGK14a] eine angreifbare (tangible) Benutzersschnittstelle geschaffen, mit der haptische Interfaces prototypisiert werden können. In dieser Diplomarbeit benutzen wir dieses System um haptische Interaktion für Augmented Reality (AR) zur Verfügung zu stellen.
Dafür haben wir zuerst ein alternatives Tracking-System für die ACTO-Roboter entwickelt, es trägt den Namen *TrACTOr*. Mit diesem neuen Tracking-System wollen wir einige Nachteile der existierenden Tracking-Lösung kompensieren und den Anforderungen eines AR-Systems gerecht werden. Aufbauend auf diesem neuen Tracking-System haben wir eine auf Augmented-Reality basierende Lernapplikation entwickelt, die haptische Interaktion anbietet. Diese Lernapplikation trägt den Namen *StARboard*. Mit StARboard wollen wir die Vorteile von haptischer Interaktion für Lernsysteme untersuchen.
In dieser Arbeit präsentieren wir das Konzept des Tracking-Systems und der Lernapplikation. Des Weiteren wird die Entwicklung zusätzlicher Hardware für die ACTO-Roboter, sowie der Systemaufbau ausführlich erklärt. Im Anschluss wird die Implementierung von TrACTOr und StARboard diskutiert. Die technische Evaluierung des Systems hat gezeigt, dass die Performance und Genauigkeit des TrACTOr Tracking-Systems hoch genug ist, um Echtzeit-Tracking-Daten für eine AR-basierte Applikation zur Verfügung zu stellen. Die Benutzerstudie, die wir mit dem fertigen System durchgeführt haben, hat gezeigt, dass haptische Interaktion in Kombination mir AR im Einsatz für Lernsysteme ähnlich gut abschneidet, wie herkömmliche Desktop-Applikationen oder gestengesteuerte Interaktion für AR-Applikationen. Die Benutzer haben diese neuartige Interaktionstechnik als sehr intuitiv, schnell und einfach zu benutzen beschrieben.

Abstract

In our modern and computer-driven environment the topic of Human Computer Interaction becomes more important, in order to design efficient and flexible interfaces. A field of increasing importance is Augmented Reality, where current interaction techniques and principles are not yet satisfying or sufficient for the great potential of these technologies. Especially haptic feedback during user interaction is an important and complex topic. With the ACTO system, Vonach et al. [VGK14a] developed a Tangible User Interface (TUI) for prototyping haptic interfaces. In this thesis we use this system to provide haptic interaction for Augmented Reality (AR).
Therefore we first developed an alternative tracking system for the ACTO robots, called *TrACTOr*. With this new tracking solution we want to overcome some shortcomings of the existing tracking solution, as well as fit the needs of an AR application better. Based on the new tracking system we implemented an AR based learning application with haptic user interaction, called *StARboard*. With StARboard we wanted to investigate the benefits of haptic interaction for learning systems for the users.
We present the concepts of the tracking solution and the learning application. The development of additional hardware for the ACTO robots, as well as the system setup is explained. Also the implementation of TrACTOr and StARboard is discussed in detail. The technical evaluation of the system showed that the TrACTOr tracking system has a high enough performance and accuracy to provide real time tracking data for an AR based application. The user evaluation we performed with the implemented system showed that for learning systems, haptic interaction in combination with AR can match the performance of conventional desktop applications or gesture based AR applications. The users rated this new interaction technique as very intuitive, fast and easy to use.

Contents

CHAPTER 1

Introduction and Motivation

Today we live in two parallel existences. The first one is the real physical world where we can interact with every object and living being. Our ancestors invented a wide variety of tools that can be used for all sorts of interaction or manipulation of this world. Our second existence is a digital one: the cyberspace. But for this virtual world we have a very limited range of interaction possibilities. The most common ones are screens for receiving information, and mouse and keyboard for generating or manipulating data. The goal in Human Computer Interaction (HCI) research is to take advantage of the diverse set of skills from the real physical world, that everyone of us brings along, and develop new methods to use them for interacting with the cyberspace [IU97].

If the virtual data is given a physical form, these types of User Interfaces are called Tangible User Interfaces (TUIs). An example given by Ishii is an urban planning workbench [UI99], where the virtual buildings are represented simply by physical models that can be touched and moved around the workbench by the user. This provides a very tight coupling between the physical and the virtual world [Ish08].

A variety of different approaches for TUIs have been developed over the past years. Most of them are based on some sort of interactive space, like a tabletop. The users can interact with the digital world by positioning and moving physical objects in that space. In that manner a lot of use cases can be covered, like an interactive synthesizer [JGAK07], or a generic TUI prototyping tool [WSJB10].

The TUI we build upon was developed by Vonach et al.[VGK14a]. It is based on small rectangular robots, so called ACTOs (Modular Actuated Tangible User Interface Objects). See section 3.2 for a detailed explanation. These ACTOs serve as tokens for the Tangible UI and can be equipped with different interfaces, like buttons, spinners, displays, etc. This allows the creation of TUIs for different scenarios. Furthermore, each ACTO has a set of wheels, so it can move around freely on the interaction surface. The tracking is based on optical marker tracking. The interaction surface is a glass table and each ACTO

is equipped with an ID-marker on the bottom, so it can be tracked by an RGB-camera, that is positioned underneath the table.

Regarding how the virtual data is represented and visualized, Augmented Reality (AR) is a great way of embedding virtual information into the real world. The direct combination of real world and virtual world enhances the users perception, compared to an ordinary computer screen [Azu97].

1.1 Motivation and Problem Statement

One great field where TUIs can be used are learning systems. The direct interaction and the usage of multiple human senses can improve the cognitive learning process, which can help the users to learn faster and more efficiently [Sha09]. These kind of learning systems are also very well suited for collaborative learning experiences.

For these reasons we want to investigate these benefits by developing a collaborative learning system based on Augmented Reality and haptic user interaction.
We choose the ACTO system as interface for the user interaction, because it allows us to modify the interaction tokens. They can be equipped with different interfaces, like buttons or spinners, which allows us to create more sophisticated user input methods than with other systems. This helps us to improve the direct interaction with the system. To provide visual feedback we use AR glasses: The user wears a Microsoft HoloLens [Holb] and sees virtual objects instead of the ACTO robots.

As the ACTO tracking currently requires a glass table, it is not possible to use the ACTO system on other surfaces like a normal wooden table or the floor. Also, the lightning situation could cause problems with the current tracking, as the ACTOs need to be well lit from below the table without causing reflections on the glass surface.
Another issue is that for technical reasons it is required to synchronize the 3D space of the ACTO tracking system and the AR system. To solve this issue we use an optical tracking marker which acts as origin of both 3D spaces. This marker needs to be positioned in a way that it is visible for the HoloLens as well as the tracking system. As the ACTO tracking system records from below and the HoloLens top down, this is a problem. Even if we would use a double sided marker, which is positioned on the glass table this would cause the tracking not to work in that area.

For those reasons we present an alternative tracking solution in this thesis. We call it *TrACTOr*. The system is able to operate on any kind of solid, flat surface, like a wooden table or the floor. We developed a depth based tracking system which tracks the ACTOs from above. This type of tracking is less vulnerable to bad lightning conditions, as it has its own infrared light source. Also the positioning of the tracking marker does not cause any problems, as it can just be placed on top of the interaction surface. There it is visible for the tracking system as well as the HoloLens.

With these changes, it is possible to benefit from the advantages of TUIs provided by the ACTOs in an AR setting. In order to explore the potential we developed the *StARboard*

system, a learning system for sailing students. It should support sailing beginners in understanding the basic concept of driven bearings and different sail trims in various wind situations.
We developed a module for the ACTO robots that allows the user to interact with them in a way that suggests direct interaction with the virtual object he sees through AR. Figure 1.1 shows both, the ACTO robot and the virtual overlay. The virtual overlay matches the ACTO module (highlighted in blue) that the beam of the ship exactly overlays the beam extension of the rotary element of the ACTO module and the virtual button overlays the real button (highlighted in red). Also the width of the virtual ship matches the width of the ACTO. So if the user tries to grab the virtual ship he will actually grab the ACTO robot.

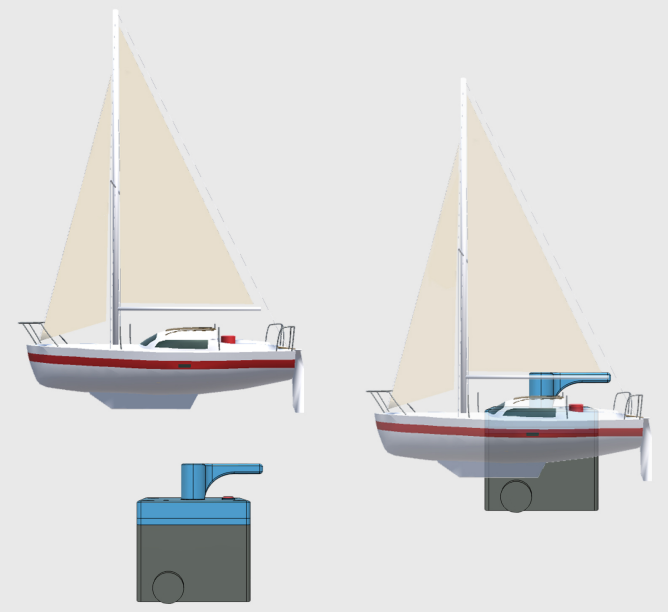


Figure 1.1: Overlay ACTO robot with virtual ship.

The user can physically pick up the ship and rotate it or move it to another place. To change the sail trim, he just pushes the beam of the boat left or right. This natural way of interacting with the virtual objects should provide an immersive experience and an intuitive handling to support the learning process and motivation.
With this simple interaction technique, the StARboard application provides the option to test various sail trims for certain wind situation easily and learn how to drive certain courses or maneuvers.

1.2 Goals and Research Questions

With this thesis we basically follow two main targets. First we want to show a novel way to combine a haptic interface with visual AR feedback. For this purpose we develop an

alternative tracking solution for the ACTO system, called *TrACTOr*. It also solves the other issues mentioned above, as it is not limited to the glass table and can be used on any kind of solid, flat surface.
We investigate how a depth based tracking system allows us to track the interaction tokens in 3D space. For this to work we equipped the ACTO robots with a small IR LED, used for identification. Also the system setup is relatively easy, as it only requires a depth sensor.
The tracking system has to be able to reliably track the ACTO robots from above. Therefore we tested the two existing versions of the Microsoft Kinect camera to see if the required accuracy can be reached. The factors which need to be tracked are the ACTOs current x, y and z position in the interaction space as well as the rotation around the vertical axis. We explore if this tracking system is able to work with multiple ACTO robots at the same time. To achieve this it is necessary to differentiate between different ACTOs, based on their id. At the same time we want to find out, if it is possible, that multiple users interact with the system at the same time.
The tracking system has to be fast enough to provide real time tracking results, to enable us to create the AR overlays. Another important aspect is the tracking accuracy. We examine if the system is accurate enough so that it is possible to overlay the real world objects with virtual objects. Only if the real world object and virtual object are well aligned, the direct interaction, mentioned above, becomes possible.
TrACTOr also has to provide an option for optical marker tracking, which is needed for synchronizing the tracking 3D space with the AR 3D space.
A problem that occurs with tracking from above is that the users hands occlude the tracked objects during the interaction process. The tracking system has to be resistant against occlusion to a certain degree.

This new tracking system allows us to follow upon our second major target. We examine whether the usage of Augmented Reality (AR) in combination with haptic interaction can be beneficial for interactive learning systems. Therefore we implemented a suitable learning system, we call *StARboard*. With this system we close the gap between virtual system and real world to provide *human-real world* interaction instead of *human-computer* interaction as described by Rakimoto et al. [RN95]. We achieve this by combining the haptic tokens and the overlayed AR content in a way that suggests direct interaction with the virtual object with haptic feedback from the real world object. Especially the movement of objects and adjustment of the sail should work as a sample for this. We hypothesize that by providing this kind of interaction we can improve the efficiency of the learning system. We also expect it to enhance the users motivation to learn.
The system should also support collaborative learning, so multiple users learn with the system together at the same time.

An empirical user study allows us to evaluate whether our described research objectives are reached and the assumptions we made prove to come true. For that purpose we conducted a study with 18 users, who tested the StARboard learning system. Every user was given several tasks to solve. In order to be able to give an objective statement

whether the StARboard system is really beneficial for the users, regarding our research goals, we compare three different interaction techniques. Besides the already described haptic interaction technique with AR, we also developed two other functionally equivalent interaction techniques. One of them is also based on AR, but it uses gestures for interacting with the virtual world. So the system does not use any physical tokens. The other interaction technique is based on the standard computer interaction tools most of us are familiar with: Mouse, keyboard and a standard computer screen for visualizing the scene. Every user tried each of the systems in different order.
This method should provide us with reliable information to make a point about the correctness of our assumptions.

1.3 Structure of the Thesis

In chapter 2 we give an overview of the research that has been done in the fields of TUI and AR, which are particularly related to this thesis. Various technologies and concepts which we use for our system are introduced in chapter 3. Chapter 4 contains the conceptual design of the TrACTOr tracking system and the StARboard learning application. In chapter 5 we discuss the implementation of the developed systems and give details about specific implementation features. Chapter 6 covers the results of the technical evaluation, as well as the execution and interpretation of the user evaluation. The outcome of the thesis is discussed in chapter 7. Finally, we give a brief summary and a outlook into the future of this topic in chapter 8.

CHAPTER 2

Related Work

In this chapter we give an overview of the research that has been done in the particular fields of TUIs and related works in AR.
First we will focus on TUIs and explain what they are and why research is performed in this area. We will also explain this with some examples that can be related to our work. After that, we will present some basic works of AR. We will illustrate why this is relevant for the thesis and give examples on how AR can be used for creating TUIs.
Finally, we introduce different tracking technologies that come into question for our tracking system. We will point out the advantages and disadvantages each of them has for our scenario and explain why we chose *depth tracking* in the end.

2.1 Tangible User Interfaces

This section will give an introduction into TUIs. After the general explanation, we will present why the research for new TUIs is important, and why they will be the future in HCI. Then we will briefly introduce the TUI we used for this thesis. Also, we will give some other examples for TUIs. Finally, we will focus on one special field of applications, which is the use of TUIs for educational purposes. This is the domain that our application belongs to.

2.1.1 What Are Tangible User Interfaces

Shaer et al. [Sha09] describes the development of *Tangible Interfaces* as a result of the efforts to develop applications for Augmented Reality and Ubiquitous Computing. The basic idea is not to force the user into a virtual digital world, but to extend the real world with digital content.
The first to talk about *Graspable User Interfaces*, which we nowadays call *Tangible User Interfaces*, was Fitzmaurice et al. [FIB95]. They introduced physical objects, so called

bricks, that represent virtual user interface elements. These allow direct interaction with the digital world.
They use the example of an ordinary car to make the difference clearer. In a car you have pedals for braking, a gear box for shifting and a wheel for steering. Each of them works differently and serves a different purpose. For the digital world we almost exclusively use mouse and keyboard to interact with all sorts of digital content. This poses a bottleneck that limits our interaction possibilities for the virtual world. By developing Tangible User Interfaces we can overcome this issues and the border between the real physical world and the virtual digital world diminishes.

The term Ishii et al. [IU97] use for Graspable User Interface elements is *Tangible Bits*. He describes them as the attempt to close the gap between physical and digital world by making the virtual information tangible. They picture the Tangible Bits as the counterpart of Augmented Reality. While they use a projector that is positioned above the desk as an output device to augment the real world with virtual content, the Tangible Bits should function as the input device to manipulate the data. They call the setup of this prototype *metaDESK*. The metaDESK is an example of an early TUI that uses tangible input devices.
In a more recent work Ishii [Ish08] describes that *Tangibles* (Tangible Bits) do not only serve as input device, but they also carry a physical state that is tightly coupled to the virtual information they represent. He uses Ben-Josephs TUI for urban planning as an example, which allows to position different models of buildings and simulate their shadow and wind behavior [BJIU+01]. Users can change the position and orientation of the buildings by simply moving the models around on the table. In this example the TUIs function as input devices, but at the same time they represent their own x- and y-position as well as their rotation on the workbench. All of these are attributes that the user can sense directly.

2.1.2 Why Tangible User Interfaces Are Useful

Our interaction with computers and digital information is inconsistent compared to how we interact with the rest of the physical world that surrounds us [Ish08]. Most of our senses, like the haptic interaction skills, are not used in the computer interaction with mouse and keyboard [IU97].

A lot of research is done towards improved tactile interaction with computer systems. MacKenzie et al. [MI94] describe how versatile the interaction between our hands and various objects is. We can manipulate, transport and feel objects with our hands, or we can even use them as tools to modify other objects and our environment. Everyone of us already brings this capabilities along, we only need to create interfaces to use them.
Another opportunity that TUIs give us is the use of both of our hands at the same time in two-handed interaction, which is another skill that we already have. Thereby the hands can either be used to perform the same task, each hand could do a different task, or both work together in a complex task, where the one depends on the other. Various studies have shown that proper design of these tasks can lead to high performance increase or, if

done badly, become an obstacle for the user interaction [BM86, KBS94].
The application StARboard we developed takes use of this knowledge. It is designed for two-handed interaction, whereas one hand controls the wind source and the other one the ship. But an alternative one-handed interaction method is also possible, where the user only uses one hand to setup one parameter after the other.

There are also less obvious advantages that we gain through the use of our hands, when interacting with TUIs. One aspect we know as *think with your hands*, which means the supportive use of your hands while performing a cognitive task. Kirsh [Kir95a] performed an experiment where users had to count coins with or without using their hands. The result was clear: the participants were much faster when they were allowed to use their hands and also the error rate was significantly lower. This is another fact we make use of in our interface. We want the user to be able to play around with the system's different parameters by directly modifying the setup with his hands. He instantly retrieves feedback from the system, as the robots move in a different speed. This short feedback loop should improve the learning curve, as discussed in chapter 2.1.5.

TUIs provide a whole new dimension for the way we interact with computers. It allows us to use the whole space around us. Kirsh [Kir95b] describes how we can manage our environment with techniques like categorization to be more productive. Many of these advantages do not apply to conventional Graphical User Interfaces but to TUIs.

Besides the already mentioned advantages, Fitzmaurice et al. [FIB95] mention even more that also apply to our interface, like:

- simultaneous, collaborative use
- more intuitive interaction by using physical artifacts
- more specialized input devices
- one to one mapping between control and controller

2.1.3 Actuated Tangible User Interface Object - ACTOs

The TUI we used for this thesis is called ACTO interface (Actuated Tangible User Interface Object), developed by Vonach et al. [VGK14b]. It is a highly modular TUI to quickly bootstrap and test various setups to develop new TUIs. It is based on small rectangular robots, so called ACTOs. These ACTOs serve as tokens for the TUI and can be equipped with different interfaces, like buttons, spinners, displays, etc. This allows the creation of TUIs for different scenarios. Furthermore, each ACTO has a set of wheels, so it can move around freely on the interaction surface.
One goal of our work is to develop a new tracking system for this TUI, that is not limited to any special surface. For the original implementation the tracking is based on optical marker tracking. The interaction surface is a glass table and each ACTO is equipped with an id-marker that sits underneath it, so it can be tracked by an RGB-camera positioned underneath the table. Read further information about the ACTO system in section 3.2.

2.1.4 Other Examples of Tangible User Interfaces

The following two examples make use of two key concepts that are part of the basic idea of the ACTO system. The first one, called *mediaBlocks*, is a very early TUI developed in 1998 by Ullmer et al. [UIG98]. Similar as in the ACTO system they use several tangible tokens that basically look the same but can serve an entirely different purpose. The second one, called *Pico*, is a more recent one. It was designed by Patten et al. [PI07] and, like the ACTO system, it is not only designed for one purpose but enables the developer to create different sorts of applications.

mediaBlocks

Ullmer et al. [UIG98] developed a Tangible User Interface for containment, transport and manipulation of online media. This so called *mediaBlocks* are made of wood and can be tagged electronically. They can be used to transport large pieces of data between input and output devices, whereas they do not function as storage themselves, like USB-sticks or CDs, but as reference for the data, which itself is transferred via the network.
Also it is possible to use these blocks as controls for tangible interfaces on manipulation devices like sequencers. This gives the user a quick and easy way of interacting with the data, while the physical token represents the actual data and can be physically carried from one device to another. A simple use case would be the recording of a short video. The user can carry the video in form of a mediaBlock from the camera to the sequencer, where he uses other mediaBlocks to manipulate it and then put the block into the projector to display the video.
Furthermore, these mediaBlocks can be attached directly to conventional Graphical User Interfaces (GUIs) to make the data available on normal computers as well. Figure 2.1 shows the interaction of mediaBlocks with different devices.

Figure 2.1: Operating with *mediaBlocks* (Ullmer et al. [UIG98]).

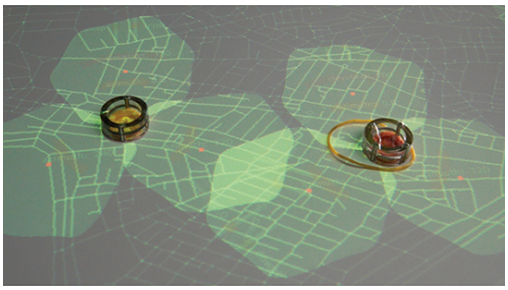
Ullmer et al. had to solve the problem that the mediaBlocks they use have the exact same color and shape. This makes it difficult to differentiate between them, for the system as well as for the user. They overcame this issue by applying electronic tags to

them to make them identifiable by the system. For the users they labeled them with text and a color indicator. The same issue applies to the StARboard interface we developed: all the ACTO-robots look the same. To make them identifiable by the computer system we use an infrared transmitter to send their ID. For the users the system provides a 3D augmentation with a virtual object (See chapter 4.2.5). With this way of displaying the information we have the advantage that we do not rely on an extra display, but the information can be directly displayed on the interaction token itself.

Pico

The Pico system (Physical Intervention in Computational Optimization) developed by Patten et al. [PI07] is a more recent TUI. Similar to the ACTO system, this interface is not designed to serve only one purpose, but to experiment with different conditions and constraints for various fields of applications.
The system is based on the *Sensetable* developed by Patten et al. [PIHP01], which provides a flat surface that can electromagnetically track (See 2.3.2) objects that are placed on top of it and also move them around on the table. These objects act as interaction tokens. The system also includes a projector that is placed above the table and gives visual feedback. This setup enables the users to place different types of tokens on the table. They are tracked by the system and depending on the underlying software which manages the scene, Pico will move them around and provide visual feedback through the projector. One interesting aspect of this system is, that the tokens can be moved by the user as well as the computer. The user can also restrict the computers movement intentions by adding constraints to the tokens, such as the position of obstacles on the table or just holding the tokens fixed with a hand and preventing automated movement (Figure 2.2a).

(a) Using physical constraints for restricting token movement.



(b) Examining possible arrangements of cell-phone towers with *Pico*.

Figure 2.2: The Pico system by Patten et al. [PI07].

As a sample application to introduce the Pico system Patten et al. implemented a system for planning the positioning of cellphone towers (Figure 2.2b). The only possible way of interacting with the system is by positioning and moving the tokens.

The StARboard interface we developed functions in a similar way. In our case the interaction tokens can also move around on the interaction surface by themselves. We have the advantage that with StARboard we can not only display visual information around the interaction tokens, but we can visually replace the interaction tokens with 3D information, which in our case is a ship and the settings are bearing and sail trim.

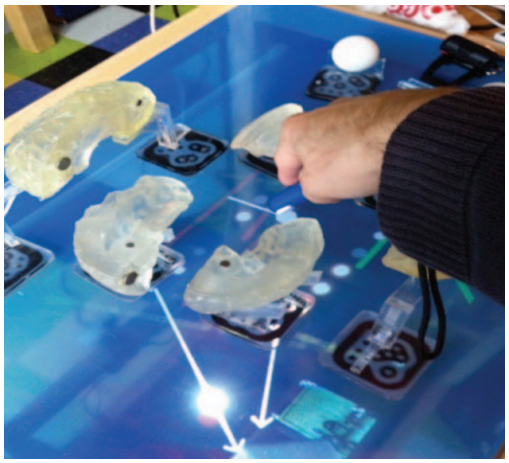
2.1.5 Tangible User Interfaces for Learning

In their book Shaer and Hornecker [Sha09] list a wide range of fields where TUIs can be used. Besides domains like *Information Visualization*, *Problem Solving and Planning*, *Tangible Programming* or *Music and Performance* they also name *TUIs for Learning* as an important field where TUIs can be applied.

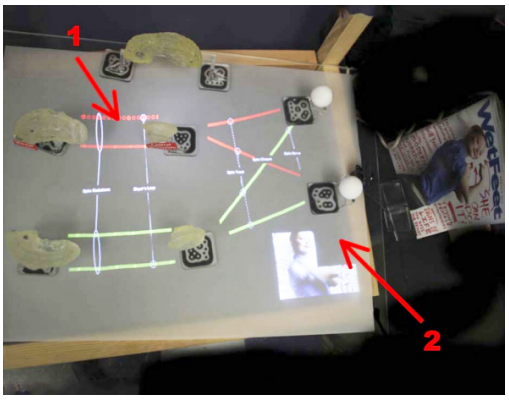
They state that learning systems which are based on tangible interaction are very well suited for children, because they require many of the human senses, which supports their development. But TUIs can also be used in learning systems for adults, to improve the cognitive processes. Many of these learning systems are in the domain of planing, problem solving or simulation, as *StARboard*, the system developed in this thesis.
An increasing number of studies focus on researching the benefits of using tangibles in learning environments. Price et al. [PFSR09, Pon09] conducted a study where children learned about the behavior of light using a tangible system. They observed different aspects of tangible learning and found that the children were highly engaged in working with the system. Also, they understood the presented concepts easily by playing around with the system and figure out the principles of light themselves.
The outcome of another study, carried out by Schneider et al. [SJZD11], can be applied directly to the *StARboard* interface. Schneider researched the learning benefits of TUIs for collaborative learning. This is relevant for our project as the StARboard application can also be operated by several people at the same time. If you think of a theoretical sailing lesson, a group of students could learn together to understand the correlation between wind, bearing and sail trim. Schneider et al. did an empirical study where the participants should be trained in planning the setup of racks inside of a warehouse. The aim was to reach maximum efficiency and optimized transportation routes with the forklifts. The apprentices did not have special preknowledge in the topic of logistics in a warehouse. The aim of the experiment was to compare a TUI with a screen based multitouch interface. In the TUI the racks were represented by little models of racks that could be directly grabbed and positioned by the participants. In the screen based solution those racks were represented by 2D graphics on the screen. The result of the study was that the apprentices performing with the TUI did not only have a significantly better learning outcome, but also could achieve better results in performance and efficiency of the designed system.

The aim of the StARboard application is to simplify the complex environment of sailing which is influenced by many different parameters. It is narrowed down to the most important ones which are then put into a TUI where tangible tokens represent objects of the real world. Those can be manipulated and brought into relation to each other to

learn about the behavior of the whole system and gain insight into the complex process that is taking place in the real world.
Schneider et al. designed their learning TUI *BrainExplorer* [SWBP13] in a similar way to allow better understanding of the neuroscientific processes taking place inside of the human brain. They created a TUI that represented the structures that are involved in processing visual stimuli. Therefore, they created a couple of tangible tokens representing the human eyes and different parts of the brain (See figure 2.3a). When these tokens are positioned in the interaction area, they are augmented with a 2D projection that visualizes connections between the components.
A webcam simulates the visual functionality of the eyes and acts as input source for the image the human sees. By manipulation of the tokens the user can now experiment with different constellations, for example examine the effect of failure of one of the involved brain areas. Figure 2.3b shows the result of a lesion to the persons field of view.
BrainExplorer uses tangible tokens which are shaped like parts of the human brain or eyes. An advantage of the StARboard application is that the interaction tokens do not need to have the exact shape of the objects they represent. We can easily use the same token for different objects, as it is only visually replaced with a virtual object.



(a) Tangible tokens represent different parts of the human brain.



(b) The broken connection (1) represents a lesion in a certain brain area and results in the loss of the top right corner of the persons visual field (2).

Figure 2.3: The *BrainExplorer* tangible learning interface by Schneider et al. [SWBP13].

Analog to the StARboard application, a designated aim of the designers is that there are little to no prerequisites for the users to use the interface and explore the topic. An active learning experience based on sensomotoric actions should be created and the system should allow different outcomes, so not only one predefined path of interaction exists, but the user can freely choose what to do and how to use the system.

2.2 Augmented Reality

In this section we want to give a short introduction about Augmented Reality (AR). First, we will describe what AR is in general, followed by a brief overview about why AR is an important, aspiring technology especially for TUIs. Finally, we will give examples of TUIs using AR to give rich visual feedback within the applications interaction context, one with 2D and one with 3D augmentation.

2.2.1 What Is Augmented Reality

An early survey about AR was done by Azuma in 1997 [Azu97]. Augmented Reality is, as the name suggests, the augmentation of the real world with virtual content, where real and virtual world coexist. Azuma describes AR as a variation of Virtual Reality (VR), whereas AR supplements reality rather than replacing it completely, as it is the case in VR. Milgram and Kishino [MK94] created the concept of the *virtuality continuum*, which defines the different abstraction levels between a real and a virtual environment (Figure 2.4). It also introduces the term Augmented Virtuality (AV), which is the converse case of AR where the virtual world is enriched with real world objects. AV will not be covered in more detail here.

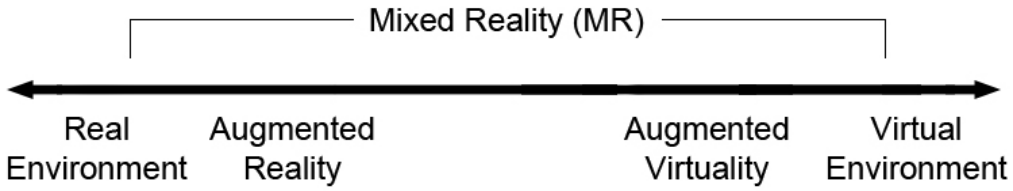


Figure 2.4: *Virtuality Continuum* by Milgram et al. [MK94]

Azuma defines three key requirements for an AR system, which must also be considered for the *StARboard* application. It has to combine the *real, 3D world* with *virtual 3D objects* and needs to be *interactive and in real time.*

The fields of application for AR are wide. While Azuma only called six of them in 1997, for example medical visualization, maintenance and repair or military aircraft navigation, a more recent complement to that survey [ABB+01] includes more, like navigation systems. With advanced mobile technologies many more will emerge in the future.

For the sake of completeness it should be mentioned that AR does not only have to be limited to the visual component. The real world can also be augmented with virtual content for other senses, like the sense of smell, taste or even touch.

Another approach based on the visual sense is the augmentation of the real world with 2D visualizations. A 2D augmentation can be achieved easier than a 3D one, as the augmentation process is less complex and the visualization does not rely on the users point of view. This allows the use of more established technology as projectors or displays. 2D

augmentations are a common approach especially for TUIs, like the *reacTable* [JGAK07]. For this reason, 2D augmentation could also be considered for the StARboard application. But the more realistic feedback and better immersion, which can be accomplished by 3D augmentations, are important benefits that support learning (read more in section 2.2.3).

2.2.2 Augmented Reality for Tangible User Interfaces

The advantages of AR are apparent: It gives a developer the possibility to create a system that merges with the real world, which enhances the user's perception [Azu97]. Users are not bound to an ordinary computer screen anymore, the whole world becomes the screen. This matches the aim of TUIs discussed in chapter 2.1.1. Ishii [IU97] described that with TUIs the whole world becomes the interface. This means that AR is the ideal output mechanism for a tangible user interface, as it perfectly fits the requirements to decrease the border between the physical and the virtual world.

Another project by Kaufmann [PAH14] supports our decision to use AR for StARboard. It is about the use of AR in a collaborative learning environment. The system he developed is called *Construct3D* and it allows students to create and manipulate different sorts of geometric primitives and experience the relation they have to each other. After several evaluations and design-improvement-loops, Kaufmann et al. [KD07] presented the result of their final user study: They tested their system against a desktop based geometry education application. The result was, that the usability of their system was rated better. However, Kaufmann pointed out that there were still issues related to the technology they used for AR, which could be improved to gain even better usability.

2.2.3 Examples of Augmented Reality Used in Tangible User Interfaces

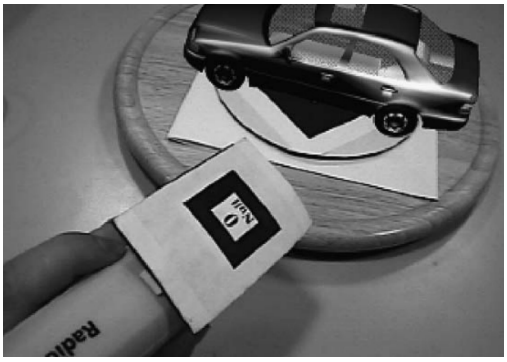
As already discussed, there are two possible augmentation strategies that we could use for the StARboard application. In this section we give a quick example for each of them and explain why we decided to use the 3D approach.

reacTable - 2D Augmented Reality

The *reacTable* developed by Jordà et al. [JGAK07] is a typical example for a 2D augmentation of real world objects (figure 2.5a). A projector positioned underneath a glass surface can augment each tangible token individually with different information, which is displayed around the token. Depending on the token type, the information displayed varies. This approach can be applied to StARboard in a similar manner. Around the wind source the current wind speed as well as the wind direction can be displayed. For the ships we can display the current bearing, speed, sail trim and its efficiency compared to the perfect trim. Another advantage is that occlusion through the user's hands interacting with the tokens does not become a problem, because the tracking as well as the projection are done from below the glass surface. The tracking will be explained in more detail in chapter 2.3.1.3. The big disadvantage for our intended application is that the tangibles

themselves cannot be visually replaced by virtual objects. Also one requirement of the *TrACTOr* tracking system is, that it should not rely on a specific interaction surface (see chapter 4.1.1). This issue could be overcome by positioning the projector above the surface, which would cause other issues, as the already addressed occlusion problem.

