Visualization of Robotic Sensor Data with Augmented Reality

Improving the observer's understanding

Mathias Ciarlo Thorstensen

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017

Visualization of Robotic Sensor Data with Augmented Reality

Improving the observer's understanding

Mathias Ciarlo Thorstensen

Visualization of Robotic Sensor Data with Augmented Reality

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

To understand a robot's intent and behavior, a robot engineer must analyze data at the input and output, but also at all intermediary steps. This might require looking at a specific subset of the system, or a single data node in isolation. A range of different data formats can be used in the systems, and require visualization in different mediums; some are text based, and best visualized in a terminal, while other types must be presented graphically, in 2D or 3D. This often makes understanding robots challenging for humans, as it can be hard to see the whole picture of the situation.

This thesis attempts to solve this issue, by creating an augmented reality system on the virtual reality platform HTC Vive, to investigate methods for visualization of a robot's state and world perception. It also investigates the effect augmented reality has in increasing a user's understanding of a robot system.

The visualization was achieved by projecting a robot's sensor data into the user's reality, presenting it in a intuitive way. Augmented reality was achieved by utilizing HTC Vive's front facing camera, and showing the *augmented* video see-through in virtual reality. To test the system's ability in increasing the user's understanding, a user study was conducted. The study tested the users' understanding of the robot's perception of its environment. This was done by comparing the augmented reality system with traditional methods.

The implemented augmented reality system was successfully tested on 31 subjects in the user study. Quantitative data was recorded to measure the understanding, and a questionnaire was conducted to get qualitative data about the system. The results show a significant increase in the subjects' understanding.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals of the thesis . . . 2

2 Background **3**
- 2.1 Virtual reality . . . 3
  - 2.1.1 Teleoperation . . . 4
  - 2.1.2 In education . . . 5
- 2.2 Augmented reality . . . 5
  - 2.2.1 Achieving augmented reality . . . 6
  - 2.2.2 Pose estimation - Computer vision . . . 7
- 2.3 Projection . . . 8
  - 2.3.1 The perspective / pinhole camera model . . . 9
- 2.4 Robotics . . . 11
  - 2.4.1 Robot perception . . . 11
  - 2.4.2 Previous work in the field of augmented reality and robotics . . . 12
- 2.5 Understanding a robot . . . 12
  - 2.5.1 Human-robot interaction . . . 12
- 2.6 System testing . . . 13
  - 2.6.1 Evaluation of an information system . . . 13
  - 2.6.2 Statistical method . . . 14

3 Tools and software **17**
- 3.1 HTC Vive . . . 17
- 3.2 ROS - the robotic operating system . . . 17
  - 3.2.1 Intel RealSense depth camera . . . 19
- 3.3 Unity . . . 22
  - 3.3.1 Scripting . . . 23
  - 3.3.2 Unityros . . . 26
- 3.4 OpenCV . . . 26
  - 3.4.1 ArUco . . . 26

4 Implementation **29**
- 4.1 System design . . . 29
  - 4.1.1 The robot . . . 29
  - 4.1.2 The augmented reality computer . . . 32

4.2 Planning the user study . . . 41
4.2.1 AR only experiments . . . 42
4.2.2 Experiments where AR is compared to the traditional method . . . 43
4.2.3 Test design . . . 43
4.3 Framework for the user study . . . 44
4.3.1 The faulty object detection algorithm . . . 44
4.3.2 The physical setup . . . 46
4.3.3 Data collection and questionnaire . . . 46
4.3.4 System architecture . . . 48

5 Experiment and results **51**
5.1 The study . . . 51
5.1.1 The invisible object test in detail . . . 51
5.1.2 Experiment setup . . . 51
5.1.3 Questionnaire . . . 55
5.1.4 Execution of the study . . . 55
5.1.5 Pilot studies . . . 57
5.2 Main experiment results and analysis . . . 58
5.2.1 Efficiency results . . . 59
5.2.2 Efficiency analysis . . . 60
5.2.3 Accuracy results . . . 60
5.2.4 Accuracy analysis . . . 63
5.3 Questionnaire results and analysis . . . 64
5.3.1 Quantitative data . . . 65
5.3.2 The 31 subjects preferring augmented reality . . . 65
5.3.3 Solving the tasks with the traditional method . . . 66
5.3.4 Difficulties with the traditional method . . . 69
5.3.5 Solving the tasks with augmented reality . . . 70
5.3.6 Difficulties with the augmented reality system . . . 70
5.3.7 Previous experience . . . 71

6 Discussion **73**
6.1 The study . . . 73
6.1.1 Thoughts on the study design . . . 73
6.1.2 A deeper analysis of the scenes and cubes . . . 74
6.1.3 Bias in the study . . . 75
6.2 Visualization in augmented reality vs. virtual reality . . . 76
6.3 Augmented reality as visualization tool . . . 77

7 Conclusion and future work **79**
7.1 Conclusion . . . 79
7.2 Future work . . . 79
7.2.1 Applying the system to a real robot . . . 79
7.2.2 Visualization of decision . . . 80
7.2.3 Human-robot interaction . . . 81
7.3 Future development . . . 81
7.3.1 Efficiency . . . 82

7.3.2 Stability 82
7.3.3 Visual quality 83

List of Figures

2.1 Reality-virtuality continuum 4
2.2 The pose of coordinate system B relative to A 8
2.3 The perspective / pinhole camera model 9
2.4 The extrinsic part of the perspective camera model 10
2.5 Perspective / pinhole camera model, the intrinsic part 11

3.1 HTC Vive 18
3.2 Screenshot from rviz 19
3.3 Intel RealSense f200 depth camera 19
3.4 Byte structure of a point cloud entry 20
3.5 Live sensor_msgs/pointcloud2 message 21
3.6 Depth image from depth sensor 22
3.7 Screenshot from the Unity editor 23
3.8 Monobehaviour flowchart 24
3.9 *ArUco* marker 27

4.1 The system setup 30
4.2 Image of the robot prototype. 30
4.3 Pixel coordinates to 3D transformation 36
4.4 First attempt at augmented reality 39
4.5 The coordinate frames of the Vive, *ArUco* and sensor 39
4.6 Comparison of point cloud sources 40
4.7 Robot sensing an obstacle 41
4.8 Object masking 45
4.9 Images of the cube prototypes. 47
4.10 ROS graph during the user study 48
4.11 Adjusting the cube masks 49

5.1 Experiment environment. 52
5.2 The subject's seat and computer mouse to control the view of the point cloud on the laptop. 53
5.3 The subject's view of the robot and scene. 53
5.4 The 5 scenes used in the experiment. 54
5.5 Main experiment data collection form. 55
5.6 Experiment form example 56
5.7 Time spent on different tasks 59
5.8 Time efficiency for the traditional method 61

5.9 Histograms showing the total number of failed tests per subject in both methods. 61
5.10 The probability of failing at a given task 63
5.11 Error distribution in the scenes 64
5.12 Quantitative results. 67

6.1 Part and point cloud, compared visualization methods . . . 76

7.1 Human and robot 80
7.2 Human and robot interacting 81

List of Tables

4.1 Overview of software used in the robot prototype. 30
4.2 Measured bandwidth usage 34
4.3 Measured frame rates in Unity 37

5.1 Flow chart of main experiment procedure 57
5.2 Summary of time spent on the different tasks 60
5.3 Summary of time efficiency 60
5.4 The number of failures per subject with the two methods . . 62
5.5 The probability of failing at a given task 62
5.6 The number of severe errors 62

Acknowledgements

My deepest appreciation goes to my supervisor: PhD Candidate Tønnes Nygaard for an incredible effort, through inspiration, support, and humor. You have made these two years a pleasure. Thank you.

A special thanks goes to postdoctoral fellow Charles Martin, who has been helping me by sharing his experience in user studies.

Another thanks goes to the staff at ROBIN, and the great dual boiler espresso machine, which has been very supportive, supplying me with 1200 lattes[1].

I would also like to thank the lecturers from the computer vision course at UNIK, for their enthusiasm and inspiration. A special thanks goes to Trym Haavardsholm for taking the time to help debugging a specific part of the system, in times of depression and despair.

Another thanks goes to Bjørn Ivar Teigen and Emilie Hallgren for exceptional modeling, and everyone who took the time to participate in the user study.

I would also like to express my sincere gratitude for everyone who are close to me, supporting me through this work. This would not have been possible without you.

[1] Access to the espresso machine was granted in September, 2015. As of May, 2017, this is 20 months ago. 3 lattes a day, 5 times a week equals 1200 lattes.

Chapter 1

Introduction

Robots are becoming an increasingly bigger and important part of our society. They are used in a wide range of applications, from consumer luxury aid, for example vacuum cleaners and lawn mowers, to military and deep sea operations, where it is dangerous for humans to operate. This range use of cases is often called *The three D's*, standing for *Dirty*, *Dangerous* and *Dull*. When creating a robot, robot engineers have to plan, design, build, program, and test the robot. While the physical part of the implementation can be hard enough, the process of testing and tuning the parameters of the algorithms is often tedious and time consuming.

1.1 Motivation

Many robots are dependent on the ability to sense their environment. Autonomous robots are a good example, since they need to make decisions based on their surroundings. Without this information, they cannot navigate or send information about their current situation to their operator. The information usually comes from different sensors on the robot, for example laser range scanners and stereo cameras.

When a robot engineer is creating a robot, he or she needs to understand how the robot makes its decisions. The decisions can be based on multiple factors, such as internal state and sensor data. Unfortunately, robots do not have a good way of expressing their internal data or state, which can make robot engineering a challenging task.

The elements of the decision base are separated. For example, the current ways of visualizing state are often text-based, while 3D sensor data typically is shown in 3D programs. Second generation information, like the output from an algorithm processing sensor data can be hard to visualize. Getting an overview of the robot's situation can thus be challenging.

What if one had the ability to see the robot's plan of action, and which elements that lead to its decision? Getting this information into our point

of view, instead of on a computer screen would likely make it easier to understand the robot. This thesis attempts to solve this problem, by presenting an augmented reality system for visualization of robotic sensor data.

1.2 Goals of the thesis

The goals of this thesis are formulated below:

1. Investigate visualization of robotic sensor data through augmented reality using the virtual reality platform HTC Vive
2. Test the system's effectiveness in improving the user's understanding of sensor data

Chapter 2

Background

This chapter gives an overview of the field of mixed reality, as well as the required background knowledge for the implementation of a robotic augmented reality system. Applied techniques for system testing and validation are also covered.

2.1 Virtual reality

Virtual reality has a wide range of applications, from surgery to entertainment. The first references to virtual reality dates back to 1935, from a science fiction story called *Pygmalion's Spectacles* by Stanley G. Weinbaum, describing a pair of virtual reality goggles showing the user fictional content, including touch and smell. In 1999, projective virtual reality was found as a new way to control and supervise robotic systems[1]. The word projective is used because actions from the user in the virtual reality system are projected into the real world, for example through a real robot.

Mixed reality In 1994, Paul Milgram and Fumio Kishino published the article *A taxonomy of mixed reality visual displays*, where they define the term *mixed reality*. The term was defined as the area between the real and the virtual environment, along the so called *reality-virtuality continuum*. This area includes the sub categories *augmented reality* and the less famous *augmented virtuality*. The latter basically means the opposite of augmented reality - augmentation with real content on top of a virtual environment, that is, see figure 2.1. Milgram and Kishino stated that a taxonomy, or a classification framework for the virtuality continuum was needed since there was no such established framework in the community.

> *"The purpose of a taxonomy is to present an ordered classification, according to which theoretical discussions can be focused, developments evaluated, research conducted, and data meaningfully compared."*[2].

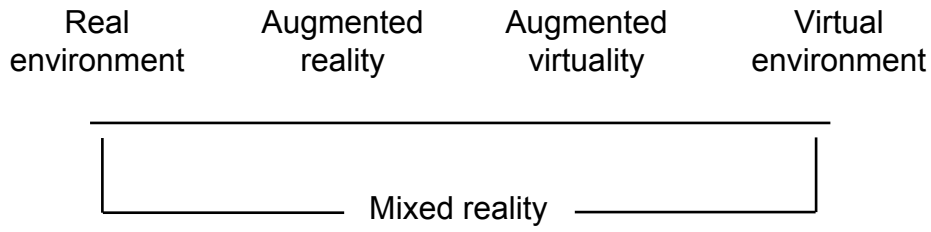


Figure 2.1: Reality-virtuality continuum

Multi-sensory feedback and operator performance A group researching human-robot interaction (HRI) studied how one can utilize the performance of a robot operator by using multi-sensory feedback interfaces. They mapped sensor data from simulated robots to different senses on the operator. The different types of sensory feedback used was visual (LCD display), audio, vibration, touch, and smell. Multiple studies were done and they found that using multi-sensory feedback was beneficial for increasing operator performance[3].

2.1.1 Teleoperation

Some places are hard to reach for humans, due to dangerous environments. In the oil industry, welders have to dive to the bottom of the sea to do maintenance. This is both extremely dangerous and expensive. By using virtual reality, the welder is able to *become* the robot, and perform the maintenance in a safe environment above the surface. Other fields include space and surgery, where space falls into the same category as deep sea. In surgery, some operations can be difficult to get done in time, due to the need of specialists who are far away from the patient. This problem can be solved with teleoperation, where a surgeon operates through a robot interface from another place in the world. Traditionally, this has been achieved through a two dimensional screen, with a joystick. However, this method suffers from the lack of precision. If the screen is replaced with a virtual reality headset, the specialist will have depth perception, making higher precision possible. To achieve even higher precision, a haptic controller can be utilized, giving the specialist the ability to feel the force applied to the patient. This can improve the quality of surgery and examination through teleoperation.

How teleoperation is done A study found three ways of achieving teleoperation[1]. The first method involves the recording of a user performing a task, and later playing it back through the robot. The second method is by direct, real time robot control through the virtual reality system. While this method is very flexible, time delay can be an issue. An example of this is when an operator is controlling a robotic arm and grasps

an object he or she sees, later discovering that the remote controlled arm in fact did not grasp the object, but rather pushed it over. A third approach, called *task deduction*, involves the virtual reality system recording the user's actions, classifying, and dividing them into sub tasks which are sent to the robot.

2.1.2 In education

Some fields are difficult to practice in. Virtual reality can be applied to simulate these situations, making it possible to train more quantitatively and methodically. For instance, pilot training is extremely expensive because of fuel and maintenance. Virtual reality can thus be applied to avoid the costs, while giving both accurate and valuable simulation.

Medicine In medicine, surgery training brings patient safety issues. The classical model of surgical education is *See one, do one, teach one*, and while giving the trainee first-hand experience from the actual operating room, studies show that this can be a suboptimal way of learning[4]. One of the reasons for this is the fact that the act is not centered on education, and must be focused on the patient. In addition, procedures cannot be repeated for the student to watch again. The operating room can also be a hostile and stressful environment for the trainee. Studies show that cognitive performance as function of stress is shaped like the Bell curve, thus showing that only moderate levels of stress is optimal[5]. Practicing in a virtual environment can therefore help by lowering the stress, while still keeping it on a moderate level, because the experience feels somewhat real for the trainee. In addition, medicine students often use videos to learn how different procedures are performed. While the 2D-videos are a good and inexpensive education form, they lack depth. In a virtual environment, the students can both get a better view on the anatomy, as well as feel it - if a haptic controller is utilized.

Summarized, virtual reality makes it possible to train more quantitatively and in a wider range of scenarios, the only limits are the technology, designers, costs, and time.

2.2 Augmented reality

Augmented reality is about changing how we sense the real world. It finds its place in the *virtual continuum*, on the opposite side of the spectrum compared to virtual reality, closer to reality, see figure 2.1. To create augmented reality, start with a given sense, for example vision, and then change something. What most people associate with augmented reality is originating from mobile entertainment. This includes using the mobile device's main camera on the opposite side of the screen, and adding - or augmenting - something into the live video feed. Concrete examples of such apps are Snapchat, and the more recent Pokémon GO. Snapchat

focuses on face recognition, and changes the user's looks with different filters. Pokemon GO adds 3D figures to the smartphone's live video feed. Another example is in medicine, where the surgeon is able to see through the skin of the patient, observing data from sensor scans[6].

2.2.1 Achieving augmented reality

Augmented reality can be achieved through a range of different methods. A common example is through a display. The display can be hand-held or head-mounted. The hand held solutions, for instance a smart phone, are typically more affordable than the head mounted displays. The built in camera in the smart phone is used to capture a video stream while the processor *augments* it, by for example adding 3D models to the scene. A challenge arises due to the high demand of computational power to run in real time, and thus reduces the maximum possible resolution. A study showed that high responsiveness and avoiding lag were key factors in a good augmented reality system[7].

Head-attached displays are the group of all displays attached to the user's head in some way. Retinal displays project the image directly onto the eye[8], offering an ultra-wide field of view, but are limited to red laser[1]. Head-mounted displays are a subgroup using a small monitor to display the image. These displays can be further separated into *video see-through* and *optical see-through*[9, 10]. Video see-through uses a display in front of the eye, working just like the hand-held solution, and has the same performance issues. Optical see-through achieves augmentation with a partially transparent mirror to reflect the image, or with a transparent LCD screen[11]. This technique does not suffer from the latency issues and the following motion sickness introduced by the video see-through display, and is only limited by the typical low resolution in the augmentation overlay.

Another approach is to use Spatial Augmented Reality (SAR). This technique differs from the others because it is not connected to the body. One example is a regular monitor showing video see-through. Another example is spatial optical see-through displays, that work by aligning the augmented images to align with the environment, for example with a transparent display. A last method is to project images directly onto a surface in the environment[12].

Ethical and privacy concerns There are ethical questions that have to be looked at as new technology is emerging, as augmented reality can be used to extract and display information about other people with facial recognition[13, 14]. Such information can be sensitive and expresses the importance of this subject.

[1]Low powered lasers in other colors are not yet available.