(a) Live music performance with the *reacTable* by Jordà et al. [JGAK07] (a TUI using 2D augmentations).



(b) Lighting a virtual car with a virtual flashlight in *MagicMeeting* by Regenbrecht et al. [RWB02] (which uses 3D AR).

Figure 2.5: 2D and 3D augmentation.

MagicMeeting - 3D Augmented Reality

The *MagicMeeting* interface developed by Regenbrecht et al. [RWB02] uses 3D virtual objects to replace the tangible tokens, which in their case are 2D fiducial markers. Some of them are attached to real world objects, as the flashlight in figure 2.5b. *MagicMeeting* is a collaborative tangible AR interface that is designed to serve multiple purposes, comparable to the *ACTO* system. It gives the users a comprehensive set of tools, which they can use to perform various tasks dependent on the actual application. Each user wears AR glasses, which enables the system to give visual output in form of 3D objects, that are positioned within the real world interaction space. The sample application they present in their paper is a tool for the automotive industry. As shown in figure 2.5b, the users can see a virtual car positioned on a token. Now they can apply different kinds of transformations to it by using some of the interaction tokens. This could be the change of the color of an object or hiding parts of the model. In the picture the user applies a virtual light beam to the car model by using a virtual flashlight.
An advantage of using a 3D augmentation is, that the system works more like the real world. This allows moving around the object to see it from different angles. The object seems to really consume space of the real world. For StARboard this means that it becomes possible to replace the ACTOs, which are the tangible tokens, with virtual objects, like ships in our case. To the user it looks like a small ship stands on the interaction surface instead of the token. Compared to the MagicMeeting interface, StARboard has the advantage that the tokens do not need to have fiducial markers

attached. The tokens can be directly tracked without a marker. A disadvantage, is the occlusion problem: While the user interacts with the token the hand should occlude the virtual object to provide a realistic effect. There are possible solutions for this problem, which are mentioned in section 7.1 future work.

2.3 Tracking for Tangible User Interfaces

A lot of research has already been done in the field of Tangible User Interfaces. Most of the developed systems require a proper way of tracking the physical tokens that are used for interaction with the virtual world. Different technical approaches exist and dependent on the specific system requirements, some are more suitable for the specific use case than others. In the following sections we will present different tracking systems that could be used in a related context.

2.3.1 Optical Tracking

One of the most common tracking technologies is optical tracking. Cameras are used to record a video stream of the interaction scene, which is processed by a computer in realtime to detect the tokens. Therefore it is required to develop according image processing algorithms, dependent on what should be tracked. One of the big advantages of optical tracking is, that the required hardware is highly available, cheap and easy to handle, for example the camera of a smartphone. In addition the tracking quality can be very good.
Basically there are three types of optical tracking, which can be distinguished by the type of camera that is used: *depth-camera tracking*, *RGB-camera tracking* and *infrared-camera tracking*.

2.3.1.1 Depth Camera Tracking

Depth cameras, as the Microsoft Kinect [Kinc], are able to create a gray-scale video stream, whereas the different gray-values represent a different distance to the camera. With appropriate image processing algorithms it is possible to separate the interaction tokens from the rest of the scene, which allows an accurate position detection in 3D space. The x- and y-position can be directly derived from the objects position on the recorded frame, whereas the z-position, which is the distance the token has from the camera or the background, can be deduced from the gray-scale value of the according pixels.
Follmer et al. use the Microsoft Kinect camera for their TUI [FLO+13] as well. They create a depth-image of the interaction area, which is used to track the user's hands and surface objects that are used for their shape display *inFORM*. Therefore, they generate a background image from the "empty" scene and use this as a reference image to compare it with the current depth image. In this way it is possible to detect objects in the tracked area. A specific challenge that has to be solved when developing a shape display is, that the interaction does not only happen on a 2D surface but in a 3D space. This requires more sophisticated tracking methods, which is why a depth sensor is used in this

case. They also use the Kinect to detect touch on the surface of the shape display. This method is not that accurate and they suggest a touch sensitive surface would provide more reliable results. We cannot use the inFORM's tracking method itself as tracking tool for the ACTO system because it relies on color tracking for objects and only uses depth tracking for the users hands. But we implemented a method that based on the same principle as their depth based hand tracking. We create a background image and compare it with the current record to create a foreground image. This image can then be used for image processing. Read more about this procedure in chapter 4.2.2.
There is a couple of advantages the depth-camera tracking has compared to the other techniques. The reasons why we decided for this solution are:

- It is capable of tracking tokens in 3D space.
- Besides the identification-LED, the ACTOs do not need any special modifications, which is utterly important for the ACTO system as its main idea is to serve as a generic basis for various modules that can be applied to it. This requires a high grade of flexibility.
- Aside from a depth-camera there is no special hardware needed.
- The tracking is basically independent of the surface the ACTOs move on, which was one of the projects requirements.

2.3.1.2 RGB Camera Tracking

A system developed by Weiss et al. [WSJB10] uses tangible magnetic widgets on an interactive table. There are several types of those widgets, so called Madgets, for example simple buttons, sliders or whole keyboards. Each of them is positioned on top of a set of circular markers that can be detected with a simple RGB-camera from underneath the table. The arrangement of those markers makes it not just possible to detect and track the exact position of the Madget, but also to identify its type as they have different arrangements for different Madgets.
Furthermore, it is possible to move the Madgets with an array of electromagnets positioned under the table's surface. The purpose of this system is to enable the researchers to develop and prototype new interactive user interfaces.
This approach is not applicable to our system as it requires a specific table and the requirement for our project is to be independent from the interaction surface.

2.3.1.3 Infrared Camera Tracking

A well-known example in this field of research is the reacTable [JGAK07] developed by Jordà et al. The system is used to create an interactive real-time audio synthesizer.
They use an infrared camera that is positioned below a translucent surface and tracks the interactive tokens placed on top of it. Each of these tokens is equipped with an optical marker that faces towards the camera and allows easy identification and position tracking. Also it is possible to track the user's fingers when they touch the surface of the table to provide a further way of interaction. At the same time a projector positioned

underneath the table gives visual feedback to the manipulation of the tokens.
Again, this tracking technique cannot be applied in case of the TrACTOr tracking. In order to be independent from the interaction surface, the tracking would have to occur from above. Also the visual feedback should not only be a 2D image, but a 3D augmentation of the real world.
The advantage of positioning the projector and camera below the interaction surface is that occlusion problems with tracking and projection can be avoided. To reduce these problems, our tracking is implemented to be more robust against interruption caused by partial occlusion.

2.3.2 Radio Frequency Tracking

Audiopad [PRI02] by Patten et al. is another tag-based tangible interface for musical performance, but the tracking technology works differently than for the *reacTable*. The tokens contain a so-called LC tag, which is basically a coil of wire and a capacitor. Depending on the inductance and capacity this tag resonates at a different frequency, which makes it possible to distinguish between different tags.
To track their position and orientation it is necessary to place several antennas in a certain geometric order. The amplitudes of the tags resonance can then be interpreted and the position and its rotation can be derived. For displaying information they use a projector that is mounted above the interactive surface.
The tracking works quite fast and accurate, which gives a good response to the user.
To be able to overlay the real world with AR content it is necessary to track the tokens in 3D space, which is not possible via the Audiopad tracking.

2.3.3 Ultrasound Tracking

The tracking of very high frequency sound, so-called ultrasound, is an approach Mazalek et al. propose for their TViews [MRD06]. They designed a digital media table that basically consists of a horizontal glass plate that serves as interaction surface. The interaction tags can be placed anywhere on this plate. In the corners acoustic transmitters are positioned. The glass transmits the acoustic signals to the tags which calculate their own position. In the second step they send their identity and position back to the table via an infrared transmitter. The big advantage of this technique is, that it enables a very high amount of tags being on the table at the same time. To give visual feedback a display is placed right underneath the glass plate.
The resulting tracking is quite accurate and fast, but is not applicable for the ACTO tracking. As soon as they are lifted the tracking would be lost, which would be unacceptable for our project.

2.3.4 Hybrid Systems

Hybrid systems use different tracking mechanisms at the same time to either increase the accuracy and performance of the tracking, or as the *STARS* interface developed

by Magerkurth et al. [MMES04], to track different aspects of the system. *STARS* is a platform to realize computer augmented tabletop games and it uses two sorts of tracking. While the user's hands are tracked with an ordinary camera positioned above the table, the actual playing pieces are detected via an integrated RF-ID antenna system based on the *interacTable* developed by Streitz et al. [STMTK01]. The table consists of a horizontal display which is used for displaying the playground, another vertical display serves as screen for giving additional information to all players and additional hand held displays give individual information to each player. In combination with physical tokens the system enables a wide variety of tabletop game scenarios.
The disadvantage of this system compared to the system developed is that it is not possible to track the vertical position of an object when it is lifted off the ground. This is necessary to provide proper Augmented Reality overlays.

2.3.5 Other Tracking Types

Other than the mentioned tracking technologies, there are also various other tracking strategies used for TUIs, like the *DataTiles* system developed by Rekimoto et al. [RUO01]. In their system the tokens can only be positioned within a predefined grid system that only allows discrete positioning. The actual position is detected by several RF-ID readers that are positioned behind every panel of the grid. Or the *Augmented Coliseum* developed by Kojima et al. [KSN+06]: In contrast to the other systems introduced so far they use inside out tracking. This means that not an outside sensor measures the position of all the tokens, but each token determines its location itself. They project certain patterns onto the interaction surface which are interpreted by the tokens and transformed to location values.
These tracking mechanisms are not applicable to the ACTO system and therefore not essential for this thesis.

CHAPTER 3

Basics

This chapter contains a collection of different technologies that are used in the *StARboard* application and *TrACTOr*-tracker. Each of them is explained in detail. The information provided in this chapter should give enough basic knowledge for understanding the concepts discussed in later sections of this thesis.

3.1 Kinect Depth Sensor

The Microsoft Kinect is a very wide spread depth sensor that was designed for computer games in the first place but is also used for many research projects (eg. [FLO+13]). The first Kinect version was released in 2010. During the development process we mainly used this version, but we designed the system to be capable of using the second version as well. For testing and evaluation we used the current version, as it has better performance and higher resolution as well as more accurate depth detection.

3.1.1 Kinect Version 1

The Kinect [Kinb] was developed by Microsoft [Mic]. Version one was released in November 2010. It is a system that basically combines an array of different sensor technologies. Besides the infrared emitter and the infrared depth sensor, which are both used for the creation of the depth image (described in section 3.1.1.1), it also contains an RGB camera, a tilt monitor and a microphone array, which is spread out over the whole sensor. It provides a synchronized color and depth image and uses a 3D human motion capturing algorithm to detect people and their movements within the camera scene [JLDS13].

3.1.1.1 Depth Measurement

Khoshelham [Kho12] describes the mathematical background that lies behind the depth sensor. The method used for depth measurement was invented by Freedman [FAMA10] and is based on triangulation. The infrared emitter is a special kind of laser that projects a constant, structured pattern of points onto the scene (See figure 3.1a). The infrared camera captures the scene with the point pattern. The Kinect then compares the pattern that is being sent with the one that gets recorded. Each area in the point cloud looks different, so based on the distances between the points the Kinect can identify each single point and compare its actual position with the expected one. Due to the fact that the emitter and the sensor are a couple of centimeters apart from each other it is possible to use triangulation to calculate the shift between the expected and the measured position (this works similar as the stereoscopic vision of humans). With this it is possible to determine the distance between the Kinect and each reference point. Figure 3.1a shows the resulting depth image.

(a) Kinect 1 point cloud and depth image. (b) Kinect 2 depth image.

Figure 3.1: Kinect depth camera.

3.1.2 Kinect Version 2

The second version of the Kinect [Kinc] was released in 2014. The basic components are similar to version one, but the technology used for depth perception is entirely different (See section 3.1.2.1). This version also houses an RGB camera, a microphone array, an infrared light source and an IR camera. Similar to the first version it is capable of providing an RGB image and a depth image of the scene, which it uses to detect human skeletons. But also more detailed information, like facial expressions, can be received.

3.1.2.1 Depth Measurement

From a technical perspective the biggest difference to the first Kinect version is the way the depth image is created. While the first version used structured light, the second version uses the Time-of-Flight (TOF) technology. This means that the IR emitter sends out bulk light with a certain frequency to illuminate the scene. The light is reflected on the objects in the scene and the IR sensor receives these reflections, just as a normal IR

camera would do. The difference is that the TOF sensor can also detect the received light's phase for each pixel. From the phase shift of the emitted light to the received light it is possible to measure the time it took the light to reach the sensor. With this information it is possible to calculate the distance between the according object and the sensor.
As this is done simultaneously for each pixel, it is possible to create a 3D image of the recorded scene (See figure 3.1b). [Li14, XP16]

3.1.3 Technical Specifications

Table 3.1 shows the technical specifications, that are relevant for the TrACTOr tracking system, of both Kinect versions in comparison:

		Kinect v1	**Kinect v2**
RGB image	**frame rate**	30 fps	30 fps
	resolution	640x480	1920x1080
depth image	**frame rate**	30 fps	30 fps
	resolution	320x240	512x424
	depth detection range	∼40cm - 4m	∼50cm - 4.5m
viewing angle	**horizontal**	57°	70°
	vertical	43°	60°

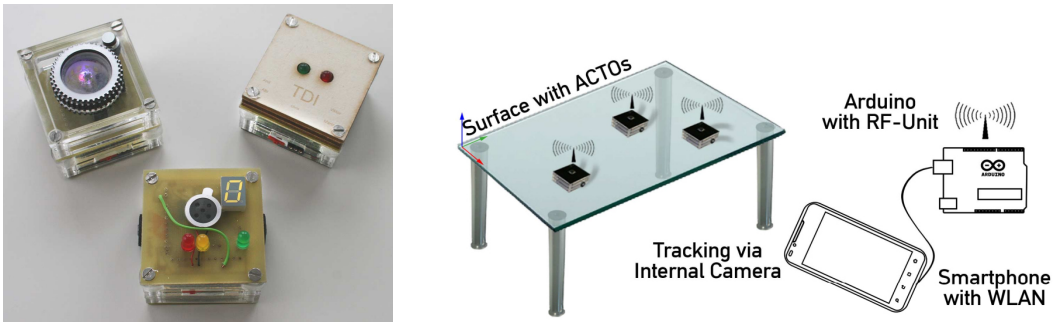
Table 3.1: Technical specifications of Kinect v1 and v2 in comparison [Kinb, Kina].

3.2 ACTO Interface

The TUI we build upon was developed by Vonach et al. [VGK14b]. It is an actuated interface, which is defined by Poupyrev et al. [PNO07] as *"an interface in which physical components move in a way that can be detected by the user"*. In the ACTO system these components are small rectangular robots, so called ACTOs (Modular Actuated Tangible User Interface Objects) (See figure 3.2a). These ACTOs serve as tokens for the Tangible UI and can be equipped with different interfaces, like buttons, spinners, displays, etc. This allows the creation of TUIs for different scenarios. Furthermore, each ACTO has a set of wheels, so it can move around freely on the interaction surface.

3.2.1 ACTO Hardware

From a technical perspective ACTOs are based on an Arduino Pro Mini [Ard] *Base Module*, which also consists of an interface for Radio Frequency (RF) communication. The whole system is designed in a modular way, so that this base module can be extended with various custom made *Extension Modules*. One of these Extension Modules is the *Motor Module*, which is positioned underneath the Base Module and allows the ACTOs to move around with a pair of rubber wheels. On top of the Base Module other modules



(a) Different ACTO interfaces. (b) The ACTO system setup.

Figure 3.2: ACTOs (Modular Actuated Tangible User Interface Objects) by Vonach et al. [VGK14b]

can be connected. Those can be customized for the requirements of the TUI, like buttons, spinners or screens.
For the StARboard interface we also designed a custom module, containing a button, a spinner and an IR LED for identification.

3.2.2 ACTO-Framework and Tracking

To make the ACTO system usable for other people, Vonach et al. developed a framework that provides basic functionality for controlling and tracking the ACTOs. Users who want to build their own custom modules and develop their own software to create an individual TUI based on the ACTO system can make use of this framework and build their application on top of it.
The ACTO-Framework basically consists of a server, that is based on Java and runs on an Android phone and the clients, which are the ACTOs.

3.2.2.1 Server - Android Software

As battery power and endurance is an issue with the ACTOs the framework is conceptualized so that most of the processing takes place on the server. It maintains all of the ACTOs positions, by tracking them and giving them basic movement commands to move them around on the table. The communication for these commands, as well as the interface commands received from the ACTOs, run over a RF connection. As most of the Android phones do not provide such an interface, an additional Arduino is connected to the phone, via USB. This so called *RF-Unit* is equipped with a RF transmitter.
The standard framework provides a basic GUI which allows the user to see the current system status and give basic movement commands. It can also be extended to provide more functions. (This step was required for development of the TrACTOr tracking.)
Furthermore, the framework provides a WiFi interface, which allows third party applica-

tions to connect and communicate with it. (This step was required for the TrACTOr tracking and StARboard implementation.)

Tracking

The tracking is based on optical marker tracking. The smartphone running the server is positioned underneath a glass table (as shown in figure 3.2b) and tracks the ACTOs with its RGB camera. Therefore each of the ACTOs has a fiducial marker (read more in section 3.4.1) positioned on the bottom, pointing towards the camera. For this to work the system needs to run on a glass table, so the camera can actually see the ACTOs. This limitation is one of the main reasons the TrACTOr tracking was developed, so the system can run on any kind of flat surface.

3.2.2.2 Client - ACTO Software

The ACTOs themselves run the so called *ACTO OS*. It provides the basic functionality for communication to the server and can execute the received movement commands. It also offers functions for developers of custom modules that make it easy for them to address input and output pins without conflicting with the ones already used by the ACTO OS itself.

3.3 OpenCV (Emgu CV)

The main library we use for the image processing tasks in the TrACTOr-tracker is *Emgu CV* [Shi13]. Emgu CV is basically just a *.NET* wrapper to the open source computer vision library *OpenCV* [BK13]. It allows us to use most of the functions OpenCV provides in *C#*. In their book *Learning OpenCV* Bradski et al. describe that OpenCV can be used for a variety of tasks connected to processing or editing stills or videos. One example could be to convert an RGB image to a grayscale image, or to remove camera motion from a video. The authors mention various fields of applications where OpenCV is used, like factory product inspection, medical imaging, camera calibration, stereo vision or robotics.
OpenCV has a main focus on real time processing of video streams (which is what we need for the TrACTOr-tracker). We use computer vision during different stages of the tracking process. For example to create a stable background image from the depth stream and compare it with the current depth image. Also the actual square and line detection works based on functionality provided by the OpenCV library.
Another task that we use OpenCV for is the calibration. In order to be able to calibrate the different coordinate systems it is necessary to use some sort of calibration image or marker, which in our case is just an ordinary picture printed on a sheet of paper. OpenCV supports us in identifying and locating the picture within the camera image. Read more about image tracking in chapter 3.4.

3.4 Image Marker Tracking Basics

A very important task for AR or systems using multiple optical devices, is to detect the cameras position relative to the coordinate system. This can be achieved by putting a 2D marker into the scene, which is detected by the camera. With this information the camera is able to estimate its own relative position to the marker which enables us to synchronize multiple optical systems or to create visually appropriate augmentations for AR. Basically there are two different types of trackings: *Marker tracking* and *Marker-less tracking*.

3.4.1 Marker Tracking

For marker tracking a special kind of fiducial marker is required. An example for such a marker is shown in figure 3.3. This marker system is called *ARTag* and was developed by Fiala [Fia05]. It contains a set of markers with different ids that are not just trackable but can also be distinguished from each other. The tool is able to detect the marker within the camera image and calculate its relative pose to the camera. The disadvantage of marker tracking is that such a marker always has to be applied to the scene, which simply does not look good in many cases and covers a certain amount of space, which is often not feasible.

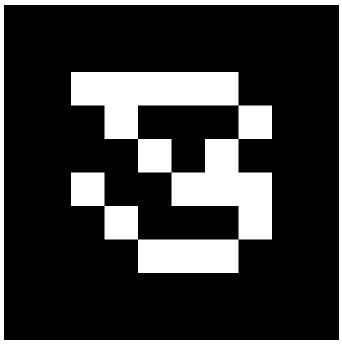


Figure 3.3: Fiducial marker, *ARTag* with id 0 (by Fiala [Fia05]).

3.4.2 Marker-less Tracking

This problem can be overcome by the use of marker-less tracking. This technique does not require any additional kind of marker, therefore natural features of the scene are used, like corners or edges. Of course it is required that the scene provides enough of those features to ensure proper tracking. It is for example not possible to track a plain white wall for this reason.
Wagner et al. [WRM+08] describe their approach of marker-less tracking for mobile devices which has to overcome special difficulties due to limited processing power and memory. They modified two different tracking methods to make them available for mobile devices, which is an important step for providing AR on smartphones and Head Mounted

Displays (HMDs).
For our system we use two different tools for tracking. As described in section 3.3 the TrACTOr-tracker uses the tracking provided by OpenCV. On the other side the StARboard client uses tracking provided by Vuforia [vuf], another provider for tracking frameworks.

3.5 Kalman Filter

A Kalman filter can be used to remove inaccuracies and disturbances caused by measuring instruments. Welch et al. [WB06] describe it as "*a set of mathematical equations that provides an efficient computational solution for the least-squares method*".
The basic idea is to use the knowledge of the previous measurements to make the current measurement more accurate. This works by creating a formula that uses all of the already existing measurements, the so called *priori* state and calculates an estimation for the next measurement, the so called *posteriori* estimation. Now the estimation is compared with the actual measurement and by weighting these two results the final value can be calculated. Afterwards this value will be included to the priori state and also be used for estimating the following measurements.
This means that every value that is calculated in this manner is probably not hundred percent accurate, but it is likely to be more accurate than the actual measurement. This only applies to systems where the measured values are more or less continual. If there are many jumps in the series of measurements the result gained by the kalman filter will become less accurate or even useless.
For example if you want to measure an object that moves steadily in two dimensional space without doing abrupt jumps from one position to another a kalman filter can be used to flatten the measured values to a smooth movement path. This is important for the tracking of ACTOs, to get smoother position and rotation values, as well as for predicting an ACTOs location if the tracking is lost for a short period of time.

3.6 Unity

Unity [Uni] is a development environment for 3D games. Besides game development it is a common platform for developing other kind of software that uses AR or VR. It was developed by *Unity Technologies* and the first release was in 2005.
Unity gives the developer a fast and easy way of creating 3D scenes and adding interactivity by applying scripts to the objects within the scene. Another great advantage of Unity is that it already provides a lot of solutions for common problems and tasks that appear for this kind of applications, like a physic engine or client server system functionality. Also it is not only bound to one platform, Unity allows the developer to easily deploy to all the common platforms, like iOS, Android or Windows.
All of this makes it a great tool for developing any kind of 3D based applications.

3.7 HoloLens

The Microsoft *HoloLens* [Holb] is an optical see-through HMD. This means that the user can see the real world at the same time as the virtual content provided by the HoloLens. This is possible as there is an optical projection system included which projects the virtual image onto a transparent surface in front of the users eyes, the so called holographic frame. In combination with a complex array of sensors, which track the 3D world around the user and its own position and orientation within it, the HoloLens can create a very good AR experience.
As the user can move and look around freely in the scene the level of immersion is relatively high for the state of the art. The resolution of the virtual image is comparatively high, as well as the opacity of the virtual image. A weakness is the field of view, which could be larger. But this fact can be neglected as the tracking is very stable which allows the user to look around smoothly.

3.7.1 Deploy from Unity

Basically HoloLens applications are developed in C#. Also there was a special version of Unity released for developing HoloLens apps. They also provide the so called *HoloToolkit* that provides many basic functions like gesture recognition and cursor interaction, which is quite useful for development. So far the deploy process from Unity is a bit complex, as there is no direct way to deploy from Unity to the HoloLens. First it is necessary to export a Visual Studio C# project from the Unity project. This can then be opened in Visual Studio and deployed to the HoloLens. Especially during the developing process this consumes a lot of time. In the future the Unity compatibility for HoloLens will probably be improved and a direct deployment process might be provided.

3.8 Sailing Basics

There is a wide range of different types of sailing boats. The way a boat is built and the type of sails that are used are the basic factors that determine the bearings that can be sailed and the possible speed. Besides those and the wind there are many factors playing a role for the speed, like current, heeling, payload, load distribution and many more. For the sake of simplicity we only consider the most important ones in our system: boat and sail type, sail trim, wind direction and speed and bearing.
Wilson [Wil08] describes how the maximum possible speed can be calculated, dependent on the boat and sail type, bearing and the wind speed. This information can be displayed by using a *polar diagram*, which displays the maximal possible boat speed per wind speed and bearing. It does not consider the current sail trim, an optimal trim is assumed. Figure 3.4 shows an example for a polar diagram of a modern yacht.
The curve of such a polar diagram can vary greatly for different boat types. Not just the speed, but also the possible bearings. While the boat in this example is able to sail every

course between 30° and 330°, a Trireme from 300 B.C. might only be able to sail courses from 80° to 280°.

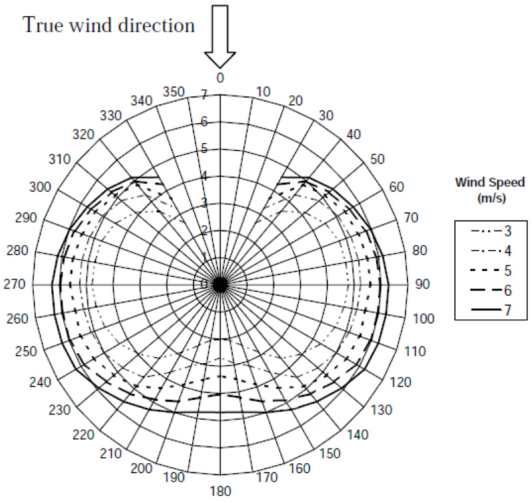


Figure 3.4: A polar diagram showing the maximal possible speed of a sailing yacht, depending on the bearing and wind speed. [Wil08]

3.9 Transform between Different Coordinate Systems with Homography Matrix

The homography is the mathematical relation between two images of the same planar surface, as described by Benhimane et al. [BM07]. It can be represented by a matrix, the so called homography matrix. This means that a homography matrix can be used to transform points from one image into the other. Every image has a two dimensional Coordinate System (COS) which allows us to specify every point of the image with an x and y value. If we transfer this point to the other image this can also be seen as transferring it from one COS to another COS. Figure 3.5 shows an example to illustrate this. In the center we see the planar surface with a highlighted point inside of the red area, at x_m/y_m. On the left and right we see two different images of that surface. Both images contain the highlighted point, but at different coordinates x_1/y_1 and x_2/y_2 in their own COS, which is pixels on the screen in this case. This is because they do not have the same COS. The homography matrix can now be used to transfer from one COS to the other. So we can transform the point at x_1/y_1 to x_2/y_2 by multiplying it with the homography matrix.
In our application the left phone would represent the Kinect camera and the right phone the HoloLens. We do not transform the points from the Kinect to the HoloLens, but all points are transformed to the marker COS, which is the one on the table.

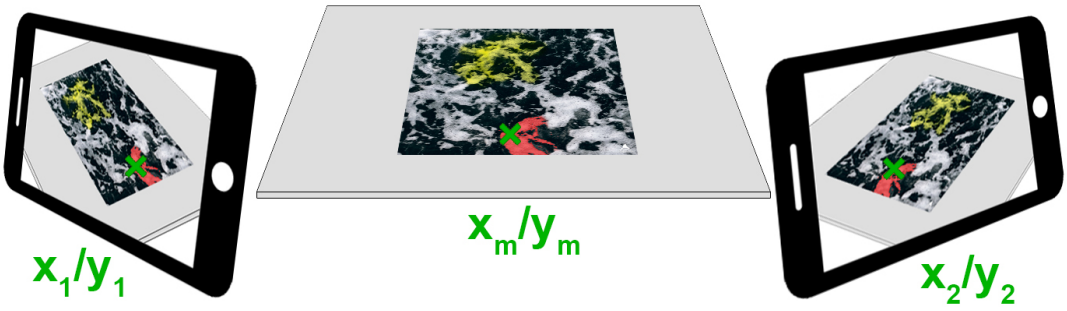


Figure 3.5: Two different images of the same surface. Transform the same point between the images with the homography matrix.

CHAPTER 4

Conceptual Design

This chapter contains the conceptual design of the *TrACTOr* tracking system and the *StARboard* application. Therefore the initial problem is briefly recapitulated, followed by the specification of requirements that results from it. Finally we give an introduction to the conceptual design of the tracker, the application and how they will be connected. Decisions about design specifications and technical approaches are reasoned and explained.
As already discussed in chapter 1 the aim of this project is to create a Tangible User Interface (TUI) which gives sailing students an easy alternative to drawings and books for learning the complex behavior of a sailing boat in different wind situations. This application, called *StARboard*, should provide an intuitive and easy to learn environment for studying various sail trims in relation to bearing, wind direction and speed.
This application is realized by using the *ACTO* interface, described in section 3.2.
In order to make the application easy to setup and install in various locations such as sailing schools or also private learning environments, we need to make the tracking independent from the surface the application runs on. The original ACTO tracking algorithm requires the use of a glass table as interaction surface, which is usually not available in such locations. Furthermore, we do not want to limit the *TrACTOr* tracking to the StARboard application, but make it available for any other ACTO applications that already exist or will exist in the future.

4.1 Requirements

The following requirements must be fulfilled in order to reach our research goals:

4.1.1 TrACTOr Requirements

Generic Implementation

The TrACTOr tracking software should not only work specifically for the StARboard application, but also for other applications that have already been developed with the ACTO interface, as well as for future implementations. It should serve as an alternative for the current tracking, which is limited to the use of a glass table.
In order to be an adequate substitution to the current tracking it is required that the changes that have to be made on the ACTOs need to be minimal. They should restrict the future development of ACTO modules as little as possible.

Rectangle Tracking

The ACTOs basic shape, if watched from above, is a square. Maybe some modules change the shape to a rectangle. So the tracking software needs to be able to detect ACTOs independent of the modules that are positioned on top of the ACTO, as long as they have a rectangular shape. Their position needs to be tracked in 3D space, so the x and y position on the surface as well as the height above the surface. Additionally the ACTOs rotation has to be tracked, but only around its vertical axis.

ACTO Identification

As the tracker should be able to track multiple ACTOs simultaneously there needs to be a way to identify each ACTO individually. This needs to be done without changing the shape of the ACTO itself. Furthermore, it is important that if the tracking of an ACTO is lost and found again it can be identified as the same ACTO. The same applies if it reappears on another place of the scene.

Independence from Tracking Surface

The tracking should be independent of the surface the ACTOs move on, as long as the surface is flat and even enough for the ACTOs to move. Especially the restriction to the glass table should be overcome.

Robust against Occlusion

As the users interact with the ACTOs it is likely that parts of the ACTOs or even the whole ACTOs are occluded by the users hands from time to time. The tracking should be robust against partial occlusion and even if an ACTO is occluded fully the tracking should be kept alive for a certain amount of time.

Robust against Different Lighting Situations

The system should be able to run in various locations with different lighting situations. Due to this reason it should be independent of the actual lighting situation and robust against changes of light intensity, color or direction.

Optimized for Moving Objects

A special focus during the TrACTOr design and implementation should be to optimize the tracking of moving objects, as the tracked objects will move around on the surface. With this special focus the accuracy of the resulting tracking should be optimized.

Real Time

As the tracker should be used for real time TUIs it is required that the tracking data (position, rotation, identity) of each ACTO can be streamed to the ACTO-Framework so it can be processed there and used by other applications, as in our case StARboard. Therefore it is required to have a highly performant tracking process and fast round trip times within the whole system.

Connection to ACTO-Framework

The already existing ACTO-Framework should be used for managing the ACTOs, so it is required to connect the TrACTOr-tracker to the ACTO-Framework and stream all the tracked ACTO data, as already stated. This requirement is also essential for allowing already existing ACTO applications to use the new TrACTOr tracking.

Synchronize Coordinate Systems

There is the need to synchronize the tracking COS and the one of the application that uses the tracking, in case visual augmentation is desired (for example for our StARboard application). So TrACTOr has to provide a way for COS synchronization.

Normalize Tracking Data

Another requirement is the normalization of the tracking data. While the ACTO rotation is automatically limited to a value between 0 and 360 degrees, the position values are not limited and can basically assume any value, dependent on the system setup. To overcome this issue it is required to normalize them. This means that a “playground” should be defined, whereas the point 0/0/0 is in its center and the border is either -1 or +1. Every tracking value will now be between -1 and +1, as long as the ACTO is located inside of the defined playground. In that way the tracked values become independent from the system setup and using them becomes much more comfortable for other systems.

4.1.2 StARboard Requirements

Real Time

In order to provide an appropriate and intuitive learning environment for sailing students it is required that the whole application runs in real time, so that modifications executed by the user instantly result in a behavioral change and appropriate visual and physical adaptions of the system. This means that when the user rotates an ACTO for example, the system should immediately recalculate its speed and movement direction and apply it to the ACTO. In the same way, when the user changes the sail trim the visual feedback and physical speed should change at once.
To achieve this requirement two specific needs have to be satisfied:

- **Receive tracking** - There needs to be a permanent connection from the TrACTOr system to the StARboard application to stream the real time tracking data (position, rotation, identity).

- **Send movement commands** - The whole physical simulation takes place in the StARboard application, so also the movement speed and direction is calculated here. This means that there needs to be another connection from StARboard to the ACTO-Framework, to transmit the movement commands.

Adjust Physical Parameters

The application needs to provide a way for the users, so they can manipulate all of the considered physical parameters, like the relative position between wind and ship, the wind speed and the sail trim. As the system should be easy to use and provide a hands-on experience the application should be implemented as a TUI. This means an ACTO module needs to be designed which provides the required interaction techniques.