2.2.2 Pose estimation - Computer vision

As presented in this section, there are different ways of achieving augmented reality. When looking at cases where added graphics or animations have a connection to the environment, the system needs to know exactly where the environment is and how it is moving relative to the user. An example case is a virtual 3D figure that is added to the room, making it look like the figure is real. If the target medium was only a single image, achieving this would be easy, as the figure could simply be drawn into the image at the desired location. This would by definition indeed be augmented reality, although this is more commonly known as simple photo editing. However, in this example, there is not only one image, but a continuous video feed, which means the floor will be moving relative to the AR goggles, as the user moves. This makes the system more complicated, since the 3D figure must move with the environment. When adding 3D models or special effects in film production, some advanced video editors have built in computer vision algorithms, making them able to automatically calculate how the camera moves relative to the scene. This way 3D models can be placed into the video to create credible film. These algorithms can be quite expensive computational vise, but since the video production does not happen in real time, this is not a big issue. However, in real time augmented reality systems, efficiency is a critical subject. Stable frame rate is important for the overall quality feel of the system; resolution is therefore often sacrificed in these applications to keep an acceptable frame rate.

Pose estimation To be able to track the environment's movement, the exact position and orientation of the headset's coordinate frame, relative to the environment's coordinate frame must be known. This combination of position and orientation is called the pose, and is a key concept in computer vision. The pose is the rotation and translation required to move coordinate system A to B. Although augmented reality requires 3D pose, 2D pose is visualized in figure 2.2 for simplicity. The resulting transformation matrix is shown in equation 2.1, and can easily be used to translate points from one coordinate system to another, by multiplying the points with the transformation matrix.

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & {}^{A}t_{Bx} \\ sin\theta & cos\theta & {}^{A}t_{By} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Back to the example, the headset's exact pose must be known to be able to draw the augmented graphics in the right position. To find the pose, different techniques can be applied. If the augmented reality system is in a motion capture lab, reflectors can be attached to the headset to track its position and orientation. This is not the case in most AR applications, so the headset's pose must be calculated through the built-in camera. This can be done in different ways. If the frame contains a known object, the distortion

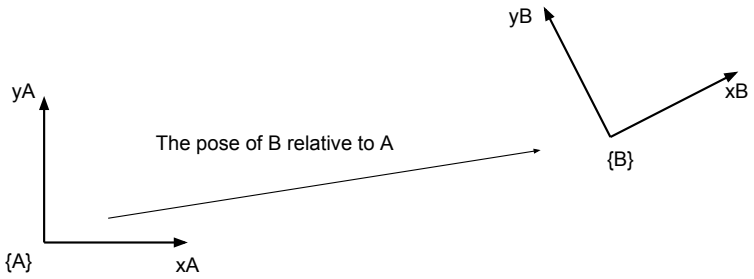


Figure 2.2: The pose of coordinate system B relative to A

of the object can be analyzed and used with the optical characteristics of the camera to calculate the pose. This method is used in a popular computer vision library called ArUco[15]. ArUco uses printed markers that are easy to distinguish from the background, to estimate the camera's pose relative to the markers. This is more thoroughly explained in section 3.4.1.

If the AR system is not dependent on knowing its exact pose relative to the environment, but rather on how it *changes*, another technique can be applied: Find points of interest in the current frame in the video feed and compare with the next frame. If at least three corresponding points are present in the two frames, the pose between the two frames can be estimated. This is called the *perspective-3-point-problem*[16]. Another possible technique is Optical Flow, where the movement of brightness patterns in the image is measured to calculate the relative motion between the camera and the scene[17].

2.3 Projection

In camera based augmented reality applications where artificial 3D objects are drawn into the scene, knowledge about how the objects would appear in the image is required. Correct rendering can be achieved by transforming, or *deprojecting*, the object's 3D points into the 2D images. This transformation can be described by a camera model. There are different models; *the generic camera model* is one of them, supporting zooming, focusing, and fisheye lenses[18]. A simpler model, *the pinhole camera model*, is covered in this chapter.

Image formation A camera is an imaging device, where photons are captured onto a detector. Cameras require a way to focus photons onto the detector, to form an image. *The pinhole camera* describes a simple camera

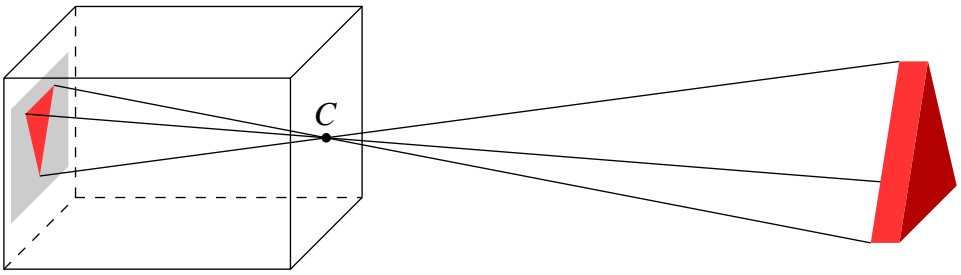


Figure 2.3: Perspective / pinhole camera model. A triangular prism in front of the camera is captured through the pinhole, onto the image sensor (gray).

without a lens, with a small opening, or *pinhole*, that focuses points from the world through the pinhole, and onto the detector, forming an image. The size of the pinhole is called the *aperture*, and should be as small as possible to produce a sharp image, although this also makes the image darker. Cameras with a lens can still produce sharp images with large apertures, making it possible to adjust the brightness and field of depth (the distance from the nearest to the furthest point in focus). The camera's *focal length* is the distance from the pinhole to where the light rays are brought to focus, and determines the field of view (how wide the imaged area is), as well as the field of depth. An image is captured by opening the pinhole, exposing the detector to photons, and closing the pinhole when the detector has been exposed to enough light to produce a bright image.

2.3.1 The perspective / pinhole camera model

The pinhole camera model describes a simple camera without a lens, where the 2D pixels from the detector in the image can be traced along straight lines through the pinhole of the camera, to their origin in the 3D space in front of the camera. The camera model consists of two parts: The *extrinsic* and the *intrinsic* part. The pinhole camera is illustrated in figure 2.3.

The extrinsic part This part handles the transformation from the world coordinate frame to the *normalized image plane* ($3D \rightarrow 2D$). The *normalized image plane* is placed normal on the z-axis at $z = 1$, in the camera's coordinate frame, see figure 2.4. The extrinsic part is the product of two matrices, shown in equation 2.2. The first matrix is a perspective projection, which transforms 3D points into 2D (3x4 matrix, far left of equation 2.2). The second matrix is the pose of world coordinate frame, relative to the camera coordinate frame (4x4 matrix, second matrix on the left side of the equation. Note that R is 3x3; t is 3x1). Recall that the pose translates from world coordinates to camera coordinates.

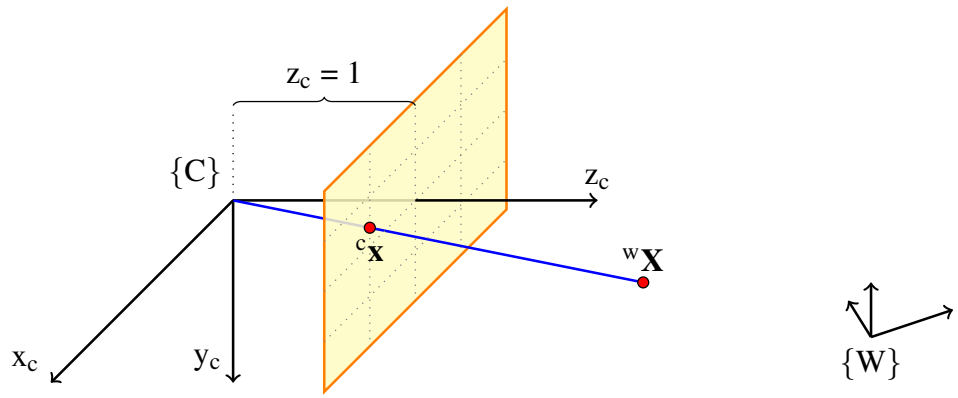


Figure 2.4: The extrinsic part of the camera model, illustrating the relationship between points in the world coordinate frame ($^w\mathbf{X}$), and their corresponding point in the normalized image plane ($^c\mathbf{x}$).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} R & t \end{bmatrix} \tag{2.2}$$

The extrinsic part thus describes the relation between a point in the *normalized image plane* $^c\tilde{\mathbf{x}}$ and the corresponding point in the world coordinate frame $^w\tilde{\mathbf{X}}$. This relation is shown in equation 2.3. Note the use of *tilde*, meaning that the points are *homogeneous*[2].

$$^c\tilde{\mathbf{x}} = \begin{bmatrix} R & t \end{bmatrix} {}^w\tilde{\mathbf{X}} \tag{2.3}$$

The intrinsic part This part translates points from the *normalized image plane* (x, y) to image coordinates (u, v), and is thus a 2D to 2D transformation. The transformation is illustrated in figure 2.5. The camera calibration matrix K describes the intrinsic transformation and is displayed in equation 2.4.

$$K = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The values in the matrix come from the camera's optical characteristics. f_u and f_v are related to the pixel densities in the u and v directions, relative to the focal length. c_u and c_v define the optical center on the sensor array. s is the *skew* parameter[18]. The relationship between the (both homogeneous) image coordinate $\tilde{\mathbf{u}}$ and the *normalized image plane* coordinate $\tilde{\mathbf{x}}$ can thus be described, and is shown in equation 2.5:

[2]Homogeneous coordinates have an extra dimension (i.e. [x, y, z, 1] in 3D), giving them the ability to be multiplied by a non-zero scalar and still represent the same point. This can be observed in figure 2.4, as the two points are at the same line, but have different scaling. This makes homogeneous coordinates especially useful in projective geometry.

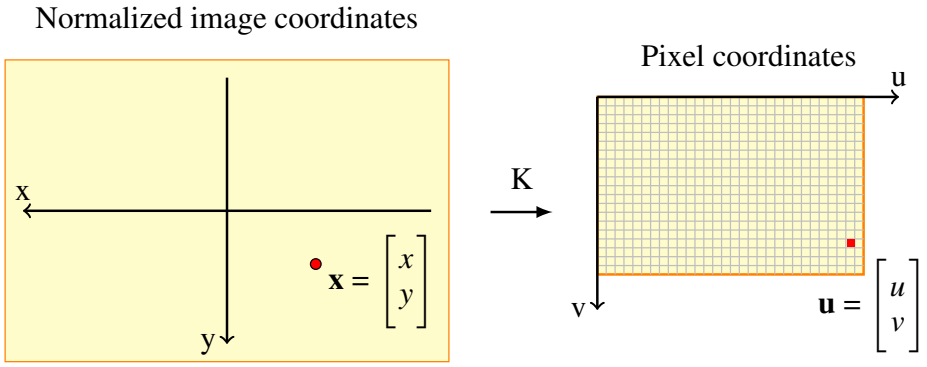


Figure 2.5: The intrinsic part of the pinhole camera model transforms points in the *normalized image plane* (left), into pixel coordinates (right). Both coordinate frames are seen from the camera's front, towards the sensor.

$$\tilde{\mathbf{u}} = K\tilde{\mathbf{x}}$$
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.5}$$

The projection matrix The projection matrix is combined by the extrinsic and the intrinsic part, and describes the correspondence, or projection, between points in the image and in the world, displayed in equation 2.6 and 2.7:

$$P = K \begin{bmatrix} R & t \end{bmatrix} \tag{2.6}$$

$$\tilde{\mathbf{u}} = P\tilde{\mathbf{X}} \tag{2.7}$$

2.4 Robotics

Robotics is a field between computer engineering, physics, and electronics, pulling in many other sciences as well. The field has numerous uses, from stationary robots used in automatic manufacturing[19, 20] and surgery[21], to mobile robots used in Urban Search and Rescue (USAR)[22, 23] and hostile environments such as Mars[24, 25].

2.4.1 Robot perception

Robots sense their surroundings with sensors. There are a great variety of sensors, which can be separated in two groups, proprioceptive and exteroceptive[24]. The former classifies sensors measuring internal data, such as temperature and torque. The latter are sensors measuring external information about the robot's environment, for example range and sonar

sensors, tactile sensors (proximity/pressure), and vision sensors (cameras). However, all sensors have limitations, and are not always accurate[26], validation and testing are thus important procedures in robot engineering.

2.4.2 Previous work in the field of augmented reality and robotics

There are multiple studies on augmented reality done within the field of human-robot interaction. Augmented reality can be applied to shared industrial environments, where robots work alongside humans, to visualize assembly procedures and show general information[27]. It can be used to aid in human-machine interaction in disaster sites, by for example visualizing a collapsed building's 3D model[28].

In applications requiring the human to see spatial sensor data from the robot, such as a point clouds, visualization have been found to be much more effective with stereoscopic displays than single displays. The reason for this is that the depth is perceived directly in a natural way for the human observer, without having to look for hints in the image to understand the depth[29].

Other studies focus on the control of robotic system through augmented reality. A study presented the use of augmented reality to quickly visualize a robot's path planning with *Programming by Demonstration*[30]. Another study used augmented reality to visualize the interactive mapping of the robot's world model, path planning, and intention, and mentions possible visualization of sensor data for future work[31]. In medicine, minimal invasive cardio-vascular surgery is possible through the combination of robotics and augmented reality[32].

2.5 Understanding a robot

This section covers the basics in human-robot interaction, and the background necessary to define *understanding* in the field of robotics.

> *Understanding is a psychological process related to an abstract or physical object, such as a person, situation, or message whereby one is able to think about it and use concepts to deal adequately with that object.*[33]

2.5.1 Human-robot interaction

An important field within robotics is human-robot interaction, or HRI. This field is about how humans interact with robots. By definition, interaction is a two-way event that occurs when two or more objects are affecting each other. There are two types of HRI, remote interaction and proximate interaction. The former is applications where the human is in a remote location relative to the robot. An example is a deep sea robot,

inspecting oil and gas pipes on the sea bed[34]. The latter is the opposite case - where the human and the robot are in close vicinity, for example in production environments[35].

Information exchange There are different types of information exchange within HRI, which can be categorized by medium and format. Examples of mediums are visual, audio, and touch. Common for all is that they are sensory based, as it is impossible to gain information for humans without the use of senses. A medium example for visual sensory information is graphical user interfaces on computer screens, in virtual reality or in augmented reality.

Situation awareness Situation awareness or SA is defined as an idea of the environment's state and its elements, in a limited volume of space and time. This includes the ability to understand what is happening in the moment, as well as in the future. SA is important in many applications, for example driving, air traffic control, and in search-and-rescue situations, to name a few. The formal definition breaks SA into three levels[36]:

1. Perception of the elements in the environment
2. Comprehension of the current situation
3. Projection of future status

2.6 System testing

Humans make errors[37]. Therefore computer programs also contain errors. These defects, or bugs, can introduce health risks for people and economic risks for companies. Medical and flight control systems naturally require more vigorous testing than a web page for visualization of different color palettes. Software testing includes multiple levels during the testing process[38]. After the software product is shipped by the producing organization, the client can perform an acceptance test, making sure the product meets his requirements. These tests can focus on non-functional characteristics such as the system's usability. This is usually done with alpha or beta testing, where the software is tested on users; either at the producing organization's location (alpha) or another place with a beta version of the software.

2.6.1 Evaluation of an information system

A system designed for human users might require testing on human users. In robotics, evaluation of systems does often not involve users. A robot learning to walk with an evolutionary algorithm can be evaluated with a fitness function. The algorithm evaluates the fitness of each generation and iterates until a desired fitness have been achieved. With users, it is not possible to simply start an algorithm to test the system in

such an automated fashion. Validating abstract metrics like understanding requires a different approach.

Usability Usability is an important measure in system design. It has a standard from the International Standards Organization (ISO). This standard defines five key concepts[39]:

Learnability This measures how easy the system is to learn for new users

Efficiency How fast users are able to perform tasks in the system. When users have learned how the system works, they should be able to work effectively.

Memorability The user's ability to come back to the system after some time, and still be able to use it, without having to relearn it.

Errors How often users makes errors, and how easy it is to recover from them.

Satisfaction A measure of the users overall feeling with the software. They should be satisfied when using the system.

In usability testing, the five key concepts should be analyzed. Because of the scope of this thesis, *Efficiency* and *Errors* will be the main focus. *Errors* can be interpreted as the accuracy of the users[40]. A study investigating the use of augmented reality to understand 3D models measured the users' accuracy by calculating their error rate[41].

2.6.2 Statistical method

Research Methods in Human-Computer Interaction[42] presents the statistical background needed to perform a good study in the field of HCI. This section is inspired by the methods presented in this book.

The null hypothesis and the alternative hypothesis The null hypothesis is normally a hypothesis stating that there is no difference in two measures, while the alternative hypothesis states that there *is* a difference. A researcher can use statistical methods to disprove the null hypothesis, and thus confirm the alternative hypothesis. An example null-hypothesis and its opposite alternative hypothesis are displayed below.

- H_0: Sleep duration does not affect reaction time
- H_1: Sleep duration does affect reaction time

Randomization Randomization is important for good experiment design. If not done thoroughly, the study's results can get corrupted, as unwanted factors are introduced. Every aspect of the study should be randomized, the sequence of the experiments, which subject are given treatment, and which are given placebo[43].

Between-group and within-group When testing multiple systems, or conditions, for instance in a study on how sleep affects work performance with two conditions: 6 hours of sleep versus 8 hours of sleep, a decision in whether to use the between-group or within-group design has to be made. Between-group means every study participant is tested in all conditions or systems. In the sleep example, if the experiment compares 6 and 8 hours of sleep, all subjects would be tested in both 6 and 8 hours. On the other hand, in a between-group study, a subject would either participate in the 6 or 8 hour group, but never in both.

There are pros and cons with both styles. In within-group design, both learning effect and fatigue (from long lasting experiments) can occur, since each subject is tested multiple times. This is not an issue in between-group design, since each subject is only tested in one condition.

A disadvantage in between-group design is that individual differences can obscure the results, making significance less likely to occur. This can cause a *type II error*, which means acceptance of the null-hypothesis, when it should have been rejected[44]. Another rather important disadvantage with between-group design is the requirement of a larger subject pool. The reason for this is that statistical significance is harder to achieve with fewer subjects. For instance, a between-groups study with two conditions will approximately require twice as many subjects compared to within-group design, since the subjects must be divided in two groups. This can make the between-group study challenging to conduct.

Errors There are two types of errors, random errors and systematic errors. Random errors will always be present in experiments due to noise. For example - someone who needs 30 minutes to travel to work will not always use 30 minutes, maybe 28 minutes on a good day, and 32 minutes on a bad day. Here the errors are the deviance from the actual value (-2 and 2). Systematic errors or *bias*, in contrast to random errors, moves the mean in one direction (random errors do not affect the mean when there are a significant number of samples). The *bias* can be caused by different issues:

- A faulty measurement apparatus, e.g., not measuring time correctly
- A within-group experiment with multiple conditions that are not randomized
- Inconsistent instructions to the study participants
- The experiment leader intentionally or unintentionally affecting the subject with wording or body language.

Significance tests When data is gathered, it can be wise to compare the means between the groups to see the tendency. Different means does not hold for concluding that the groups are in fact different, but can be a good indication that a significance test should be conducted.

Student's t-test The t-test is a simple method to measure the statistical significance of a hypothesis where the means of two groups are compared. One must be aware of whether the two groups are independent or paired(dependent in some way), as the test is performed differently for each of the cases. A paired t-test yields higher precision and thus stronger tests than the independent one[45].

To suggest whether there is a significant difference in the two means or not, the calculated t value is compared with the t value from a *t table*(table with different values depending on degrees of freedom and chosen confidence interval, e.g. 95%). The null-hypothesis can be rejected if the calculated *test statistic* value is higher than the corresponding value from the table.

The test can be two-tailed or one-tailed. The former is used when the goal is to investigate if there is a difference between two groups. However, sometimes testing if one of the groups is better than the other is more appropriate. In this case, a one-tailed test should be applied. An important matter to remember is that when using a t table, the t value for a 95% confidence interval in a one-sided t-test is the same as the t value for a 90% confidence interval in a two-sided t-test.

Investigator bias Bias from the investigator is an important challenge in all research[46]. Studies motivated by economic or political goals can have poorly documented details, making it hard to recreate the experiments. However, other good studies can still find false conclusions, because of investigator bias. This can happen if an experiment has a poor hypothesis, or is designed to prove a point.

Learning effect During a user study where subjects are tested multiple times in the same system, in multiple conditions, a *learning effect* may occur. The effect can make users perform better after multiple tests have been conducted, when they have learned, or become more familiar with the test conditions. If this effect is not taken into consideration, a false conclusion can be drawn. A method to counter this issue is to randomize the sequence of the conditions[42].

Chapter 3

Tools and software

This chapter presents the different tools and software used in the system. It covers a brief introduction to the robotic operating system, computer vision libraries, and a game engine often used for controlling augmented reality goggles.

3.1 HTC Vive

This section covers the chosen platform for augmented reality, the HTC Vive. The Vive is a virtual reality platform, but it can be used to achieve augmented realty. This is possible because it has a front-facing camera, and transforms the Vive into a head-mounted see-through display, as covered in section 2.2.1. In 2016, the goggles were one of the leading commercially available virtual reality systems, offering high resolution displays, at 1080x1200, with a refresh rate of 90 Hz. The headset comes with two hand controllers, making it possible to interact with the virtual environment in an intuitive way. The headset and controllers' poses are tracked by two sensors. The system is displayed in figure 3.1.

3.2 ROS - the robotic operating system

The robotic operating system, or *ROS*, is a vastly popular open source system for development of robotic systems[47]. It supports multiple languages, including C++ and python, and cross-language development, which means a robot can have its components programmed in different languages. This works by using an *Interface Definition Language* (IDL), requiring message definitions in text files. An example of such a message definition file is displayed below.

Figure 3.1: HTC Vive. Goggles and sensor in the right corner.

```
# This expresses velocity in free space
# broken into its linear and angular parts.
Vector3 linear
Vector3 angular
```

Listing 3.1: ROS message (geometry_msgs/Twist Message).

Nodes and topics A *node* is a core concept in ROS. It is a small process, and can be responsible for different tasks, for example controlling a sensor, or managing the navigation of a mobile robot. Nodes can communicate with each other with messages. The sending node *publishes* a message on a *topic*. Another node, which needs the published information, can *subscribe* to the said *topic*. The listening *node* subscribes to the *topic* and defines a callback function, automatically called when a message arrives on the topic. *Services* is another method for communication, working in a similar fashion as a function with a return value. The *service* is defined by two messages: A request and a response.

rviz rviz is a powerful 3D visualization tool for ROS, offering real time monitoring of a robotic system. The program makes it possible to observe the robot model in relation to its environment, as well as its sensor data. Additionally, the software offers control of the robot in the graphical user interface, for example by drag-and-dropping an arrow to make the robot navigate to a desired position in the map. In figure 3.2, sensor data from an Intel RealSense depth camera, in form of a point cloud, is visualized. The sensor's origin is also displayed in the 3D environment, but below the visible frame in this image. The topic *tf* contains information on how the cloud's coordinate system is positioned and rotated relative to the sensor's base. This makes rviz able to draw the point cloud and the robot, at correct relative positions. However, this requires a 3D model of the robot.

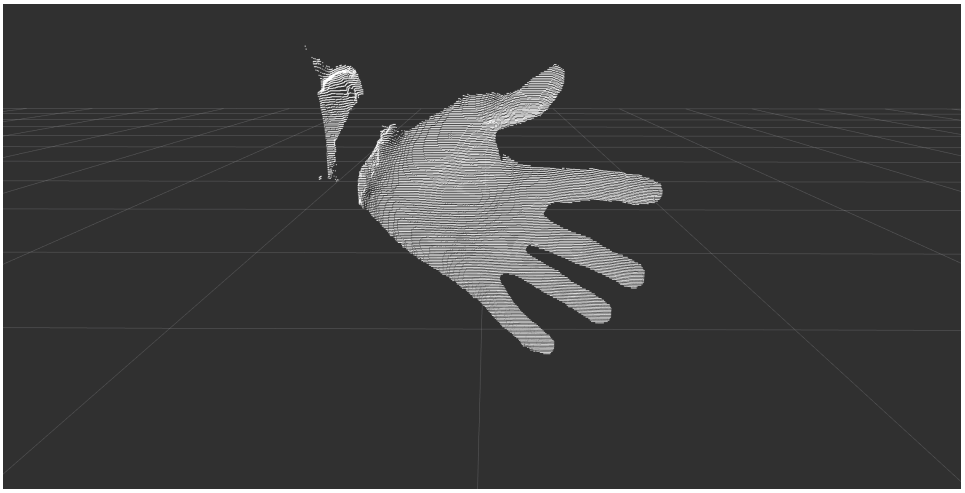


Figure 3.2: Screenshot from rviz, inspecting the real time point cloud from an Intel RealSense depth camera

Figure 3.3: Intel RealSense f200 depth camera

RosBridge *rosbridge_server* is a ROS package making communication between ROS nodes on different computers possible. The server uses *WebSocket* as transport layer. The WebSocket protocol dates back to 2011, and enables a two-way communication between a client and a server[48]. Unlike HTTP, WebSocket keeps the connection open, and do not require HTTP handshakes for every message.

3.2.1 Intel RealSense depth camera

This section attempts to go through the most important parts of the depth camera used in this thesis. It starts with a brief explanation on how the sensor works, followed by the software part, including driver control and published topics. The full name of the sensor is Intel RealSense f200, and is pictured in figure 3.3.

Sensing the depth The RealSense gets depth information by projecting a grid in the infrared spectrum into the room in front of the camera. It then

```
Field name: [   x   ][   y   ][   z   ][ unused ][r][g][b][ unused
Byte index: 0        4        8        12          16 17 18 19
```

Figure 3.4: Byte structure of a point cloud entry

measures how the grid hits surfaces with an infrared camera. The sensor does not see black objects or glass, since it depends on reflected rays from the surface. Black surfaces absorb light, and glass reflects very little back to the sensor.

Depth information from the sensor The RealSense ROS package publishes three important topics. The first one is */camera/depth/image_raw*, this topic contains the raw depth images from the sensor, 640x480 pixels of 16 bits. Each pixel holds a depth measure in millimeters. The second important topic is */camera/depth/camera_info*, this topic holds information about the sensor's optical characteristics - the camera matrix and distortion coefficients. The third topic is */camera/depth/points*, and contains the complete point cloud.

The point cloud message The first message to look at is the *pointcloud2* message. A live data packet, fetched with *rostopic echo -n1 /camera/depth/-points* is illustrated in figure 3.5. The following paragraphs will explain its contents.

fields The first part to look at is the field explaining the internal structure of the point cloud. It is called *fields:* and contains an array of type *sensor_msgs/PointField*. This array describes one entry in the point cloud. In our live packet we observe, by looking at *fields:*, that each entry in the point clouds contains the four fields x, y, z, and *rgb*, each encoded as float32 (*datatype: 7* means float32[1]). It is further important to notice the field *offset*. This shows where the x, y, z, and rgb starts, and is set to 0, 4, 8, and 16 respectively. The numbers make sense for x, y, and z, since each of them needs 32 bit, or four bytes. Even though the z field ends at index 11, the *rgb* field starts at index 16. Inspection of the source code in the ROS package shows that the *rgb* field consists of three bytes. The resulting byte structure of a point entry is visualized in figure 3.4.

point_step The next important field is *point_step*. This field covers how many bytes each entry in the point cloud uses. Its value is 32, which means there are 13 seemingly unused bytes between rgb (ending at index 18) and the next entry (starting at index 32). As seen in figure 3.4, each entry has a total of 17 unused bytes [2]. A consequence of this is that each point

[1] http://docs.ros.org/api/sensor_msgs/html/msg/PointField.html

[2] The documentation for the RealSense ROS package is quite minimal, the unused bytes are not mentioned.

```
header:
  seq: 537
  stamp:
    secs: 1478272396
    nsecs:  68496564
  frame_id: camera_depth_optical_frame
height: 480
width: 640
fields:
  -
    name: x
    offset: 0
    datatype: 7
    count: 1
  -
    name: y
    offset: 4
    datatype: 7
    count: 1
  -
    name: z
    offset: 8
    datatype: 7
    count: 1
  -
    name: rgb
    offset: 16
    datatype: 7
    count: 1
is_bigendian: False
point_step: 32
row_step: 20480
data: [0, 0, 192, 127, 0, 0, 192, ...]
is_dense: False
```

Figure 3.5: Live sensor_msgs/pointcloud2 message

cloud occupies $480 * 640 * 17 \approx 5$ MB unused space, which can lead to a significant bandwidth issue in real-time applications.

row_step The next field is *row_step*. This tells us how many bytes one row contains. It is set to 20480. The number comes from 640 pixels multiplied by 32 bytes.

data The last important field is *data*. This is the actual point cloud data encoded into an unit8 array. Its size is $row\_step * height = 20480 * 480 \approx 9.8$ MB. This means we can expect a bandwidth usage around 295 MB/s (at 30 fps).

The image message Compared to the point cloud message, the image messages also contains depth data, but in a simpler fashion. Each pixel has a depth measure, in millimeters. Each depth measure can be used to

Figure 3.6: Raw depth image from an Intel RealSense depth camera. The distance is coded in gray scale, dark areas are close, light areas are far.

calculate a 3D point in the room in front of the sensor, since the camera lens characteristics are known. Each message contains width, height, step (the length of an image row in bytes), as well as *encoding*. The latter is an important field, since it declares which encoding the pixels have. For the image topic from the RealSense ROS package, the encoding is *mono16*, meaning a gray scale 16-bit integer. This means each message use $640 * 480 * 2 \approx 614$ KB per message, corresponding to 18 MB/s at 30 fps. Figure 3.6 illustrates what a depth image looks like.

The compressed image message The RealSense ROS package also publishes depth images on the topic */depth/image_raw/compressedDepth*. This topic contains PNG (Portable Network Graphics) compressed images of message type *sensor_msgs/CompressedImage*. Its bandwidth usage is thus smaller than the image message, but varies depending on the image content.

3.3 Unity

Unity is a popular cross-platform game engine, available for Windows, OS X, and Linux. A game engine is essential for development of advanced games; it controls the video stream, the audio mix, controls, to name a few. Unity has a powerful GUI (Graphical User Interface), allowing users with little or no programming experience to create games. This is a key factor helping Unity become as popular as it is[49]. Unity also has a large pool of tutorials, making it easy to get started with game development as a beginner. It offers a large, well-documented API (application protocol interface), as well as a large online community and forum, making help easily available. Unity offers scripting in both C# and UnityScript. The latter is designed for Unity and is modeled after JavaScript.

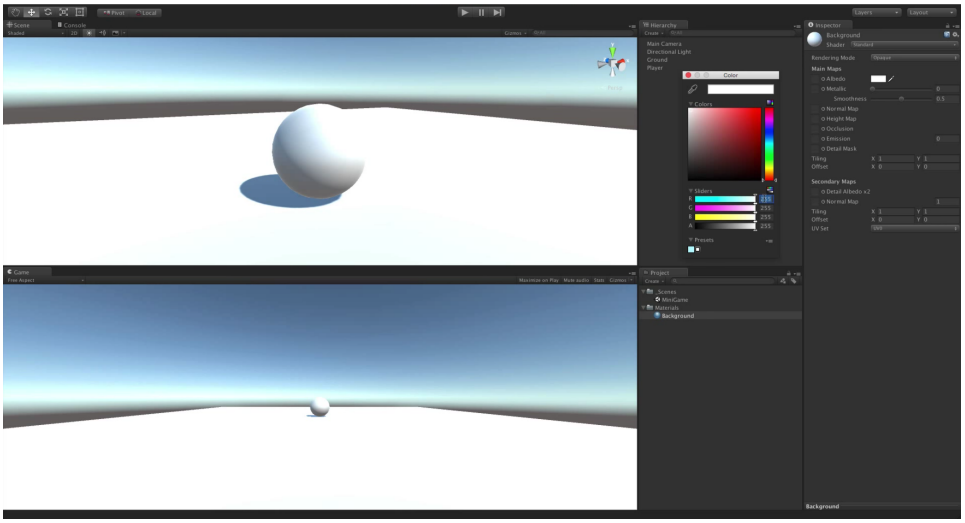


Figure 3.7: Screenshot from the Unity editor

Asset store Unity has an *asset* store, which is a large collection of pre-built components. Two of these assets were required for this thesis. The first is the SteamVR Plugin[3], making creation of virtual reality games with Unity possible. The second is OpenCV for Unity[4].

Game objects and coordinate systems The ball from the Unity editor (figure 3.7), is a *game object*. This is the base class for all entities in a scene[5]. All instances of this class have basic attributes such as *transform*. The transform contains the object's position, rotation, and scaling, according to the base coordinate system in Unity. Game objects can be nested, making linked systems of joints simple to work with, by visualizing the different coordinate systems in the editor, and how they affect each other.

3.3.1 Scripting

Game objects can have scripts attached to them. *MonoBehaviour* is the base class all scripts inherit from. There are two important functions in a *MonoBehaviour* script to be implemented, *Start* and *Update*. *Start* is called before any *Update* calls, to initialize the script and scene. *Update* is the most common function for implementing any game behavior, and is called every frame[6]. The diagram in figure 3.8 shows the execution order in a script's lifetime. To make a fluent and responsive game with a stable frame rate, it is important to make sure the *Update* function is efficient and includes no waiting calls or large, time demanding tasks.

[3]SteamVR Plugin: https://www.assetstore.unity3d.com/en/#!/content/32647
[4]OpenCV for Unity: https://www.assetstore.unity3d.com/en/#!/content/21088
[5]https://docs.unity3d.com/ScriptReference/GameObject.html
[6]https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

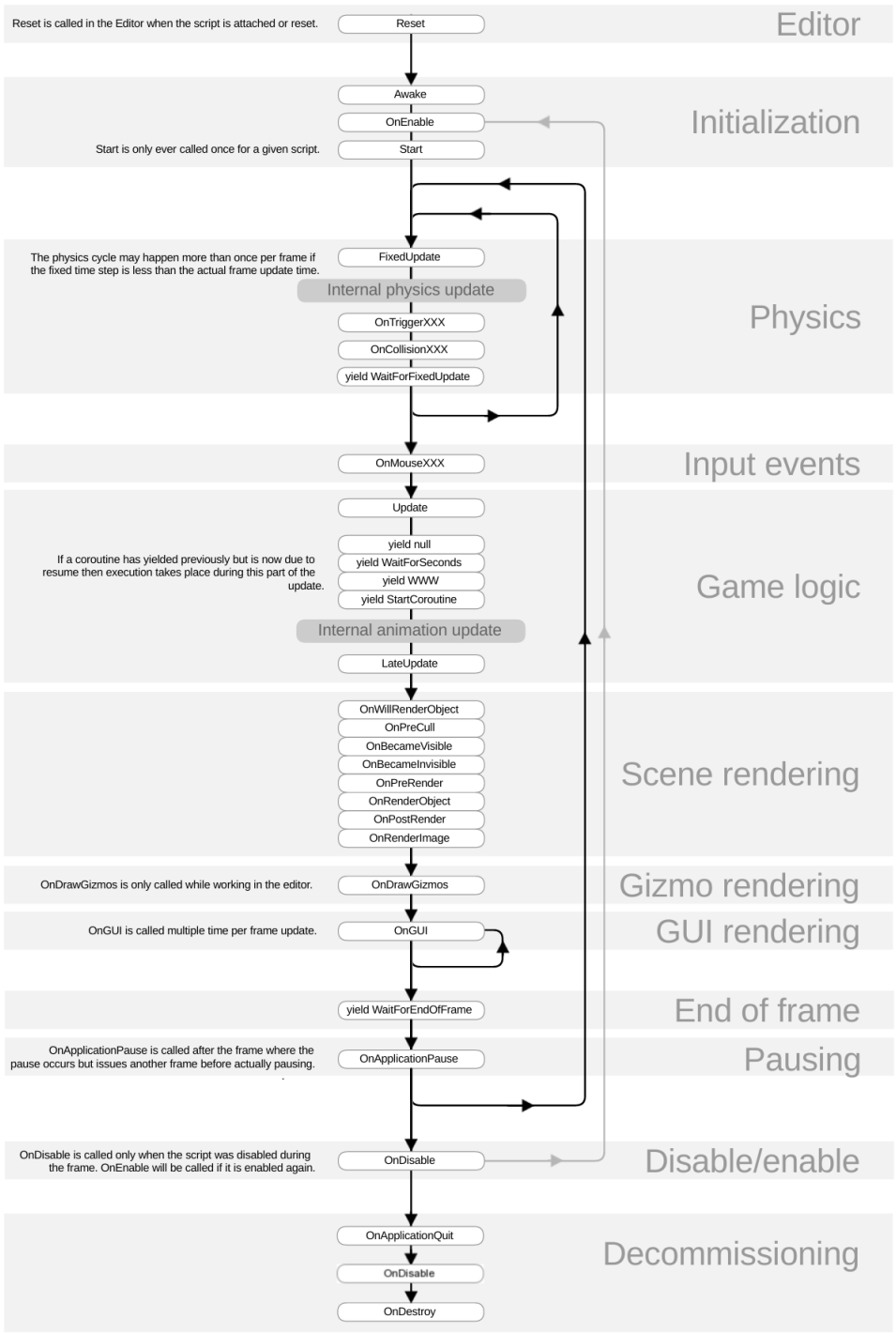


Figure 3.8: Monobehaviour flowchart[a]. The diagram shows the ordering and repetition of event functions during a script's lifetime.