Visual and Physical Feedback

Besides the appropriate input method the application also requires suitable output techniques to provide rich feedback to the users. Basically there are two different output techniques that should be implemented:

- **Physical output** - To create a TUI it is required that the objects of the systems are represented by physical interaction tokens. For the StARboard application these represent the ships and the wind source. While the wind source is static and can only be moved by the user, the ships need to automatically change their physical position during interaction sessions. So moving around on the interaction surface is the one type of feedback the application should provide.

- **Visual output** - The other one is visual feedback. To create an immersive application it is required to visually decorate the physical tokens with virtual content. So the real world scene should be augmented with virtual objects that

adapt to the users input. For example the sail trims should be visualized and change when the user manipulates the according parameter.

Multiple Users

In a sailing school setup usually multiple people learn together. The application should therefore allow multiple users to interact at the same time.

Multiple Ships

Furthermore the application should not only allow the concurrent interaction of multiple users, but also support the use of multiple ships. This allows the direct comparison of different trim constellations between multiple ships side by side.

Synchronize Coordinate Systems

A requirement that results from the necessary system architecture is the synchronization of the COSs between StARboard and TrACTOr. As these two systems run individually there needs to be a way to calibrate their COSs, so both of them have the same origin and scale. This is needed in order to display the visual feedback in the correct location and size.

4.2 Concept

In this chapter each of the given requirements is addressed with a specific solution strategy. Decisions in application design are reasoned and algorithmic approaches explained.

4.2.1 Architectural Design

Figure 4.1 shows the architecture of the whole system. It shows how the main components are connected to each other and what information they share. While these connections are discussed in this section, the components themselves are described in detail in the following sections of this chapter.

The ACTOs are the interfaces interaction tokens. They are positioned on the interaction surface and controlled by the *ACTO-Framework* which runs on an Android phone. The connection is based on RF and happens via an extra Arduino, as described in chapter 3.2.2.1, as most Android phones do not provide RF functionality.
For the ACTO-Framework to work properly it needs to know the exact position and rotation of each ACTO at any time. This task is fulfilled by the *TrACTOr* tracking application. It receives a depth image stream from the connected Kinect camera. TrACTOr creates position, rotation and identity information from this stream and sends it to the ACTO-Framework via a WiFi connection. To synchronize the tracking COS and normalize the tracked data, TrACTOr also tracks an image target which is used as origin

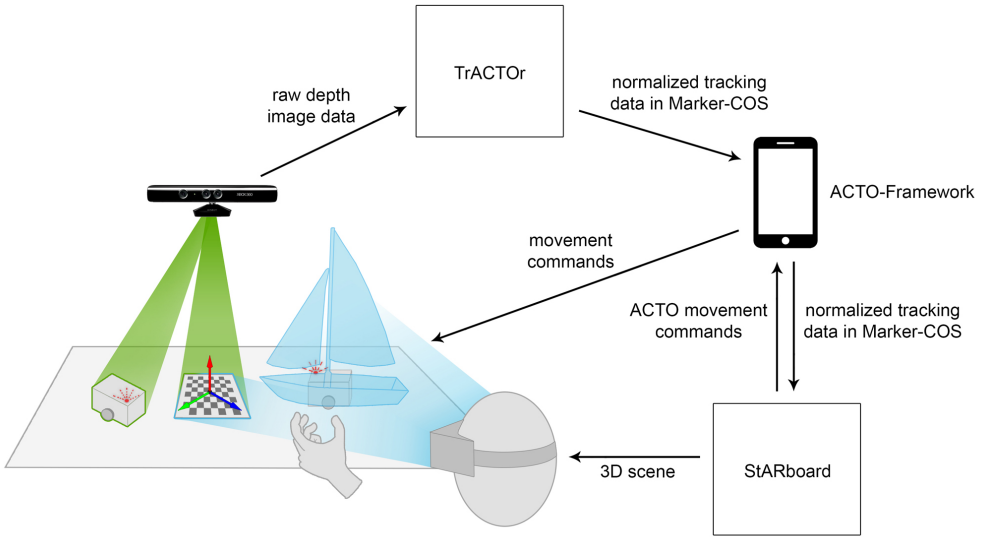


Figure 4.1: StARboard and TrACTOr system setup.

for the COS (read more about COS synchronization in chapter 4.2.2.3 and about tracking data normalization in chapter 4.2.2.4).
The only remaining component is the *StARboard* application. It consists of two components itself, the server and the client. The server is responsible for managing the virtual sailing scene and the game logic. Therefore it receives the tracking data from the ACTO-Framework via a WiFi connection. The whole 3D scene is then sent to the client which is responsible for rendering the virtual scene in the real world (AR). In order to be able to render the scene in the correct place the client also needs to visually track the image target to synchronize its COS with the tracking COS. The client application can run on different kinds of devices, as smartphones or HMDs. In our case we use the Microsoft HoloLens to run it.

4.2.2 TrACTOr Concept

TrACTOr is the component responsible for identifying and tracking the ACTOs while they move on the interaction surface. The generated tracking data should then be streamed to the ACTO-Framework where it is used to control the ACTO movement and is also passed along to the StARboard application for further use. The process required to achieve this task can be broken down into four key functions that TrACTOr has to provide:

- tracking
- identification
- synchronization
- normalization

4.2.2.1 Tracking

As described in chapter 2.3 there are various technologies that can be used for creating tracking solutions. The reasons already discussed in chapter 2.3.1.1 are crucial for our decision to use a depth camera for our tracking algorithm. The depth camera we use is the Microsoft Kinect camera, as explained in chapter 3.1. It is an easy to use depth camera and also provides an RGB or IR image along the depth image. Furthermore the Kinect SDK provides a good entry point for developing Kinect applications with C#.
The basic idea of the tracking process is shown in figure 4.2. The Kinect camera is positioned about 70 to 90 centimeters above the interaction surface and points downwards in an angle of about 90 degrees. This height distance is an ideal compromise for tracking accuracy and size of the tracked area. With this setup the following process should enable us to generate the tracking data:

Step 1 - Create Foreground Image

The aim of the first step is to create a grayscale image stream from the depth stream that only contains the objects of interest, which are the ACTOs (shown in figure 4.2a).

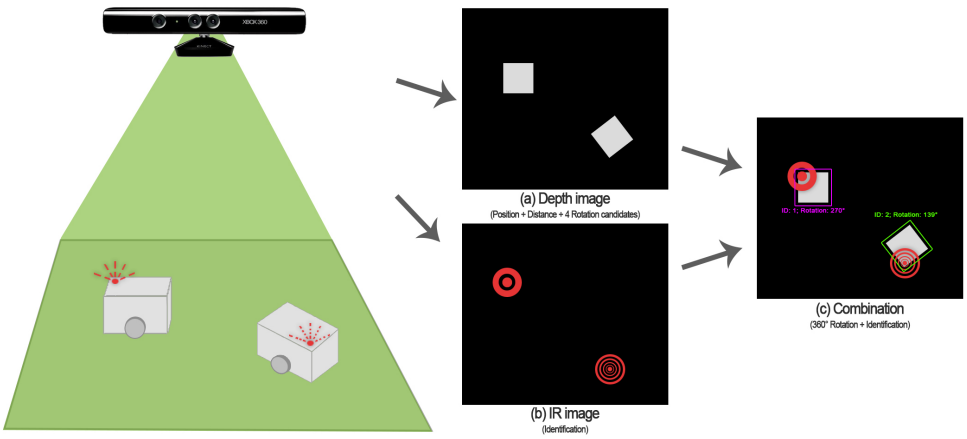


Figure 4.2: Concept for tracking and identification.

The basic idea to retrieve this so called *foreground image* is to first create a so called *background image* containing the empty scene. The whole process is shown in 4.3. Picture (a) shows the depth image of the empty scene, which functions as background image. To create a stable background image there should be an initialization process that accumulates the average image over a couple of frames. This makes the resulting background image more stable against camera flickering, compared to just using a random single frame.
Picture (b) shows the current depth image which already contains an ACTO prototype. To create the foreground image it is required to build the difference image between the

background image and the current depth image by subtracting them from each other. The resulting foreground image is shown in picture (c). It only contains the objects of interest, and can be processed in the next steps.
A major advantage of this strategy is that static objects that are located within the interaction space do not interfere with the tracking, as long as they are already there during the background initialization phase and do not move during the tracking process.

(a) Accumulated depth background image.

(b) Current depth image stream.

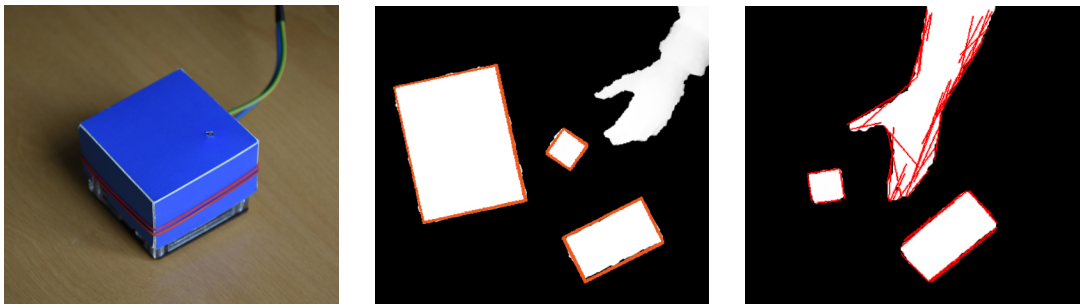
(c) Calculated depth foreground image stream.

Figure 4.3: Reducing depth stream to potentially interesting objects.

Step 2 - Two Level Tracking Process

In the next step the foreground image should be analyzed to retrieve ACTO candidates from it. An ACTO candidate is a detected object that might be an actual ACTO or some other object. This should be done in a two level process, whereas the aim of the first level is to detect rectangular objects in the foreground image. The second level should improve the tracking quality in case the tracked objects are partly occluded, for example by the user's hand. This second level can only be applied after an ACTO has already been identified successfully and can not be used for initial ACTO detection. This level becomes more important during step 5 of the tracking process (read more about it there 4.2.2.1)˙
For the first level it is required to detect all the rectangles in the foreground image. This is a complex task which can be supported by using a appropriate computer vision library. For this and other image processing tasks we use the *OpenCV* based library *Emgu CV* which is described in section 3.3. It can easily be integrated into the project and provides a wide range of image processing functions, e.g. shape detection.
To evaluate if this concept also works practically and if the size and shape of ACTOs is suitable for tracking with the Kinect, mockup tests have been performed. Therefore an empty ACTO case was used and positioned underneath the Kinect (see figure 4.4a). A single frame of the depth image stream, already with background removal performed, was used to try out the EmguCV shape detection algorithm. The result was promising, the ACTO and also other rectangular objects could be detected, as shown in figure 4.4a. Also the line detection algorithm was tested and different configurations were tried out

to figure out if it would be suitable for our needs. The result can be seen in figure 4.4c where an ACTO, another rectangular object and an arm of a user can be seen. If the information of both methods is combined the reliability and stability of the tracking can be improved.



(a) ACTO prototype for mockup tracking tests.

(b) Detection of rectangles in foreground depth image.

(c) Detection of lines in foreground depth image.

Figure 4.4: Mockup tests for image processing algorithms used for tracking.

As shown in the images of the mockup tests in figure 4.4b the square detection not only detects ACTOs, but also other rectangular objects. All of those are candidates for potential ACTOs and the next step is to distinguish between ACTOs and other objects.

Step 3 - Filter Actual ACTOs from ACTO Candidates

The aim of this step is to filter out the actual ACTOs from all the ACTO candidates found in the previous step. To achieve this task different solutions have been discussed. For example the ACTO size and shape could be used, as well as the color or similar parameters that could be detected with the depth, RGB or IR camera. But all of those would restrict the development of future ACTO modules, either in size, shape or visual occurrence. To reduce the required adaptions to a minimum we decided to just include an IR LED that is positioned on top of the ACTO. This LED serves several purposes. It is not only used in this step but also in step 4 and also for identification as described in section 4.2.2.2.
To decide if an ACTO candidate is an actual ACTO or not the IR image's region of the detected rectangle is observed. The whole area within the rectangle is searched by using according image processing algorithms to detect a bright spot which is created by the flashing IR LED. The images in figure 4.5 were taken during mockup tests. They show the IR image of an ACTO candidate with the IR LED on and off.
Figure 4.4a shows the ACTO prototype with the IR LED positioned on top of it.

Step 4 - Choose from Rotation Candidates

It is not just the actual position of the ACTO which should be tracked, but also its rotation. This is the goal in this step.

(a) ACTO candidate with IR LED on. (b) ACTO candidate with IR LED off.

Figure 4.5: Mockup tests detecting IR LEDs in the IR image.

An ACTO always has quadratic shape. The detected rectangle is aligned with the ACTOs edges, but only from the depth image it is not possible to tell what direction the ACTO is actually facing. As we already know the orientation of the rectangle, there are basically four possible directions the ACTO could be facing. We call those rotation candidates.
To choose which of those candidates is the right one we again use the IR LED. Now the LED's actual location on the ACTO is very important. It always has to be positioned close to one of the four corners so it is clear which corner is the closest, if seen from above. In our application the LED should always be placed in the left front corner of the ACTO. This allows us to clearly choose which of the four rotation candidates is the correct one.
As an example, the ACTO prototype in the mockup test in figure 4.5a faces to the bottom right of the image.
Mockup tests have shown that the LED must not be positioned too close to the ACTOs edges, because in that case it could cause interferences with the rectangle detection. This is due to the fact that the IR LED disturbs the Kinect's depth tracking in a small area, as it is also based on IR light. If this disturbance is too close to an ACTO's edge, the resulting depth image might have disturbing artifacts which could cause errors in rectangle detection.

Step 5 - Compare with Tracking Result from Previous Frames

The aim of the final step is to improve the tracking quality by using information over time. This means that if an ACTO is present in a certain location in one frame it is very likely that it will still be there (or close) in the next frame. This information is very useful for two reasons:
First, it can be used to overcome missing tracking results in single frames. So if an ACTO is detected in a certain frame but cannot be found in the following frame, the tracker should still expect it to be at the same position and send tracking information for it for several frames. It is much more likely that the tracking is lost for a couple of frames than that the ACTO disappeared. This issue is especially important in case of partial or complete occlusion of an ACTO by a users hand. TrACTOr should still estimate the position even if it can not *see* the ACTO anymore. This is also the basic assumption, why a two level tracking process (discussed in step 2, 4.2.2.1) should be implemented. The

tracker knows where to expect the ACTO to be and with this information it is enough to only detect one or two of the edges, instead of the whole rectangle.
Secondly, it can help to make the tracking more stable in terms of jitter reduction. As the depth image's resolution, as well as the tracked ACTOs are quite small the tracked position and rotation can vary greatly from one frame to the other, due to measuring inaccuracy. This will result in a jitter of the tracking data. To reduce this artifacts the position, as well as the rotation, should be averaged over several frames. Therefore a predicted value, that is based on the tracking of previous frames, should be calculated and combined with the actually measured value of the current frame. The resulting value should be chosen as current tracking. This can be done by using a Kalman filter (see chapter 3.5) which is especially applicable for moving objects.

4.2.2.2 Identification

As the tracker has to be able to track multiple ACTOs at the same time, some sort of identification is needed to distinguish between the different ACTOs. The ACTOs themselves should need as little modification as possible for this, to avoid limiting the future development of ACTO modules. So for example it is not possible to give every ACTO a different color or shape. A couple of solutions have been considered for solving this problem. One of them was for example to put different colored stickers on top of the ACTO which can be detected by the RGB camera. Another one suggested to have a certain amount of LEDs positioned somewhere on the ACTO. Depending on the amount of lights that are on the identity could be determined. However, the solution we chose is more pragmatic and enables us to use an unlimited amount of ACTOs at the same time. As already discussed in the previous section each ACTO should be equipped with a single IR LED. This LED can also be used for identity detection. Therefore each ACTO receives a numerical id. This id is visually transmitted from the ACTO to the tracker. This can basically be done by blinking the binary code of the id in an endless loop, whereas light on stands for *1* and light off for *0*. TrACTOr is then able to decode the bit string to the id again and differentiate between different ACTOs (see figure 4.2b).
This task sounds trivial, but for achieving it, many factors need to be considered:
The most challenging problem that has to be solved is that the data can only be transferred asynchronously because there is no way of synchronizing the ACTO, which is the sender, and the Kinect camera, which is the receiver. Basically this is a common problem in electronics and information technology but in our case it becomes a bit more complex as we use a camera as measurement device. This means that the current signal status can not be measured continuously, but only at discrete points in time, as we can only measure every frame. This issue leads to two problems: The one is that the receiver does not know when a single bit starts and ends, the other one is that it is also unknown when a data sequence starts and ends.
The first problem happens because the LED status is not discrete. It can not only be *1* and *0* (On and Off), but also every value in between. So if the sampling rate would be the same as the send rate, in the worst case the first measurement would happen exactly while the LED changes status from *0* to *1*. The result of this measurement is not

predictable, so it could be either *0* or *1*. In that case the next measurement would again happen exactly while the LED is turning down from *1* to *0*, which again could result in both measurement results. So a whole bit could be missed as shown in figure 4.6.

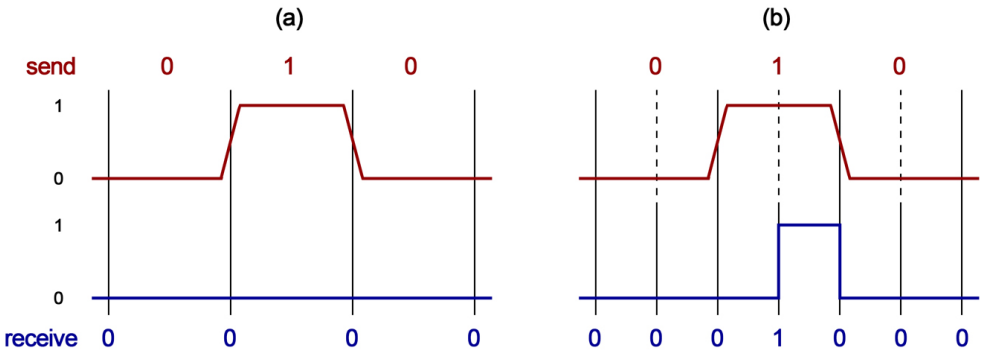


Figure 4.6: Worst case situation for sampling where sampling rate is equal to the send rate (a) and sampling rate is twice the send rate (b).

This problem was already discussed by Nyquist [NYQ28], who stated that the sampling rate has to be at least twice the send rate. As the sampling rate is fixed 30fps due to the Kinect hardware specifications the send rate has to be adapted accordingly. This means that the maximal possible send rate is 15 bits per second.

To address the second problem, that it is unknown when a data sequence starts and ends, we developed a stable method that allows us to perform reliable id transmission. The id is not repeated permanently, but with a break in between. Additionally a *burst bit* (1) is placed before each id sequence to mark its start and another *end bit* (1) follows afterwords. Figure 4.7 shows an example where a four bit id is sent at a bit rate of 15 bits per second. Before the actual data sequence a burst bit is sent to notify the receiver that data bits will follow afterwords. As soon as the receiver, who reads with 30 frames per second, receives the first *1* he ignores every second bit and uses the other ones as data bits. After the four bit data sequence an end bit is sent. If this is missing the sequence is invalid. After this a series of at least eight *0* bits should follow to make sure the receiver can distinguish between sequence and break. If we would not use the end bit and the receiver would start reading in the middle of the sequence a wrong id could be read. Figure 4.8 demonstrates this problem. The receiver starts reading in the middle of the sequence, misconstrues one of the data bits as the burst bit and reads the id 12 instead of 3. If the end bit would be used the receiver would expect it, but it does not appear. So the sequence would be declared as invalid and ignored.

If we would use this method exactly as it was explained here another issue would arise. Due to the breaks in between the sequences most of the bits would be *0*. So most of the time the IR LED would be off. This is a problem, as it is also used for rotation detection. To improve this we simply invert the whole stream. So *0* becomes *1* and vice versa. Now the LED should be on most of the time.

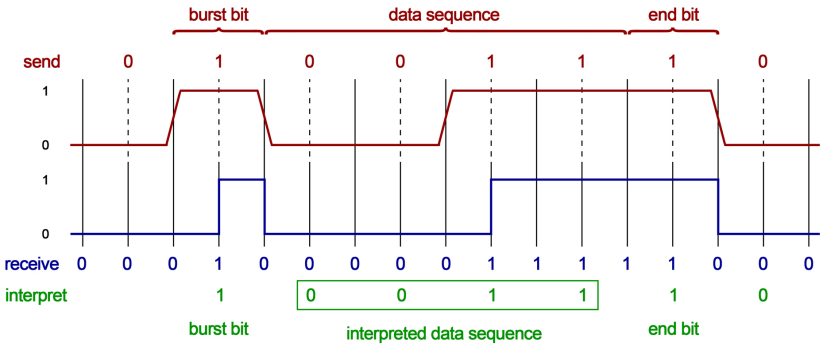


Figure 4.7: Sampling and interpreting of a whole id sequence, starting with a *burst bit* followed by the 4 bit data sequence and ends with an *end bit*. The sampling rate is twice the send rate and shows the worst case situation, as discussed in figure 4.6.

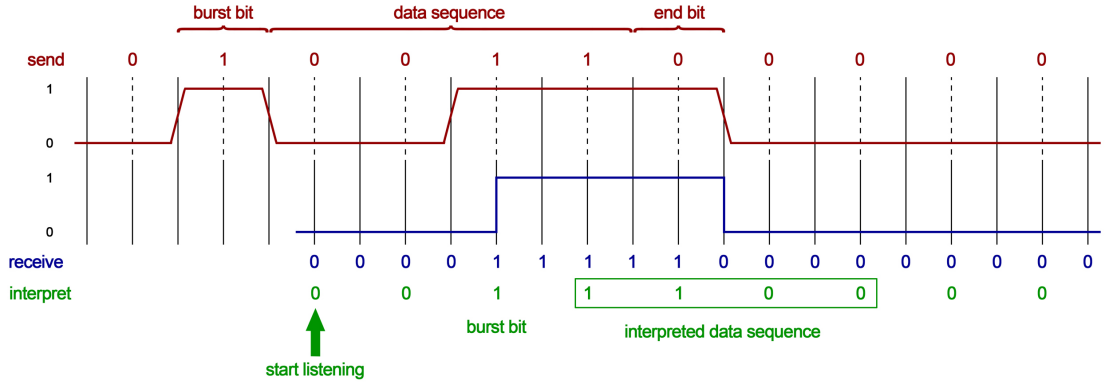


Figure 4.8: The possible error if no end bit would be used. The receiver starts to listen in the middle of the sequence and due to the lack of the end bit does not recognize the error. The wrong id is interpreted.

Another issue that needs to be considered is that the ACTOs are also not synchronized with each other. This means that their bit rates will be shifted to each others. So the tracker needs to handle each of them individually, which should not be a problem with the method presented above.
To evaluate the reliability of this method mockup tests have been performed. The ACTO prototype shown in figure 4.4a was connected to an Arduino board which controlled the IR LED. A Kinect camera was used to create the IR video stream of the scene. This setup was used to test various bit sequences with a promising result. The id detection worked properly most of the time, but sometimes the detected id was not correct. To overcome this problem the id detection runs until the same id is detected twice. As soon as the same id is detected twice it is assigned to the ACTO.

4.2.2.3 Synchronization

When the tracker is initialized it will create a COS with its origin (0/0) in the top left corner of the video stream. This means that the user can not really control which area should be used as interaction space. Every time the camera is moved also the interaction space would shift. But the biggest problem appears when TrACTOr should work together with an AR system, because it will most likely have different COSs and the virtual objects would be placed on the wrong place. In order to make this possible both COSs have to be synchronized.
We also call this process calibration. In order to make it fast and easy we use an image tracker that is detected with natural feature tracking (read more about natural feature tracking in chapter 3.4.2). So almost any kind of image can be used for this process. The actual image tracking is done with the OpenCV library (see chapter 3.3). As soon as the image is detected the origin of the tracking COS is moved to the center of the image and the tracking data that is generated will also be based on this new origin.

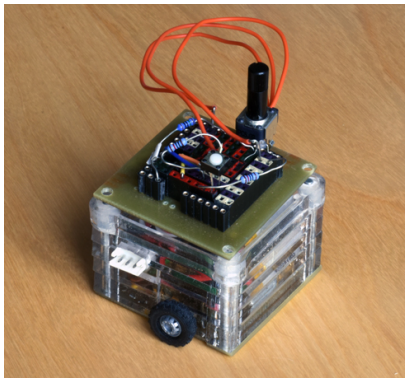
4.2.2.4 Normalization

The COS normalization can also be seen as part of the calibration process. It is not enough to only synchronize both COSs so they have the same origin, because the distances in both COSs will still be different. This means that one distance unit in the on COS is not equal to one distance unit in the other COS. Also the camera setup can still be a problem. If the camera is not positioned in a perfect 90 degree angle the image will be distorted. So one distance unit in x direction could have a different distance in the real world as one distance unit in the y direction.
To solve this issue we use the same visual marker as for synchronization. For this task not the position, but the size of the marker is used. The distance from the origin (marker center) to the markers edge is defined to be 1. This new distance system is then applied to all the tracking values. Also the ACTO rotation will be based on the rotation of the image marker, whereas a rotation of 0 means that the ACTO has the same rotation as the image marker itself. From now on the resulting tracking values are not dependent on the camera setup and image resolution anymore. They are based on the position, size and rotation of the image marker which enables us to use them across multiple systems. As soon as this step is done the calibration process is finished and the tracking data can be called *normalized*.
To apply the calibration to the original tracking values, a homography matrix should be used (see chapter 3.9).

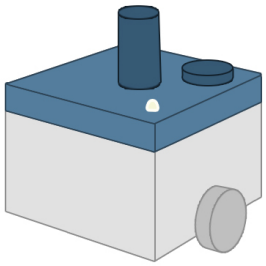
4.2.3 Concept for ACTO Adaptions

The *ACTOs* are the tangible tokens for the StARboard interface (see chapter 3.2). Their basic implementation is very generic, which allows us to use them and develop the functionality we need on top of it. To make them usable with the TrACTOr tracking we need to include the IR LED, described in the previous sections, and use it to transfer the

ACTOs id. Therefore also the core ACTO code needs to be extended with the according algorithms to transmit the id information via the LED.
Additionally, an ACTO module should be created that provides the additional functionality required for the StARboard application. This is basically a stepless rotary encoder and a simple button. The aim of this module is to give the user direct control of the physical parameters used in StARboard, without being to complex. The conceptual design of this module is shown in figure 4.9b
The prototype used to test this functions can be seen in figure 4.9a.



(a) StARboard ACTO module prototype.



(b) StARboard ACTO module design.

Figure 4.9: Prototyping of the ACTO module required for StARboard. The module is equipped with an IR LED, a rotary encoder and a button.

4.2.4 Concept for ACTO-Framework Adaptions

The ACTO-Framework is the server part of the ACTO interface, as described in chapter 3.2. The standard implementation by Vonach et al. already provides most of the functionality needed for the StARboard application. The two main changes that are required are the ability to stream tracking data via WiFi in real time and an individual GUI which provides the functionality needed for managing the ACTOs.
The tracking data which is generated by the TrACTOr-tracker is required by the ACTO-Framework as well as the StARboard application server. Basically there are two possible options to solve this. The tracker could either send the data to both of them or it is just sent to the one and is then forwarded to the other one from there. As shown in the system architecture in figure 4.1 we have decided to use the second solution. There is a stream transferring the data from TrACTOr to the ACTO-Framework. The ACTO-Framework then uses it to manage the ACTOs and forwards it to the StARboard server in real time. The reason to implement it like this is because with this solution it is possible to replace the tracker without having to change anything else in the system. To achieve this a WiFi connection for streaming the data to StARboard is required.
The implemented GUI provides two screens. The one screen is the management interface for controlling the system. The network connections should be controlled, as well as the

USB connection to the other Arduino board, used for communication with the ACTOs. The second screen shows the currently tracked situation. The playground is shown with the different ACTOs on their current position. Also the identity of each ACTO should be visible. This screen should make testing easier and briefly tell the user whether the tracking works sufficiently or not.

4.2.5 StARboard Concept

The *StARboard* component is the actual application. It uses and controls the ACTO interface to create a tangible application. Therefore it has to take control of the whole scene and provide the interaction mechanisms. It basically serves two purposes. The one is to handle the current situation on the interaction surface and send movement commands to the ACTO-Framework, based on the current constellation of the interaction tokens in the scene. The other one is to provide visual feedback to the user, to display the current state of the most important physical parameters, as well as display augmentations for the interaction tokens.

4.2.5.1 Client Server Concept

A requirement of the application is, that it can be used by multiple users simultaneously. For this reason the StARboard application should be created with a Client Server architecture. The basic concept for this can be seen in figure 4.10.

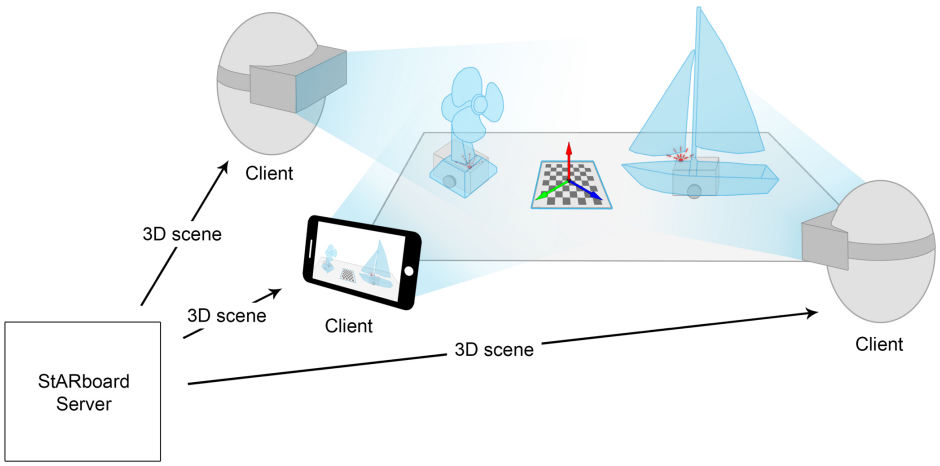


Figure 4.10: StARboard Client Server concept.

There is a central server that controls the whole scene. It is aware of the current location and state of each object within the scene. Based on this knowledge it applies the physical model that lies behind the sailing application to each object and manipulates

the according parameters, e.g. position and movement speed (read more about the game logic in section 4.2.5.3). The whole scene is permanently streamed to each of the clients via WiFi connections. The clients themselves are conceptualized as *thin clients*, so their only job is to display the whole scene in the right place. They mainly do rendering and tracking tasks (see section 4.2.5.5). This means that most of the processing effort is done on the server side, which is very important in our situation, as the devices running the clients have significantly less computing power as the server. The client application can run on different devices as smartphones or HMDs.

4.2.5.2 Unity

The whole StARboard application is developed in Unity (see chapter 3.6) as Unity is not only a very powerful development tool for 3D applications, but also has the big advantage that the deployment to different platforms is easy and straight forward. Another great benefit of Unity is that it provides many functions that are helpful for the StARboard implementation itself, such as sharing of the scene between the server and the clients. Those tasks can be done without much effort, as Unity is also designed for multiplayer applications.

4.2.5.3 Scene Logic

Basically the scene logic running on the server should be split up in two separate areas. The one is a central *Game Controller*, the other one the *Ship Objects*.
The Game Controller is a central unit that receives the tracking information and the users input commands, like button clicks. It is responsible for monitoring all the objects in the scene, applying the received tracking information and passing the user input to the according objects.
Each object itself is responsible for maintaining their current physical state. This means the current sail trim, the current bearing to the wind and the current sailing speed. With the received input parameters these values need to be adapted accordingly. Based on this information, changes in speed have to be calculated and transmitted to the Game Controller. There they can be passed to the ACTO-Framework, to actually affect the physical ACTO tokens. In order to be able to calculate the current speed based on the bearing and the sail trim, each ship object needs to know the polar diagram (read more about polar diagrams in chapter 3.8) data which is related to the respective boat type.

4.2.5.4 Visual Feedback

StARboard is not only responsible for processing the scene logic and passing movement commands to the ACTO-Framework. Another main functionality is visual representation of the current system state and additional information needed for teaching purposes. As reasoned in chapter 2.2.3 we should use 3D augmentations for our application.
There are several parameters that should be visible to the user, according to the state of wind source and ships and each of them need a visual representation themselves:

Wind Source

Basically the wind source can be represented by the 3D model of a fan, since this single metaphor is self-explanatory. The current wind direction is shown by the rotation of the 3D model, so the fan simply faces the same direction as the wind should go (just as it is with a real fan). The actual wind speed can be visualized with the rotating fan blades. The faster they move the stronger is the wind. If they stand still there is no wind at all.

Ships

A more complex visualization is required for the ships. While each different boat type should have a different 3D model for the ship itself, also the sails should be displayed according to the currently defined sail trim. The bearing is shown with the actual position and rotation of the ships 3D model.
To give the user more information and a better understanding for the boats behavior it should be possible to display the polar diagram at the ships current position. The diagram is different for each boat type and should be rotated according to the current wind direction. This should make it easier for the user to understand the concept of polar diagrams and provide insight into the relation between bearing and speed.
Another level of information should be given with the visualization of the true and the apparent wind. This is a very important concept in sailing theory and usually displayed as direction vectors. There should be one vector showing the true wind, one showing the wind from boat motion and a third one the resulting apparent wind. This visualization should be displayed alternatively to the polar diagram and also be rotated according to the current wind direction.

4.2.5.5 Augmented Reality

In order to be able to display the augmentations on the right position the COSs of the StARboard client applications need to be synchronized with the one used for tracking. There are different solutions how those two systems could be synchronized, like manual synchronization or marker tracking. We decided to use an automated synchronization process based on image markers, because it does not require additional effort and allows quick setup of the whole system. Also image markers are more esthetic than fiducial markers. In our case a marine map could be used for example.
As described in chapter 4.2.2.3 the tracker uses an image target and natural feature tracking (read more about natural feature tracking in chapter 3.4.2) to place the origin of the scene to the center of the image. To also synchronize the StARboard clients they also need to track this image and position the whole scene based on it. The library that can be used for the image tracking task is Vuforia [vuf]. It is free to use for small application and provides fast and robust tracking for mobile application. As an alternative solution Metaio [met] was considered, but they do not provide their service anymore and therefore had to be discarded. With this tracking step done the augmentations can be correctly displayed on top of the ACTO robots.