[a]Monobehaviour flowchart [Digital image]. (n.d.). Retrieved November 23, 2016, from https://docs.unity3d.com/Manual/ExecutionOrder.html

Coroutines Sometimes there is need for a waiting call, or a time demanding task. In those situations *Coroutines* should be used. For example, if a ball should move from point *a* to point *b* when the user presses space, a common technique to program this would be to give the ball an x, y, and z velocity; to make the ball move in the desired direction. It is tempting to write the following code:

```
void Update() {
    if (Input.GetKeyDown("space")))
        Move();
}

void Move() {
    // The ball moves 100 steps in the x direction
    for (int i = 0; i < 100; i++) {
    ball.x += 0.1f;
    }
}
```

Listing 3.2: Naive ball movement without *coroutines*. This results in all the movement happening in the same frame, meaning no visible animation.

The problem with this code is that the *Move* function has to finish before the update function finishes. The result of this is that the ball's movement from point a to point b completes before the update function is finished. From figure 3.8 we observe that the scene rendering happens *after* the update function, under "Game logic". A solution to this is to let the ball move a little bit between every update call. This can be achieved by using *coroutines*. A *coroutine* stops execution before it completes, and continues where it stopped in the next frame, see the modified code below:

```
void Update() {
   if (Input.GetKeyDown("space")))
        StartCoroutine("Move");
}

IEnumerator Move() {
    // The ball moves 100 steps in the x direction
    for (int i = 0; i < 100; i++) {
        ball.x += 0.1f;
        yield return null;
    }
}
```

Listing 3.3: Ball movement using *coroutines*. Animation happens over multiple frames.

Since *coroutines* can run over multiple frames, they can be used for background tasks in Unity. An example use case is when the game waits

for a network event, since the time of the event is unknown, *coroutines* has to be used to regularly check if data has arrived.

3.3.2 Unityros

Unityros is a Unity project hosted on GitHub, created by Michael Jenkins in 2015[7]. The project connects to the *TurleBot* ROS tutorial. *TurtleBot* is a simple robot platform with functionality such as teleoperation and navigation, controlled by commands sent to its topics[8]. The ROS computer running *TurtleBot* requires a ROS package called *RosBridge*, covered in section 3.2. This package allows connections to other computers. *Unityros* is not well documented, the following quote from the project page's *readme* reflects this well:

> "[...] *Then fire up the unity program. with luck (?) you should see a checkerboard with a robot on it.* [...]"

Testing the system After some trial and error, the project was up and running, with the *TurtleBot* ROS package running on a laptop and *unityros* running on another. The robot was controlled with the arrow keys from the Unity computer, showing synchronized movement on both machines. The system used *subscription*, *publishing*, as well as *services*, covering all the methods for communication with ROS. A problem with this project was the lack of documentation and explanation, meaning it required a lot of time to understand.

3.4 OpenCV

OpenCV is a large, open-source computer vision library, available in multiple languages, on multiple platforms[50]. The library supports common computer vision techniques, such as *feature extraction*, *structure from motion*, and *facial recognition*. *OpenCV* is used in many visual systems, for example augmented reality applications[51], gesture recognition[52], and motion tracking[53].

There is a Unity plug-in called *OpenCV For Unity*[9]. It is not complete, but already has a lot of functionality, including *ArUco*, a core component in this thesis.

3.4.1 ArUco

ArUco is a module for augmented reality that can be included in *OpenCV*[54]. It provides a way to generate markers, such as the one in figure 3.9. The markers can be detected in an image, and the pose of

[7]Unityros: https://github.com/michaeljenkin/unityros
[8]TurtleBot: http://wiki.ros.org/Robots/TurtleBot
[9]https://www.assetstore.unity3d.com/en/#!/content/21088

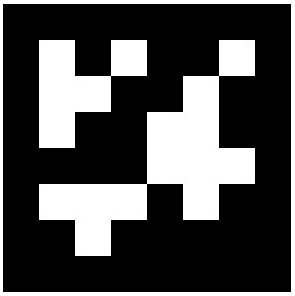


Figure 3.9: *ArUco* marker

the camera relative to the marker can be calculated. Multiple markers can be used at the same time, making it possible to track multiple objects simultaneously, and from different angles.

Chapter 4

Implementation

This chapter covers what was implemented, and how it was done. Creating a proof of concept required many choices to be done regarding how the prototype was going to be implemented.

4.1 System design

The implementation of the system is complex, and is therefore divided in different sections. It consists of two main parts: A computer running ROS, simulating a real robot, and a second computer controlling the augmented reality goggles. The setup is illustrated in figure 4.1.

4.1.1 The robot

This section covers the implementation of the robot prototype. Instead of a full scale robot, a minimal robot prototype was created, as this was found to be sufficient for investigation of sensor data.

Physical robot The robot prototype was built out of a plywood sheet, dimensions 40x20x0,5 cm. The sheet was designed to be attached to the live quadrupedal robot, *Dyret*[55], to have the opportunity to test the system in a real setting. A big ArUco marker, dimensions 20x20 cm, was glued on top, see section 3.4.1. A sensor was secured to the sheet with screws. A hole was drilled in the sheet for the sensor cable to go through. The sheet was equipped with four steel legs, to make the platform stable. The robot prototype can be seen in figure 4.2.

Sensor and ROS computer The robot prototype was connected to a laptop, running Ubuntu 16.04, with ROS kinetic installed. The chosen sensor on the robot was an Intel RealSense f200. This sensor is a close range, depth sensor, which produces 3D point clouds, and works well in an indoor environment, see section 3.2.1 in the Tools and software chapter. In addition to the infrared camera, used to sense depth, the sensor has an RGB (red, green, blue) camera, but this was not used in this prototype. Required software for this system can be seen in table 4.1.

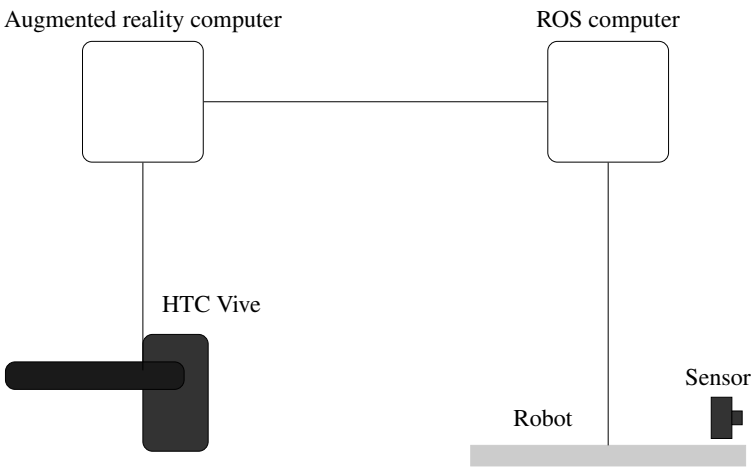


Figure 4.1: The system setup

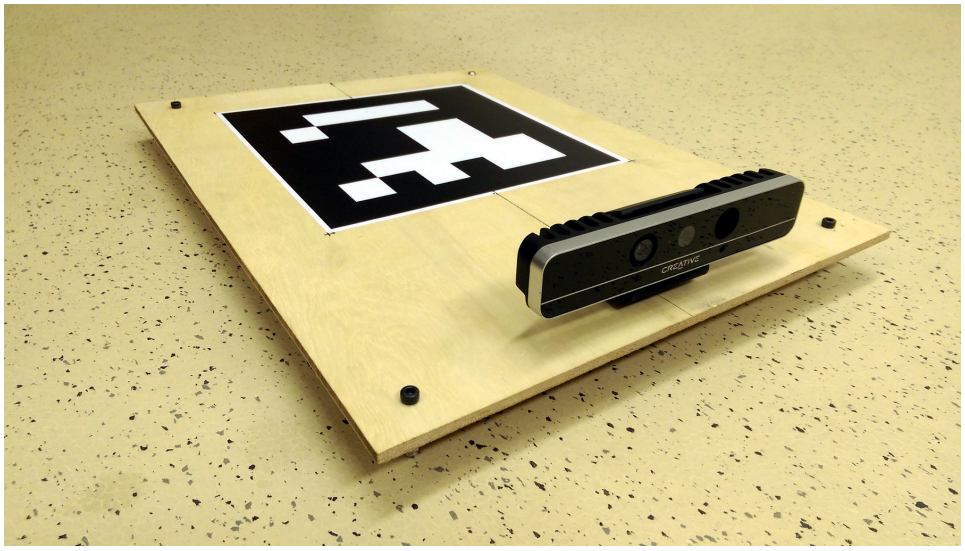


Figure 4.2: Image of the robot prototype.

Software	Purpose	Version
Ubuntu	Linux operating system	16.04
ROS	The robotic operating system	Kinetic
rviz	3D visualization tool for	1.12.4
rosbridge_server	Package for sending data over network	0.7.16
realsense_camera	Package with drivers for the sensor	6c8f08a

Table 4.1: Overview of software used in the robot prototype.

Testing the RealSense When the robot was set up with the sensor and an operative ROS configuration, testing of the sensor's capabilities could begin. The ROS package for the sensor was run with a *launch file*, setting parameters like resolution, frame rate, enabling of the point cloud, and more. The default frame rate was set to 30 frames per second. To look at the actual sensor data, rviz was used. Visualization can done by choosing one or multiple topics.

There were two main topic types that could be used for displaying the point clouds: */camera/depth/points* and */camera/depth/image_raw*. As explained in section 3.2.1, the *image_raw* topic simply contains the depth image, while *points* contains the actual point cloud. However, rviz can generate point clouds from the *image_raw* topic, so both topics can be used to visualize the point clouds, with no visible difference.

Bandwidth issues As seen in section 3.2.1, the point cloud topic required 2.4 Gb/s at 30 frames per second, which did not work, as modern network adapters typically are rated for 1 Gb/s. Methods for decreasing the bandwidth were thus investigated. The easiest method was to lower the camera's frame rate. This augmented reality system did not require high frame rates in the point cloud; the frame rate could thus be lowered to 5 frames per second. This also lowers the bandwidth, by a factor of 6; down to 400 Mb/s. Further, the sensor resolution was decreased from 640x480 to 640x240[1], effectively cutting the bandwidth in half; down to 200 Mb/s. Unfortunately there was a bug in the *realsense_camera* ROS package, locking the frame rate to 30 fps, and thus the bandwidth to 1.2 Gb/s.

Sending the sensor data to the augmented reality computer The next step was to send the depth data to the computer responsible for the augmented reality goggles. This was done with *rosbridge_server*, making communication between ROS nodes on different computers possible, as covered in section 3.2. The server uses port 9090 as default, and sends data with WebSocket. This was a problem on the university local network, as most ports were blocked. Two options were considered to solve this problem, the first was to use port-forwarding. A project called *ngrok*[2] makes this possible by forwarding connections to their server in the cloud through port 80, and then to the desired port, in our case 9090. A big advantage with ngrok is that the ROS computer could be wireless, removing the requirement of the two computers to be connected, however, this did not matter as the augmented reality goggles were wired, meaning that the robot had to be within 5 meters from the machine.

Communication with an Ethernet crossover cable Since the augmented reality goggles were wired, connecting the two computers directly with

[1]The resolution was decreased by cropping the top and bottom.
[2]https://ngrok.com

cable was also an option. An Ethernet crossover cable is a type of cable that is used to let two computers communicate directly. In the earlier days, crossover cables were made by crossing some of the internal wires in an Ethernet cable. Today this is not necessary, since automatic crossover was introduced in 1998. This makes a direct connection possible without the use of a special cable[56]. The connection was then implemented by setting up a unique static *IPv4* address on both computers.

4.1.2 The augmented reality computer

The augmented reality computer is necessary for two things. First and foremost it drives the augmented reality goggles. Secondly it communicates with the robot computer, receiving the point clouds. The computer was installed with Windows 10, since Windows is the only supported operating system for HTC Vive.

Controlling the goggles The next step was to find a way to control the augmented reality goggles. The HTC Vive developers have created a Unity *asset*, making this fairly simple. It contains a game object representing the Vive play area, which can be dragged and dropped into the scene in Unity. When the game object is imported and the play button is pressed, the game starts, and the scene is perfectly rendered in the Vive. This means that the only difference from a "normal" Unity game and Vive Unity game, is how it is observed, through goggles, instead of a screen.

Connecting Unity to the robot computer One repository on GitHub that had implemented communication between ROS and Unity was found. The project, called *unityros*, is described in tools and software, section 3.3.2. It is important to notice that the project is not a library for connecting Unity to ROS, but a specific sample application for *TurtleBot*.

Applying unityros to this project To make the system applicable for this thesis, a lot of changes had to be done. First, all unnecessary files were stripped from the system to make it as clean as possible. Then all *TurtleBot* related files were deleted. The remaining files consisted of *SimpleJSON*, a library for working with JSON (JavaScript Object Notation), and code for setting up a connection to a *RosBridge* server. The code responsible for the ROS connection was located in a folder called *ROSBridgeLib*. The implementation only included the most important ROS messages and those necessary for the project. This thesis would require implementation of new messages. At this point, a decision to create a bare bone library based on this code was made. A new GitHub project was created, forked from the *unityros* project, called *ROSBridgeLib*[3].

[3]ROSBridgeLib, available at https://github.com/MathiasCiarlo/ROSBridgeLib

Collaboration with the University of the Balearic Islands When the development of *ROSBridgeLib*, was close to finished, including new messages and documentation, another GitHub project called *arsea* (Augmented Reality Subsea Exploration Assistant)[4], was discovered. The project is created by the systems, robotics & vision group at the University of the Balearic Islands. The authors also had branched out from the *unityros* project, and implemented the same missing ROS messages. Since both groups were working on the same thing, collaboration was agreed on. *ROSBridgeLib* has thus become a more complete library, making it easier for others in the future.

Testing the connection After the library was complete, getting sensor data into Unity was the next step. A new Unity project was created, with the library imported as a *git submodule*[5]. The two computers were connected wirelessly with *ngrok*, described in section 4.1.1. The project was then set up to subscribe to the *depth/points* topic. The point clouds arrived successfully, and were disposed upon arrival, as it was the connection that was being investigated at this point.

Firewall issues Once the first connection between the computers was complete, the messages were received at approximately 1 fps, even though the messages were published at 30 fps. Even though the frame rate was stable, the delay kept increasing as the system was running. This latency was first believed to originate from the third party server (*ngrok*). Therefore the two machines were connected with an Ethernet crossover cable instead. After the machines were correctly set up, it became clear that the latency issue was still present. It turned out that the increasing delay came from the Windows firewall, as disabling it solved the problem. Since the wired solution required less set up, and the two computers had to be close, it was decided to continue with the wired solution. Note that the frame rate was still only 1 fps.

Visualizing point clouds in Unity After the firewall issue was resolved, code for transforming the point clouds to Unity objects was written. Since the point clouds can be massive (up to 153.600 points), the creation of them can be quite slow. However, going from only receiving point clouds to actual rendering in Unity did surprisingly not decrease the frame rate, suggesting the bottleneck was the data transfer over the network.

[4]https://github.com/srv/arsea
[5]https://git-scm.com/docs/git-submodule

Topic	Bandwidth
/depth/points	1181 Mb/s
/depth/image_raw	79 Mb/s
/depth/image_raw/compressedDepth	7 Mb/s

Table 4.2: Measured bandwidth usage for different *ROS topics* at 640x240 resolution, 30 fps

Countering the bandwidth issue The point cloud topic required 1.2 Gb/s at 30 fps (even after half the resolution was cut). Since the frame rate was fixed to 30, two options to solve this problem were considered:

1. Use a lighter image topic at 30 fps
2. Modify the source code to make the ROS package work at lower frame rate

The first option was to use a lighter *topic*. There are two alternative depth image topics, */depth/image_raw* and */depth/image_raw/compressed_image*. Their bandwidth usages are shown in table 4.2. Although this approach is the most efficient considering bandwidth, it does not come without a cost. Since the messages only contain depth images, their corresponding point clouds have to be constructed when they arrive at the augmented reality computer. Projecting 2D pixels into 3D space would be a challenging task to get right. On the other hand, the other option, modifying the ROS source code, would also be challenging. To be able to do this, a much deeper understanding of ROS was necessary. Considering both options, choosing a lighter *topic* was found as the best solution, as this was the most elegant one. Both the raw and the compressed *topic* are small enough to avoid network saturation. The compressed one was chosen due to its smaller size, even though it would have to be decompressed back into raw images upon arrival in Unity.

Using compressed images To access the individual depth values, the images had to be decompressed. In Unity this was attempted with the built-in image load function[6]. However, the data seemed to be corrupted, as no image was created. The first ting to be tested was if the data already was corrupt before it was sent. Back on the robot computer, rviz showed that the compressed image topic looked just like the raw image topic, meaning the issue originated from the loading into Unity. PNG images start with an 8 byte long PNG signature, but the signature was not present. Help was requested in the Unity forums and revealed that the signature was located at byte index 13[7]. By removing the first 12 bytes, the image was loaded successfully. This introduced a new issue, as there were only

[6]Texture2D.LoadImage creates a Unity texture from a byte array.

[7]https://forum.unity3d.com/threads/need-help-decoding-a-16-bit-1-channel-png.442317/#post-2861464

five different shades of gray in the images. The whole dynamic depth range had been reduced to five levels[8]. The issue was likely the result of a too powerful compression, and could not be solved without changing the source code in the *realsense_camera* ROS package itself. Therefore, the raw depth image topic was used instead.

Using the raw images Although the compressed images would have been optimal, the raw image topic still offered an acceptable bandwidth usage. Extracting the depth measures from the pixels was straightforward. The byte array was traversed with a double for loop (240 rows x 640 columns), reading 16 bits at the time. This way each depth reading got linked with its corresponding pixel coordinates.

Construction of point clouds Each of the pixel coordinate-depth pairs was then sent to a function to calculate their corresponding 3D points. The returned 3D points were then pushed to the point cloud data structure. The function was designed to take a pair of pixel coordinates and a depth measure. The transformation was done as follows:

In section 2.3, the pinhole camera model, including the relation between 3D points and pixel coordinates, was covered. To calculate 3D points from pixel coordinates and depth measures, the first thing that had to be done was translation of image coordinates into the normalized image plane. This translation can be done by multiplying the image coordinates $\tilde{\mathbf{u}}$ with the inverse of the calibration matrix K^{-1}, giving the corresponding point in the normalized image plane $\tilde{\mathbf{x}}$. See equation 4.1. Note that the *skew* parameter in the calibration matrix was set to 0.

$$
\begin{aligned}
\tilde{\mathbf{x}} &= K^{-1}\tilde{\mathbf{u}} \\
\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} &= \begin{bmatrix} \frac{1}{f_u} & 0 & -\frac{c_u}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{c_v}{f_v} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \\
\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} &= \begin{bmatrix} \frac{u}{f_u} - \frac{c_u}{f_u} \\ \frac{v}{f_v} - \frac{c_v}{f_v} \\ 1 \end{bmatrix}
\end{aligned}
\tag{4.1}
$$

Since the point in the normalized image plane is homogeneous, it can be scaled to its correct length, or depth, by multiplying it with the corresponding depth measure z. Recap that the normalized image plane is in the camera's coordinate frame, at $z = 1$, as illustrated in figure 4.3. The scaled point can be interpreted as the translated 3D point $\tilde{\mathbf{X}}$ (ready to be put into the point cloud). The final scaling is shown in equation 4.2, which was used directly to calculate the 3D points in the point clouds.

[8]The 5 shades of gray: 0.000, 0.004, 0.008, 0.012 and 0.016. An interesting detail is that these numbers are the numbers you get if you divide the numbers 0-4 by 256.

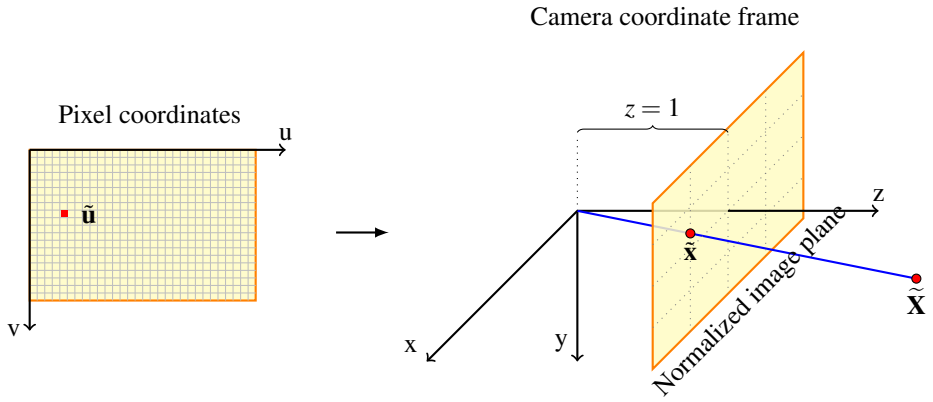


Figure 4.3: The transformation from pixel coordinates and a depth measure to the corresponding 3D point.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{u}{f_u} - \frac{c_u}{f_u} \\ \frac{v}{f_v} - \frac{c_v}{f_v} \\ 1 \end{bmatrix} z \tag{4.2}$$

Note that this model is fairly simple and does not take distortion into account.

Point cloud rendering At this point, every pixel in every raw depth image was transformed into a 3D point, and then added to the current point cloud. Once a new point cloud had been constructed, the old was deleted. Real time inspection of the point clouds in Unity showed that the frame rate was quite low. This does make sense, since the point clouds can be massive (up to 130.560 points, at 640x240 resolution). Both the raw image topic and the point cloud topic were tested, and the measured frame rates are shown in table 4.3. Note that the point cloud resolution was set to 640x480 during the experiment. The point cloud topic's low frame rate is likely low because of bandwidth saturation, as there was not much difference with and without rendering. With the raw image topic, the frame rate was close to the source frame rate (at 30 fps) without rendering, but dropped to 8 with rendering. This confirms that the image topic does not saturate the bandwidth, and that the maximum frame rate that can be achieved is 8, at 640x480 resolution. Since the system's resolution is set to 640x240, the maximum frame rate should be 16.

Experimentation with virtual reality Since the chosen platform for augmented reality was mainly a virtual reality headset (HTC Vive), it was natural to test its VR capabilities. The *SteamVR* plugin, mentioned in section 3.3 in the tools and software chapter, makes it easy to add the headset to the Unity scene. The headset acts like a camera in the scene, moving in Unity as the user moves and looks around in the real world, giving the user a first person experience in 3D. The first experiment was

Topic	Only network	With rendering
/points	1.7	1.6
/image_raw	25	8

Table 4.3: Measured frame rates in Unity with different topics, with and without actual rendering of the point clouds. The ROS computer was publishing messages at 30 Hz and with 640x480 resolution. This shows how much of the latency comes from the network.

to model the RealSense camera in the scene, and project the point cloud in front of it.

Implementation of augmented reality The first step was to get hold of the video feed from the front facing camera on the HTC Vive. This was achieved by using the Unity plug-in *OpenCVForUnity*. The camera is a fish-eye lens, making the video feed quite distorted (straight lines appear as curved lines). To make the see through experience as natural as possible, the camera was calibrated in MATLAB with its built-in calibration app, using multiple pictures of a checkerboard. The resulting model had a mean reprojection error of 0.19, meaning that the model was not optimal. Although undistorting the video feed makes it appear more natural, some of the resolution is sacrificed, resulting in a cropped and more blurry image. The video stream was then put 1 meter directly in front of the user in virtual reality, and scaled up to cover about 70 percent of the user's field of view.

Locating the robot in the frame To be able to draw the point cloud at the correct position, a way to locate the robot had to be decided. The simplest solution found was to use *OpenCV* to locate an *ArUco* marker on the robot, see section 3.4.1. Every frame in the video feed was analyzed to detect the marker. If the marker was detected in the frame, the camera's pose, relative to the marker was calculated.

Adding the point cloud to the video feed At this point in time, the virtual environment was a completely empty, black void, except for the rectangular screen with the video feed, positioned 1 meter in front of the user. The first attempt to draw the point cloud to the view was to simply place it with correct scaling between the video feed plane and the user, making it look like it is in the image. This did not work since the video feed was 2D, and the point cloud was 3D. If the user was focusing on the video feed, the points were out of focus and doubled. Similarly, when the user attempted to focus on the points, the same effect happened to the video feed. This effect caused severe nausea in about 30 seconds, making it clear that this was not the right direction to continue in, and showing the point cloud in 3D was thus discontinued. The next option was to draw the point cloud into the video feed. This was done by using the *OpenCV* function *Calib3d.projectPoints*, taking a list of 3D points (in the world coordinate frame), the camera's pose relative to the world coordinate frame, distortion

coefficients (for the camera). The function returns a list of 2D points, which are the image coordinates the 3D points were projected into. The 3D points are all the points in the point cloud, and the pose is fetched directly from the *ArUco* marker (world coordinate frame). The next step is to simply draw all the returned points into the image, and with that, achieving augmented reality. The points were drawn in pink, as this was an easily distinguishable color in the environment.

Adding the point cloud at the correct position Figure 4.4 shows the first attempt at drawing the point cloud in the image. In the image, the sensor is sensing a red wire roll (lower right). The clearly wrong position of the point cloud originated from the fact that it was drawn directly into the *ArUco* marker's coordinate frame. As explained above, the projection function takes the pose of the camera relative to the coordinate system of the 3D points to be projected. The pose sent was the pose of the Vive, relative to the *ArUco* marker, marked as T_1 in figure 4.5. To place the point cloud in the right position, the pose of the Vive, relative to the sensor has to be calculated. Since the pose of the *ArUco* marker, relative to the sensor is known (T_2, measured with ruler), the two poses can be multiplied, to find the combined pose, shown in equation 4.3.

$$T_{\text{Sensor} \rightarrow \text{Vive}} = T_1 * T_2 \tag{4.3}$$

Unfortunately, in the Unity implementation of *OpenCV*, the representation of poses made multiplication unnecessary complicated, and was thus discontinued. Instead, a simpler method was applied. Every point in the point cloud was transformed with the inverse pose of the *ArUco* marker, relative to the sensor (T_2^{-1}), moving them to the correct positions. Although this was not the most efficient technique[9], it aligned the point cloud in the correct position.

Debugging the inaccurate point clouds The projected points did not align perfectly with their corresponding real objects. The deviations increased with the distance to the sensor. The first thing that was done to identify the issue was to draw the sensor's coordinate system into the image, to verify its position and orientation. The next possible error was the point cloud construction from the depth images. To verify if this was the issue or not, the manually constructed point cloud was compared to the prebuilt point cloud from the ROS package. To avoid network saturation, the frame rate issue first had to be resolved. A bug report describing the issue was sent[10]. The issue was not resolved, as the report uncovered that the sensor was unstable. A temporary fix was to set the ROS parameter *motion_range* to 100, effectively reducing the frame rate to 10. While sacrificing quality, this made the comparison possible. Figure 4.6 shows the two point cloud sources, and confirms that the deviations did

[9] Transforming each point in the whole cloud can be slow on large point clouds.
[10] Bug report: https://github.com/intel-ros/realsense/issues/152

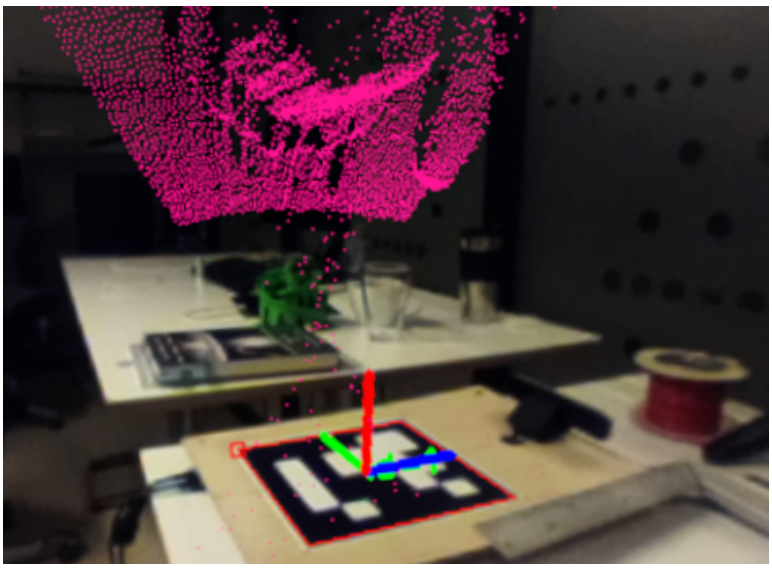


Figure 4.4: First attempt at drawing the point cloud into the video. The sensor is facing the red wire roll in the lower right. The system is configured as if the sensor was placed in the *ArUco* marker's center, facing up. Red=z, blue=x, green=y. This image is from the Vive's camera.

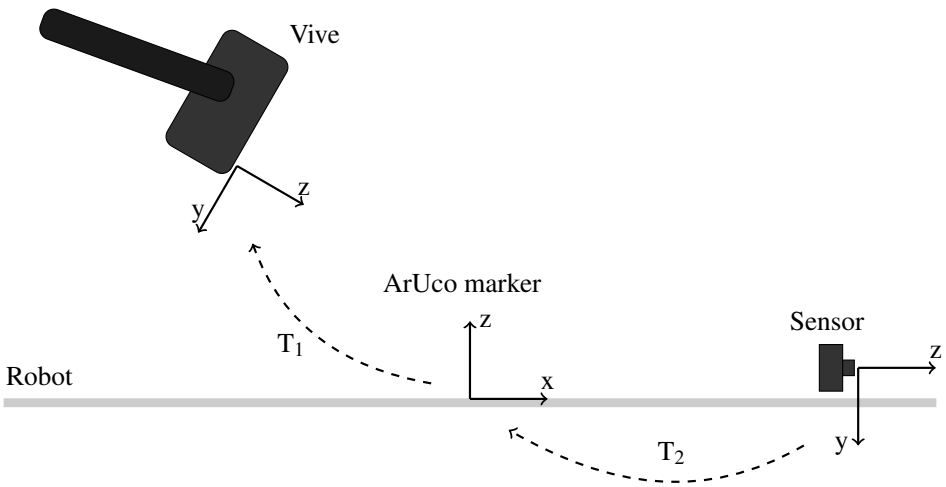


Figure 4.5: Side view of the coordinate frames for the Vive, *ArUco* marker, and sensor. The poses T_1 and T_2 describe the relations between the three coordinate systems.

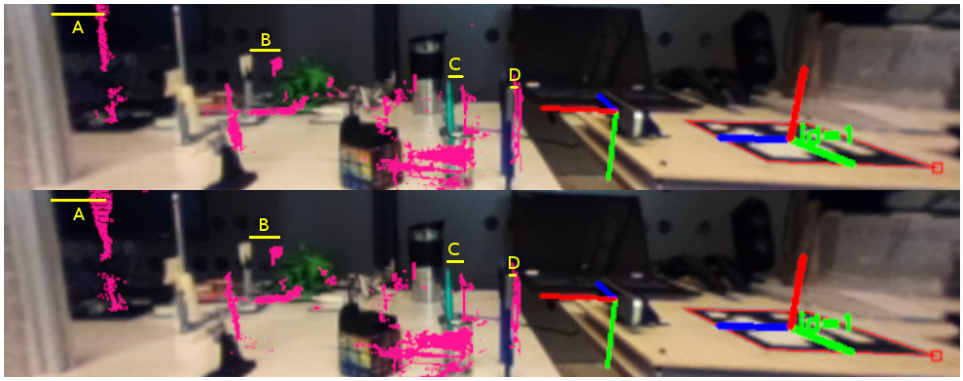


Figure 4.6: Comparison of the pre-built point cloud from the ROS package (top), and the manually constructed point cloud from the raw depth images (bottom). A-D show errors in depth, present in both point clouds. Notice the error increasing with distance from the sensor.

not originate from the point cloud construction, since the deviations were present in both point clouds. Another possible cause was the projection of the points into the Vive's camera. Recall that the camera model used in calibrating the Vive camera was not optimal. The last hypothesis was that the sensor was inaccurate. A test rig with known lengths was put in front of the sensor, and the lengths were verified in rviz, rejecting that hypothesis.

Finding a workaround for the prototype Although the origin for the inaccuracies in the point clouds was not identified, the point clouds could be aligned sufficiently at around 40 cm in front of the sensor, by performing small adjustments on the transformation matrix. The settings were made easily adjustable as sliders in the Unity editor. This made it possible to modify the sensor's coordinate system and scaling the point cloud in real time, making fine tuning very easy. For example, the sensor was not secured to the robot completely straight, and had a slight rotation of about 3 degrees. Although this does not seem like much, it creates a 5 centimeter deviation one meter in front of the sensor. This error was quick to correct with the real time settings. The corrected model is shown in figure 4.7. In the image, robot prototype was only connected to the quadrupedal robot for testing purposes.

Stabilizing the point cloud The point clouds were not stable. If the viewing angle on the *ArUco* marker was low or if there were light reflections in the marker, the point cloud started to flicker. It was programmed to only be visible when the *ArUco* marker was visible. This means that there always has to be a clear line of sight between the AR headset and the marker. However, even in good light conditions and with angles that were good for the marker detection, there was a constant small, but rapid movement in the point cloud. Different marker sizes were tested, but the marker was still unstable. This issue may originate in the Vive's poor camera quality. This issue was particularly annoying when trying to

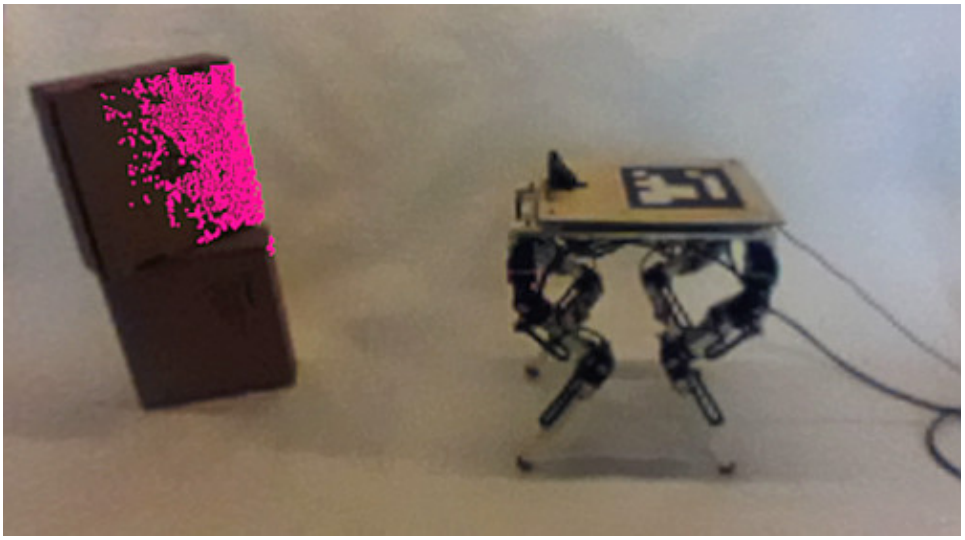


Figure 4.7: The robot sensing an obstacle. This image is what the user of the augmented reality system sees. From putting on the goggles, the user understands exactly what the robot sees. The user sees that the robot only senses the top box, and only half of its side. Note that the image is from a continuous video feed, the user is free to move around and inspect the scene from other angles.

adjust the settings to debug or align the point cloud with the objects in the scene. To counter this problem, a stabilization algorithm was implemented. The algorithm looks at the last 5 poses of the marker, and calculates a weighted average, favoring the newest poses. The formula is shown in equation 4.4.

$$\mu = \frac{\sum w_i x_i}{\sum w_i} \quad \text{, where } w_i = \frac{1}{1+i^2} \tag{4.4}$$

A simple average over the 10 last poses was also implemented. While this was more effective than the weighted average, it was not as responsive, giving bad results when either the user or the robot was moving. Therefore, both stabilization modes were available while the system was running, letting the user choose the most fitting mode, depending on the situation.