4.2.5.6 Network Communication

The StARboard server and clients need to communicate to various other players in the system. Therefore, according interaction mechanisms are required.

Server

As shown in figure 4.1 the StARboard server needs to provide three different information channels. All of them should be based on WiFi, as it is the most widespread technology for wireless data transmission. It also provides high data rates and is the only wireless technology that is supported by all of the devices that should run the client, like smartphones and the HoloLens.
The first channel is the tracking data input stream received from the ACTO-Framework. The second one is another connection to the ACTO-Framework. It is used for sending movement commands for the ACTOs. The third channel is between the StARboard server and client. It is an unidirectional connection that streams the whole virtual scene from the server to the client.

Client

The addressed connection between the server and the client is the only connection the client needs. It does not need to send any kind of data. It connects to the server and receives a stream with the current scene.

CHAPTER 5

Implementation

In this chapter we describe the implementation of the entire system. At first we give an overview about the system architecture and introduce the main system components briefly. Then we explain the networking situation, how the components communicate with each other and what sort of data we actually transfer between them. Afterwards we describe how the system calibration works and why it is required.
In the next sections we give a detailed introduction in the implementation of the main components *TrACTOr*-tracker, the *ACTO* robots software and hardware, the *ACTO-Framework* and the *StARboard* server and client. We display the software architecture for each of them and point out problems and challenges we faced during the implementation and experience that we collected. We also state differences to the concept and why we decided to implement a few things differently. For delicate problems we display brief extracts of the source code to give a better understanding of some implementation details. We also included a couple of screenshots and photographs of important details of the final system.

5.1 System Architecture

The architecture of the whole system we implemented can be seen in figure 5.1. It shows how the main components are connected to each other and gives a first idea of the information flow. The main components are explained in detail in the following section 5.1.1.

5.1.1 Main Components

As shown in figure 5.1 the system consists of four main components:

- **TrACTOr** - the Kinect based tracking system, which tracks the exact position, orientation and identity of the ACTO robots. Its main components are the *GUI*

for user interaction, the *Tracking Core* which contains the actual tracking process and the *WiFi Transmitter* for transferring the tracking data. TrACTOr runs on a desktop computer.

- **ACTO-Framework** - responsible for controlling the ACTO movement and transmitting interaction commands to and from the ACTOs. The ACTOs are controlled in the *ACTO Control* component, the communication to the other parts of the system is handled in the *Communication* component. The *GUI* component provides control and feedback functions to the user. It runs on an Android based smartphone.
- **ACTO Robots** - the interaction tokens themselves.
- **USB Arduino Server** - an Arduino based interface for the communication between the ACTO-Framework and the ACTO Robots.
- **StARboard Server and Client** - the AR based learning application which uses the ACTOs as interaction interfaces. The server major components are the *ACTO Management* component to manage the ACTOs, the *ACTO-Framework Connection* for communication with the ACTO-Framework, the *Game Logic* component to control the virtual objects and learning system and the *Maneuver Management* to run the maneuver demonstrations. While the StARboard Server runs on a desktop computer, the Clients run on the Microsoft HoloLens.

The implementation details of all four components will be discussed more specifically in the following sections of this chapter.

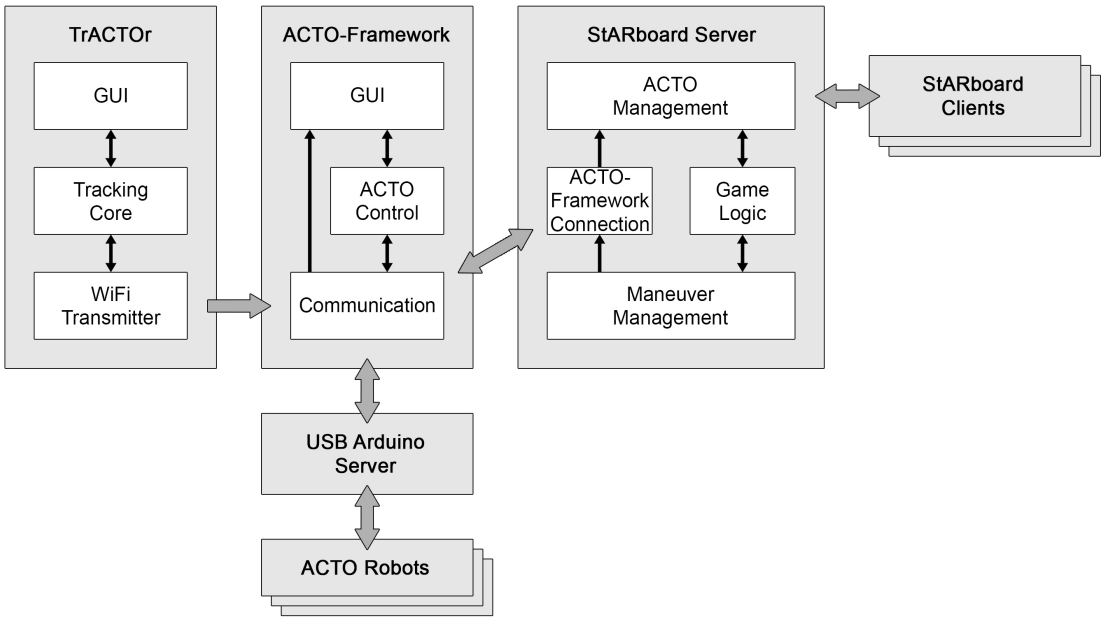


Figure 5.1: Architecture of the whole system.

5.1.2 Network and Communication

As the system consists of multiple programs which also run on different machines it was necessary to develop proper ways to allow real time communication between the individual components. Figure 5.2 gives an overview about the communication channels between the units.

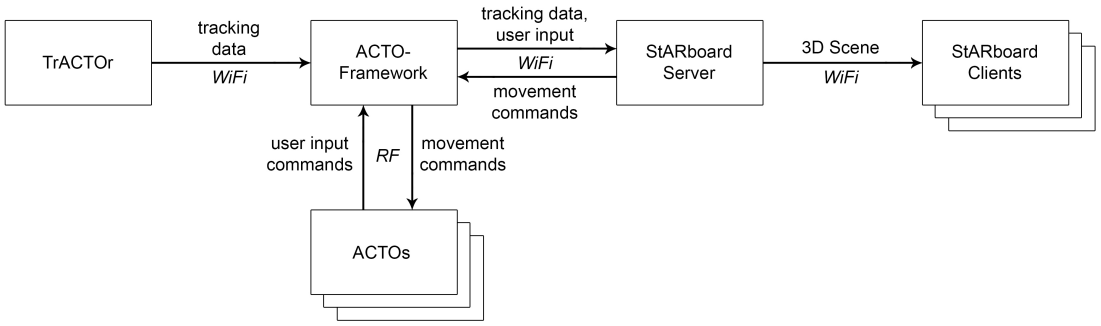
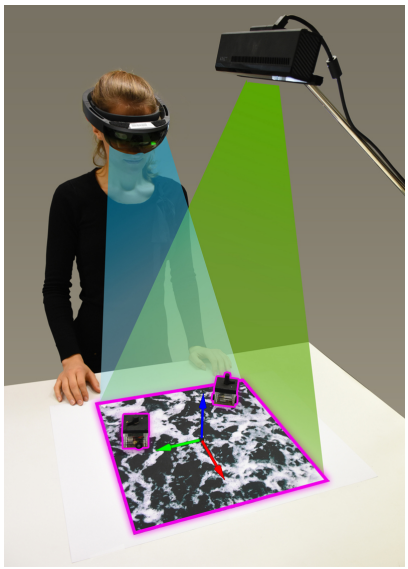


Figure 5.2: System communication structure.

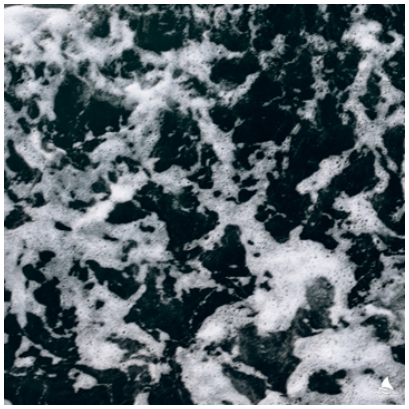
Most of the connections run via Transmission Control Protocol/Internet Protocol (TCP/IP) over an Ethernet or wireless network connection. This way the TrACTOr-tracker is connected to the ACTO-Framework. The current tracking data (position, orientation, identity) of each identified ACTO is transmitted for every frame of the Kinect camera.
The ACTO-Framework has another permanent TCP/IP connection to the StARboard server application. This channel is used to forward the tracking information received from TrACTOr, but also in the other direction to transmit movement commands towards the ACTOs. These movement commands are generated in the StARboard server, based on the current situation in the 3D scene and the according game logic.
They are processed by the ACTO-Framework into basic motor control commands, which are then delivered to the actual ACTO units. The communication medium this time is RF. As the ACTO-Framework runs on a standard Android phone that does not support RF connections, an extra Arduino unit is connected to the phone via USB. This Arduino is equipped with a RF transmission module, that allows communication between the phone and the ACTOs. The RF connection is also used to transmit the user input commands from the ACTO to the ACTO-Framework, which are created when the user clicks the button on the ACTO or rotates the potentiometer.
These user input commands are then also forwarded from the ACTO-Framework to the StARboard server, where they are processed.
The whole 3D scene is sent via wireless network connections from the StARboard server to the StARboard clients.

5.1.3 System Calibration

As already described in chapter 4.2.1 a calibration process is required before the system can be used. During this calibration process the TrACTOr's tracking COS and the COS of StARboard are synchronized. We achieve this synchronization process with a visual marker that is detected by both systems, as shown in figure 5.3a. This allows us to transfer the 3D coordinates of both COS into one shared COS.
The visual marker we use is a photo of ocean waves, seen from above (figure 5.3b). In order to make the visual tracking fast and reliable we choose an image with strong contrast and irregular patterns, so many visual feature points can be detected by the software. We printed the marker with the dimensions of 39 by 39 centimeters onto a sheet of paper, whereas we had to make sure that we use matt paper, as reflections could cause problems for the tracker (read more about reflections in chapter 5.3.1.3).



(a) COS synchronization.



(b) The visual tracking marker.

Figure 5.3: For synchronizing multiple COSs a visual marker is used, which is detected by both devices.

The technical implementation of the tracking process is described in more detail in the following chapters 5.2.3.2 and 5.5.3.1.

5.2 TrACTOr Implementation

One major component of the entire system is the TrACTOr tracking system. It was developed in the programming language *C#* and runs on a desktop computer.
The purpose of TrACTOr is to track ACTO robots within a certain interaction area. As described in chapter 4.1.1, three different parameters are tracked: the x, y and z position

in relation to the tracking marker, the orientation along the camera axis and the ACTO's identity.
The retrieved tracking data is normalized with the tracking marker's size and orientation, in order to be able to properly synchronize 3D position data across the entire system. The normalized tracking data is then forwarded to the other components of the system. There they are used as input data for further processing steps.
As described in earlier chapters the depth based TrACTOr tracking is an alternative tracking solution to the already existing optical tracking, which requires the usage of a glass table. It can also be used for other ACTO applications, if the ACTO modules are equipped with the flashing IR LED, which is used for identification.

5.2.1 Tracking System Setup

The Kinect camera is mounted approximately 70 centimeters above the center of the interaction surface, which is defined by the tracking marker. It faces down towards the table with an angle of about 90 degrees. The TrACTOr software is robust to slight changes of these values, especially the mounting height. But this setup will provide the best tracking accuracy in our experience. The tracking marker is positioned directly underneath the Kinect on a flat surface, in our case a table. We used duct tape to attach it to the table, so it can not move during tracking sessions. The whole setup is shown in figure 5.4.

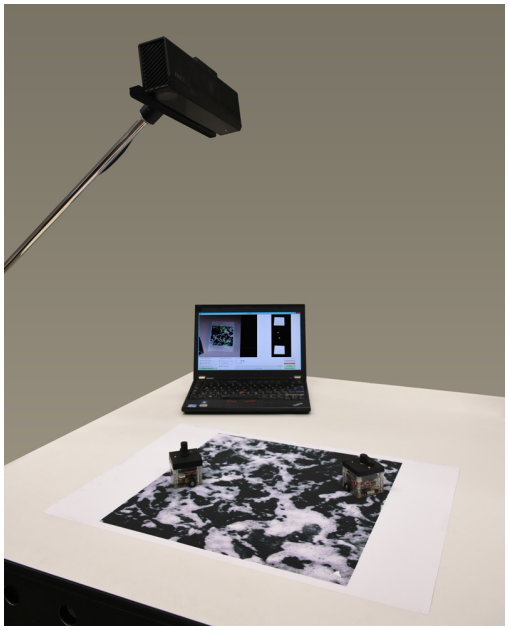


Figure 5.4: TrACTOr system setup.

For the tracking process we use the depth image stream, provided by the Kinect camera, as well as the infrared image stream. The exact usage of both streams is explained in

chapter 5.2.3.
In the first development phase we used a Kinect 1 for receiving the depth and infrared image. Because of higher accuracy we now use the Kinect 2 instead. The TrACTOr software is capable of handling either of them.

5.2.2 TrACTOr Architecture

The TrACTOr software itself can be split in three functional components: the *GUI*, the *Tracking Core* and the *WiFi Transmitter* (see figure 5.5). The GUI component contains the whole user interface, which allows the user to control and monitor the current tracking situation. The GUI is broken down in more detail in section 5.2.4. The ACTO tracking process itself is done in the Tracking Core component, which is the main unit of the program. The input image streams are handled in the *Kinect Controller*. They are passed to the *Image Processor*, where they are processed and converted to the formats we need. The prepared streams are forwarded to the *Calibration Service*, where the image marker is detected and the COS is calibrated. Also the *ACTO Manager* uses the streams to perform the actual tracking and identification tasks with support of the *Square Finder* service. It also manages the already known ACTOs and provides them to the *WiFi Transmitter*. The *WiFi Transmitter* is the third main component of TrACTOr. Its task is to reliably transmit the detected ACTOs with position, rotation and identity to the ACTO-Framework.
This whole procedure is explained in the following chapter 5.2.3.

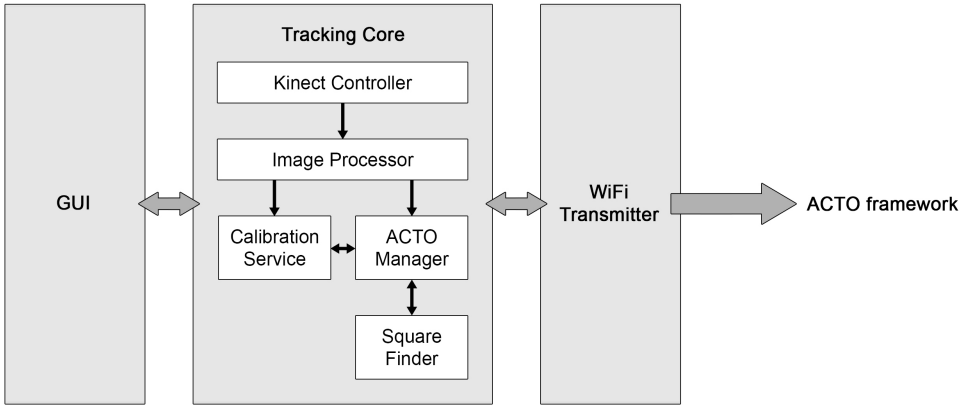


Figure 5.5: TrACTOr software architecture.

5.2.3 Tracking Process

In this section the technical implementation of the entire tracking process is explained in detail. It starts with depth and IR image streams and ends with the transmission of normalized tracking data.

5.2.3.1 Tracking and ID Detection

The first important step is the actual tracking of the ACTOs, based on the input video streams. Three pieces of information need to be acquired, the position, the orientation and the identity. We mainly use the Kinect's depth stream for the position recognition. For the identity recognition we use the infrared image and for the orientation both streams are used in combination. Figure 5.6 gives an overview of the process explained in this section.

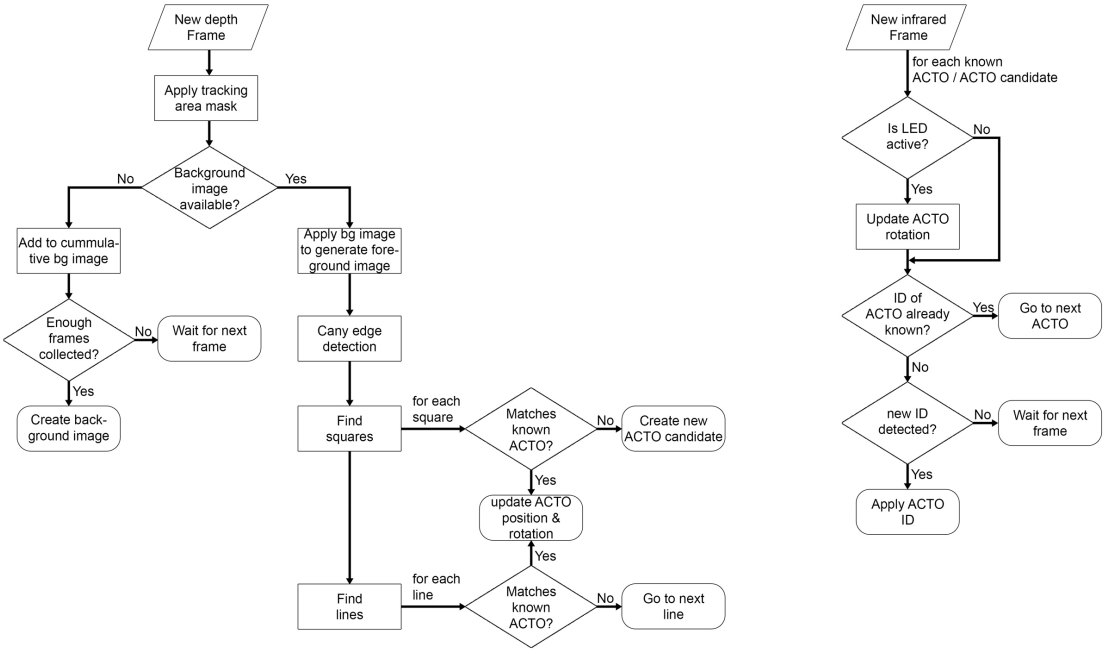
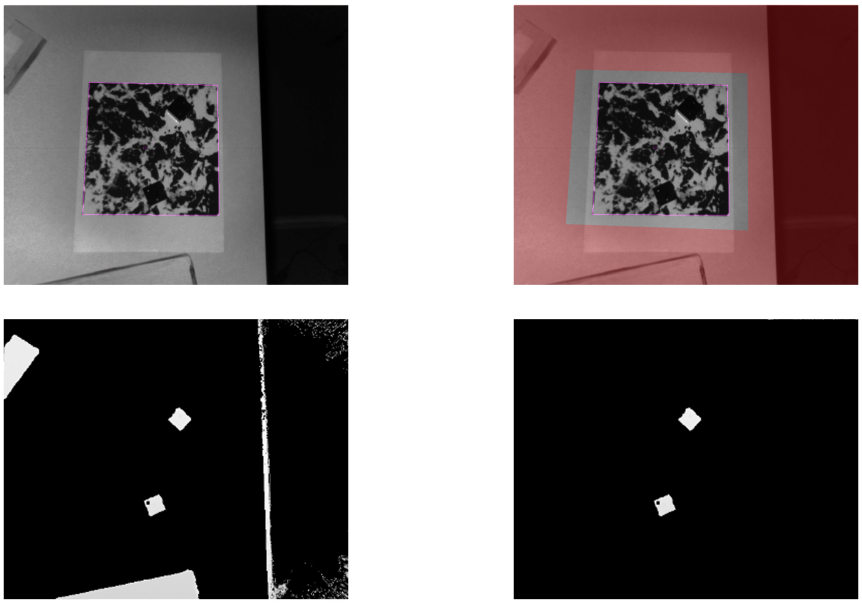


Figure 5.6: Flowchart of processing incoming depth and infrared frames.

Image Streams and Tracking Area

Both of these streams are received from the Kinect sensor, they can be streamed and used simultaneously. Especially the Kinect 2 has a very wide Field Of View (FOV) of 70 by 60 degrees. So the captured images show a lot of information which is not interesting for the tracking process, as seen in figure 5.7a. In order to drastically reduce the processing overhead, we implemented an option to define an area of interest, the so called *tracking area*. Everything outside of this area is masked and ignored for all of the following processing steps, as shown in figure 5.7b.
For tasks like masking of areas in an image or other image manipulation and conversion steps we use the computer vision library *Emgu CV*, which is described in chapter 3.3.

Another advantage is that this functionality also simplifies the system setup. The positioning of the camera can be more flexible, as the desired camera image cutout can be chosen within the software and does not have to be defined by hardware adjustment of



(a) Original Kinect IR and depth stream without tracking area.

(b) IR stream with defined tracking area and accordingly masked depth stream.

Figure 5.7: The tracking area can be defined by the user in order to save processing power.

the camera itself. This allows a camera setup based on the requirements of the interaction space and less on technical prerequisites.
The tracking area can be defined by the user in the GUI and is restored every time the program is started.

Create Background Image

As already described in chapter 4.1.1 the first step in the tracking process is the creation of a background depth image. Therefore a cumulative image is created over a period of several frames. This is done by recording a sequence of the depth stream and building the average image of all frames. The advantage of using the average over several frames, compared to using just a single frame, is that several visual artefacts of the stream, as flickering and noise, can be reduced drastically. As our tests showed it is enough to use 10 frames to create a good background image. This means that the initialization process of the whole tracking system needs less then half a second. During this initial phase of the tracking the interaction space has to be empty. This means there should be no ACTOs present and also the users arms and all other objects need to be outside of the tracking area. Only the tracking surface, in our case the table, and the tracking marker have to be in the scene.
With this method we can create a very robust and reliable background image that only

needs to be created once in a tracking session.
The process of creating the background image is automatically started when the TrACTOr application is started, but it can also be triggered manually during the tracking process, if desired. This could be necessary if the tracking camera is moved.

Create Foreground Image

As soon as the background image is created it can be used to generate the current foreground image, based on the current depth frame. Therefore a binary mask is created, based on the difference image of the current depth image and the background image. This can be seen in figure 5.8a. This mask then covers all the areas, where the current depth frame is different to the background image, which corresponds to the area of interest for ACTO detection. Finally this mask is applied to the current depth image and the result is the current foreground image figure 5.8b, which is the basis for all further processing steps.
An important detail is, that the foreground image still contains the depth information of the relevant areas of the original depth frame. This is needed in one of the next processing steps, where the vertical position (z-position) of the ACTOs is determined.

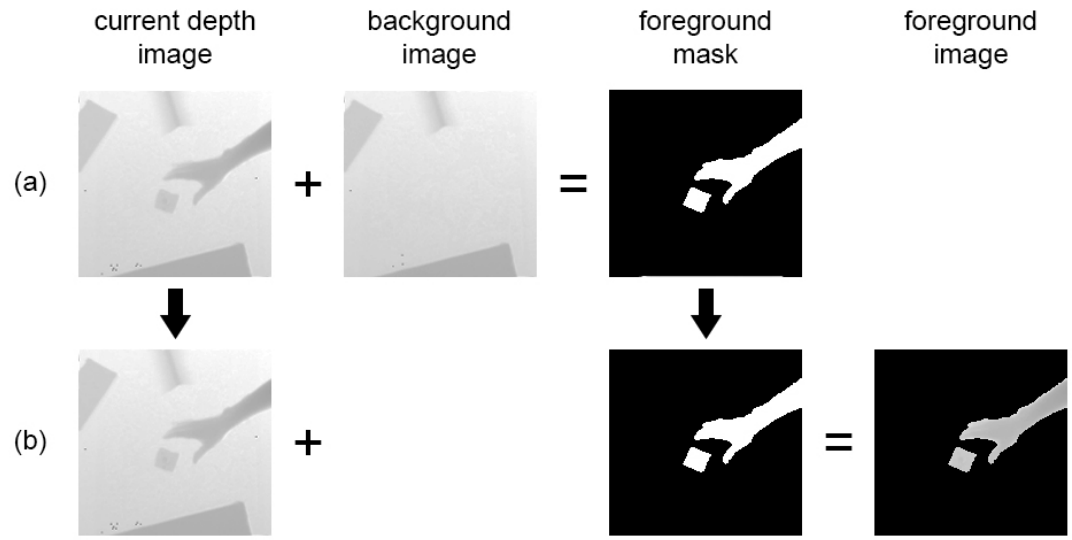


Figure 5.8: Foreground image creation process: a) Creation of binary foreground image mask, b) Generation of foreground image

Two Level Tracking

As mentioned in chapter 4.1.1 we use a two level tracking system for the identification of potential ACTOs, so called candidates. This is done in the Square Finder. The first step, which is required for both of the levels, is the detection of edges in the foreground depth image. Therefore we use the Canny edge detection algorithm, developed by

Canny [Can86]. The resulting edge image is then used for both of the tracking levels, but in a different way (See listing 5.1).

```
1  // Find Canny edges in the foreground image and store them to 'cannyEdges'
2  CvInvoke.Canny(
3    foreground,
4    cannyEdges,
5    cannyThreshold,
6    cannyThresholdLinking
7  );
```

Listing 5.1: Generate image containing edges from depth foreground image.

Level 1 - Square Detection In the next step we try to find squares in the edge image. Therefore we first try to find contours (listing 5.2, lines 2-5), which are then evaluated to see if they can potentially be ACTOs or not. The contour needs to have a minimum area size (line 15) and exactly four vertices (line 17), as only rectangles can be ACTO candidates. Finally the remaining contours corner angles are calculated. Only if all four corners are around 90 degree the contour (lines 24-35) is added to the final list of ACTO candidates (line 39). The list is then passed to the ACTOManager where further processing takes place.

```
1  // Find contours in edge image
2  CvInvoke.FindContours(
3    cannyEdges, contours, null,
4    RetrType.List, ChainApproxMethod.ChainApproxSimple
5  );
6
7  for( int i = 0; i < count; i++ ) {
8    using( VectorOfPoint contour = contours[i] )
9    using( VectorOfPoint approxContour = new VectorOfPoint() ) {
10     CvInvoke.ApproxPolyDP(
11       contour, approxContour,
12       CvInvoke.ArcLength( contour, true ) * 0.05, true
13     );
14     // only consider contours with a certain minimum area size
15     if( CvInvoke.ContourArea( approxContour, false ) > minAreaSize ) {
16       // only consider contours with four vertices
17       if( approxContour.Size == 4 ) {
18         // only consider contours where all four corner angles are
19         // between 80 and 100 degrees
20         bool isRectangle = true;
21         Point[] pts = approxContour.ToArray();
22         LineSegment2D[] edges = PointCollection.PolyLine( pts, true );
23
24         for( int j = 0; j < edges.Length; j++ ) {
25           double angle = Math.Abs(
26             edges[( j + 1 ) %
27             edges.Length].GetExteriorAngleDegree( edges[j] )
28           );
29           if( angle < 80 || angle > 100 ) {
30             isRectangle = false;
```

```
31                  break;
32                }
33              }
34
35              if (isRectangle) {
36                RotatedRect newRr = CvInvoke.MinAreaRect(approxContour);
37                // make sure all angles are in the range [0;180[
38                newRr.Angle = ACTOHelper.normalizeAngle(newRr.Angle, 180);
39                boxList.Add(newRr);
40              }
41            }
42          }
43        }
44      }
```

Listing 5.2: Find rectangles in edge image.

Figure 5.9a shows the process of creating the edge image from the depth image and finding the rectangular candidate in it.

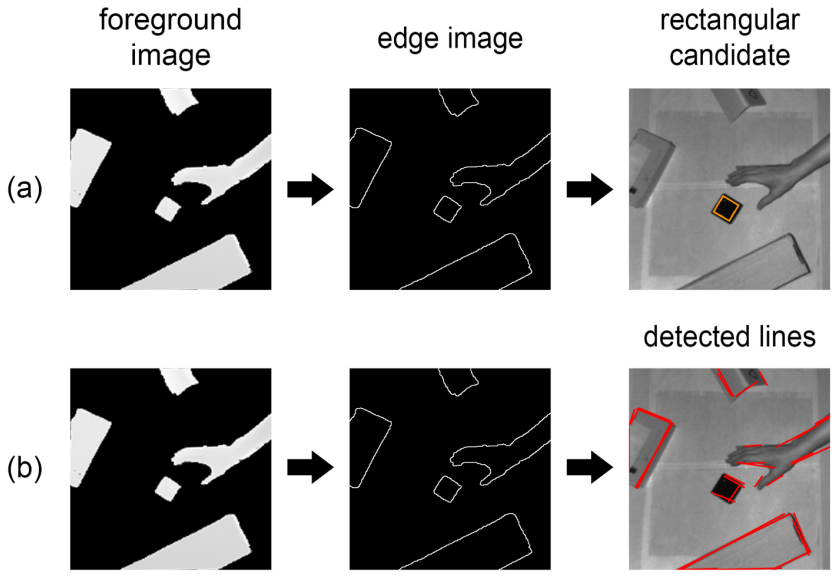


Figure 5.9: a) Creation of an edge image out of a foreground image and find rectangular candidate, b) Detect lines in the same edge image.

Level 2 - Line Detection The ACTO candidates generated in level 1 can also be used to detect ACTOs which are unknown to the tracker until then. This level is only used to maintain ACTOs that are already known. This is usually necessary when the ACTOs are partly occluded by the users hands during the interaction. New ACTO candidates can not be found only based on line detection, because there are usually too many lines detected in the image and also the length of the detected lines is not reliable

enough to draw proper conclusions on ACTOs.
Listing 5.3 shows the source code, how the lines are extracted out of the edge image. Figure 5.9b visualizes the creation of the edge image based on the foreground image and how the lines are detected in the second step. See section 5.2.3.1 to see how these lines are used to maintain known ACTO positions.

```
1  // Detect lines in the edge image
2  LineSegment2D[] lines =
3    cannyEdges.ToImage<Gray, byte>().Dilate(1)
4      .HoughLinesBinary(0.5, Math.PI / 180, 6, 20, 3)[0];
```

Listing 5.3: Use Hough Transformation to detect straight lines in the edge image.

ACTO Management

From the previous steps we receive a list of rectangles, which now need to be evaluated. In the ACTO Manager we store already identified ACTOs for continuous tracking, but also evaluate ACTO candidates to see if they actually are ACTOs or not.
In the first step we try to match the detected ACTO candidates as well as the tracked lines with the already known ACTOs:

Maintain Known ACTOs For every known ACTO we first try to match any of the rectangles we found. Therefore we mainly use the distance between the ACTO and the rectangles. If the distance of the closest rectangle is small enough, we assume that the ACTO has moved since the previous frame. We then use the position and rotation of the rectangle to update the ACTOs position and rotation. An important step here was to not just replace the ACTOs position with the rectangles position, which could be assumed to be the new position. As the tracking has a failure of several millimeters from frame to frame this would cause strong jitter in the final tracking. So we use a Kalman Filter for position and rotation to make the tracking much smoother. This means the previously measured positions area also taken into account for the determination of the new position, which is then estimated based on these values. The resulting, so called *predicted* position, is more likely to fit to the actual position, than the measured one, due to measuring inaccuracies.

For every ACTO which could not be matched with a rectangle we try to find matching lines from the lines detected in level 2 of the tracking process. This most likely happens if the user interacts with an ACTO and occludes parts of the ACTO. We try to match each of the detected lines by matching the end points of the line to the corner vertices of the known ACTO. If both ends of the line are closer than a defined minimum value to two adjacent corners, we accept the line as a match and use the same Kalman Filter as before to move and rotate the ACTO towards the newly detected position.

For every known ACTO that could not be rematched we reduce the Time To Live (TTL) by one. The ACTO will still be considered as known, which means it will be sent over the WiFiTransmitter to the ACTO-Framework along with the other known ACTOs. We

also try to match it again in the next frame, as long as the TTL is greater than zero. Then the ACTO is finally dropped. We use an initial TTL of 20, so the ACTO can stay alive for almost a second (20 frames) without actually being tracked. Every time it is tracked, the TTL is reset. This mechanism improved the stability of the tracking a lot, because it happens quite often that an ACTO isn't detected for one or more frames.

Identify New ACTOs While the detected lines, which are not used for ACTO matching, are dropped, the remaining rectangles are used to create new *unidentified ACTOs*. These unidentified ACTOs are stored in the ACTOManager in order to try to identify them during the following frames, but they are not sent over the WiFiTransmitter. While this identification step is only carried out for unidentified ACTOs, the rotation determination, which is also explained in the following section, is also applied to already identified ones. For each of these two steps we need to find the IR LED in the infrared image.
Every frame of the IR image is also forwarded to the ACTOManager. For every ACTO we cutout the relevant section of the IR image to see if the IR LED is currently on or off. If the light is on we additionally identify the closest corner of the ACTO rectangle to the IR LED. This information is then passed to the ACTO model itself where the following two steps are performed:

ACTO Identification

The ACTO identification is the process of ensuring that an ACTO candidate actually is an ACTO and retrieving its id. As described in section 5.3.2.1, we could not implement the id detection as planned. Tests showed that pattern transmission was much more reliable than binary id transmission.
When the detection is about to start we wait for a series of *1*, followed by a single *0*, because all of the ids we use end on *1* and start with *0*. This raises the chance to start the id detection at the right point.
Also we found out that the maximum bitrate that could reliably be detected is ten bits per second. This is due to the asynchronous process of sending and receiving the visual information. As the tracker works with 30 frames per second, we can try to read the same bit three times in a second. In the ideal case we read three times *0* or three times *1*, but in some cases we receive two times *0* and one time *1*, or the other way around. In that case we use the value that occurs more often. This showed to improve the reliability again.
Another problem is that sometimes the IR LED is occluded by the user or another object, which can interrupt the detection stream. In the worst case this could lead to a wrong id detection. To prevent this error we included another mechanism. The ACTO id is only set after the same id is detected twice. From that moment on, the ACTO will be seen as identified. As soon as the same id is detected a third time the id detection is considered as reliable and final. After that the tracker does not try to identify this ACTO anymore, it stays fixed as long as the tracking is not lost again.

Determine Rotation

The only remaining task of the tracking process is the determination of the rotation. As explained in section 4.2.2.1 the tracked rectangle gives us four possible rotation candidates for the ACTOs actual rotation. If the IR LED is detected, it can be assigned to one of the four corners of the rectangle. We also know that the IR LED is always positioned in the left front corner of the ACTO. This allows us to choose the correct one of these four candidates.
This step is not only applied to new ACTOs, it is executed for every ACTO in every frame. If the IR LED is off for that frame, it is simply ignored. This method proofed to be very reliable. The only issue we encountered, was with reflections on the surface of the ACTO, which were hard to distinguish from the IR LED itself. See chapter 5.3.1.3 to see how we fixed this problem.

5.2.3.2 Normalize and Transform to Marker COS

As already discussed, it is necessary to synchronize the COS of the tracking system, the so called tracking COS, with the camera COS of the StARboard AR application. Therefore we use a visual tracking marker (see figure 5.3b), which allows us to create a shared COS, the so called marker COS. We use this marker for having a shared origin, the so called pivot point of the scene. But it is also used for normalizing distances, so one unit in the tracking COS is equivalent to one unit in the camera COS. This process will be described in the following section.

Detect Tracking Image

The first step of this synchronization, or calibration process, is to detect the visual marker. To achieve this we use the Kinect's infrared camera. It would also be possible to use the RGB camera, but the position of the RGB camera inside the sensor is a bit different. As the depth stream is also created with the IR camera we use it for tracking as well. Then we have the exact same origin and prevent an offset of several centimeters.
For the step of visually identifying the image in the IR frame we use *Emgu CV* again, as shown in listing 5.4. The result of this step is the *homography matrix*, described in the next section.

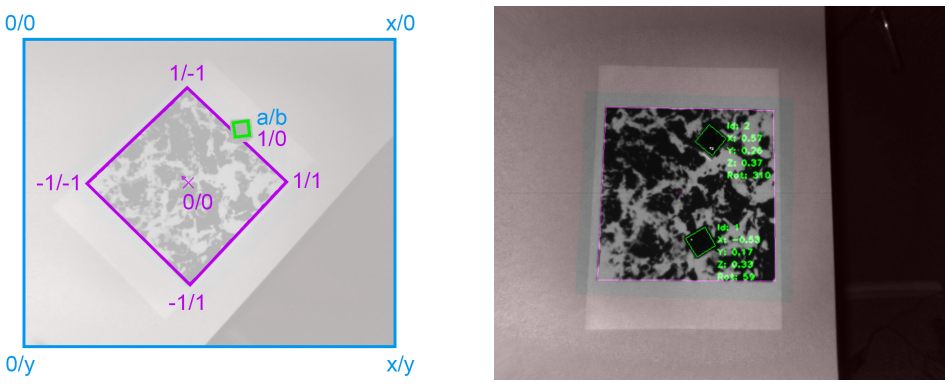
```
1  // Find the calibration image in the IR frame
2  Mat homographyMatrix = ImageHelper.findSubimage(
3    ( (Image<Gray, ushort>) irFrame).Convert<Gray, byte>(),
4    calibrationImage
5  );
```

Listing 5.4: Find the image marker within the IR Frame.

Homography Matrix

The homography matrix (see chapter 3.9) can be used to transform between the tracking COS and the marker COS. We can use the homography matrix not only for transforming

the origin of the COS, also the distances can be normalized. The square marker has a size of 2 x 2 units in the marker COS, whereas the center point is *(0/0)*. So the top right corner of the marker would be *(1/-1)*. Figure 5.10a compares the two COSs. In figure 5.10b you can see an example of a real tracking situation where the tracking information is already transformed into the marker COS.



(a) Tracking COS and marker COS in comparison. ACTO with coordinates in both COS.

(b) Live tracking image with the detected marker and two identified ACTOs with normalized coordinates.

Figure 5.10: Transform from tracking COS to marker COS with homography matrix.

5.2.3.3 Stream Tracking Data via WiFi

The TrACTOr-tracker has a WiFi socket connection to the ACTO-Framework. For every frame the id, position and rotation of all currently known ACTOs is sent to the framework. All this data is already in the marker COS, so the other components of the system can directly use it for further processing.

5.2.4 TrACTOr Graphical User Interface

In order to be able to monitor and control the tracking process we also developed a Graphical User Interface (GUI) for TrACTOr. Figure 5.11 shows the main GUI window of the tracking software. In the upper section you see different video streams which are mainly for monitoring the system.
The *IR Image* shows the live IR image stream with the currently detected ACTOs shown in green color. Next to each ACTO you can see its id as well as the current coordinates and rotation. The purple square shows the detected visual image marker, in the middle you can see a little arrow, which shows the orientation of the marker. It always points upwards. Also the active tracking area is visualized in the IR image. It can be seen around the tracking marker. The whole area outside the tracking area is overlayed with a transparent red color. It is also possible to display more information, like the currently detected rectangles and lines. This features can be switched on and off in the *Tracking Controls* section.

The three frames on the right show different depth streams. The *Depth Image* shows the original current depth stream. The *Foreground Image* is the processed depth image which is used for tracking, the white squares you can see are the ACTOs that are being tracked. The other white object is the users hand. The *Background Image* shows the empty tracking scene which is used to generate the foreground image. In order to safe processing power each of this streams can be frozen by clicking on it. The tracking will still run in the background, but the displayed frame is not updated anymore.

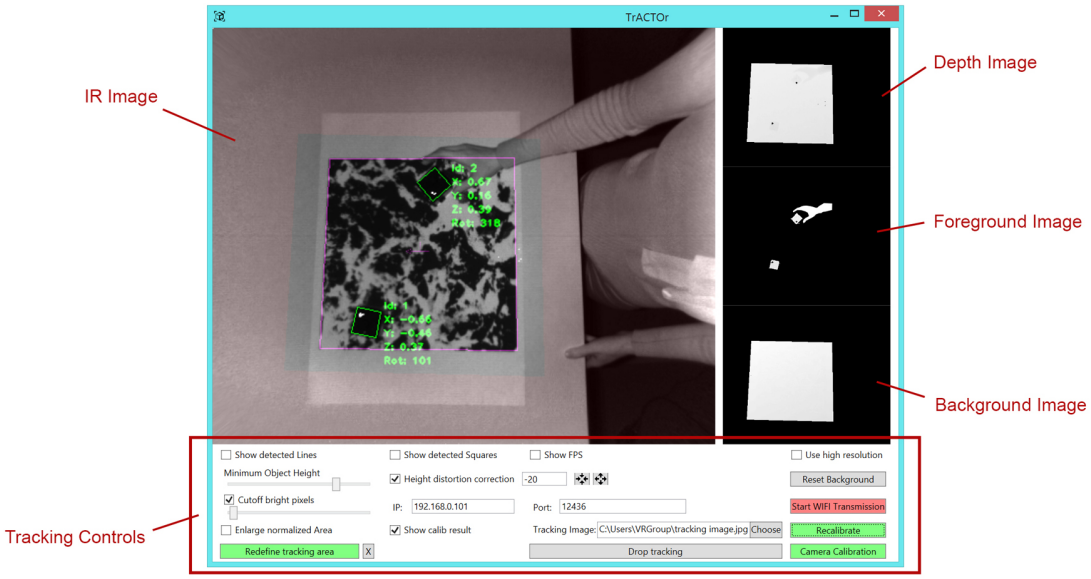


Figure 5.11: Main GUI window of the TrACTOr tracking software.

For managing the tracking process we provided the *Tracking Controls* section. The most important features are the WiFi module, which is used for creating the connection to the ACTO-Framework. Also the image marker can be changed here, if the user wants to use a different one. Every time the camera setup changes it is also necessary to redefine the tracking area, which can be initiated here as well. As soon as this process is started, the user can define the new tracking area by simply clicking its new corners in the IR image. Another feature which can be controlled from here is the tracking calibration. If the tracking is offset, due to bad camera calibration or setup, the error can be reduced by manually calibrating the tracking.

5.3 ACTO Module Development and Core Code Adaptions

In order to make the ACTOs work with the TrACTOr-tracker and also provide the required interaction functionality for the StARboard application, an ACTO module had

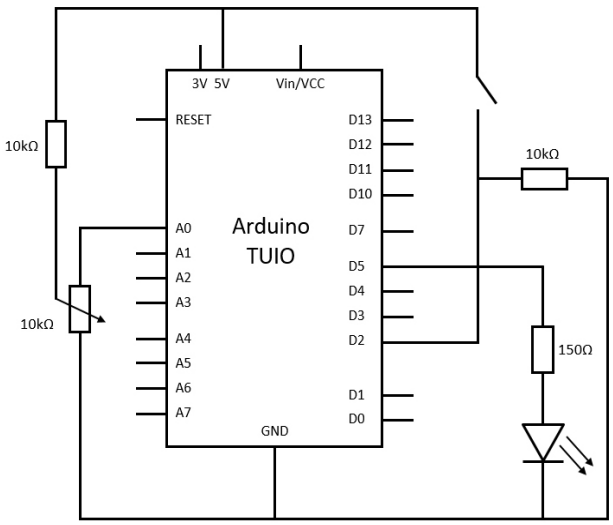
to be developed. Also the ACTO's Operating System (OS) code had to be extended with the according functionality for TrACTOr id detection.

5.3.1 Hardware

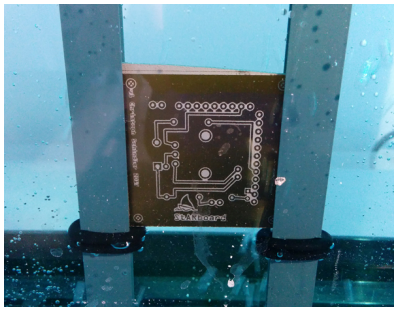
As already mentioned, the hardware of a module houses three main components: an IR LED, a rotary encoder and a button. We developed a compact module that can easily be attached to the basic ACTO robot. The module consists of two main hardware components that had to be designed and developed: the circuit board, containing all the electronic parts and the housing, which covers all the sensible parts and allows stable mounting to the ACTO.

5.3.1.1 Circuit Board

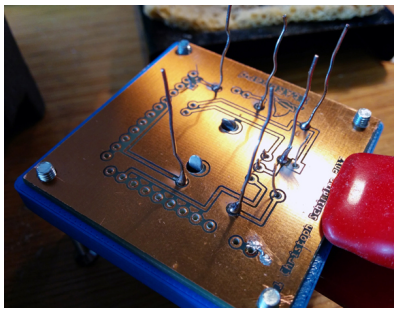
The circuit board had to be developed from scratch. Figure 5.12a shows the final circuit diagram which was designed based on the prototype which we developed earlier. This circuit diagram was used to design the circuit board with the software *fritzing* [fri].



(a) Circuit diagram.



(b) Etching process of the first version.

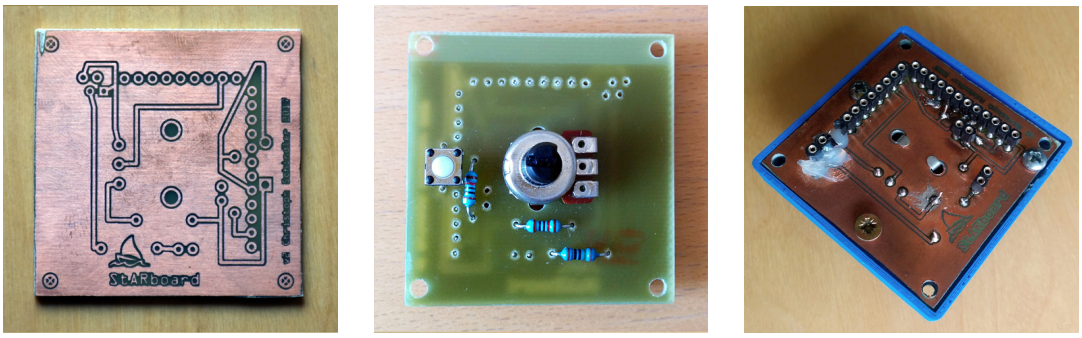


(c) Soldering work.

Figure 5.12: Manufacturing of the circuit board.

The final template for the board etching was printed on a transparent foil, which we used to transfer the template to the raw board, by exposing it to UV light. After this the

board was put into an etching liquid to remove the unwanted copper (see figure 5.12b). The result was the finished circuit board as seen in figure 5.13a. After drilling the holes the electronic parts were added to the board (see figure 5.12c).
During testing with the first finished version of the circuit board, as seen in figure 5.12b, we found out that there were some problems with the circuit design. The button and the rotary encoder were not connected the in right way. We revised the circuit diagram and fixed all the problems in the second and final version of the board. Figures 5.13b and 5.13c show the final board with all the electronic components already attached (the IR LED was added later).



(a) Etched circuit board (final version).

(b) With components from above (LED missing).

(c) Finished board with components from below.

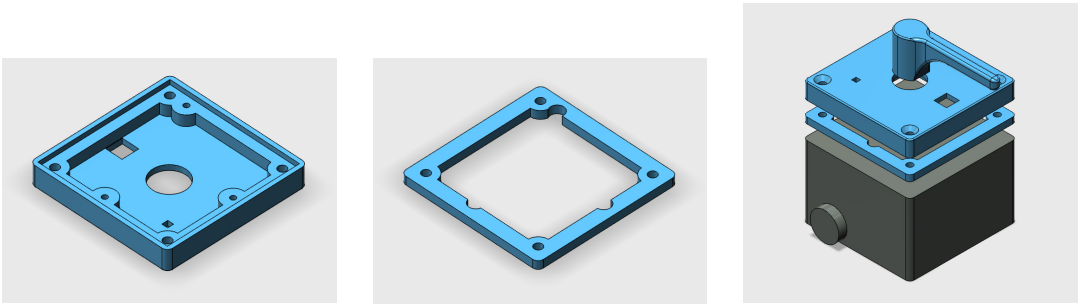
Figure 5.13: Final version of the circuit board.

All in all we manufactured three identical modules to give the users the possibility to interact with one wind source and two ships at the same time.

5.3.1.2 Housing

One major target during the design of the housing was to keep it as compact as possible, to minimize the size of the final ACTO. Also it should be easy to mount and dismount and be built in a robust way so that nothing breaks when being mounted and unmounted several times. With the outer dimensions we stuck to the dimensions of the ACTO case, to get one compact object in the end.
The whole housing consists of two main parts that have to be attached to each other. All of the 3D parts were designed with *Autodesk 123D Design* [aut] and printed with an *Ultimaker 2* [ult] printer. Figure 5.14a shows the 3D design of the piece which the circuit board is attached to. In the image the 3D model is upside down, as preparation for the 3D printing process. Each corner has a hole which is used for the screws, to solidly attach it to the ACTO base component. On the same level three smaller holes can be seen. Their purpose is to hold the screws that permanently attach the circuit board to the case. On the lower level three more holes are visible. They are needed for the electrical components which are placed on the board. The square one will contain

the button, the large circular hole in the center is for the rotary encoder and the smaller square shaped hole is for the IR LED. The circuit board and the housing were designed in a way that the components perfectly fit into the designated holes.
The other main part can be seen in figure 5.14b. It is placed between the circuit board and the ACTO base components to fill up the remaining space between the housing of the ACTO and the StARboard component. It is not required for every ACTO, as they are not perfectly equal.



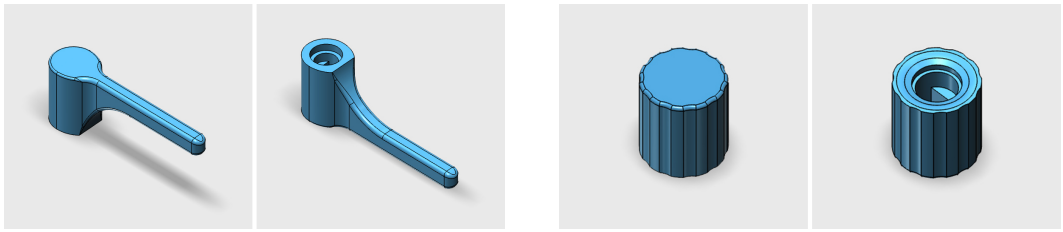
(a) The base component (upside down).

(b) The spacer component.

(c) Assembly of all components to the ACTO.

Figure 5.14: The main components of the StARboard housing.

Two more components were designed. They can be attached to the rotary encoder and improve the user experience by giving a better haptic feedback and allow more intuitive interaction. The first one can be seen in figure 5.15a and is called *boom*. It is used for the ACTOs representing boats. The ACTO will be overlaid with a virtual ship in a way that the boom is placed exactly where the boom of the virtual ship can be seen. This gives the user the impression of directly moving the virtual boom left and right, in order to adjust the sail trim. The other component is called *knob*. It serves a similar purpose and is used for the fan (See figure 5.15b).
Figure 5.14c shows how the 3D printed components are attached to each other.

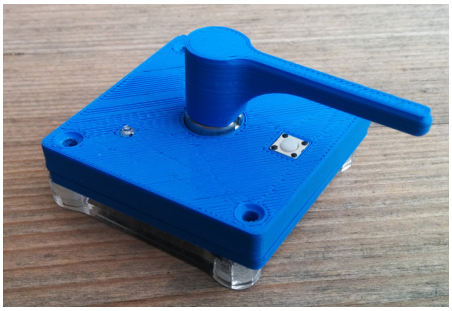


(a) Boom, top and bottom view.

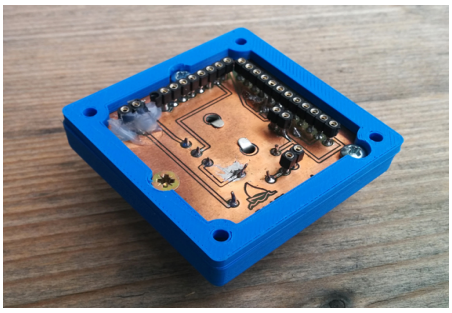
(b) Knob, top and bottom view.

Figure 5.15: Interaction components for the rotary encoder.

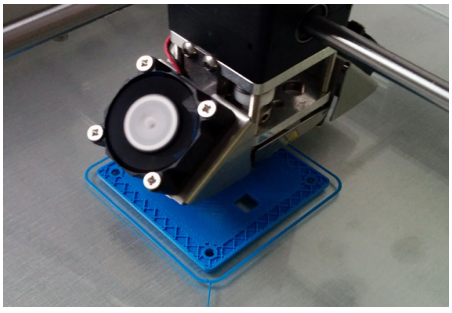
All the final parts, printed with the 3D printer, can be seen in figure 5.16d. The printing process can be seen in figure 5.16c. The fully mounted StARboard ACTO module can be seen in figures 5.16a and 5.16b. The finished module can be applied to the ACTO base module and screwed together. The final ACTO is shown in figure 5.16e.



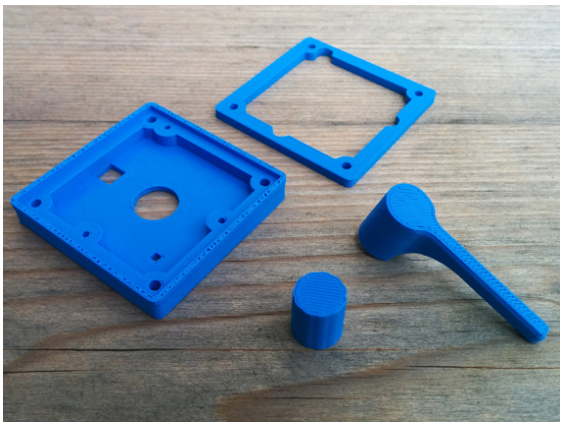
(a) Assembled StARboard-module (top view).



(b) Assembled StARboard-module (bottom view).



(c) 3D printing of a base component.



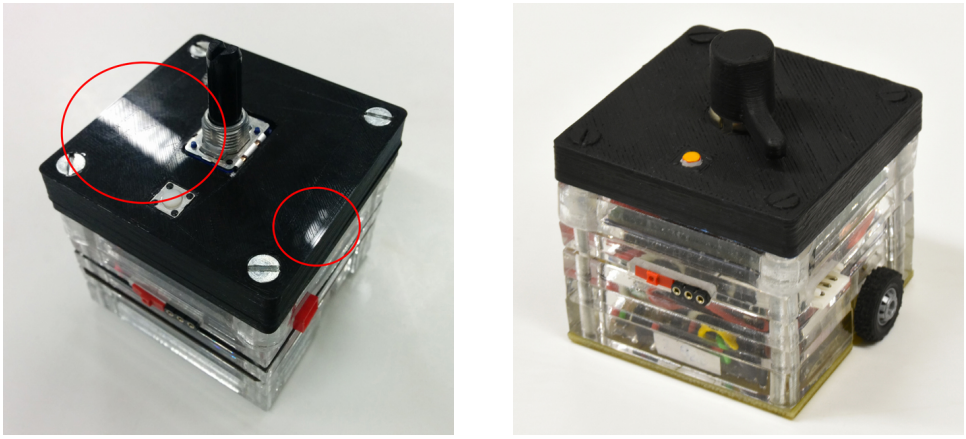
(d) 3D printed parts.

(e) Fully assembled ACTO for StARboard application.

Figure 5.16: Final StARboard ACTO module components and assembled module.

5.3.1.3 Reflections

A problem that came up with the ACTO tracking was reflections. The surface of the 3D printed objects was very smooth, which caused strong reflections of the light, as seen in figure 5.17a. Also other shiny parts of the ACTO as the metal screws or parts of the potentiometer and button were a problem. The Kinect emits infrared light to create the depth image. If this light is reflected the depth image can get severely distorted in the affected areas, as well as the infrared image, which we use for id detection. Especially when the ACTO was positioned directly underneath the camera, so the reflections could go straight back, this caused problems with the rectangle detection and the id recognition. To solve this issue we painted the surfaces which faced to the camera. We tried different colors, to find one that is matt enough to not cause any reflections. We finally used black acrylic color. We choose black in order to provide a background with high contrast for the AR overlay. This solved the problem and the result was the final ACTO, as it can be seen in figure 5.17b.



(a) Optical reflections on the ACTO surface can disturb the tracking.

(b) The black acrylic paint prevents reflections on the surface.

Figure 5.17: Reflective surfaces on upside facing components of the ACTO caused tracking problems.

The same problem occurred with the surface of the interaction space, which is the table and the marker. We had to use a matt paper and ink for the tracking marker to prevent reflections.

5.3.2 Software

As already discussed in chapter 3.2, the ACTOs are based on Arduinos. The ACTO code is written in C. For our purposes the code basically required adaptions of two kinds. First of all the ACTO OS had to be extended with a new module to provide the functionality required for the TrACTOr tracking and id detection process. The other implementation which had to be done was the functionality required by the StARboard application. We

did it in the form of a StARboard Extension. The resulting ACTO software architecture can be seen in figure 5.18.

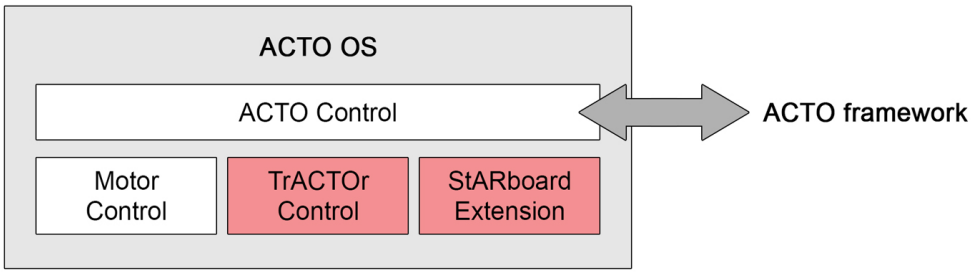


Figure 5.18: ACTO software architecture with the new *TrACTOr Control* module and the *StARboard Extension*.

5.3.2.1 TrACTOr Control Module

As already described in chapter 5.2.3.1, the identification process could not be implemented as planned in the beginning. The detection of single bits was not robust enough, so we had to change it from binary id transmission to pattern transmission. We use a two dimensional array to define the available patterns. Listing 5.5 shows the defined id patterns from 1 to 8. The first number in each line defines the length of the code and is required for looping the series, as the length can not be retrieved programmatically. The patterns themselves always consist of a series of *0* followed by a series of *1*, which we found was the most reliable type of pattern from the ones we tested. We use variable length patterns to minimize the amount of data to transfer and save time during the identification process, especially for ACTOs with a low id.

```
1  const int ID_CODES[8][6] = {
2    { 2, 0, 1 },                  // 1
3    { 3, 0, 1, 1 },               // 2
4    { 3, 0, 0, 1 },               // 3
5    { 4, 0, 0, 1, 1 },            // 4
6    { 4, 0, 1, 1, 1 },            // 5
7    { 4, 0, 0, 0, 1 },            // 6
8    { 5, 0, 0, 0, 1, 1 },         // 7
9    { 5, 0, 0, 1, 1, 1 }          // 8
10 };
```

Listing 5.5: ACTO id patterns from 1 to 8.

For sending the id pattern in a permanent loop an extra module, called *TrACTOrControl*, was developed. Its main functions are *trACTOrSetup* and *trACTOrLoop*.
trACTOrSetup runs only once during the initialization phase. It is called at the end of the *setupActoFramework* function in the core *ACTOControl* module. The setup function is only responsible for defining the Arduino's digital output pin, which should be used for the LED and reading the id from the StARboard code. We use a digital pin and not an

analog one, because the LED only ever needs to have two different states, *ON* and *OFF*. The more complex function is the loop function *trACTOrLoop*. It is also called from the ACTO core module, but in the loop function *loopActoFramework* which runs in an infinite loop as long as the ACTO is activated. The purpose of the loop function is to send the id pattern in a permanent loop. Therefore an important parameter is required, the *bitrate*. The bitrate is the amount of data bits that is sent per second. We found that the ideal bitrate is at 10 bits per second. With this bitrate we can achieve an optimized balance between a high data rate and a low error rate. So depending of the ACTOs actual id the identification pattern can be sent between two and five times per second, which is enough for a quick id detection through the TrACTOr-tracker.
The function uses a timestamp to detect when the next bit should be sent and checks every loop if this timestamp is already reached. As soon as the timestamp is exceeded the next data bit is read and the status of the IR LED is set accordingly with the *digitalWrite* command. If the bit is a *0* the pin state is set to *LOW*, if it is a *1* it is set to *HIGH*.

5.3.2.2 StARboard Extension Code

For the StARboard application the ACTO needs to be equipped with a stepless rotary encoder and a button. Therefore the ACTO code needs to provide the functionality to read from this input sources and pass the resulting values to the ACTO-Framework.
In case of the rotary encoder we use an analog input pin. This is necessary in order to be able to control the sail trim or fan speed fluently, between their minimum and maximum value. The *ACTOControl* module provides a very useful function to retrieve input events, called *onAnalogInputEvent*. The desired pin needs to be specified at the beginning of the module, by adding it to the *analogInputPins* array. Then the function is called every time there is an input event on one of the specified pins, which in our case happens whenever the current rotary encoder value changes.
The resulting value will always be an integer, but so far we do not know its meaning as it is dependent on the actual rotary encoder component that is used. Before the value can be used we have to normalize it, so that the minimum possible value is *-1* and the maximum value is *+1*. Therefore we have to test the minimum and maximum values and use them as constants for the calculation of the normalized result.
This final value is then sent to the ACTO-Framework. For this task the ACTOControl provides another function: *writeFloatMsg*. It takes three arguments: the *usb_msg*, which specifies the type of the message that is being sent. In our case this is *USB_SET_AN_UNIT*, which is used for sending the analog status of a unit. The second parameter is the ACTO id and the third one the actual value as float.
A similar approach is used for the button, except that we use a digital input pin for it, which can only be *0* or *1*.

5.4 ACTO-Framework Adaptions

In order for the TrACTOr-tracker to work properly and also to provide all the functionality we needed for the StARboard application we had to adapt the ACTO-Framework to fit our needs. The ACTO-Framework is an Android application, written in Java and runs on a smartphone. The software architecture of the updated version of the framework can be seen in figure 5.19.
The core components are the *Communication* component, the *GUI* component and the *ACTO Control* component. We use the *Communication* component to share all sorts of information as tracking information or user interaction with the TrACTOr-tracker, the StARboard server over WiFi. The communication with the ACTOs is over an Arduino Server that is connected via USB. The *GUI* component contains the interfaces for the user interaction: *ACTO GUI*, *ACTO Activity* and *TrACTOr Activity*.
All the changes that were made in comparison to the original version are described in the following sections.

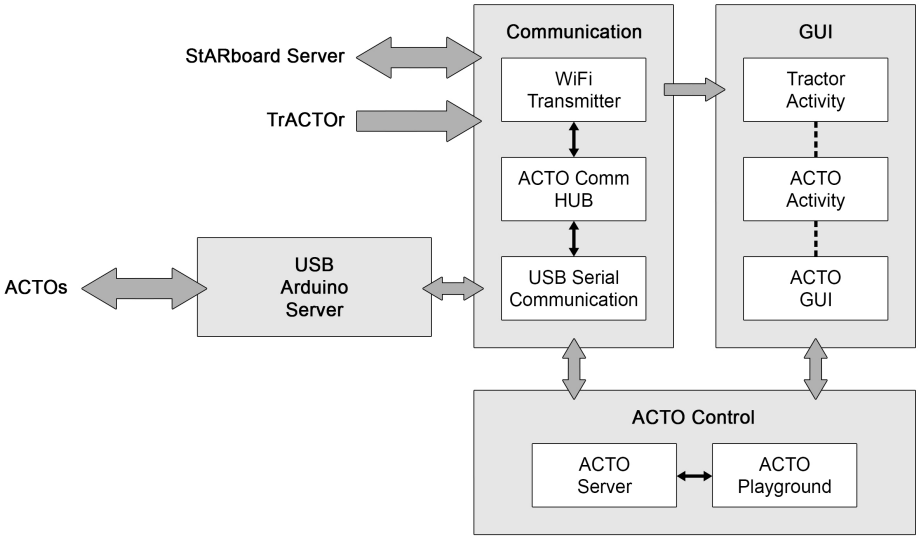


Figure 5.19: ACTO-Framework software architecture.

5.4.1 Message Enhancement

Internally the ACTO-Framework uses a predefined set of system messages for sending WiFi and USB data. Other system messages are used for sending internal messages between Services and Activities. One important improvement we had to make is to introduce new system messages to create the new functionality we needed.

5.4.1.1 Receive Tracking Data

In the original version the tracking data was created by the framework itself. It used optical marker tracking to track ACTOs with the phones built-in camera. This tracking data could be streamed in real time to other application with a WiFi socket connection. For our setup we needed the framework to also receive tracking data via WiFi, as we use the TrACTOr-tracker for generating it. Therefore we implemented a new WiFi interface, which allows the TrACTOr-tracker to establish a socket connection and stream the live tracking data to the framework. A new view was implemented as well, to graphically display the current tracking situation. This view is described in detail in section 5.4.2.

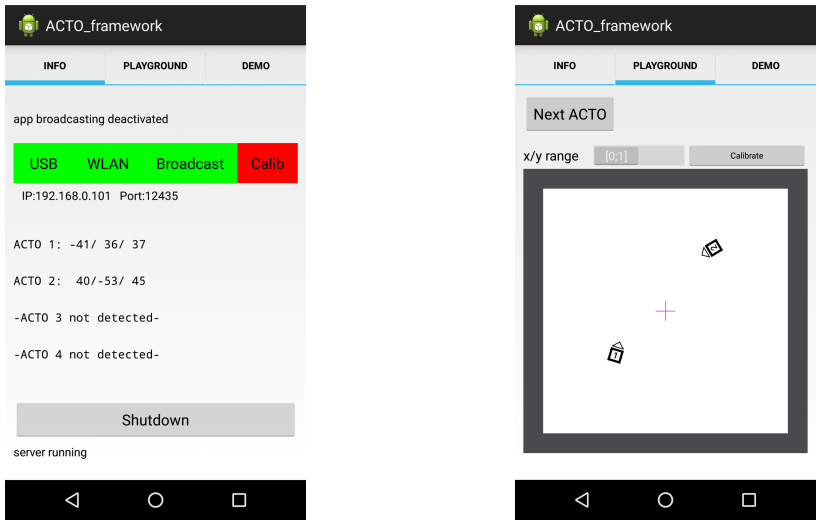
5.4.1.2 ACTO Commands

Also the set of available commands for ACTO robots was extended. So far the framework was just used to let robots drive to a certain point and rotate by a certain angle. The main feature we needed for StARboard was to let the robots drive straight at a certain speed. So we introduced three new commands: *driveStraight*, *setSpeed* and *stopMovement*.
The *driveStraight* command is used to let an ACTO drive straight ahead. The *setSpeed* command can be used to change the ACTO's speed while driving, which is very useful for us as the ships in the StARboard application repeatedly change their speed based on the current wind situation. The *stopMovement* command is used to stop the ACTO when the wind is switched off and also for safety reasons. When the ACTO leaves the tracking area, it stops as well, so it cannot fall of the table.
The ACTO movement functionality itself is implemented in the *ACTOPlayground* class. It uses destination vectors as target position of the ACTO movement. In order to fit into this system and make the drive straight functionality compatible and simple, we also use target vectors. When a drive straight command is received, we use the current tracking position of the ACTO and calculate a target point, which is straight ahead of the ACTO. This solution lets the ACTO drive straight, but we still use the provided functions of the existing implementation.
Also the stop functionality is implemented in a similar manner. As soon as a stop command is received, the movement target position is set to the ACTO's current position. This means the ACTO will have immediately reached its new position and stops where it currently is.