4.2 Planning the user study

This section describes the design process of the user study. Different experiment types will be covered, rounding off with the chosen experiment for the user study. The section starts by defining the term understanding to reduce the scope.

Understanding To fit our context we define understanding to be the following:

> The ability to create a mental picture of the robot's situation, meaning its position, state, and view of the world around it.

This thesis focuses on point clouds. We therefore wish to investigate to what degree point clouds in augmented reality affect the user's understanding of the robot. Another term that will be used a lot in this chapter is *the traditional way*, which can be defined as follows:

> The current common technique of inspecting robotic sensor data in ROS, is by using the visualization program rviz. The user looks at the robot and scene, and inspects the sensor data on a computer monitor.

Experiment types Different options were considered. Both tests where the augmented reality system was tested isolated, and tests where it was compared with the traditional way of examining sensor data.

4.2.1 AR only experiments

An early idea was to study how the combination of visual quality and point cloud resolution affects an observer's understanding of the robot's situation. These two metrics are the two major variables in the system. The plan was to create a 3D color graph, showing understanding as a function of point cloud resolution and visual quality. The visual quality was controlled by adding Gaussian blur in combination with contrast reduction. By creating such a graph, the lower boundary for useful visualization can be located, and help show how the different factors affect understanding. Data would be gathered by letting the users perform a test, getting scores based on efficiency and accuracy. An example of such a test is the *counting-objects test*, where the users count objects in front of the robot.

A problem with the counting-objects test The user might favor full visual quality on the video feed, without the lowest point cloud resolution. This is because the points can get clustered together into a pink smudge, completely obscuring the object behind. This happens when many points are drawn in a small area in the image. Additionally, in good lighting conditions, the user has most of the information he or she needs, just by looking at the image, so there is no need for a point cloud. An exception is when objects hide behind other objects, making them possible to see in the point cloud, but not without.

A problem with AR only experiments in general An important problem with these types of experiments is that they only evaluates the AR system. They find which combinations of settings that work best, but they do not show how AR performs in relation to the traditional method. This type of experiment was thus abandoned, to further investigate experiments that compare the two methods, as this makes it much easier to answer the second goal of the thesis.

4.2.2 Experiments where AR is compared to the traditional method

Another experiment type is when augmented reality is compared to the traditional method. This type was found to be a better option, because it allows investigation on how AR performs compared to the traditional method. Considering the test where the users count objects, scores could be gathered based on efficiency and accuracy from *both* groups, which further makes it possible to apply statistical techniques to determine if there is a significant difference in the two groups.

4.2.3 Test design

Designing a test that measures understanding was challenging. Some keywords were chosen to help narrow down the test possibilities. These keywords were found based on previous work seen in user studies from related work. The first thing is perspective, as the user has to watch two different places(at the physical robot and at the monitor) to understand what the robot sees in relation to what actually is in front of the robot. The second thing is correspondence, which translates to the user's ability to understand how the point cloud on the screen corresponds to the scene in front of the robot. The last thing is dependency; to be solved, the test should require the user to look at both the point cloud and the physical scene in front of the robot. These key points are formulated in a list below.

1. The experiment should test the user's understanding of perspective in both methods.
2. The experiment should test the users' ability to understand what each point in the cloud corresponds to in the real world.
3. The tests should **not** be solvable solely by looking at the scene in front of the robot.

After the limitations were formulated, possible tests were looked at. The counting-objects test makes sense in an AR only experiment, but conflicts with the third point above, as the user can simply look at the scene in front of the robot to solve the task. Additionally, it would likely be an experiment in disfavor for AR, since the resolution in the goggles is quite low.

The invisible object test Another idea was to rely on the fact that the RealSense (robot sensor) has issues in detection of glass and black objects. This test would require the users to identify objects present in the scene that are invisible to the robot, and thus lacking from the point cloud. However, tests based on invisible objects originating from material usage were abandoned because of two reasons: The first, and most important reason was because of the *learning effect*, see section 2.6.1. As there was desirable to test each user several times with different object setups, there

is a risk of the user learning that the black and glass objects are the ones that are invisible, corrupting the following tests. The problem here is not that the users are getting slightly better at each test, but rather that they change their algorithm for solving the tasks once they discover that they only need to look for black and glass objects. The second reason this test was discontinued, was because the test is based around the fact that the RealSense has issues sensing some materials. This is a rather specific case, and it was thus more desirable to find a more general test.

The modified invisible object test The final result of the design process was based on the same principle as the *invisible object test*, but instead of using special materials to make objects vanish, a *faulty object detection algorithm* was implemented. This algorithm is faulty because it does not detect all the objects in the scene, making some of them invisible. This is described further in section 4.3.1. All the objects to be used in a scene should be of equal size, shape, and color. The choosing of which of the objects that are made invisible should be random. The user's task is to identify the objects visible to the robot by examining the point cloud.

Metrics Two metrics were used in the experiment: Efficiency and accuracy. Efficiency measures how fast the users solve the tasks in seconds. Accuracy measures how many errors they do. It is possible to make multiple errors on a single scene, since all the present objects have to be identified. A scene can produce from 0 to 6 errors, depending on the scene size. This makes it possible to analyze the data in multiple ways. The simplest method is to use a binary measure, that is, error or no error. This can be used to calculate an error rate.

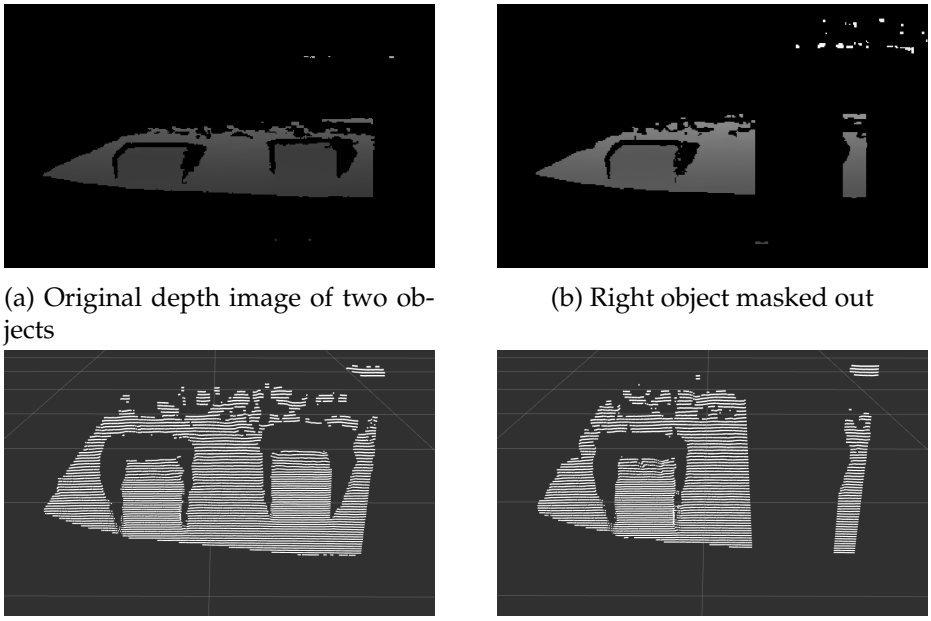
Another method is to sum the number of misclassifications in the scene. This method should be used carefully, since misclassification of one object can lead to misclassification of the rest of the objects. However, this fact can be used to measure the severity of the errors.

4.3 Framework for the user study

This section covers the materials and programming necessary in the user study. First, implementation details of the experiment found in the previous section is covered. Then, the physical objects to be used, data collection method, and the required programming is reported.

4.3.1 The faulty object detection algorithm

To make the experiment as similar to a real life case as possible, it was desired to use an object detection algorithm to detect the objects in the scene, and then hide a random subset of the objects from the point cloud. The user's task would then be to identify the remaining objects visible in the point cloud. This however was not done, since there is no guarantee



(a) Original depth image of two objects

(b) Right object masked out

(c) Original point cloud

(d) Masked point cloud

Figure 4.8: Masking of objects by erasing a rectangle from the depth image (b). The rectangle translates into a truncated square pyramid hole in the point cloud, as seen in (d).

for an optical object detection algorithm to always behave correctly. Since the user's accuracy is calculated by comparing the actual present objects with the user's answer, there can be no uncertainty about the actual present boxes. Therefore, another technique had to be developed.

The solution was to note the position of each object in the scene and mask out the corresponding areas in the point cloud. The masking was done by removing a square from the depth image, resulting in a truncated square pyramid in the point cloud, with the top side closest to the camera, and the base side furthest away, see figure 4.8. This introduced an unwanted effect when the surface the objects are sitting on is visible in the point cloud, as the resulting point cloud will have a "hole" in the surface where the removed object was. This required the surface beneath the objects to be hidden in the point cloud. Two methods to achieve this were conducted:

1. Filter the point cloud by defining a plane below the objects and removing all points below the plane
2. Use a invisible material as base for the objects

The latter was chosen, as matte, black, plastic trays were easily available and invisible to the RealSense.

4.3.2 The physical setup

This section describes how the objects to be used in the experiment were chosen, and their design process, from low to high fidelity prototypes.

Volatile setup Since the masking areas are predefined, the experiment setup is sensitive to movement. The relative positions of the robot and the scene in front it thus had to be constant. This required the scenes and their objects to be robust.

Selecting a fitting object The objects had to be large enough to easily be identified through the AR goggles, which suffer from low resolution. They also had to be made of a material visible to the RealSense. White 3D printed cubes were selected, as they are quick to produce, and tough.

Identification of objects The users needed a way to identify the cubes in the scene. The study was going to use different scenes, each having different numbers of cubes, with a maximum of six cubes. Two options on how to do this were considered. The first one was to write numbers on the cubes. This was tested with the AR goggles, and the numbers were found to be hard to read, because of the low resolution in the goggles.

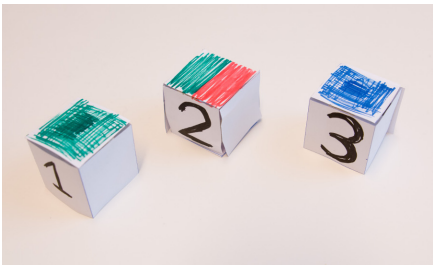
The next option was to mark the cubes with color. This limits the number of cubes in the same scene to the number of different colors available. Since the goggles suffered from poor visual quality, only colors easily distinguishable could be used, leaving red, green, and blue. This did not give enough combinations, so two colors for each cube was tested. Unfortunately the colors blended, making this approach impossible without using larger cubes.

The end result was larger cubes (size 4x4x4 cm) with big, black numbers written on them. A very thick line thickness was found to be crucial for the numbers to be readable through the goggles. The numbers also needed to have some margin from the edges on the cubes, to avoid blending with the black background. The prototypes, from low fidelity paper models, to high fidelity 3D printed are visualized in figure 4.9.

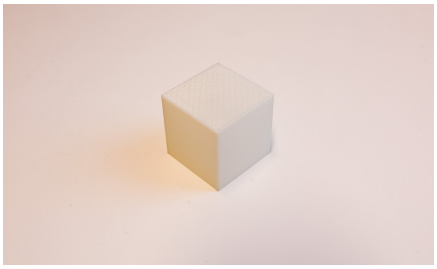
Different scenes A scene is defined as a black plastic tray with numbered cubes glued on top. Multiple scenes were needed, with a varying number of cubes, and different positioning of the cubes, both scattered and clustered. 6 scenes were constructed, in which scene *1* had one cube, scene *2* had two cubes, scene *3* had three cubes, and so on.

4.3.3 Data collection and questionnaire

Two measures were to be collected during each experiment: The number of errors, and the time spent. To make the experiments as efficient



(a) Low fidelity prototype.



(b) High fidelity prototype, number not shown.

Figure 4.9: Images of the cube prototypes.

and less prone to errors as possible, the data collection had to be simple. While the time spent was easy to measure, the errors were not. For example, scene *4* has 4 cubes. When the experiment starts, only a random subset of these cubes will be visible in the point cloud. The user compares the physical scene in front of the robot with the point cloud and identifies the visible cubes. At once the user states an answer, the time and answer has to be noted. To be able to validate the answer, the actual visible cubes had to be written down for comparison. Multiple options were considered to solve this challenge.

Option 1 The first option was to program the whole user study. For each scene, the computer chooses a random subset and then takes the user's answer as input, writing the result to file. Although this would be the easiest way to conduct the experiments, it would be a time consuming task to program, because of Unity's architecture.

Option 2 Another option was to generate the random subsets prior to the experiment and input them to the computer to make it mask the scenes either before or during the experiments.

Option 3, the chosen A third option was to generate the random subsets during the experiment, making the computer output the correct answer for the experiment operator to note. Back to the example with scene 4, the operator inputs that scene 4 is about to be tested, the computer generates the random subset which in this case was cube 1 and 3. Below the operator's interaction with Unity is shown.

```
> User study started.
> Mask activated on scene 4, cubes [1, 3] are visible.
```

When the operator has noted the correct answer (1 and 3), the scene is presented to the user, and the time is started. To avoid bias in the experiment, the users must not see the scenes before the time is started.

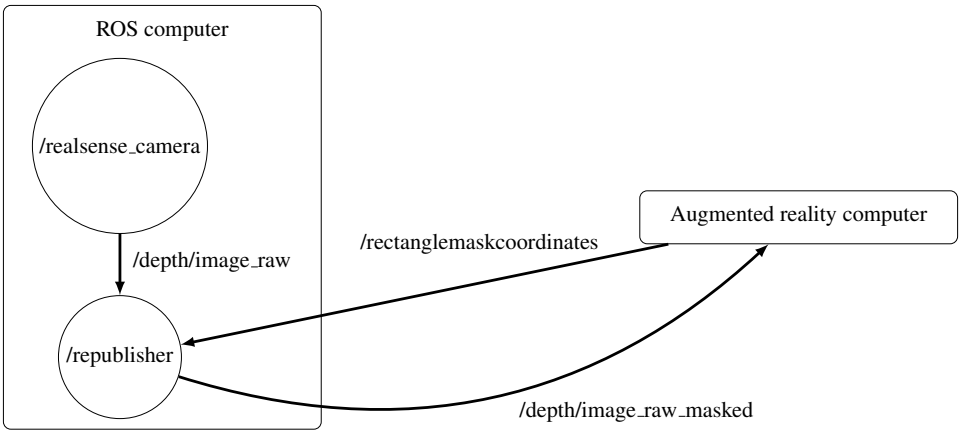


Figure 4.10: ROS graph during the user study. *rviz* is also subscribing to the republisher, but is not shown in the graph.

This was solved by letting the users sit with the goggles on, and with blacked screens, between the experiments. When an augmented reality test started, the screen was turned on. When a traditional test started, the subject removed or put the goggles on his or her forehead.

Questionnaire A questionnaire was created to learn more about the system. It included some quantitative questions, to determine which system they preferred and similar, and some qualitative questions, to find out *why* they preferred the chosen system and to identify problems with the two systems.

4.3.4 System architecture

The simplest way to mask out the cubes would be on the Unity computer. When a depth image arrived, *OpenCV* could be used to do the masking, before cloud construction. However, since the traditional method requires the users to inspect the point cloud on the ROS computer, the masking had to take place there. This meant that the depth images had to be edited on the ROS computer, before they could be visualized as point clouds in rviz, and sent to the Unity computer.

The masking ROS node A new ROS node was created. It was set to subscribe to the depth image topic (/depth/image_raw), and a new topic for masking instructions (/rectanglemaskcoordinates). When a depth image arrives, it is either left untouched, or masked, depending on the node's state. After the possible masking operation was done, the depth image was republished to a new topic (/depth/image_raw/masked). The system is visualized in figure 4.10.