5.4.2 Graphical User Interface Extension

For having more control and better observation and debugging options, we also extended the Graphical User Interface (GUI). On the main control page (shown in figure 5.20a) we added an info section, which shows the currently tracked ACTOs with their x, y and z tracking position.
A more graphical representation can be displayed in the *Playground* tab of the app. There the user can see the normalized tracking area with all the tracked ACTOs on their current position. With this view it is also possible to actually move the ACTOs. Therefore the

user has to click the *Next ACTO* button until the desired ACTO is highlighted. Then he can tab on the desired position of the tracking area. The ACTO will instantly move to the chosen position. This feature is not used in the final StARboard setup, but it is especially helpful for testing, debugging and probably for future applications, which use the TrACTOr tracking.



(a) Control and info menu. (b) Live tracking situation.

Figure 5.20: ACTO-Framework GUI.

5.5 StARboard Implementation

The new possibilities provided by the TrACTOr system allows us to develop the *StARboard* application. As discussed in section 4.2.5, we developed this client server based application with Unity in the programming language *C#*. The server application runs on a desktop computer. It can either run on the same computer as the TrACTOr tracking or on a separate machine. To optimize the performance we run it on a separate computer. The StARboard clients can run on various devices that support AR. We mainly used the HoloLens and Android smartphones.
StARboard is a learning application for sailing students. It gives especially beginners a better understanding of the correct sail trim for several wind situations and maneuvers. It also allows the user to explore how different types of ships behave. Figure 4.10 shows the basic system setup, the architecture is described in the following section.

5.5.1 StARboard Architecture

The StARboard architecture basically consists of two main components, the *Server* and the *Client*. It is shown in figure 5.21. We designed the system based on a centralized design principle, which means that most of the work is done on the server side while the

clients purpose is mainly to display the content.
To receive the actual tracking data we establish a connection from the *ACTO-Framework Connection* component to the ACTO-Framework. This connection is described in section 5.5.2.4. Based on the tracking data the *ACTO Manager* creates different types of virtual objects, like ships or fans, which are managed as *ACTOs*, see section 5.5.2.1. These virtual objects are controlled in the *Game Controller* and moved based on the physics and game logic we implemented. See section 5.5.2.2 for a detailed description. For an improved learning experience we developed a module for the specific training of certain sailing maneuvers, which is explained in section 5.5.2.3. These *Maneuvers* are controlled in the *Maneuver Manager*.
Multiple clients are able to connect to the same server instance simultaneously. This allows multiple users to use the system at the same time, so we can provide a shared learning experience.

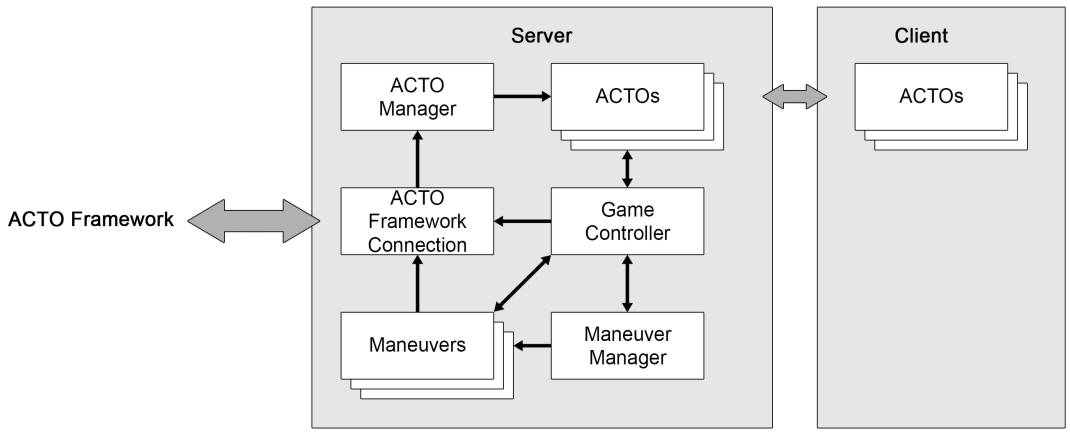


Figure 5.21: StARboard software architecture.

5.5.2 StARboard Server - Physics and Game Logic

On the server we process the received tracking data and setup a virtual scene with different kinds of objects.

5.5.2.1 Virtual Objects - Ships and Fans

While we use the ACTO robots as real world objects for tracking and as input devices, we want the user to see other, virtual objects instead. Therefore we use AR to overlay the ACTOs with virtual objects. While the user sees another object, he can still touch and interact with the ACTO as if it was the virtual object he sees.
The system supports two different kinds of virtual objects, which can be actively used by the user: *Ships* and *Fans*. For each base type different instances of objects can be implemented easily. We developed two different ships and one fan. All of the virtual objects are in a clear inheritance hierarchy, which allows us to bundle shared functionality

in parent classes and easily develop new ship and fan types. The inheritance structure can be seen in figure 5.22. The basic positioning, rotation and interaction functionality is implemented in the *ACTO* base class. We implemented the ship specific functions, like sail trim, driving speed and visualizations in the *Ship* class. The ship specific differences like the physical behavior and visual details of the 3D model are implemented in the ship child classes. The same principle applies to the *Fan* class. General fan specific functions are implemented there, while different types of fans can be implemented based on that. So far we implemented two different ships and one fan. Without much effort more ships and fans could be added.

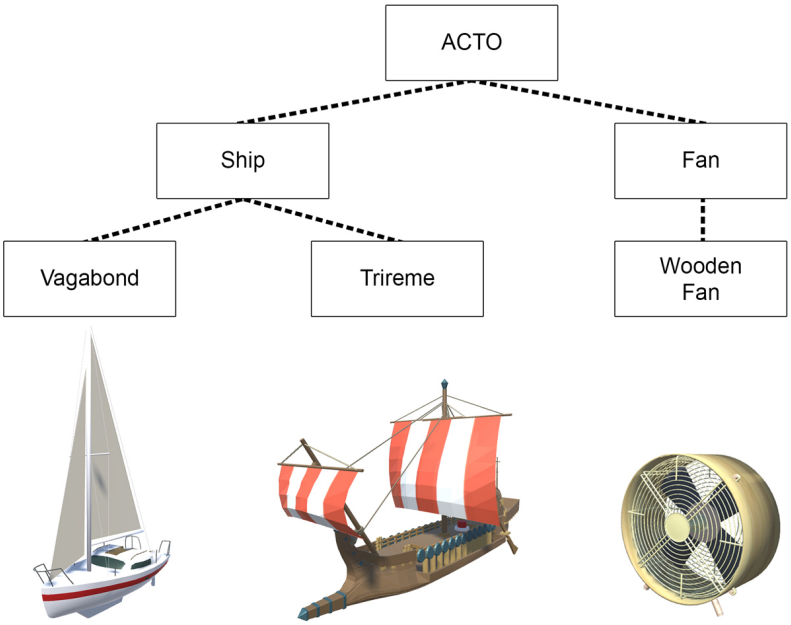


Figure 5.22: Inheritance structure of virtual objects.

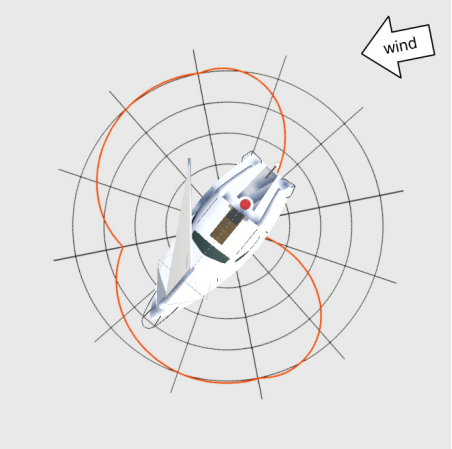
For instantiating virtual objects while the application is running we use Unity's *Prefab* feature. It allows us to predefine objects in the editor beforehand. They can then be instantiated and loaded into the scene at runtime. This feature is also very useful for sharing virtual objects across multiple projects and keep them synchronized. As we used two different Unity projects for the server and the HoloLens client, we had to synchronize the virtual objects during the development process. At runtime the server and each client instantiate their own prefabs. If they are not the same due to bad synchronization, various problems can occur.

Diagrams

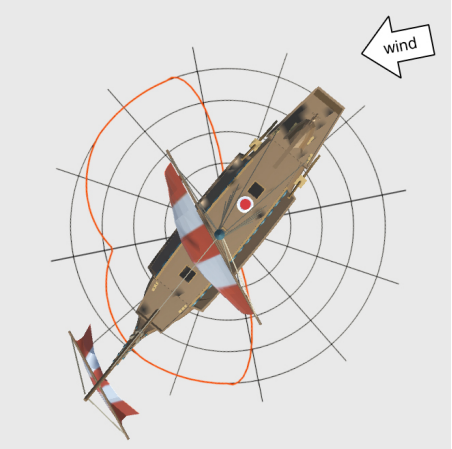
In order to support the user's learning process we implemented two different kinds of visualizations, the *polar diagram* and the *wind diagram*. The user can display them by clicking the button on the ACTO. Read more about the ACTO interaction in chapter

5.5.4.

Polar Diagram The polar diagram supports the user in finding the ideal bearing, dependent on the current wind situation. It does not consider the current sail setting. Figure 5.23 shows how we integrated the polar diagram into the system. It is displayed directly underneath the ship and rotates if the wind source is rotated, so it always points in the direction of the wind. The symmetric orange line shows the maximum possible boat speed per bearing, which can vary for different boat types.



(a) Vagabond with polar diagram.



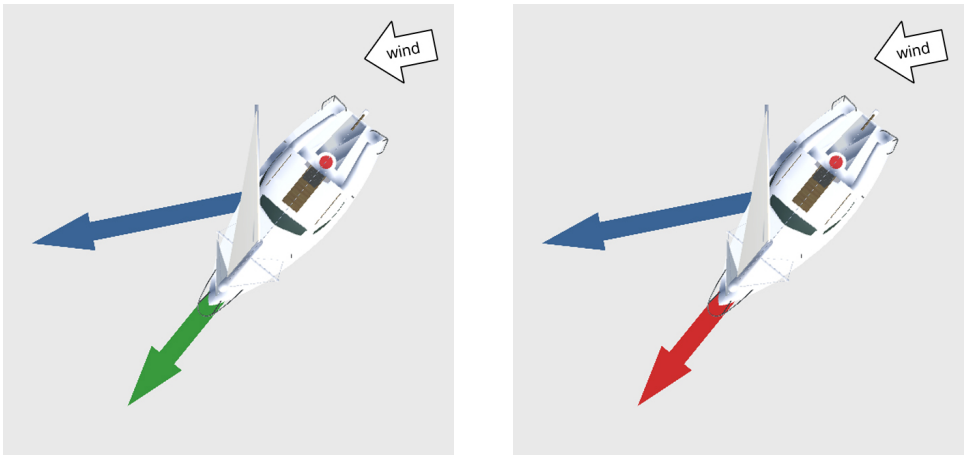
(b) Trireme with polar diagram.

Figure 5.23: Ship specific polar diagrams displayed underneath the virtual objects.

Wind Diagram The wind diagram is supposed to give assistance in the next step, which is finding the ideal sail setting for the current bearing. Figure 5.24 shows that the diagram consists of two different arrows. The blue arrow represents the wind. It points into the direction of the wind and its length indicates the current wind strength.
The second arrow can either be red or green, dependent on the current game mode (read more about game modes in section 5.5.2.2). It always points into the current driving direction of the boat. The length represents the current boat speed. This means the longer the arrow gets, the faster the boat sails. This function allows the user to change the sail setting and immediately see the response through the length of the arrow.

5.5.2.2 Physics and Game Logic

The sailing physics and the game logic are implemented in the *Game Controller* as well as in the *ACTO* class and its deriving classes. Each tracked ACTO is represented by a virtual object in the scene. When a new ACTO is tracked, the user can choose, which kind of object it should represent. He can rotate the potentiometer handle on the ACTO to switch between the available objects and click the button to select one of them. The ACTO will then gain this object's behavior immediately. It can either be a ship or a



(a) Wind diagram in live mode. (b) Wind diagram in planning mode.

Figure 5.24: The wind diagram shows the wind situation and the current boat speed.

fan. The system can only have one fan at the time, because multiple wind sources do not make sense.
The physics of the fan are quite simple, as we only store the orientation and wind speed, the fan itself does not move. The ships on the other side are way more complex, because they should move according to the current bearing, sail trim and wind speed. Therefore we call the *updateSpeed* method of every ship in each update loop of the Game Controller. In the ship model we calculate the current speed, based on the influencing parameters and return it to the controller. In the controller we send the updated movement commands to the ACTO-Framework, to apply them to the ACTO robots.
The calculation of the current ship speed is quite complex, as various parameters need to be taken into account. As the physical behavior of a sailing boat is very complex we simplified the system and implemented a physical approximation which is sufficiently accurate for our application. Listing 5.6 shows the core of the speed update function's implementation.
First we calculate the current bearing (lines 2-9). The bearing is then used to calculate the most interesting parameters, the *maxBearingSpeed* and the *newSpeed*.
We calculate the *maxBearingSpeed* based on four factors (lines 12-16):

- **windSpeed** - the current speed of the wind.
- **maxShipSpeed** - the maximum possible ship speed between *0* and *1*, which is higher for faster ship types and lower for slower ones.
- **ACTO_MOVEMENT_SPEED_FACTOR** - a factor that allows us to control the overall game speed easily.
- **polardiagram.getMaxSpeedForBearing(...)** - the maximum possible driving speed for this specific boat type on the current bearing. This is the most interesting and complex parameter for this calculation. Section 5.5.2.2 shows how it is calculated in detail.

The *newSpeed*, which is the ships new speed, is calculated based on two parameters (lines 19-21):

- **maxBearingSpeed** - the maximum possible speed for the current bearing, calculated above.
- **sailtrim.getSpeedFactor(...)** - the speed factor based on the current sail trim, which is explained in detail in section 5.5.2.2

In order to make the movement smoother and more realistic, we do not just apply the new speed to the ship. We also consider speed changes. So we calculate the according acceleration or deceleration and apply it to the ship speed accordingly.

```
1  // get current bearing (angle between boat and wind)
2  float bearing =
3    windSource.gameObject.transform.rotation.eulerAngles.y -
4    this.gameObject.transform.rotation.eulerAngles.y +
5    180.0f;
6  bearing %= 360;
7  if (bearing > 180) {
8    bearing = bearing - 360;
9  }
10
11 // calculate the maximal possible speed for the current bearing
12 maxBearingSpeed =
13   windSpeed *
14   maxShipSpeed *
15   ACTO_MOVEMENT_SPEED_FACTOR *
16   polardiagram.getMaxSpeedForBearing(bearing);
17
18 // apply the current sail trim to the maximum possible speed
19 newSpeed =
20   maxBearingSpeed *
21   sailtrim.getSpeedFactor(getTrimDegr(), bearing);
22
23 // apply acceleration and deceleration to make movement changes smooth
24 ...
25 }
```

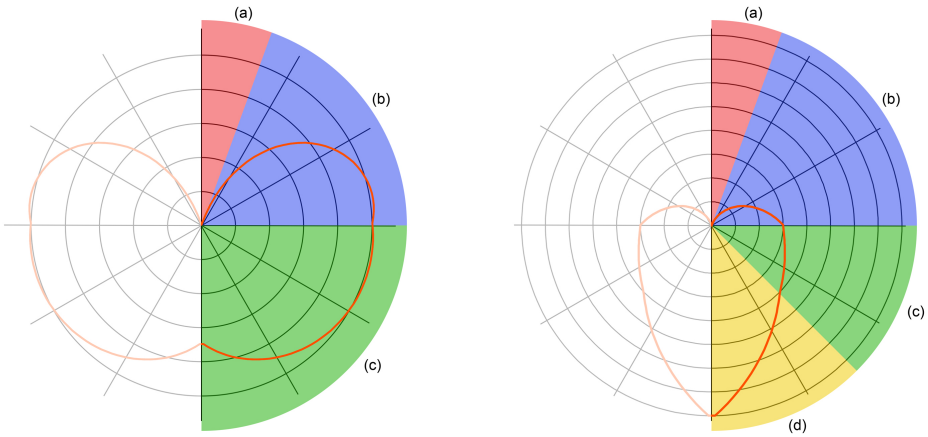
Listing 5.6: Calculate the new driving speed of a ship.

getMaxSpeedForBearing()

This function returns the ship specific maximum speed for a certain bearing, independent of the sail setting. So it basically returns the value of the polar diagram for a certain bearing (See polar diagram 3.8). The polar diagram is a symmetric curve. In our approximation we divided it in several sectors, each of which is represented by an equation, which allows us to calculate an exact value for each bearing. Figure 5.25a shows the three sections *a*, *b* and *c*, we use for calculating the bearing speed for one ship type. While every value in section *a* is *0*, as the ship can not drive against the wind, sections *b* and *c* are represented by a quadratic equation, we defined over several static points along the curve.

getSpeedFactor()

We use a similar approach for calculating the sail trim specific speed factor for a certain bearing. First we divided all bearings in sections, similar as for the maximum bearing speed. This time each section represents an equation that defines the perfect sail trim per bearing. Figure 5.25b shows the four sections we use, while the orange line represents the sail trim. Values close to the center mean the sail is very tight, while values on the outside of the diagram mean that the sail is far opened. All the values in section a are 0 again. Sections b, c and d are represented by quadratic equations.
After calculating the perfect trim for a bearing we compare it with the actual trim and based on the difference between these values we generate the resulting speed factor.
A special case occurs on bearings around 180 degrees, because there the ship should also move when the sail is on the other side. In those cases the wind comes from behind, so it should not make a big difference on which side the sail is. We also consider this special case when comparing the ideal sail trim with the actual sail trim.



(a) Sections of a ship specific polar diagram used for calculating the maximum speed for a certain bearing.

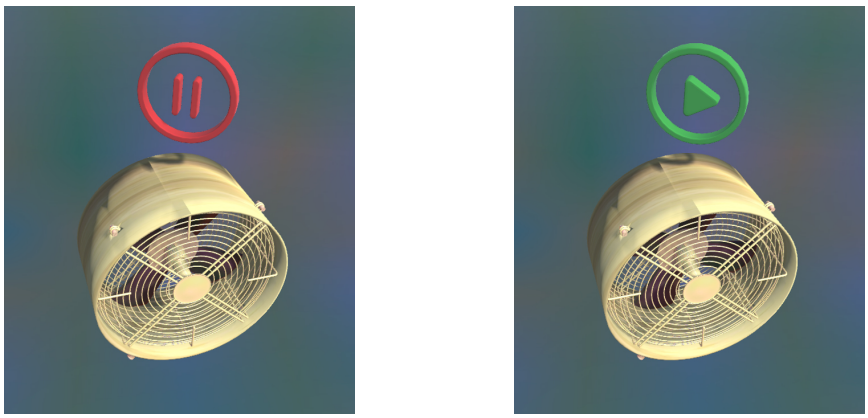
(b) Sections of a ship specific diagram which is used for calculating the perfect sail trim per bearing.

Figure 5.25: Mathematical construction of polar- and sail diagram used for calculating the ship speed.

Game Modes

While testing our system and the interaction we found out that it is often useful to play around with the sail setting without the ship to be moving. Usually if the sail trim is more or less correct, the ship will instantly start moving. But the user sometimes wants to try out other sail settings first, before the ship starts moving. Also it becomes more difficult for the user to set the sails as desired if the ship is already moving.
For this reason we introduced two different *Game Modes*: When the system is in *Live mode*, the ships will move according to the current wind and sail situation. If it is in

Planning mode the ship will stay static, even though it is possible to adjust the sail and use the visualizations to see how the sail setting would affect the driving speed. The user can switch between both modes by clicking the button on the fan. The symbol hovering above the fan will change, as well as its color, according to the current mode (see figure 5.26). Also the color of the second arrow of the wind diagram changes to green or red accordingly (see figure 5.24),



(a) Planning mode. (b) Live mode.

Figure 5.26: The symbol above the fan indicates the current system state to the user.

5.5.2.3 Maneuvers

Besides the user being able to play around with various parameters we also wanted to improve the learning experience by teaching some sailing specific content to the user. Therefore we developed the *Maneuvers* functionality. It allows to introduce the user to various situations that often occur while sailing, by showing him certain courses and standard maneuvers in a Demo mode. The system will visually show how a certain maneuver or course is driven and what the perfect sail setting looks like. Additionally it gives verbal audio explanation and guidance throughout the demonstration. The most important pieces of information are pointed out so the user is able to drive the same maneuver or course himself.
Besides this *Demo* feature we also implemented a *Test* feature, where the user is asked to repeat the previously explained maneuver or course. We implemented a functionality that will give verbal audio feedback to the user's attempts to solve the task. This should improve the learning experience and bring the user close to real world sailing situations. In order to be able to play audio information, we created all the required recordings with a text to speech engine (*TEXT2MP3* [tex]) and stored the results as audio files in the application. All of them are available in German as well as in English language, so the system can also be experienced by international users. The audio files are triggered on the server side but played on all the clients as well, so it is also possible that multiple users solve tasks and learn together.

The maneuver feature was implemented in a way that new courses or maneuvers can easily be added or the existing ones updated. We used a hierarchical model, where most of the work is done in the base class, while the child classes, which represent the actual maneuvers, mainly define the ACTO's waypoints, the sail trim and the audio files to use. With this method we defined three different courses as well as three maneuvers, which we also used for the user study, see chapter 6.2.

5.5.2.4 Networking

For the StARboard server we had to implement two different types of network connections. The connection to the ACTO-Framework and the connections to the StARboard clients. Both types are WiFi connections.

ACTO-Framework Connection

The *ACTOFrameworkConnection* handles both incoming and outgoing networking messages to the ACTO-Framework. The outgoing messages are mainly ACTO driving commands, like moving to a certain point, setting a specific speed or stopping an ACTO. These messages are called from the Game Controller and the Maneuvers, if active.
The incoming messages can be of different types, like tracking information or user interaction, like a button click or a change of an ACTO's potentiometer. All of the received messages are treated as events and collected by the *ACTODataEventQueueManager*. In each game loop all the cumulated events are fired and handled in the *ACTOManager*. There we update existing ACTOs with new tracking data, create new ACTO instances, if they are not yet known or invoke the ACTO objects with updated user interaction, as a button click event.
Similar as in the TrACTOr tracking system we do not just place the virtual objects to the exact new position we received from the tracking. We just use it as a target position and move the 3D object towards it. The further the distance to the new target position the faster it is moved. This reduces inaccuracies from the tracking system and makes the 3D objects appear more stable.
User input events like button click or potentiometer change are passed to the ACTO model itself, which is facilitated by our inheritance model. Dependent on the actual object type, ship or fan, the model initiates the according actions.
A special case occurs if tracking data of unknown ACTOs is received, which means ACTO IDs that the StARboard server did not receive during the current session yet. In this case we create a new ACTO instance from a prefab. This *GameObject*, as they are called in Unity, needs to be instantiated network wide, so it also has to be instantiated on every client, which uses the second type of network connection, explained in the next section.
As the received tracking data is normalized to the tracking marker, see chapter 5.1.3, we needed to make sure, the normalized position data is applied correctly on the server. Therefore we created a GameObject on the server that also renders the tracking marker. We added a child GameObject, called *MarkerCos*, to it, which is used to transform from the normalized tracking data to Unity units. We did this with a very simple trick. The

tracking marker plane always has a size of *10* by *10* units, while the tracking data ranges from *-1* to *+1*. So we just gave the MarkerCos a scale of *5* units. All the ACTO models are moved into the MarkerCos GameObject and in there the normalized tracking data can just be applied directly, with the desired result.
Another problem that arose was the size of the 3D objects we used. After instantiating the new ACTO GameObjects they are moved into the MarkerCos GameObject, which causes a change of their local scale. We have to reapply the original prefab scale after this step to keep the desired 3D model size and stay independent to changes of the tracking marker.

Client Connection

The client connection is a *one to many* network connection. One server provides the data for many clients. All the calculations and interactions are made on the server side, so every change in the visual appearance of the scene needs to be passed to every client. This starts with the exact position and orientation of each ACTO, but also covers changes like an updated sail trim or showing and hiding of polar diagram and wind diagram. To share this sort of information we use *SyncVar*s within the ACTO models. These are basically variables whose value is shared over all the network instances. While these SyncVars are updated on the server, the clients will check for changes of their instance of the SyncVar and update the scene accordingly. An example for a SyncVar as we use it is is shown in listing 5.7.
The shared variable *curSailTrim* is updated on the server with the latest user input data received from the ACTO-Framework. Each client stores a local copy of the last known value in the *localSailTrim* variable. In every update loop each client compares its local copy with the shared one. As soon as they are different, the client applies the new value to the visualization and stores it in the local copy again, until the next change is registered.

```
1  // shared variable
2  [SyncVar]
3  public float curSailTrim = 0;
4
5  // local copy of the variable to detect updates
6  private float localSailTrim = 0;
```

Listing 5.7: SyncVars as shared variables.

We use a similar approach for playing the audio files network wide, which are used in the maneuvers.

5.5.2.5 Graphical User Interface

The GUI of the StARboard server was designed to fulfill two needs. First to give an observing person an overview of the current tracking situation and scene. And second to provide the control mechanisms for connecting to the ACTO-Framework, starting the Unity server and managing other StARboard relevant actions. This can be done by the

user himself or if available, by an operator who controls the system so that the user can focus on the learning system.
The reason why a convenient way of observing the system for outsiders or the operator is important is, because the system is based on AR. This means the person operating the session does not know what the user, who is interacting with it, currently sees. Unless he also wears AR glasses (or other AR display). This is why we implemented the server interface as seen in figure 5.27. It provides a top down view on the whole virtual scene on the StARboard server PC. With the arrow keys of the keyboard the camera can be moved. It can also be zoomed in, out and tilted to allow better viewing angles, which might be helpful in certain situations.

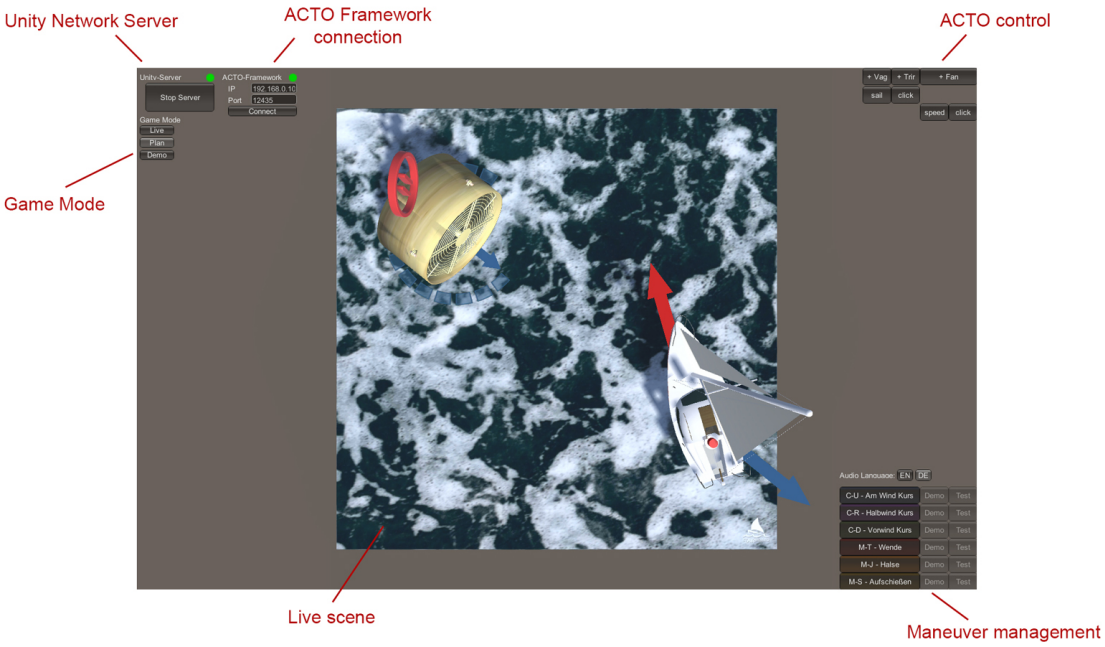


Figure 5.27: StARboard server Graphical User Interface (GUI).

In the top left area of the screen, we placed the networking controls. Here the operator can start the Unity network server PC, as well as establish a connection to the ACTO-Framework. The current status of both network controllers can be seen at any time.
Underneath the current game mode can be changed, e.g. between Live, Planning or Demo mode.
The maneuver controls are located in the bottom right area of the screen. Each row represents one maneuver or course. When the operator clicks on one of the left buttons the according maneuver will be prepared. The ACTOs will move to the starting position of the demonstration. As soon as they are in place the other two buttons of the row, *Demo* and *Test*, get enabled. If the operator clicks the *Demo* button, the maneuver will be explained and demonstrated to the user. A click on *Test* will launch the test for this maneuver, which will also be guided through an audio voice.

On the top right the currently tracked ACTOs are listed. Here the operator is also able to add additional virtual instances of the ships or fans. This is mainly used for testing purpose and the other interaction techniques, we developed for the user evaluation. Read more about this in section 5.5.4.1

5.5.3 StARboard Clients

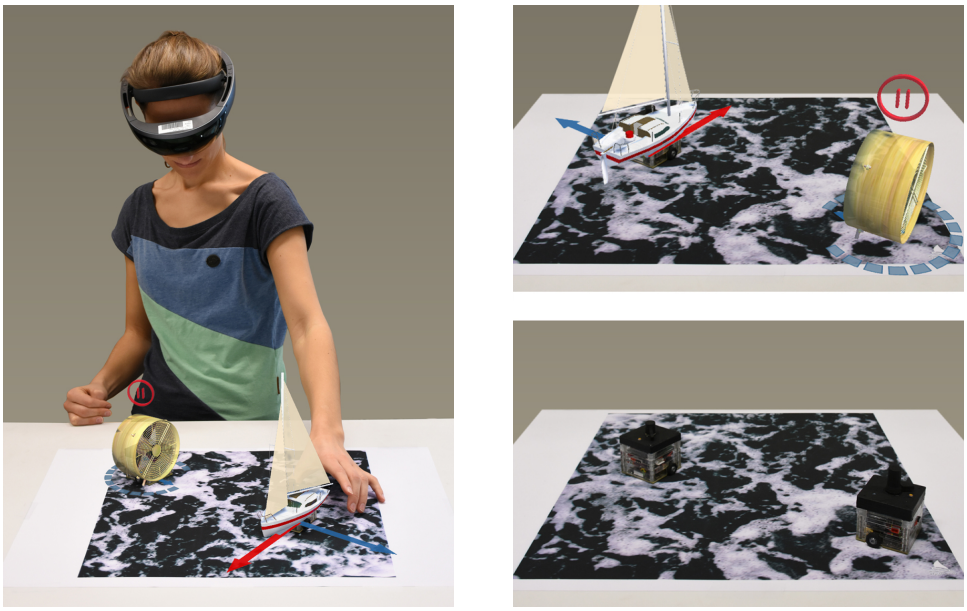
As already explained, the clients we implemented are thin clients. Most of the system logic runs on the server, while the clients are mainly for displaying the shared content. For some interaction techniques they also transfer the interaction to the server (not needed for the standard ACTO interaction technique). The client we developed can be deployed to different kinds of targets with different interaction techniques. Besides the main HoloLens client we also developed an Android client and a desktop client, mainly for the user evaluation, to be able to compare different interaction techniques with each other. All of these applications are based on the same Unity project which can be deployed to different platforms and act as client or server.

5.5.3.1 Marker Tracking

One key feature which is only needed for the HoloLens client and the Android client is the marker tracking. In order to be able to render the scene at the correct position in the real world the tracking marker, which is positioned in the interaction space, needs to be detected. Therefore we integrated the Vuforia tracking library into the StARboard application, as explained in section 4.2.5.5. The center of the marker will act as the origin of the whole scene. While the marker tracking has to be continuous for the Android application, we also use spatial tracking on the HoloLens. This means that the HoloLens creates a 3D scene of the environment the user is in, while using the system. This spatial environment map is used to improve the tracking. So even when the tracking marker is not detected anymore, as soon as its position is determined once, the system will still be able to project the scene to the right point. We still try to detect the marker simultaneously during the whole session in order to improve the accuracy.
During the implementation we faced a problem here, as it was not possible to reposition the whole scene to the center of the tracking marker. The HoloLens Unity application would always have its origin at the position where the HoloLens was in the room, when the app was started. Therefore we used the same principle as we used on the server side for transforming the normalized tracking positions to Unity space. The image marker target in the Unity scene is represented by a GameObject which is positioned at the place where the marker is in the real world. We used a separate GameObject called *MarkerCos*, which we positioned inside it. Every ship or fan GameObject that is created during a session is moved into this *MarkerCos* GameObject. In this way we achieve the right position and scale of the 3D objects in the real world.

5.5.4 Interaction

The main interaction technique we developed uses the ACTO robots as interaction tokens. The user wears the HoloLens and sees virtual objects instead of the ACTOs, as seen in figure 5.28



(a) User interacts with ACTOs in AR.

(b) View from user perspective with and without AR.

Figure 5.28: A user experiencing StARboard with the HoloLens.

The idea is that the user can touch the virtual object and will receive tactile feedback as he touches the ACTO. For the user it should seem as if he would be able to touch the virtual object. Therefore three different ways to interact are supported. He can take the virtual object in his hand, lift it up, rotate it or put it somewhere else. He can touch and push the beam of the sail boat, which is represented by the rotary knob with the beam-extension on top of the ACTO. This will change the actual sail setting of the sail boat. Furthermore, the user can click the red button on the boat in order to activate or deactivate the visualizations polar diagram and wind diagram.
In order to give the best possible haptic feedback to the user the virtual object needs to be overlayed on top of the real ACTO as precise as possible. So the real button should be where the virtual button is displayed and the extension of the rotary knob has to be at the position of the virtual beam. When the user reaches for the virtual object he should touch the real world equivalent.
For the fan the rotary knob represents the rotational knob of a real fan. The same way as for a real fan the user can use it to change the speed of the wind. If he clicks the button on the fan the system switches between the *Live mode* and the *Planning mode*.

To improve the usability of the system even more, we do not only use visual and tactile feedback, we also implemented audio feedback. When the user clicks the button on an object he hears a click sound as feedback.

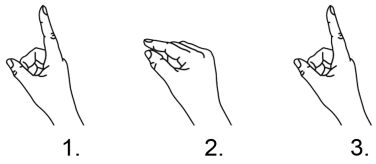
5.5.4.1 Alternative Interaction Techniques for Evaluation

In order to be able to evaluate the presented haptic interaction method we also implemented two other interaction techniques based on more conventional interaction strategies: gesture based interaction and mouse-keyboard interaction. In the user study we compared these three techniques to see if the haptic interaction provides advantages or disadvantages compared to the others.

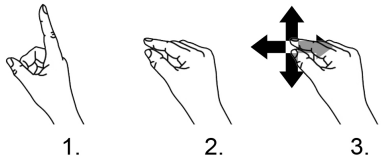
Gesture Based AR Interaction

For the gesture based interaction technique the user also wears the HoloLens, but the ACTO robots are not involved. This means that the objects the user sees are purely virtual and he can not touch them physically (see figure 5.29a).

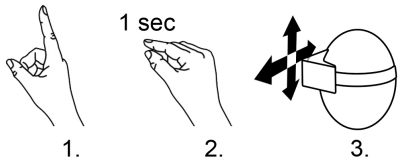
(a) User interacting with virtual objects.



(b) Click to change visualization or game mode.



(c) Pinch to rotate objects and change wind speed and sail setting.



(d) Long click to move objects.

Figure 5.29: Gesture based interaction technique. (Gesture images based on graphics from HoloLens documentation [Hola])

For interaction we used the available standard HoloLens gestures and mapped them to our system. A difficulty was the limited amount of available gestures, as we wanted to provide the same range of possible actions the user has with the ACTO interaction techniques. These are: rotating objects, changing sail setting and fan speed, moving objects and clicking objects.

- **click objects** - All gestures start in the same way, as shown in image 1 in figure 5.29b. If the user brings the index finger and thumb together and apart again, he clicks the object the cursor is currently focusing. The effect is the same as if the user would click the button on the ACTO. At a ship the visualization changes and at the fan the game mode changes.
- **rotate objects** - To rotate an object the user gazes at the desired object and puts the two finger together as before. Instead of moving them apart from each other again, the user can move the hand left or right to rotate the object accordingly (see figure 5.29c).
- **change sail setting and fan speed** - For changing the sail setting the same interaction technique is used, only that the user gazes on the sail, instead of the boat's hull. When gazing on a fan the user can manipulate the fan speed by moving the hand up or down.
- **move objects** - In order to move objects a long click has to be performed, as seen in figure 5.29d. The gesture starts equal than the gestures before, but the fingers stay together and the hand steady for one second. After that the user can move the object by gazing at the intended location for the object. It can only be moved on the surface, so it cannot be lifted into the air.

Mouse Based Desktop Interaction

For this interaction technique the user works with a conventional desktop computer with mouse and keyboard (see figure 5.30). The scene is shown from above, similar as on the server. Again, we wanted to provide the full set of possible interactions, but this time we had to add an additional one. In the previous interaction techniques the user could physically move around to change the perspective on the scene. To achieve a similar effect, the user can move the camera with the arrow keys of the keyboard. All other interaction is done with the mouse.
With the left mouse button the user can click on an object, which is the same as if he would click the button on the ACTO. He can use drag and drop with the left mouse button to move the objects around in the scene. The right mouse button can be used to rotate objects, by dragging them. If the sail is focused and the user drags with the right mouse button he can change the sail setting. The mouse wheel can be used to adjust the wind speed while the cursor hovers over a fan.

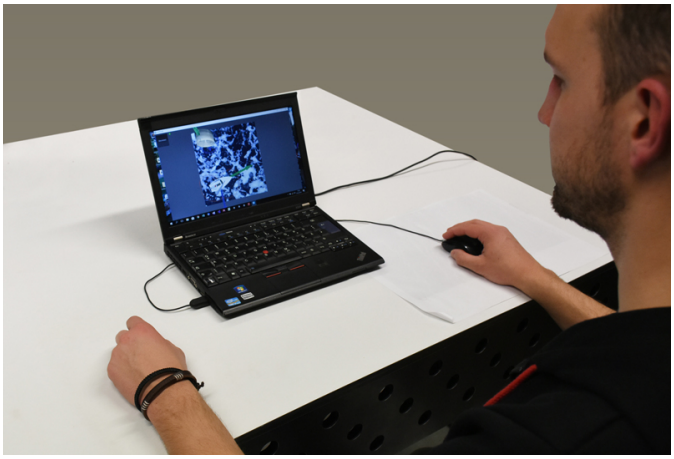


Figure 5.30: Desktop based interaction with mouse and keyboard input.

CHAPTER 6

Evaluation

In this chapter we first describe the results of the technical evaluation of TrACTOr and StARboard. We tested the performance of the tracking algorithm, especially the accuracy and speed of the tracking. The tracking resolution and frame rate are also important parameters we evaluated.
Afterwards we present the user evaluation we conducted. First we give a general introduction into the goals of the study and an overview of the participants and their demographic characteristics. Then we introduce our study design. We explain the components we used and how we sequenced them to receive a meaningful result. Afterwards we describe the tasks for the users. Afterwards we give an overview of the testing process itself and how we carried out the actual tests. Then we present the study results and also interpret them based on our experience. At the end of that section we also give a summary about all the results.

6.1 Technical Evaluation

With the technical evaluation of our system we want to outline the technical capabilities as well as the limitations of the system. For the technical evaluation, as well as for the user study described in the next section, we used the hardware shown in table 6.1.

For the depth tracking we used the Microsoft Kinect 2 as input device. It was placed 72cm vertically above the tracking surface. The optical marker we used was a square of 39 by 39cm. It was positioned centrally underneath the Kinect.
In the TrACTOr tracking software we defined a tracking area of about 50 by 55cm with the tracking marker in its center. This area acts as *region of interest* and everything outside of it is ignored for the tracking process.

One of the evaluated parameters of the tracking system is the frame rate. In order to provide accurate real time tracking the frame rate has to be sufficiently high so that the

	TrACTOr tracking	**ACTO-Framework**	**StARboard Server**	**StARboard Client**
System	Windows 8.1	Android 6.0.1	Windows 10	HoloLens
Processor	Intel Core i7-4790K CPU @ 4.00GHz	Quad-core 2.3 GHz Krait 400	Intel Cor i7-6800K CPU @ 3.40GHz	Custom Microsoft Holographic Processing Unit HPU 1.0, Intel 32-bit architecture
Graphic card	NVIDIA GeForce GTX 980 Ti, 6GB	Adreno 330	NVIDIA TITAN Xp, 12GB	
RAM Memory	32GB	2GB	32GB	2GB

Table 6.1: Hardware Specification Used for Technical and User Evaluation.

virtual image always sticks to the real ACTO robot. This can be achieved with a frame rate of more than 25 frames per second, because this is roughly the amount of frames a user can perceive. To provide an even smoother experience we set 30 frames per second as our target frame rate. This is also the frame rate of the depth camera we used for the tracking, the Kinect 2.
The frame rate is related to the amount of ACTO robots tracked at the same time, because the processing effort increases with every detected ACTO. Therefore we measured the frame rate for different amounts of ACTOs, to be able to compare them. We took measurements for zero, one, two and three ACTO robots tracked at the same time. Two or three ACTOs is the usual amount of robots required for the StARboard application, therefore these are the most relevant results for us.
Our results show that the frame rate of the tracking data sent from the TrACTOr system, for zero, one, two or three ACTOs is always 30fps. This is the maximum possible value, due to the limitation by the camera frame rate.
Additionally we measured the received frame rate at the ACTO-Framework, where the tracking data is processed first. The received frame rate there is always 30fps as well.
This means that the tracking frame rate is sufficiently high for the usage in the StARboard application. As the bottle neck in the tracking system is the frame rate of the camera we also wanted to investigate how fast the actual tracking algorithm is and how it correlates with the tracking of multiple ACTOs. Therefore we measured the time between receiving a frame and when the tracking result is finished. We measured this for zero to three ACTO robots and the processing effort increases linearly, from mean 3.8ms (SD=0.89) for zero ACTOs, to mean 7.1ms (SD=”1.04) for 3 ACTOs. Figure 6.1 shows the linear estimation of how the processing time would increase with additional ACTO robots in the tracking area. With a frame rate of 30 frames per second the algorithm could take 33.3ms for processing, before the tracking frame rate drops under 30fps. The graph shows that the critical point is reached after 26 ACTOs. This means that with this system

configuration it should be possible to track 26 ACTOs at 30fps.

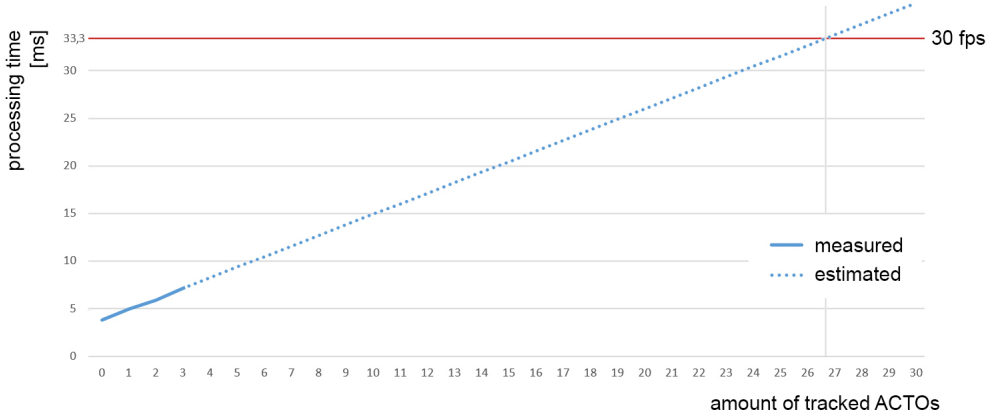


Figure 6.1: Processing time of the tracking algorithm by amount of tracked ACTO robots.

Another important factor of the tracking system is the accuracy of the tracking. To measure this we positioned two ACTOs in known distance of 7.9cm to each other and tracked them. Then we transformed the tracking result to real world distance values. To achieve this we used the known tracking marker size and converted the relative distance values of the tracking system to real world distance values. We repeated this procedure for several positions and relations within the tracking area. The resulting average distance error is mean 1.5mm (SD=0.95) in x as well as y direction.

To measure the jitter we tracked stationary ACTOs on different positions of the tracking surface. We recorded the tracking data over time and then calculated the difference to the average. The mean jitter was 0.46mm (SD=0.42) in x as well as y direction. The mean error in z direction was 4.7mm (SD=2.9) and the mean rotational error was 0.7 degrees (SD=0.54).

As described in chapter 5.2.3 we implemented a two level tracking approach, whereas the first level needs the whole ACTO to be visible for the camera, while in the second level tracking is still possible despite partial occlusion. The jitter measurement above was executed without any occlusion of the ACTO robots. We also performed the same measurement with partial occlusion. To simulate usual user interaction we occluded the ACTO with a hand as if the user would rotate the rotational interface on the ACTO. This resulted in only two of the ACTOs four edges being visible for the depth camera. The edges are the important parameters for level two of the tracking algorithm.
The resulting mean jitter with partially occluded ACTOs is 1.4mm (SD=1.22) in x as well as y direction, 6.3mm (SD=1.25) in z direction and 5.1 degrees (SD=3.1) for the rotation.
A comparison between the jitter error for unoccluded and partly occluded ACTOs can be seen in table 6.2.

The tracking resolution depends on the depth camera resolution and its distance to the

	x [mm]	y [mm]	z [mm]	rotation [deg]
unoccluded (level 1)	0.46	0.46	4.7	0.7
occluded (level 2)	1.4	1.4	6.3	5.1

Table 6.2: Mean jitter error of unoccluded and partly occluded ACTOs in comparison.

tracking surface. In our setup we positioned the Kinect 2 with a depth tracking resolution of 512 by 424 pixels 72cm above the tracking surface. This results in a resolution of 2mm per pixel. As stated previously the tracking accuracy and jitter are below this value. We can achieve this high accuracy due to the rectangle and line detection, as well as the tracking over time.
The tracking area for this setup is about 1 by 0.83m. In order to track a larger interaction space the Kinect could be put further away from the interaction surface. According to the Kinect 2 specifications it has an approximate maximum range of 4.5m. With this distance the maximum possible tracking area size is about 6.3 by 5.2m. But the higher distance of the depth camera would also result in a lower resolution per pixel of 12.3mm per pixel. As the ACTOs have a size of about 5.5 by 5.5cm, the tracking accuracy would be lower accordingly.
The ideal distance between depth camera and interaction surface depends on the use case and the application. For the StARboard application we recommend a distance of about 72cm, based on our practical experience.

The time between the first frame when an ACTO robot is visible for the depth camera until the first tracking result is sent, is the latency of the ACTO identification. It is also dependent on the ACTOs actual id, because the length of the id codes differs, which means that the optical transmission process does not take the same amount of time for different ids. The mean measured latency for the initial ACTO recognition with the id 1 is 420ms. This means that 420ms after the ACTO first becomes visible to the depth camera, the first tracking data package is sent. As soon as the ACTO is identified once, this id detection does not have to be performed anymore, so it does not play a role in the continuous tracking process, only for the initial detection. For the ACTO with the id 3 this initial detection time is 650ms. For an ACTO with id 26 it would take about 1680ms. This time could be reduced if the used patterns for id detection would be changed. We choose this pattern for the StARboard application as it is optimized for a lower number of ACTOs and for StARboard we only need two or three ACTOs at the same time.

For the StARboard application the pattern is sufficient, as only two or three ACTOs are used at the same time.

For the StARboard application the whole system round trip time is an important factor. This is the time between receiving a depth frame at the tracking system until the 3D model is rendered at the according position on the HoloLens. So the time between an action happens until it is visually reflected to the user within the AR environment. During this process the tracking data needs to be calculated on the tracking system, transmitted to the ACTO-Framework, where it is processed and transmitted to the StARboard server.

There it is applied to the 3D scene and finally it is sent to the StARboard HoloLens clients. The mean measured time for this process is 110ms (SD=42).

6.2 User Study

The aim of the user study was to evaluate whether the usage of haptic feedback in combination with Augmented Reality (AR) could improve learning with a learning system. Therefore we wanted to know how easy and intuitive the system is to use. The aim was to enable the user to focus on the learning task and not on handling the system. Another aspect we wanted to explore is how much fun the users have while using it, because learning is much more efficient if the user enjoys what he is doing.

A total of 18 people between the age of 22 and 57 participated in the study (mean 31.5, SD 8.4), 5 of them female, 13 male. Most of them stated that they had little to no previous experience with the HoloLens, with an average of 1.9 (SD=1.1) on a scale from *1 - Not at all* to *5 - Very much*. On the other hand many of them already had some experience with AR in general, with an average of 2.9 (SD=1.4). The general usage of computers was rated relatively high amongst most of them, with an average of 4.6 (SD=0.9).
As the learning system deals with sailing we also asked about the preknowledge there. Most of the users had no sailing experience at all, some had a little and a few had much, with an average of 1.8 (SD=1.1).

6.2.1 Study Design

In order to be able to give a statement about whether the usage of haptic feedback and AR in a learning application is useful or not, we decided to compare this interaction technique with two others. As described in section 5.5.4.1 we implemented two additional interaction techniques for the same learning system. As the quality of these alternative interaction techniques has a high impact on the result of this study we tried to implement them as good as the haptic one. We also made sure to provide the same range of functionality for all three. The learning system stays the same, only the way of interaction changes. This allows us to compare them objectively. The three interaction techniques are:

- AR with haptic feedback, using the HoloLens for visualization and the ACTO robots for interaction. From now on also referred to as *AR-H*
- AR with gesture control, using the HoloLens. *AR-G*
- Conventional desktop application with mouse and keyboard control. *D-M*

See section 5.5.4 for a detailed description.

Each user tested all three of the interaction techniques. In order to compensate learning effects the order was changed for each user. Also the order of the given tasks was changed in a way that after all 18 users each combination was tested equally often.

After each interaction technique the users tested, they had to fill out the same questionnaire, which consisted of a combination of different tests: The first section was the System Usability Scale (SUS), by Brooke [Bro96]. In the second section we mixed several questions of other standardized questionnaires with questions we designed ourselves. We used four questions of the Task Load Index (TLX) by Hart [Har06], about *Frustration Level*, *Mental Demand*, *Performance* and *Temporal Demand*. Furthermore we added questions to measure the *absorption by activity*, which is a component of the *flow experience*, defined by Rheinberg et al. [REV02] in the Flow Short Scale (FSC). We also included some questions to evaluate the degree of immersion during the experience.
Additionally we measured if the users physical condition changed during the AR sessions, due to Visually Induced Motion Sickness (VIMS). Therefore we used the method introduced by Hwang et al. [HDGP17], to measure VIMS with one single question, which is based on the Wong-Baker FACES [IC03]. It is a simple version to evaluate the physical wellbeing in a single question, which the users had to answer directly before and after using the HoloLens.

6.2.2 Tasks

Every user got several tasks for each interaction technique. We divided the tasks in two sets, the warmup tasks and the tasks we actually measured.
The intention of the warmup tasks was to make the user more familiar with the interaction technique and let them explore all the available possibilities. Four different warmup tasks were given for each technique, whereas the first one was the easiest and they grew more complex one by one. The tasks covered all available types of interaction, as well as all the relevant game mechanics that are needed to solve the actual tasks. The result of these tasks was not measured.
After finishing the warmup tasks the user had to do the two actual tasks. The first one was a certain bearing that had to be driven, the other one a maneuver. The exact process of these tasks was as follows: At first the objects (fan and boat) moved to their starting position. Then the system introduced the user to the maneuver that will follow by explaining it orally. After that the boat moved along the given path, showing the correct sail setting. While the user watched the boat move, the system explained more about the sail trim and the driven maneuver. After the demonstration was finished the boat moved back to its initial position and the testing mode started. The system asked the user to apply the correct sail trim for the given course. For maneuvers where the boat has to rotate during the task (tack, jibe, shoot up), the system drove the first part of the maneuver itself for the test. The sail trim had to be adjusted by the user after the boat rotated to the new bearing. We measured the success of the task based on the amount of tries it took the user to successfully finish it. A task was classified as successful if the chosen sail trim lead to a new boat speed of at least 60% of the maximum possible speed for this bearing.

6.2.3 Testing Process and Execution

As already described, the main part of the test consisted of three sections, one for each interaction technique. The author guided the participants through the user test as the instructor. The whole process of one test session can be seen in table 6.3.

Introduction	General Introduction	
	Demography and Preknowledge questionnaire	
	Introduction StARboard	
Main Test	Interaction Technique 1	Introduction
		(Pre questionnaire)
		Training
		Warm up tasks
		Tasks
		Post questionnaire
	Interaction Technique 2	...
	Interaction Technique 3	...
Recapitulation	Sailing questionnaire	
	Comparing questionnaire	

Table 6.3: Sequence of a user test.

After the general test introduction the users had to fill out the introductory questionnaire, to collect demographic information and also receive information about their preknowledge about sailing, computers, AR and the HoloLens specifically.
Before we started with the first interaction technique, we gave them an introduction to the learning environment StARboard itself. We explained its purpose, all the available elements, how they function and what they can do with them. We also showed them examples of the visualizations (polar diagram, wind diagram) and explained their meaning and how they could be used.
After that the main section of the test started. As already explained, the order here was different for every user, but the general structure was the same over all three interaction techniques. First we gave them an introduction to the current interaction technique. We explained them the available interaction mechanisms and what they are used for. Especially the two AR techniques required an explanation for users unaccustomed to the HoloLens. For AR-H we showed them the robots and how they can be handled, for AR-G we explained the gestures already before they were wearing the HoloLens.
For the AR techniques they had to fill out the pre question about VIMS and then the interaction technique was started. We went through all available interaction mechanisms together again, so they had enough time to try out each of them. Then we gave them the four warm up tasks, to get accustomed to the handling. After this we executed the actual tasks, as described in the previous chapter 6.2.2. Then they filled out the post questionnaire about the according interaction technique. Afterwards we started with the second and third interaction technique in the same manner.

After the user had tested all the three interaction techniques we came to the final section of the user test. There the user first had to answer some short test questions about sailing, in order to evaluate the learning result. It consisted of 9 questions in randomized order, about correct sail trim and bearings. Finally the users filled out the last section of the questionnaire, which was about the comparison of the three techniques. Finally we asked the testers for a brief oral comparison of the three techniques. The average time of a full test session was about 1 hour 15 minutes (SD=0.22).

The test setup can be seen in figure 6.2. The test person stands in front of the interaction space or sits at the laptop for D-M. The instructor controls the whole setup, the TrACTOr, the StARboard server, the ACTO-Framework and starts the ACTOs.

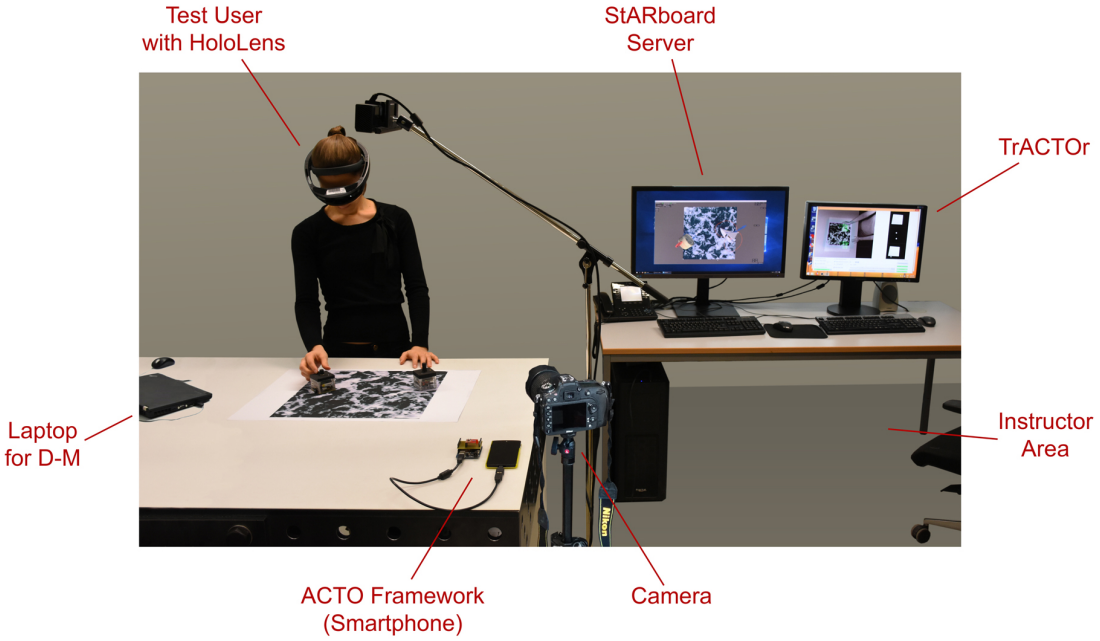


Figure 6.2: Session of a user test.

For better evaluation we recorded all test sessions with a video camera, and we used screen recording to record the StARboard server and TrACTOr tracking. Additionally the instructor made notes during the tests.
In order to prevent mistakes and to ensure a consistent test procedure we prepared a printed overview of all the steps the instructor had to do for each user. This also included the correct order of the interaction techniques and tasks for each user. After every section the instructor ticked the according box on the sheet. Each interaction technique and task got a separate color to avoid mistakes and to make finding things easier.
Before every test we prepared water and snacks for the user, to keep the motivation higher.

6.2.4 Results

In this section we present the outcome of the user study. We show the results of the questionnaire evaluation and explain our interpretation of the resulting values.

6.2.4.1 SUS

As already described, the first section of the questionnaire was the standardized SUS test. The result is shown in figure 6.3.

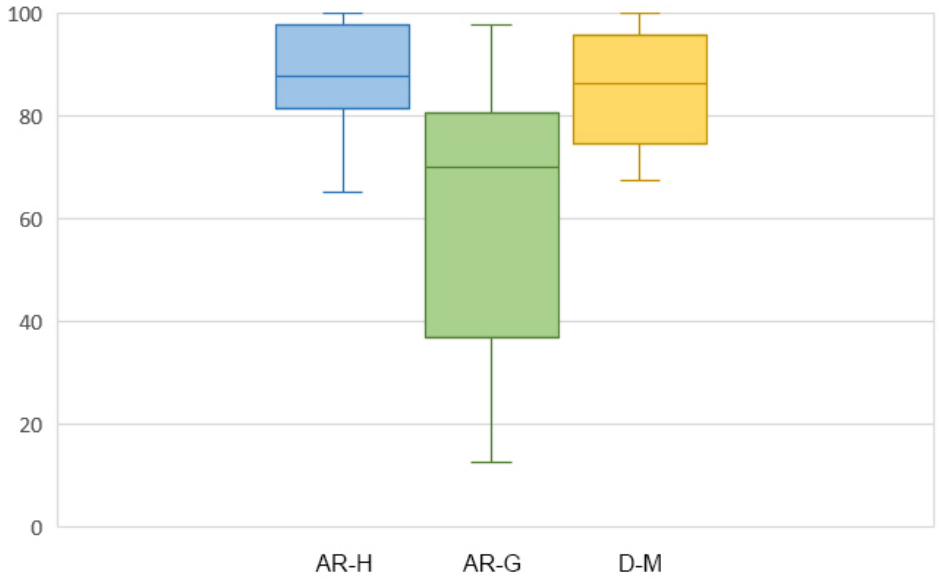


Figure 6.3: SUS results overview.

It shows that the mean of the interaction technique *AR-H* (AR with haptic robot interaction) is 86.7 (SD=11.3). According to Sauro [Sau11] this can bee seen as a very good result.
The interaction technique *AR-G* (AR with gesture interaction) got a mean of 59.6 (SD=26.8). The mean result of this technique is below average and has to be considered as not good enough. The high standard deviation correlates to the observations that we made during the user tests. For some users the gestures worked quite well, for others they did not work properly at all. The range here was very broad. We assume this is due to the fact that most of the users have not worked with the HoloLens or other gesture based systems before and are not familiar with this kind of interaction. This indicates that this kind of interaction technique has a steep learning curve which resulted in problems for some of our users as the actual testing sessions were fairly short. At this point we need to mention that we gave the users proper instructions to the different gestures and did a short training without and later with the HoloLens before starting the actual tasks.
The mean SUS result of *D-M* (desktop interaction with mouse and keyboard) was 82.9

(SD=15.2). This can also be considered as a good result, similar to *AR-H*. Most of the test users work with desktop computers very often and are quite accustomed to this kind of interaction. We assume that this fact allowed the users to learn and understand this interaction technique quickly, which might play an important role in this outcome.

The result of the SUS evaluation shows that the techniques *AR-H* and *D-M* have a good usability. They can be learned quickly and are easy to understand and accomplish. In comparison *AR-G* has a far worse usability, which makes the interaction with the system more difficult. Especially for a learning system this could become a problem, because users can not focus on the learning content sufficiently, if they have troubles with the system interaction.

6.2.4.2 TLX

From the results of the TLX questions especially the questions about frustration level and mental demand show significant differences throughout the three interaction techniques. The results can be seen in figure 6.4.

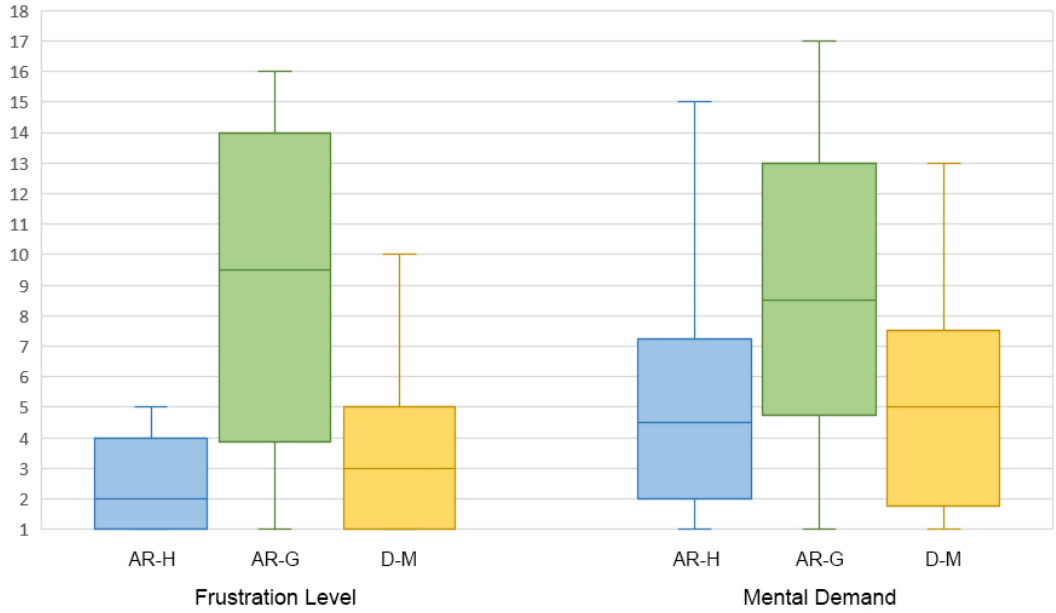


Figure 6.4: Results of the most significant TLX questions, frustration level and mental demand.

The mean frustration level of *AR-H* is 2.7 (SD=2.1). A similar result can be seen at *D-M*, with a mean of 4.0 (SD=3.6). Even though *AR-H* has a slightly better result, both can be seen as very good ratings on a scale from 1 to 18. It shows that both interaction techniques are very fluent and fulfill the users expectations. *AR-G* on the other hand

received an average frustration level of 9.0 (SD=5.3). This again means that the score varies greatly over the users. While for some users the frustration level was equally low as the one of the other two techniques, for many it was very high. It seems that for a lot of users, this interaction technique does not support them sufficiently in what they intend to do.
A similar result can be seen in the mental demand. While *AR-H* has a mean of 5.6 (SD=4.2) and *D-M* a mean of 5.2 (SD=3.9), the result of *AR-G* is a lot higher. The mean is 8.8 (SD=4.5). The higher the mental demand for an interaction technique is, the less focus can be put on the learning system and the actual tasks. This means that the learners can not concentrate on the actual tasks that good, as they are busy with handling the system. For us this is an indicator that *AR-G* would not be as suitable for a learning system, at least without more intensive usage and training of the interaction technique.

6.2.4.3 Immersion

With these questions we wanted to measure how well the interaction technique boosts the degree of immersion into the system. The result can be seen in figure 6.5.

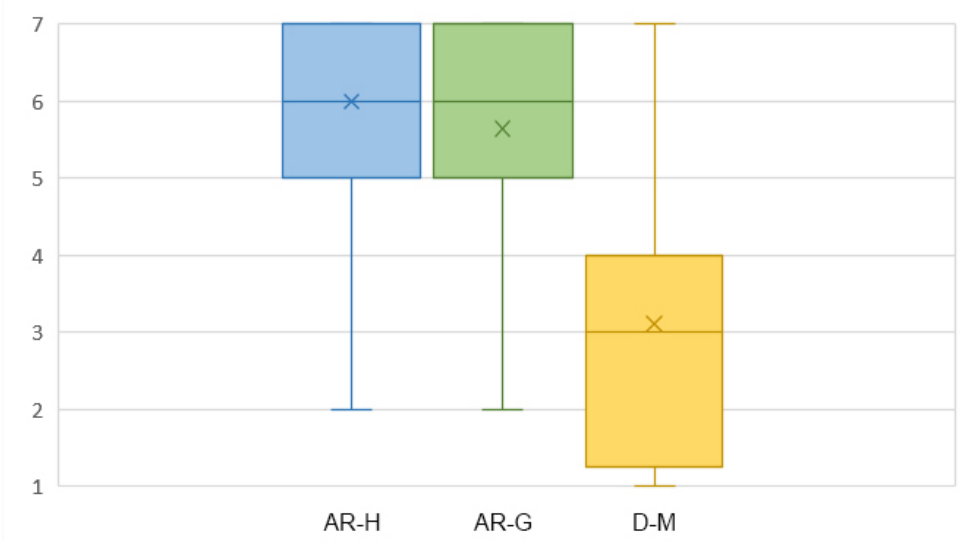


Figure 6.5: Result of the questions about the degree of immersion.

It shows that both AR techniques reached a very high rating. The mean value of *AR-H* was 6.0 (SD=1.3). *AR-G* received a mean of 5.6 (SD=1.4). The scale was 1 to 7. The mean value is represented by the *X* in the diagram. While both of the AR techniques seem to have a very high level of immersion, the result of *D-M* was notably lower, with a mean of 3.1 (SD=1.9).

We assume that the reason for this result is that the AR techniques provide a three dimensional environment which allows the user to move around freely, by physically

changing his position in this environment. Also the user is an active part of this environment. For *D-M* the user only sees a two dimensional representation of a 3D space. He is outside of the space and can only look at it from above. A more complex camera control might have slightly improved the result here, but it would also have made the interaction technique more complex, which might have caused downsides as well.

6.2.4.4 FSC - Absorption by Activity

With parts of the FSC questionnaire we wanted to measure the *absorption by activity*. We think that this plays a crucial role, especially for learning systems. The outcome can be seen in figure 6.6.