Controlling the masking Since the Unity computer generates a random subset of cubes to mask, it was most convenient to send the masking

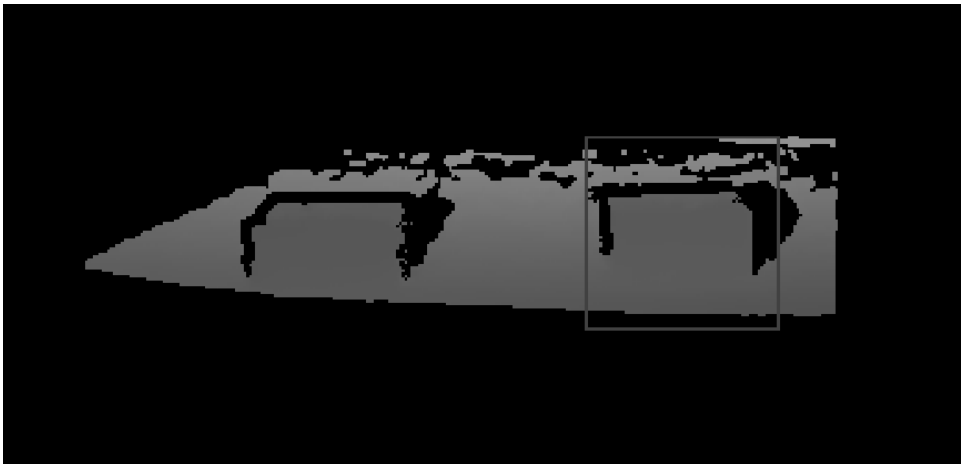


Figure 4.11: To find the correct coordinates and size of the each mask, the border of a dummy mask was drawn into the depth image. The mask was controlled with sliders in Unity (real time), making it easy to find all the mask positions.

instructions to the ROS computer, instead of having to input scene id and cube ids manually on the ROS computer. To achieve this, a new ROS message was defined, called *RectangleMaskMsg*. This message contains the coordinates (2D) and sizes of up to 6 squares to be masked out. On the receiving part, the masking node simply draws a black square covering the requested coordinates. The node holds this mask configuration until the masks are removed with an empty message or a new configuration is received. This required the positions of all the cubes to be known. There were 6 different scenes, with a total of 21 cubes. Every single cube's coordinates had to be found. This was solved by setting up Unity with parameter sliders to control the coordinates and size of a single mask, sending continuous masking updates to the masking node. The node was set up to draw the mask's border, to make it easier to use the smallest masks possible[11]. The mask adjustment is shown in figure 4.11. When all the cube positions had been located, a mask controller module was programmed in Unity. The controller could take a scene id as parameter, lookup the scene's cube positions, choose a random subset of them, print the chosen numbers to terminal, and generate and send a mask message to the masking node.

[11]Small masks were desired, as they allow cubes to be close, without the masks interfering with other cubes.

Chapter 5

Experiment and results

This chapter explains how the user study was conducted, its results, and analysis. The study is split up in two major parts, the main experiment, where accuracy and efficiency were measured, and a questionnaire.

5.1 The study

This section covers details of the study, and experiment setup. See section 4.2 for the planning of the study and reasoning in its design.

5.1.1 The invisible object test in detail

In the planning phase, the experiment was designed as follows:

> The experiment involves a robot with a faulty object detection algorithm making it unable to see all the objects in front of it. The user's task is to identify the objects that the robot *does* see.

The *objects* were numbered, white cubes. The cubes were glued onto black, plastic trays. These trays, or *scenes*, are invisible to the sensor, making the cubes the only visible objects for the robot.

Around 30 participants were found to be a sufficient number for the study. Based on this, it was decided that 5 different scenes would be sufficient. This translates to 10 tests for each subject, which should translate to below 30 minutes. To draw a statistical conclusion with the t-test, both accuracy and efficiency were to be calculated for each subject.

5.1.2 Experiment setup

This section describes how the experiment was set up in detail, including the experiment environment and how data was collected.

Figure 5.1: Experiment environment. Experiment controller (back) and subject (front).

Experiment environment The experiment was conducted in a narrow, but quite long room. The subject was seated in one end of the room, while the experiment controller was seated on a control computer in the other end, see figure 5.1. The laptop visible in the center of the image in figure 5.1 is the ROS computer, which is showing the point cloud in full screen mode. The subject can control the zoom and rotation of the view with a computer mouse, see figure 5.2. Approximately 80 centimeters in front of the subject, the robot prototype was placed, perpendicular to the subject, see figure 5.3. In front of the robot there was a platform with a black plastic tray on top. This tray will from here on be called the scene. The robot was angled in such way that the scene is the only object it could see.

Different scenes The 5 different scenes are shown in figure 5.4. Note that the plan was to use 6 scenes, but scene *1* was removed after the pilot study was conducted, more on this in section 5.1.5. The scenes were designed to be of increasing difficulty, assuming difficulty increases with the number of cubes in the scene. The cubes were arranged in different ways, both in a pattern (scene *6*), and in more random arrangements.

Randomization of the tests Because a between-group design was used, there were 10 tests in total (each of the 5 scenes was tested with both AR and the traditional method). To counter unwanted biases like learning effect, the sequence of the tests had to be random. The randomization was done with a JavaScript implementation[1] of the Fisher-Yates shuffle[57].

[1] https://github.com/Daplie/knuth-shuffle

Figure 5.2: The subject's seat and computer mouse to control the view of the point cloud on the laptop.

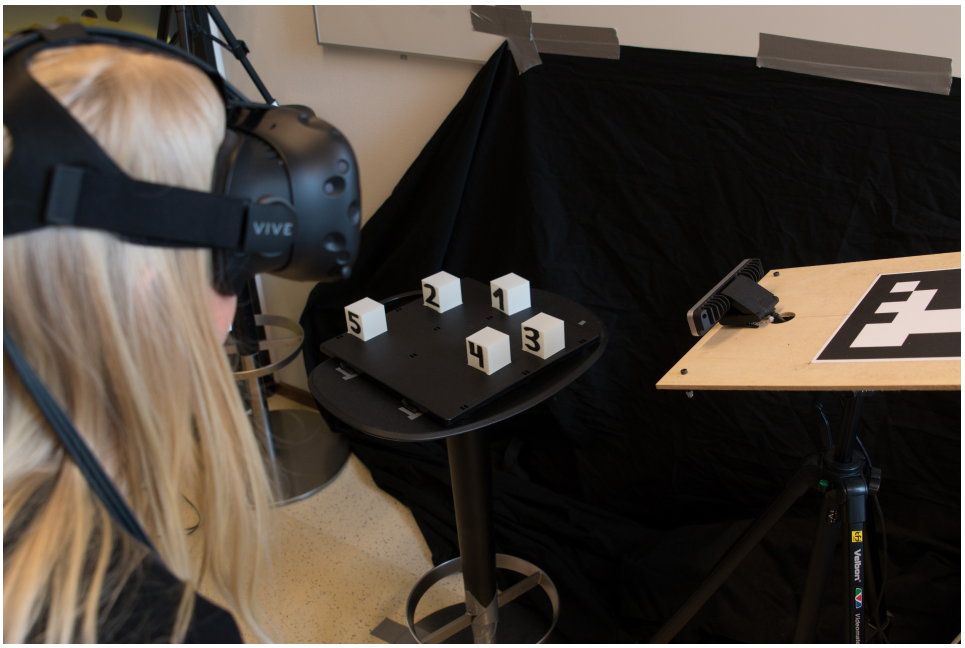
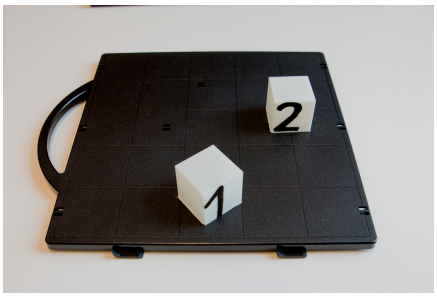
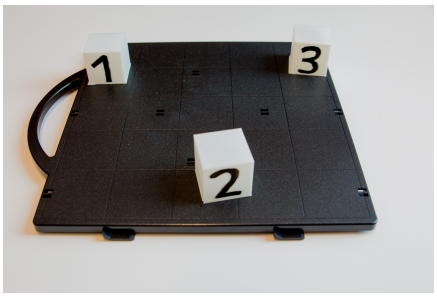


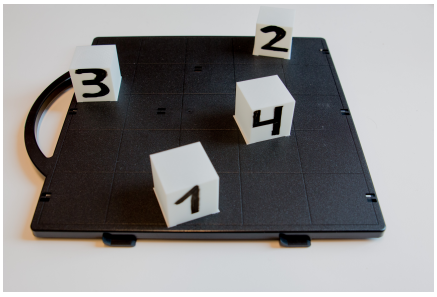
Figure 5.3: The subject's view of the robot and scene.



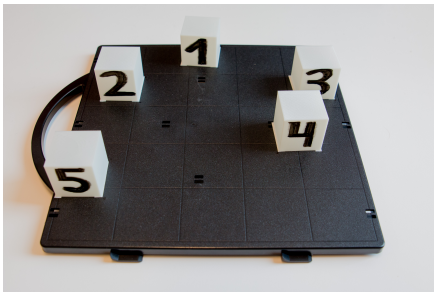
(a) Scene 2



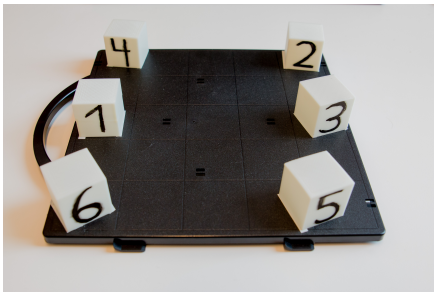
(b) Scene 3



(c) Scene 4



(d) Scene 5



(e) Scene 6

Figure 5.4: The 5 scenes used in the experiment.

Type	Scene	Actual visible boxes	Answer	Time (s)	Error
AR	2	(1) (2)	(1) (2)		
	3	(1) (2) (3)	(1) (2) (3)		
	4	(1) (2) (3) (4)	(1) (2) (3) (4)		
	5	(1) (2) (3) (4) (5)	(1) (2) (3) (4) (5)		
	6	(1) (2) (3) (4) (5) (6)	(1) (2) (3) (4) (5) (6)		
Trad	2	(1) (2)	(1) (2)		
	3	(1) (2) (3)	(1) (2) (3)		
	4	(1) (2) (3) (4)	(1) (2) (3) (4)		
	5	(1) (2) (3) (4) (5)	(1) (2) (3) (4) (5)		
	6	(1) (2) (3) (4) (5) (6)	(1) (2) (3) (4) (5) (6)		

Figure 5.5: Main experiment data collection form.

Form for data collection To be able to manually collect data during the experiments, forms with the 10 test combinations and columns for noting their time and error was produced. Each row has a corresponding condition, see figure 5.5. Note the 5 scenes, or conditions, numbered 2-5, appearing for both augmented reality and the traditional method.

5.1.3 Questionnaire

To learn about challenges with augmented reality and the traditional method, as well as how the users approached the problems, a questionnaire was designed. The study both included quantitative and qualitative questions, put in a natural order, according to the experiment. Quantitative questions are relatively simple to analyze and visualize, since the answers are predefined, e.g. yes/no. Qualitative questions cannot be visualized in the same way, but can identify important issues with the system, as well as uncovering the thought process of the individuals, using the different systems. Summarized, the questions were designed to investigate the following issues:

- Which system the users preferred
- What technique they used for solving the tasks
- Problems encountered with both systems
- Previous experience from similar systems

5.1.4 Execution of the study

This section describes how the user study was performed, including preparations before each experiment and the following execution. To make sure the experiments were executed as consistent as possible, an experiment procedure was created and printed.

Type	Scene	Actual visible boxes	Answer	Time (s)	Error
AR	2	(1) (2)	(1) (2)		
	3	(1) (2) (3)	(1) (2) (3)		
	4	(1) (2) (3) (4)	(1) (2) (3) (4)		
	5	(1) ~~(2)~~ (3) ~~(4)~~ ~~(5)~~	(1) ~~(2)~~ ~~(3)~~ (4) ~~(5)~~	6	1
	6	(1) (2) (3) (4) (5) (6)	(1) (2) (3) (4) (5) (6)		

Figure 5.6: Cutout from the experiment form. The first test is AR-5, augmented reality, scene *5*. The computer generated the subset [2, 4, 5] and made the cubes visible in the point cloud. The subject reported that he saw cube [2, 3, 5], after 6 seconds. This means that the subject at least misclassified one cube, hence the one error.

Before the experiment The first ting checked was the tidiness of the experiment room. Then a verification that all position critical equipment (the subject's chair, ROS computer, robot, scene, computer mouse.) were in the correct positions, indicated with tape markings on the floor/table. The next thing was to make sure a new data form with a random test sequence strip glued onto it[2] was ready. The ROS computer was confirmed to have the correct screen angle, and that it visualized the point cloud in full screen correctly in *rviz*,[3]. Rotation of the view with the mouse was also tested. The Unity computer was tested by generating random masks for scene *5* a few times, verifying that the masking was working, both seen through the Vive, and in *rviz*. The blackening of the goggles was tested, and the goggles were left in that mode. The Vive headset was placed by the computer mouse. Then all masks were removed, and a test scene was placed in front of the robot. The test scene had two cubes, and was used in the demonstration during the experiment brief.

The experiment The subject was welcomed and placed in the chair. After getting information about how the experiment was going to be conducted, a demonstration was done, letting the participant try both systems on the test scene, to understand how the missing cubes were visualized. After the demonstration, the subject got a chance to ask questions if something was unclear. The 10 tests were then run according to the random test sequence. Each test was performed strictly according to the flow chart in table 5.1. *Write answer* and *write time* means to note the users answer and time on the form, as shown in figure 5.6. After the 10 tests were done, the headset was removed, and the questionnaire started. The questionnaire was formed as an interview, except for the fact that it was not a dialogue. The experiment operator transcribed the subjects' answers as accurate as possible[4]. When the experiment was over, the subject was thanked sincerely and given a small chocolate.

[2]Paper strips with random test sequences were printed and glued onto the data forms.

[3]A default view was created to make sure all the users had the same view.

[4]Transcription of audio or video recordings after the experiments would have required too much time.

Input current scene to Unity to update mask	
Write actual visible cubes	
Change the physical scene	
Reset rviz view and verify that the masking was successful	
"The next test is [AR / Traditional]"	
AR	**Traditional**
Get ready	Take off headset when I say start
“Start” + start timer + screen on	“Start” + start timer
Stop timer + write answer	Stop timer + write answer
Press space (black screen)	"Put the headset on again please"
Write time	Write time
Thank you	Thank you

Table 5.1: Flow chart showing the main experiment procedure. Both AR and traditional tests have the same start. A quoted phrase implies that it should be read out loud to the subject.

5.1.5 Pilot studies

To improve the study, two pilot studies were performed. As recommended in [42], multiple pilot studies should be performed. The users should be from the target group, not from the department. However, because of limited time, there were only conducted two pilot studies, each with one subject. The first subject was from the department, and the second was from the target group. The pilot studies were performed to identify possible *biases* by analyzing the experiment procedure. As discussed in section 2.6.2, *biases* can affect the results of a study and must therefore be avoided.

Pilot 1 The first pilot was performed with a subject and the experiment leader, as well as an observer. The observer was participating to provide a different view on the conduction of the experiment, and detecting undesired factors, like bias from the experiment leader. After the experiment, the subject said it was uncomfortable to sit in complete darkness between the experiments. Therefore a simple white rectangle was placed in the middle of the darkness, providing something to look at. The subject reported that the script was lacking information, as he did not feel he knew what was coming when the experiments started, introducing stress. Another issue with the insufficient script was that the view in rviz was not explained well enough, as it was unclear where the sensor and robot was, relative to the cubes. The script was thus expanded to cover this. Lastly, the subject said he was a little disappointed that he did not get anything after the experiments, like for instance a small chocolate. Chocolate was thus included, starting in the second pilot. In general, it was hard to explain the experiment procedure to the subjects, and a demonstration part of the briefing was thus introduced in the second pilot.

Edge cases In the first pilot study, the subject noted that he got confused if either none or all of the cubes in a scene were visible. A lot of discussion followed. These two cases were defined as edge cases. The number of cubes to mask out was initially random, which produced edge cases. It was hypothesized that the traditional way of solving an edge case simply was to count the cubes in the scene and on the screen. The user can thus conclude that all the objects are present in the scene. The point of the experiment was to test the user's understanding of the relationship between the point cloud and the real world; this is however not tested efficiently when the users simply count the objects.

By excluding the edge cases, it was believed that the results from the tests would be more consistent, since it would exclude the events where the subject use extra time, in confusion. More consistent results are beneficial because it reduces the variance, and thus makes it easier to draw a statistical conclusion. Another consequence of removing the edge cases was that scene *1* had to be excluded, since it would have always been an edge case.

Wording The experiment was originally designed to identify the *missing* cubes, not the visible ones. However, the subject reported that this was confusing, and made the task harder. The wording was thus changed from *"Identification of the cubes that are missing in the point cloud"* to "Identification of the cubes that are present in the point cloud", as this was believed to be more intuitive for the users.

Pilot 2 The second pilot study was conducted a few days after the first one, and the changes had been implemented, with a new subject. No observer was present in this pilot. The subject reported that the demonstration before the experiments worked well, as it removed his uncertainties in what was about to happen. The subject said the white rectangle implemented after the first pilot was too bright and straining to the eye. The rectangle was thus colored with a dark shade of gray.

After the two pilot studies were conducted, the script read to the subjects at the start of the experiments was heavily expanded. For example, both subjects said it was hard to see the numbers with the AR system, and the script was thus expanded to explicitly warn about this issue, and to lean forward to see the numbers better.

5.2 Main experiment results and analysis

This section presents the results and analysis from the main experiment. 31 students (16 males and 15 females) from the Department of Informatics at the University of Oslo participated in the study. The study was conducted over three consecutive days. Each experiment took approximately

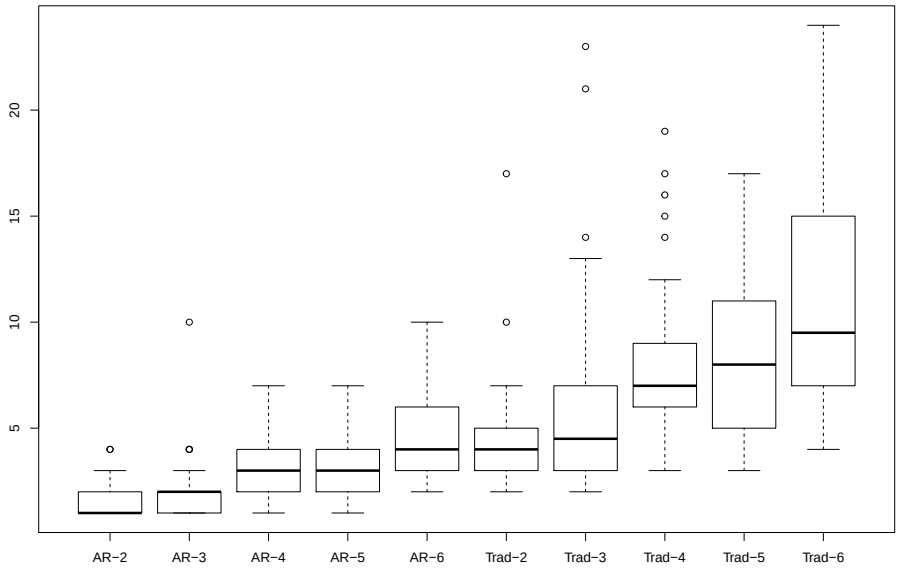


Figure 5.7: Time spent on the different tasks, with augmented reality (left hand side) and the traditional method (right hand side). The data clearly shows that augmented reality is more time efficient than the traditional method.