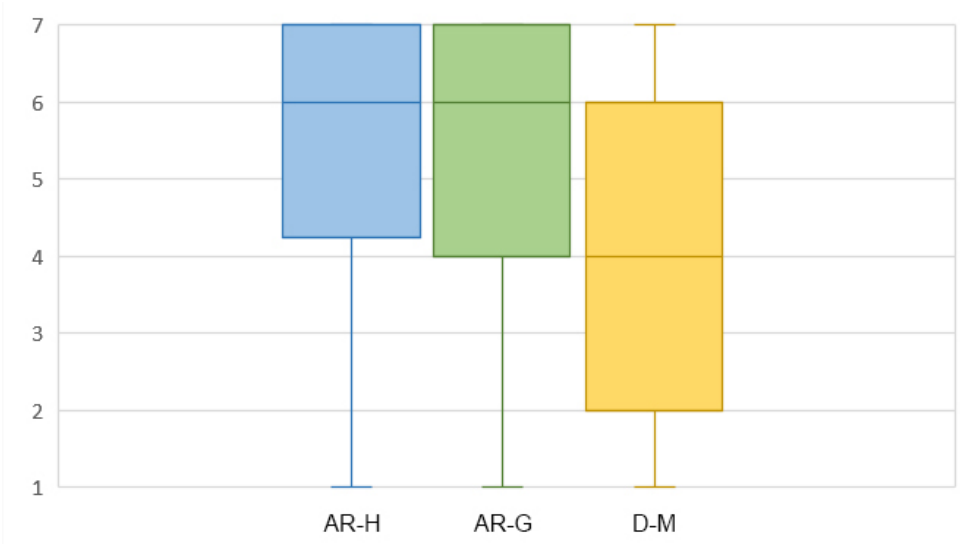


Figure 6.6: Resulting values of the *absorption by activity* from the FSC.

It shows that the results of *AR-H* and *AR-G* are quite similar. While *AR-H* has a mean of 5.4 (SD=1.7), *AR-G* has a mean of 5.3 (SD=1.5). The scale was 1 to 7. This shows that the users enjoyed to work with the interaction techniques. They were absorbed in what they were doing and had fun with the interaction. The mean value of *D-M* is much lower at 4.2 (SD=1.9).

In a real learning situation this could mean that the users would be more likely to study more and longer with one of the AR techniques, as they seem to be more absorbing.

6.2.4.5 Task Results

We also measured the outcome of the tasks we gave to the users. They were counted as + if the result was correct with the first attempt, ~ if they needed two or three attempts and - if it took them four or more attempts.
The result can be seen in figure 6.7. It does not show notable differences between the

three interaction techniques. We assume that for measuring the efficiency in resolving a task a bigger group of test people with longer sessions on each interaction technique would be required.

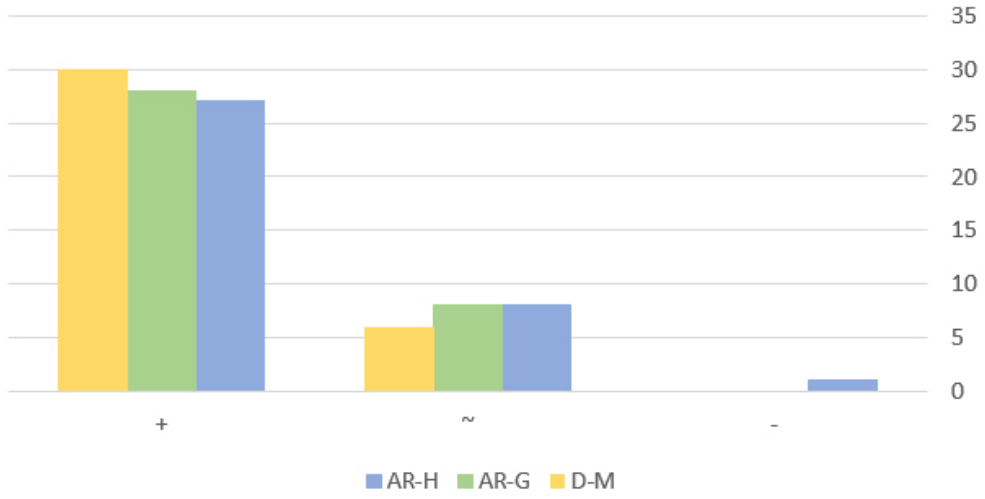


Figure 6.7: Task result based on amount of attempts.

The graphic also shows that most of the tasks were fulfilled at the first attempt. This could indicate that the difficulty of the given tasks was rather low. Since the test sessions should not be excessively long and we targeted beginners, we avoided too complex tasks. In the next development step of this learning system the tasks could become more elaborate.

6.2.4.6 Learning Result

We found a similar situation in the result of the questionnaire we designed for measuring the learning outcome. The questionnaire was designed in a way that every user got nine questions. Three groups of three questions related to the tasks they did with one interaction technique. In average the users got 6.4 (SD=1.8) of 9 questions right, which is about 71%. There was no big difference in the results between the interaction techniques. The average result for *AR-H* was 70%, for *AR-G* 69% and for *D-M* 74%. As already discussed in the previous section we think that the learning sessions were too short to get a reasonable result about their influence in the learning process.

6.2.4.7 Open Questions

In the last section of the questionnaire 13 of the 18 users stated that *D-M* interaction was easy to use. Many justified this with the fact that they were used to working on desktop computers. A total of 9 users mentioned that *AR-H* was intuitive and easy to use. Many found it very useful that they could move around in the interaction space freely, which gave them a very good overview and good control for both, *AR-H* and

AR-G. On the other hand 11 users noted that the interaction with *AR-G* was difficult or the gestures did not work properly.
In general the users found *AR-H* and *D-M* easier to learn than *AR-G*. They stated that due to the short time of usage for each interaction technique they often had to concentrate on the interaction itself so much, that they could not sufficiently focus on learning. With longer usage cycles we expect that they would become more familiar with the interaction and they could concentrate better on learning. This applies to all three techniques.
The question about which technique was most fun to use was answered by 13 users with *AR-H*. 3 chose *AR-G* and none of them *D-M*. Many state the reason that it was easy to use and it was something new, which they had not used before. The interaction with the robots was mentioned particularly often positively, especially in combination with AR.
The question which interaction technique they would find most useful for learning for themselves was answered with *AR-H* by 9 users. 4 named *D-M* and 2 *AR-G*. The high immersion of *AR-H* would allow to focus on the learning tasks and it would also be fast and easy to use.
Both AR techniques, *AR-H* and *AR-G* were mentioned to be very immersive by several users. They found the scene very realistic and easy to picture, which allowed them to put more focus on the tasks and learning itself. Some users noted that *D-M* was not very immersive, especially the camera angle was criticized a few times.
8 users mentioned that the tracking started to jitter, especially while they interacted with the robot. This sometimes impacted the immersion for them and made precise adjustments more difficult. According to some of them, an improvement in the tracking would enhance the whole learning system.
Also the technical limitations of the HoloLens affected the results. As several users stated, the FOV was so small that it limited their immersion and made the interaction for *AR-G* and *AR-H* more complex. We observed that many users stepped further back from the table during the interaction with *AR-G*. This enabled them to see the whole scene at once, without having to turn their head from one object to the other. When standing directly in front of the table they were not able to see all virtual objects of the scene at the same time. This was a problem for *AR-H*, because there some of them wanted to step back as well, but then reaching for the ACTOs to carry out the interaction would be difficult.
Some users also mentioned that the weight of the HoloLens caused them light pain in their neck, which might become a serious problem for longer learning sessions.

6.2.4.8 Summary

In summary *AR-H* and *D-M* received similarly good results in system usability and task performance sections of the questionnaire, with a slightly better tendency for *AR-H*. But in immersion and *absorption by activity* *AR-H* was rated much better than *D-M*. This gives us the conclusion that *AR-H* is potentially superior to *D-M*. *AR-G* reached similar results for immersion and *absorption by activity* but was rated far worse in system usability and task performance.

The technique *AR-H* offers a high degree of immersion, is very intuitive to use, has a flat learning curve and provides a fun alternative to conventional computer systems. On the other hand it requires a complex setup. Considering it is just a prototype, many things could be simplified there. But some of the hardware is not as far developed, that it could provide a trouble-free usage. For example the small FOV and heavy weight of the HoloLens.
Also *AR-G* has a very high degree of immersion and provides an easy way to drop into a virtual scene. Beside the HoloLens, and its problems we already mentioned, almost no hardware is required. Also the interaction was very smooth and intuitive for some users. But for many others this was not the case. We believe that with more experience this issue might become less prominent.
D-M comes along with the convenience of a standard computer setup as most people are used to work with. This advantage allows users to quickly adapt to the system with less barriers at the beginning. It is also easy to run in various locations because the required hardware is present in many places. For longer sessions this technique might be less fun and less interesting to the users, also because of its smaller degree of immersion.

In regard of the research topic, whether haptic feedback in combination with AR would improve learning with a learning system, or not, we conclude that it most likely would. It is more fun to use, the direct interaction makes it possible to focus on the learning tasks and gives a very high level of control of the system. All of these factors provide a promising basis for a good learning environment. In order to backup this proposition better, more research needs to be done to measure the actual learning effect, when such a system is used over a longer period of time, in comparison to a conventional system.

CHAPTER 7

Discussion

In this chapter we discuss the outcome of our research. We compare the results of the evaluation and our lessons learned with the research questions we wanted to answer in the beginning. Afterwards we briefly talk about possible improvements of the system.

First we wanted to investigate whether we could replace the existing ACTO robot tracking system with a new depth based system, which would eliminate several problems of the old system and allow us the combined usage of the ACTO interface and AR. The technical evaluation and also the result of the user evaluation show that we reached this goal with the implementation of the *TrACTOr* tracking system. It allows us to replace the previously required glass table with almost any kind of solid, flat surface. We found only one limitation, which is reflecting surfaces, which disturbs the depth image of the Kinect camera.
We could also show that the Kinect based tracking system offers high accuracy of 1.5mm in x and y direction for a 50 x 55cm tracking area, for the tracking of ACTO robots with the Kinect version 2. The only downside we could asses was the problem with occlusion of the ACTOs during user interaction. As long as the ACTO robots are fully visible to the depth camera, the tracking is very accurate and stable. With our two level tracking algorithm the tracking of ACTOs is also possible when the ACTO is partly occluded by the users hands. But during this occlusion phases the tracking sometimes shows more jitter. The jitter for partly occluded ACTOs increases from 0.46mm to 1.4mm in x and y direction. This was also mentioned as an issue by multiple users during the user evaluation. Especially the ACTO rotation becomes inaccurate to several degrees, from jitter 0.7° to 5.1° when occluded. Even small inaccuracies of a few degrees can cause the virtual objects to appear a bit shaky. The main reason for this problem is that the depth image is not that accurate and the ACTOs' borders in the depth image change a bit from frame to frame. We tried to ease this out with a Kalman Filter (see section 3.5), but the issue still persists to a certain extent.
With the depth based tracking system we could show that we are able to accurately track

x, y and z coordinates. Besides the issue already mentioned also the rotation along the z axis is functional and accurate. We could show that the method of determining the exact rotation with the additional IR light, which is mounted on top of each ACTO, works very good (jitter 0.7°). For the combined usage with the AR system we had to synchronize the COSs to have the same origin and distance unit scale. We achieved this with a visual tracking marker which allows us to transform all of the tracked x, y and z position values, as well as the z rotation, to the marker COS. This gives us much freedom with the positioning of the Kinect and the tracking marker. It also makes the setup process very fast and easy.

We reached the goal of being able to track multiple ACTO robots at the same time. This was a very important step in order to be able to compete with the original tracking system and allow the designed user interfaces to be a lot more complex, as multiple ACTOs can be used. We use the same IR LED as for the rotation detection. It flashes in certain predefined patterns which allows us to distinguish between them. As already discussed in section 6.1, the critical factors for the amount of ACTOs is the duration of the detection algorithm and the camera frame rate. As we could show the amount of ACTOs that could be used simultaneously, without a reduction of the tracking frame rate of 30fps, is about 26. The IR LED of each of them flashes in a different pattern, so we can differentiate between them. The amount of ACTOs tracked at the same time does not influence the tracking accuracy.

This leads us to our next goal. We wanted to explore whether it was possible to implement this tracking system in a way, that it is fast enough to deliver real time tracking data. This is required to provide the AR overlays and also to control the ACTO movement. As already mentioned the performance of the tracking algorithm is very good. In tracking sessions with one ACTO it takes 5.9ms from receiving a frame to having the final tracking data. This increases linearly by about 1.1ms per ACTO. With a target tracking frame rate of 30fps for a real time tracking system, TrACTOr is able to track 26 ACTOs simultaneously. This means that TrACTOr is able to collect and stream tracking data in real time over the network, in order to be processed and used by the other components of the system. The amount of 26 ACTOs should be sufficient for most use cases.

Tests also showed that the system can be used by multiple users at the same time without any problems, because the game logic is centralized. The whole scene data is distributed from this central server to as many clients as desired, where the rendering is carried out independently. This is important for creating collaborative user interfaces, such as collaborative learning systems.

Another question was, if the accuracy and speed of the tracking is high enough to provide good alignment of the AR content and the real world ACTOs. The very good result of the SUS questionnaire, the low Mental Demand in the TLX questionnaire, as well as the high rating of immersion, give us good reasons for assuming that we reached this goal. The TrACTOr system's good tracking accuracy, already mentioned above, in combination with the high quality tracking of the HoloLens, using Vuforia [vuf] for marker tracking, allows us to align the AR objects and the real world objects very accurately. This allowed us to achieve our goal of building an interface that provides human-real world interaction.

The second topic we wanted to examine was whether haptic interaction in combination with AR would be beneficial for learning systems. The user evaluation we executed shows that the StARboard system provides a very good usability, even slightly better than the one of the desktop interaction. We could show that the system provides a very immersive experience and an intuitive way of interaction. The users rated the system as very absorbing and the frustration level and mental demand were rated quite low. All of these factors can be a very good basis for a learning application, as described in section 2.1.5. A system that is very absorbing and fun and easy to use is likely to be used more often and with a positive attitude, which are also beneficial factors for a learning environment. With the user evaluation we could show that a learning system with haptic interaction and AR is very likely to have positive effects on learning efficiency and the learning results, but the observed tendency is not sufficient to solidly support this conclusion. In order to be able to give a certain answer to this question a more detailed investigation in the learning outcome would be necessary. The focus of our evaluation was on the comparison of this interaction technique with an equivalent gesture based interaction technique and a desktop based interaction technique with mouse and keyboard interaction. The result shows that it has several advantages compared to the gesture interaction and got also slightly better results than the desktop based interaction. This leads us to the conclusion that out of these three possibilities, the haptic interaction with AR is the most suitable interaction technique for a learning system.
Concerning the gesture based interaction there is one additional issue we want to point out. The result of the gesture based recognition during the user evaluation was very diverging. We observed that for some users, the gesture based interaction worked quite good, while others had huge problems with this interaction technique. This was not due to preknowledge, because most of the test users did not have used gesture based interaction before. The range of different gestures provided by the HoloLens is quite limited and the available gestures are very similar. For example it is not possible at the moment, to distinguish between left and right hand.
The amount of time where the users actually worked with the interaction technique was relatively short, about five to eight minutes. And even though we gave them detailed instruction about the different gestures before they started wearing the HoloLens and repeated the gestures together, as soon as they were wearing it. We think that with longer sessions the users would become more accustomed to this interaction technique. Longer test sessions could also affect the results of the other interaction techniques, but especially for the gesture based interaction we think that it might make a difference for some users.
Another question was, if we would be able to provide human-real world interaction with StARboard. Especially for the sail boat this target could be reached. The alignment of the virtual object and the interaction objects is very good. The high degree of immersion and low mental demand, as well as the general user feedback indicate that this kind of interaction was very intuitive. Especially the setting of the sail was very realistic, as it appears to the users to push the virtual sail, whereas they really push the physical beam extension. This direct interaction was a bit worse for the fan, because of an unideal

choice of the 3D model. The rotary knob of the ACTO is not reflected by the visual appearance of the 3D model. This was not a big issue, because this problem only occurs when the user tries to change the wind speed, but the main focus of the interaction was moving the objects and changing the sail trim, which was very well reflected by the virtual 3D objects. A different 3D model for the fan would probably increase the ease of interaction even more.
We also wanted to examine, whether a system like this could increase the learning motivation. The observed high level of immersion and absorption could indicate that a system with this kind of interaction causes higher learning motivation, compared to similar systems.
Another question we wanted to answer was, if collaborative learning would be possible with such a system. The system meets all necessary requirements for multiple users to share the learning experience. It is possible that multiple users interact with the ACTO robots simultaneously. Also the demos and tasks, which the system gives, can be experienced by multiple users at the same time, even with different kinds of AR displays (e.g. HoloLens, smartphone). They can cooperate to solve the tasks together or use individual ACTOs to try out certain sail trims.
All in all the StARboard application can be seen as a prototype that indicates the potential of a haptic learning system in combination with AR. In general the usage of haptic interaction in combination with AR proved to be a very promising concept for building good learning systems.

7.1 Possible Improvements

Event though the TrACTOr tracking is very stable if the ACTOs are fully visible to the depth camera, it sometimes becomes unstable during user interaction, if the users hands partly or fully occlude the ACTO robots. As already mentioned, this was mentioned by a few users to interfere with the immersive experience. One possible solution to solve this issue would be a second depth camera, mounted in a different angle, so that every ACTO is always visible from at least one camera.
Another detail which is currently limited by the technical possibilities of the HoloLens is the visually correct imaging of the users hands and the virtual 3D objects. Currently it is not possible to occlude the 3D models with the hands. The 3D models are always rendered on top of all the objects in the interaction space. This is not a major issue, but it leads to visual incorrectness between the real and virtual 3D scene. To solve this issue the users hands would need to be tracked and reflected by occlusion models in the 3D scene. The use of different AR glasses could be considered, but currently these features are in development phase for various AR systems. At the moment there is no better solution. But in several months new AR glasses generations will probably solve this issue out of the box.
So far the StARboard application is only a prototype of a learning application. Besides the functionality of testing different sail settings and bearings in various wind situations we implemented the maneuver feature, where the user has to apply the correct sail trim

during different maneuvers. To improve the learning experience it would be a good idea, if the user would also have to apply the bearings and bearing changes during the maneuvers, as if he would drive them himself. We covered the three most important maneuvers and the three most important bearings, but more of them could be implemented.
As already described the 3D model of the ship perfectly fits the physical construction of the ACTO module. On the other side the fan model could be improved, so that its controls also exactly match the ACTO module.

CHAPTER 8

Conclusion and Future Work

In this thesis we implemented a depth based tracking system for ACTO robots. It is capable of tracking x, y and z position coordinates, as well as the rotation along the y axis of multiple ACTO robots at the same time. The tracking system can be used for various ACTO based applications, such as StARboard, which we also developed in this thesis. The StARboard application is an Augmented Reality (AR) based learning application, which uses haptic feedback to improve the user experience and learning effects.

The technical evaluation showed that the TrACTOr tracking system has a very high accuracy with a mean tracking error of less than 1.5mm. Up to 26 ACTO robots can be tracked simultaneously with a tracking frame rate of 30 frames per second, which is the upper limit of the depth camera we used. These factors enabled us to develop the collaborative learning application StARboard, where multiple users can gain basic sailing specific knowledge about how to drive certain courses and maneuvers. This was evaluated in a user study with 18 participants. They tested three different interaction techniques for the sailing application. One was a desktop based interaction with mouse and keyboard, another one a gesture based AR application using the HoloLens, and the AR based application using our haptic interaction technique. The results of the evaluation show that the haptic based interaction is very easy and convenient to use and can be learned and understood quickly, similar to the desktop based interaction. In terms of immersion and absorption by activity the haptic interaction outperformed the desktop interaction. Out of these three, the haptic interaction can be seen as the most promising technique for learning systems of this kind.
Although the results on the efficiency of the learning itself, or the actual effects on the learning outcome are not conclusive, many users chose the haptic interaction as their desired interaction technique for a learning system and the overall results seem very promising that such an interaction technique can have positive effects on learning. An additional study with focus on the learning outcome would be very interesting here.

In the future the TrACTOr tracking system could be improved in terms of occlusion resistance, so that the jitter of 3D objects gets reduced during user interaction. This would allow us to achieve even better immersion. The StARboard application could be extended, so that it is capable of teaching even more maneuvers and other situations that typically occur during sailing. Another interesting improvement would be teaching of sailing specific vocabulary, such as naming of certain parts of the ship. It could also be used to show and teach different sailing relevant knots and how to tie them. Then the StARboard application could be used in sailing schools to teach sailing students and improve the theoretical sailing lessons with practical examples and more realistic situations than drawings in books or displayed videos. It could also be considered to focus the application development more on children's needs, by giving it a more playful character, like little games or tasks, which they need to solve.

The principle of haptic feedback for AR based learning systems could also be applied to other fields, such as aeronautic, medicine, music, arts, industry or building engineering. An interesting use case would be to experience the statics of buildings and different materials. Building engineering or architecture students could use this technology to get a better feeling for the load carrying capacity of different types of materials or structure techniques. They could use their own force or experience the pressure which the virtual buildings are exposed to and how the situation changes if different materials are used or if the structure is planned differently. This could give them a very different access to this topic, in comparison to the currently used techniques, such as computer simulations or drawings and calculations.

Of course haptic interaction for AR cannot only be used in learning environments but also for totally different scenarios. One field where haptic feedback and AR would be very interesting is product design, so that the designers can quickly and easily try out different shapes or structures during the design process. Another large and interesting field is entertainment. The technology could be used to provide a new kind of interactive 3D experience similar to theater or movies. The users could become active parts of the scene and sense the 3D scene not only with their eyes and ears, but also with their hands. They could even become the most important players in the scenes themselves, which leads us to the next field of games and sports. With highly interactive interfaces, the border between computer games and physical sports starts to diminish and new types of activities could be thought of, which have a game like character and are beneficial for the users health at the same time. Prototyping tools as the ACTO robots could play an important role in developing those kinds of new interfaces.
Augmented Reality (AR) with haptic feedback is a relatively new field of research with a lot of potential, but also not negligible problems which needs to be tackled before it can be applied to new areas. If we are able to solve these challenges, this technology has the potential to revolutionize many different fields.

List of Figures

List of Tables

Listings

Acronyms

AR Augmented Reality. 2–5, 7, 14–16, 26–28, 36, 44, 52, 64, 71, 76, 77, 86, 88, 89, 96–99, 101, 103, 104, 106, 107, 109–112, 115, 116

AV Augmented Virtuality. 14

COS Coordinate System. 29, 33, 35, 36, 44, 48, 54, 56, 64, 65, 110, 118

FOV Field Of View. 57, 106, 107

FSC Flow Short Scale. 98, 104, 119

GUI Graphical User Interface. 10, 24, 45, 51, 52, 56, 58, 65, 66, 74–76, 85, 86, 118

HCI Human Computer Interaction. 1, 7

HMD Head Mounted Display. 26, 28, 36, 47

OS Operating System. 67, 71

RF Radio Frequency. 23, 24, 35, 53

SUS System Usability Scale. 98, 101, 102, 110, 118

TCP/IP Transmission Control Protocol/Internet Protocol. 53

TLX Task Load Index. 98, 102, 110, 118

TOF Time-of-Flight. 22, 23

TTL Time To Live. 62, 63

TUI Tangible User Interface. 1, 2, 5, 7–17, 20, 23, 24, 31, 33, 34

VIMS Visually Induced Motion Sickness. 98, 99

VR Virtual Reality. 14, 27

Bibliography

[ABB+01] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, 2001.

[Ard] Arduino. https://www.arduino.cc/, accessed on: 10th feb 2017.

[aut] Autodesk 123D Design, v 2.2.14. https://www.autodesk.com/solutions/123d-apps.

[Azu97] Ronald Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[BJIU+01] E Ben-Joseph, H Ishii, J Underkoffler, B Piper, and L Yeung. Urban Simulation and the Luminous Planning Table: Bridging the Gap between the Digital and the Tangible. *Journal of Planning Education and Research*, 21(2):196–203, dec 2001.

[BK13] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd edition, 2013.

[BM86] W. Buxton and B. Myers. A study in two-handed input. *ACM SIGCHI Bulletin*, 17(4):321–326, apr 1986.

[BM07] S. Benhimane and E. Malis. Homography-based 2D visual tracking and servoing. *International Journal of Robotics Research*, 26(7):661–676, 2007.

[Bro96] John Brooke. SUS - A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[Can86] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[FAMA10] Barak Freedman, Shpunt Alexander, Meir Machline, and Yoel Arieli. Depth Mapping Using Projected Patterns. *WO Patent WO2008*, 1(19):12, 2010.

[Fia05] Mark Fiala. ARTag, a Fiducial Marker System Using Digital Techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 590–596. IEEE, 2005.

[FIB95] George W. Fitzmaurice, Hiroshi Ishii, and William A. S. Buxton. Bricks. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*, pages 442–449, New York, New York, USA, 1995. ACM Press.

[FLO+13] Sean Follmer, Daniel Leithinger, Alex Olwal, Akimitsu Hogge, and Hiroshi Ishii. inFORM. In *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*, pages 417–426, New York, New York, USA, 2013. ACM Press.

[fri] fritzing, v 0.9.3. http://fritzing.org/.

[Har06] Sandra G. Hart. Nasa-Task Load Index (NASA-TLX); 20 Years Later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 50(9):904–908, 2006.

[HDGP17] Alex D Hwang, Hongwei Deng, Zhongpai Gao, and Eli Peli. Quantifying Visually Induced Motion Sickness (VIMS) During Stereoscopic 3D Viewing Using Temporal VIMS Rating. *Journal of Imaging Science and Technology*, 61(6):60405—-1, 2017.

[Hola] HoloLens Documentation. https://docs.microsoft.com, accessed on: 12th apr 2018.

[Holb] Microsoft HoloLens. https://www.microsoft.com/microsoft-hololens/en-us, accessed on: 15th feb 2017.

[IC03] The National Initiative and Pain Control. Wong-Baker FACES Pain Rating Scale. *National Initiative on Pain Control*, pages 2–3, 2003.

[Ish08] Hiroshi Ishii. Tangible bits: beyond pixels. *Proceedings of the 2nd international conference on Tangible and Embedded Intreaction (TEI '08)*, pages xv–xxv, 2008.

[IU97] H. Ishii and B. Ullmer. Tangible bits: towards seamless interfaces between people, bits, and atoms. *Proceedings of the 8th international conference on Intelligent user interfaces*, (March):3–3, 1997.

[JGAK07] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reacTable. In *Proceedings of the 1st international conference on Tangible and embedded interaction - TEI '07*, page 139, New York, New York, USA, 2007. ACM Press.

[JLDS13] Jungong Han, Ling Shao, Dong Xu, and Jamie Shotton. Enhanced Computer Vision With Microsoft Kinect Sensor: A Review. *IEEE Transactions on Cybernetics*, 43(5):1318–1334, oct 2013.

[KBS94] Paul Kabbash, William Buxton, and Abigail Sellen. Two-handed input in a compound task. In *Proceedings of the SIGCHI conference on Human factors in computing systems celebrating interdependence - CHI '94*, pages 417–423, New York, New York, USA, 1994. ACM Press.

[KD07] Hannes Kaufmann and Andreas Dünser. Summary of Usability Evaluations of an Educational Augmented Reality Application. In *Virtual Reality*, pages 660–669. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Kho12] K. Khoshelham. ACCURACY ANALYSIS OF KINECT DEPTH DATA. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-5/(August):133–138, sep 2012.

[Kina] Kinect2 Specs. https://developer.microsoft.com/en-us/windows/kinect/hardware, accessed on: 5th feb 2017.

[Kinb] Microsoft Kinect v1. https://msdn.microsoft.com/en-us/library/jj131033.aspx, accessed on: 5th feb 2017.

[Kinc] Microsoft Kinect v2. http://www.xbox.com/en-GB/xbox-one/accessories/kinect-for-xbox-one#fbid=1OFOTWGbdlB, accessed on: 7th dec 2016.

[Kir95a] D Kirsh. Complementary Strategies : Why we use our hands when we think. *Seventeenth Annual Conference of the Cognitive Science Society*, (Lave 88):212–217, 1995.

[Kir95b] David Kirsh. The intelligent use of space. *Artificial Intelligence*, 73(1-2):31–68, feb 1995.

[KSN+06] Minoru Kojima, Maki Sugimoto, Akihiro Nakamura, Masahiro Tomita, Masahiko Inami, and Hideaki Nii. Augmented Coliseum: An Augmented Game Environment with Small Vehicles. In *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems (TABLETOP '06)*, volume 2006, pages 3–8. IEEE, 2006.

[Li14] Larry Li. Time-of-Flight Camera–An Introduction. *Texas Instruments - Technical White Paper*, (January):10, 2014.

[met] Metaio. http://www.metaio.eu/, accessed on: 24th mar 2017.

[MI94] Christine L. MacKenzie and Thea Iberall. Chapter 5. Movement Before Contact. In *Advances in Psychology*, volume 104, pages 109–201. 1994.

[Mic] Microsoft. www.microsoft.com, accessed on: 5th feb 2017.

[MK94] Paul Milgram and Fumio Kishino. Taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, E77-D(12):1321–1329, 1994.

[MMES04] Carsten Magerkurth, Maral Memisoglu, Timo Engelke, and Norbert Streitz. Towards the Next Generation of Tabletop Gaming Experiences. *Proceedings of Graphics Interface 2004*, pages 73–80, 2004.

[MRD06] Ali Mazalek, Mathew Reynolds, and Glorianna Davenport. TViews: An Extensible Architecture for Multiuser Digital Media Tables. *IEEE Computer Graphics and Applications*, 26(5):47–55, sep 2006.

[NYQ28] H. NYQUIST. Certain Topics in Telegraph Transmission Theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928.

[PAH14] Danakorn Nincarean Eh Phon, Mohamad Bilal Ali, and Noor Dayana Abd Halim. Collaborative Augmented Reality in Education: A Review. In *2014 International Conference on Teaching and Learning in Computing and Engineering*, pages 78–83. IEEE, apr 2014.

[PFSR09] Sara Price, Taciana Pontual Falcão, Jennifer G. Sheridan, and George Roussos. The effect of representation location on interaction in a tangible learning environment. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction - TEI '09*, page 85, New York, New York, USA, 2009. ACM Press.

[PI07] James Patten and Hiroshi Ishii. Mechanical constraints as computational constraints in tabletop tangible interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, page 809, New York, New York, USA, 2007. ACM Press.

[PIHP01] J. Patten, H. Ishii, J. Hines, and G. Pangaro. Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 253–260, 2001.

[PNO07] Ivan Poupyrev, Tatsushi Nashida, and Makoto Okabe. Actuation and tangible user interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction - TEI '07*, page 205, New York, New York, USA, 2007. ACM Press.

[Pon09] J.G. Pontual Falcão, T., Price, S., Sheridan. Extending concepts of engagement in tangible environments, 2009.

[PRI02] James Patten, Ben Recht, and Hiroshi Ishii. Audiopad: A Tag-based Interface for Musical Performance. In *Proceedings of the Conference on New Interfaces for Musical Expression (NIME'02)*, pages 1–6, 2002.

[REV02] F. Rheinberg, S. Engeser, and R Vollmeyer. Measuring Componants of flow: The Flow-Short-Scale, 2002.

[RN95] Jun Rekimoto and Katashi Nagao. The World through the Computer: Computer Augmented Interaction with Real World Environments. *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 29–36, 1995.

[RUO01] Jun Rekimoto, Brygg Ullmer, and Haruo Oba. DataTiles. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '01*, pages 269–276, New York, New York, USA, 2001. ACM Press.

[RWB02] H.T. Regenbrecht, M. Wagner, and G. Baratoff. MagicMeeting: A Collaborative Tangible Augmented Reality System. *Virtual Reality*, 6(3):151–166, oct 2002.

[Sau11] Jeff Sauro. Measuring Usability With The System Usability Scale (SUS). *Measuring Usability*, pages 1–5, 2011.

[Sha09] Orit Shaer. Tangible User Interfaces: Past, Present, and Future Directions. *Foundations and Trends® in Human–Computer Interaction*, 3(1-2):1–137, 2009.

[Shi13] S Shi. *Emgu CV Essentials*. Community experience distilled. Packt Publishing, 2013.

[SJZD11] Bertrand Schneider, Patrick Jermann, Guillaume Zufferey, and Pierre Dillenbourg. Benefits of a Tangible Interface for Collaborative Learning and Interaction. *IEEE Transactions on Learning Technologies*, 4(3):222–232, jul 2011.

[STMTK01] Norbert A Streitz, Peter Tandler, Christian Müller-Tomfelde, and Shin Konomi. Roomware - Towards the Next Generation of Human-Computer Interaction Based on an Integrated Design of Real and Virtual Worlds. *Human-Computer Interaction in the New Millennium*, pages 553–578, 2001.

[SWBP13] Bertrand Schneider, Jenelle Wallace, Paulo Blikstein, and Roy Pea. Preparing for Future Learning with a Tangible User Interface: The Case of Neuroscience. *IEEE Transactions on Learning Technologies*, 6(2):117–129, apr 2013.

[tex] TEXT2MP3. https://www.texttomp3.online/, accessed on: 23rd aug 2018.

[UI99] J. Underkoffler and H. Ishii. Urp: A luminous-tangible workbench for urban planning and design. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 386–393, 1999.

[UIG98] Brygg Ullmer, Hiroshi Ishii, and Dylan Glas. mediaBlocks. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, volume 32, pages 379–386, New York, New York, USA, 1998. ACM Press.

[ult] Ultimaker 2. https://ultimaker.com/.

[Uni] Unity. https://unity3d.com/de, accessed on: 22nd mar 2017.

[VGK14a] Emanuel Vonach, Georg Gerstweiler, and Hannes Kaufmann. Acto. *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces - ITS '14*, pages 259–268, 2014.

[VGK14b] Emanuel Vonach, Georg Gerstweiler, and Hannes Kaufmann. ACTO. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces - ITS '14*, pages 259–268, New York, New York, USA, 2014. ACM Press.

[vuf] Vuforia. https://www.vuforia.com/, accessed on: 10th feb 2017.

[WB06] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. *In Practice*, 7(1):1–16, 2006.

[Wil08] Ryan M. Wilson. The physics of sailing. *Physics Today*, 61(2):38–43, feb 2008.

[WRM+08] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Pose tracking from natural features on mobile phones. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 125–134. IEEE, sep 2008.

[WSJB10] Malte Weiss, Florian Schwarz, Simon Jakubowski, and Jan Borchers. Madgets. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*, page 293, New York, New York, USA, 2010. ACM Press.

[XP16] Z Xu and T Perry. Time of flight camera, 2016. US Patent App. 14/524,814.