25 minutes to conduct. Even though the results come from the same experiment, the time (efficiency) and error (accuracy) results are split up to provide a better overview.

Handling of subjects with impaired vision Some of the subjects used glasses, which did not fit inside the AR headset. They were comfortable with taking them off, since their vision was not more nearsighted than -2. In the traditional tests, they were allowed to take off the headset while holding their eyes closed and to put on their glasses before the experiment started.

5.2.1 Efficiency results

In figure 5.7, all the recorded times are plotted by scene. Scene 2-6 with AR can be seen on the left side, and the same scenes with the traditional method on the right side. The data clearly indicates that the subjects used less time with the AR method than with the traditional method. The results are listed in table 5.2.

The efficiency was found by calculating each subject's mean time for both AR and traditional method. This means that the 5 AR time measures and the 5 *trad* time measures for each subject were converted into one

Scene	Min	Max	Mean	S.D.
AR-2	1	4	1.645	0.877
Trad-2	2	17	4.677	2.821
AR-3	1	10	2.161	1.734
Trad-3	2	23	6.290	5.054
AR-4	1	7	2.871	1.408
Trad-4	3	19	8.226	4.121
AR-5	1	7	3.097	1.620
Trad-5	3	17	8.290	3.977
AR-6	2	10	4.548	2.142
Trad-6	4	24	10.970	5.671

Table 5.2: Summary of time spent on the different tasks, with augmented reality (top half) and the traditional method (bottom half).

Method	Min.	Max.	Mean	S.D.
Trad	4.400	15.600	7.690	2.934
AR	1.200	5.000	2.865	0.956

Table 5.3: Summary of time efficiency for the traditional method, and augmented reality group in seconds.

AR measure, and one *trad* measure. All answers were included, both the correct and the incorrect ones. These two measures, AR efficiency and traditional method efficiency are illustrated in figure 5.8, details can be found in table 5.3. Calculating a mean time for each method (AR and *trad*) on each subject makes it possible to apply a t-test, making comparison of the two methods possible.

5.2.2 Efficiency analysis

To test if augmented reality is significantly better than the traditional method, a null hypothesis and an alternative hypothesis was formed:

> $\mathbf{H_0}$: Using augmented reality goggles does not improve recognition time above traditional methods.
>
> $\mathbf{H_A}$: Using augmented reality goggles improves recognition time above traditional methods.

Under H_0 the test statistic is t-distributed with n-1 = 30 degrees of freedom. $t = 9.276$, corresponding to the one-sided p-value of 1.28e-10. The null hypothesis may therefore be rejected, and we conclude that augmented reality is significantly faster than the traditional method.

5.2.3 Accuracy results

The accuracy was measured in the user's ability to correctly identify which cubes the robot saw. The number of misclassifications for each

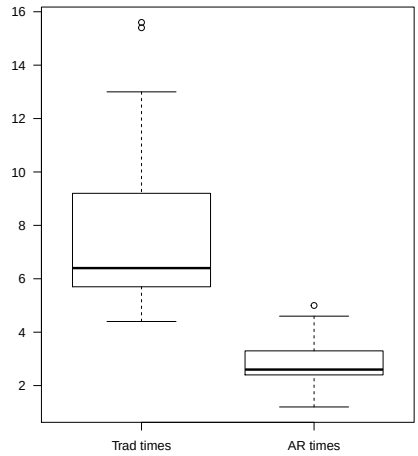
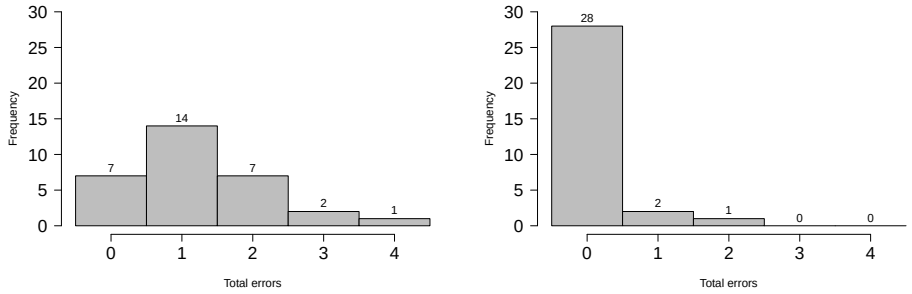


Figure 5.8: Time efficiency for the traditional method, and augmented reality group in seconds. The data clearly indicates that augmented reality is faster than the traditional method.

task was recorded, both for AR and the traditional method. Each subject did 5 tests with AR, and 5 tests with the traditional method. Although the number of misclassifications was recorded for each test, the dataset's complexity was simplified by classifying each test as a pass or a fail. The total number of failed tests per subject, in each method, is shown in the histograms in figure 5.9 and table 5.4.

28 out of the 31 subjects (90%) of the subjects passed all the tests with augmented reality. This is much higher than the result from the traditional method, where only 7 passed all the tests (23%). The probability of failing at a given task with the traditional method was 0.245, compared to 0.026 for



(a) Traditional method (b) Augmented reality

Figure 5.9: Histograms showing the total number of failed tests per subject in both methods.

Method	Min	Max	Mean	S.D.
Trad	0	4	1.226	0.990
AR	0	2	0.129	0.428

Table 5.4: The number of failures per subject with the two methods. Out of the 10 tests (5 AR, 5 *trad*) conducted with each subject. On average, the subjects failed once with the traditional method, and did not fail with augmented reality.

Method	Probability of failure
Trad	0.245
AR	0.026

Table 5.5: The probability of failing at a given task with the traditional method, and augmented reality. The data shows that failure is almost ten times more likely with the traditional method.

augmented reality. This means that failure is almost ten times less likely with augmented reality; see figure 5.10 and table 5.5.

To make the t-test applicable, only two accuracy measures for each subject was desired; one for AR and one for the traditional method. This was achieved by calculating the mean of the 5 measures for each method, and results in a new dataset with 31 AR measures and 31 *trad* measures.

This method of examining accuracy can be called the error rate, which is how often the users make errors. With this method, each scene can only produce one error. Another way of investigating the accuracy is by examining the severity of the errors, as the subjects can make multiple misclassifications on a single scene. Table 5.6 shows that there were few severe errors in general; 4 in total. All of them originated from the traditional method, one from scene *5* and three from scene *6*. The scenes can be found in figure 5.4. The highest severity recorded was 2 (two misclassified cubes in a scene).

Error distribution Another interesting result is how the errors are distributed. A box plot is in this case not particularly useful, since the data

Method	Severe errors
Trad	4
AR	0

Table 5.6: The number of severe errors, defined as multiple misclassifications in the same scene/task. There were few severe errors in general, in which none occurred with augmented reality.

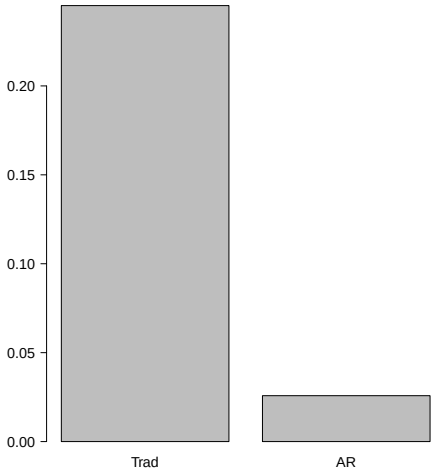


Figure 5.10: The probability of failing at a given task with the traditional method, and augmented reality. The data clearly shows that failure is much more likely with the traditional method.

has very few levels, thus, a bar plot was chosen. The error distribution is visualized in figure 5.11.

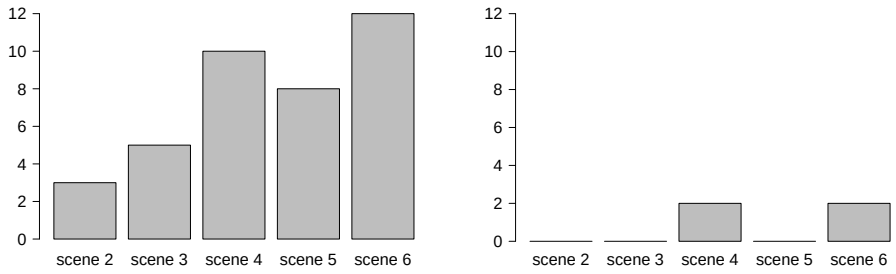
Although the scenes were designed to be of increasing difficulty, where scene 2 is easiest, and scene *6* is most difficult, the data shows something else. The error distribution in figure 5.11, show that scene *4* produced more errors than scene *5*. The rest of the scenes' difficulties appear as intended.

5.2.4 Accuracy analysis

To test if augmented reality is significantly better than the traditional method, a null hypothesis and an alternative hypothesis was formed:

> $\mathbf{H_0}$: Individuals using augmented reality goggles have an equal or worse understanding of what the robot sees than with traditional methods.
>
> $\mathbf{H_A}$: Individuals using augmented reality goggles have a better understanding of what the robot sees than with traditional methods.

Under H_0 the test statistic is t-distributed with n-1 = 30 degrees of freedom. $t = 5.677$, corresponding to the one-sided p-value of 1.724e-6, which is lower than 0.001. The null hypothesis may therefore be rejected, and we conclude that augmented reality gives significantly better understanding than the traditional method.



(a) Traditional method (b) Augmented reality

Figure 5.11: Plot showing how the errors were distributed in the scenes/-tasks with the traditional method (a) and augmented reality(b). The data shows the sum of all the errors that occurred in each scene/task. Although the scenes were designed to be of increasing difficulty, the data show that scene *4* produced more errors than scene *5*.

5.3 Questionnaire results and analysis

This section presents the results and analysis of the data collected from the questionnaire. First the quantitative (yes/no questions) data will be presented, followed by the results from the long answers (the qualitative data). All of the 31 subjects who participated in the study answered 10 questions after the main experiment was finished. Answering the questionnaire took approximately 7 minutes per subject. The questionnaire was conducted as an interview, where the researcher asked the questions. All questions asked are listed below:

1. Which system do you think was better for understanding the robot's situation? [Trad, AR, Equal]. Why?
2. Do you feel that the augmented reality method gave you a better understanding of the robot's situation? [Yes, No]. Why / why not?
3. How did you solve the tasks with the traditional way?
4. Did you control the camera in the traditional way? [Yes, No] Why/why not, and did it help?
5. How did you solve the tasks with augmented reality?
6. Did you encounter any problems with the traditional way? Rate seriousness. [Low, medium, high]
7. Did you encounter any problems with the augmented reality way? Rate seriousness. [Low, medium, high]
8. Which system do you prefer for solving these tasks? [Trad, AR]

9. Do you think Augmented reality can improve understanding of robot systems like this? [Yes, No]

10. Do you have any previous experience that you think might have affected your ability to solve these tasks?

5.3.1 Quantitative data

The quantitative questions can easily be visualized. The results can be inspected in figure 5.12. 5.12a and 5.12b both represents questions about which system the users preferred. Although they are almost equally formulated, question 5.12a also asked the subjects to reason their decision, while question 5.12b was toward the end of the questionnaire, allowing the users to reflect on difficulties with the two systems before answering the second question. The two plots show no difference, all subjects preferred augmented reality.

5.3.2 The 31 subjects preferring augmented reality

The participants reasoning in why they preferred augmented reality over the traditional method is best summarized in quotes:

> "It was so much better to see what I was looking at. The glasses highlighted the correct boxes, it was so easy. In the traditional way, I had to think. The camera's perspective was different than my own, so I had to imagine that I was where the camera was."
>
> "I got the depth that was linked to my movements, instead of the mouse's movement, in the traditional method. The connection with reality was great." (Translated)
>
> "I know where I'm in the room, so I did not have to see where the robot is and put me into its eyes. I saw myself as the robot when I wore the glasses." (Translated)
>
> "Easier to understand how the robot sees the world, in the way we humans see the world." (Translated)
>
> "The fact that I get all the information at the same time made it a lot easier. With the traditional method, I had to look back and forth, take the model in 3D and transfer it to what I see. It would be much harder in a real situation, since things do not move here." (Translated)
>
> "Like seeing it yourself, just a little further away." (Translated)

The above quote addresses the fact that the field of view with the goggles are a little shallower than human's field of view, making the scene appear further away than it really is.

> "You could see the boxes in relation to each other. Although it was difficult to see the whole model because I saw it from the side, unlike the traditional way, where you saw the whole." (Translated)

The above quote addresses an advantage with the traditional way, at least when the observer is seated and unable to move.

Do you feel that the augmented reality method gave you a better understanding of the robot's situation? The response to this question is visualized in figure 5.12c. Only 2 of the 31 participants said they did not feel they got a better understanding of the robot's situation. Their reasons:

> "No, it is easy to mix what the robot sees with what you see in AR. In the traditional method you see exactly what the robot sees. In AR it kind of looks like the robot just selected something." (Translated)

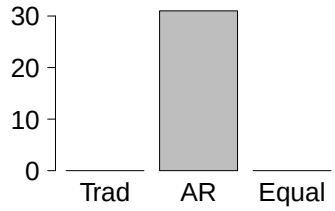
> "No, I understood it OK." (Translated)

5.3.3 Solving the tasks with the traditional method

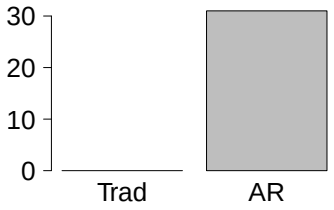
To improve the current methods of solving a problem, it is important to first understand the current methods and what difficulties there are. This section summarizes what the participants reported as their techniques to solve the tasks with the traditional method.

Centerline As presented in section 5.1.4, before the experiments started, the subjects were introduced to the 3D program. This included two important things: Where the sensor was in the model, and in which direction, relative to the sensor, the view was aimed. The participants where explicitly told which of the grid lines was the centerline. The centerline is the imaginary line pointing straight out in front of the sensor. 17 out of the 31 participants (~55%) reported that they solved the tasks by examining the cube's positions relative to the centerline in the 3D model. They saw which of the cubes in the model were to the left and to the right of the centerline, and compared this to the real cubes in the scene in front of the sensor, taking note of which seemed to be to the left and to the right. This was most likely not very easy, since they had to stay in their chair, limited to lean around.

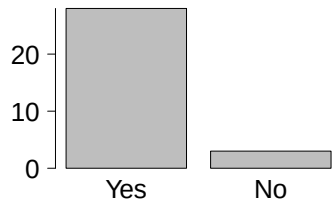
The relative positions of the cubes Another common reported technique was to compare the relative positions, or pattern, between the cubes in the 3D model with the physical cubes in the scene. If three cubes were visible in the 3D model, and they formed a perfect equilateral triangle, the subjects looked for such a relation among the physical cubes in the scene. As human beings are good at pattern recognition[58], this technique is effective.



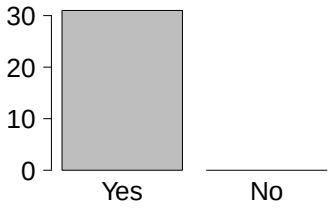
(a) *Which system do you think was better for understanding the robot's situation?*



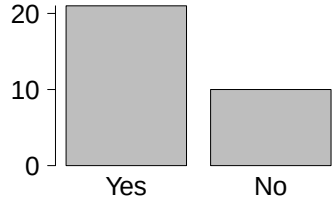
(b) *Which system do you prefer for solving these tasks?*



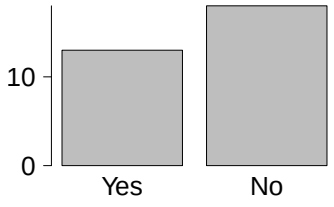
(c) *Do you feel that the augmented reality gave you a better understanding of the the robot's situation?* 2/31 participants answered "no"



(d) *Do you think Augmented reality can improve understanding of robot systems like this?*



(e) *Did you control the camera in the traditional way?* 21/31 participants controlled the camera during the traditional method



(f) *Do you have any previous experience that you think might have affected your ability to solve these tasks?* 13/31 of the participants said they had previous experience

Figure 5.12: Quantitative results.

Seeing from the robot's point of view 8 of the 31 subjects said they tried to put themselves in the robot's position, imagining how the physical cubes would look from the robot's angle. A quote from one of the subjects goes as follows: "[...] I put myself into the sensor's place. How would it look to me?" (Translated)

A comment on the scene's orientation relative to the sensor One of the subjects commented the following: "Used the center line to see right/left. A bit difficult because the board was not aligned with the sensor [...]" (Translated). The plastic trays the cubes were sitting on were intentionally rotated slightly. The subject implied that the tasks would be easier if the square trays were perfectly perpendicular in relation to the sensor's centerline. The reason for the slight rotation was to avoid reflection from the sensor, as well as it represented a more realistic situation.

Rotation of the view in the traditional way 21 out of the 31 participants tried to rotate the view in the 3D program, which of 15 said it helped them in gaining information. Most said it helped in seeing the depth between the cubes better.

The rotators 15 of the 31 subjects (~50%) found it useful to rotate the view. One of the participants said rotating helped see the grid better. The grid is default in the 3D program, and makes it easier to see the depth. Below, some of the answers are quoted (all translated from Norwegian):

- "Yes, sometimes, it helps to see things from another perspective. It helped when it was difficult to see the distances between the boxes."
- "Yes, to get a better understanding of how the boxes were in relation to each other."
- "Yes, to check if it helped. It did not help so much. And to see the routes between clearer, I mean the grid."
- "Yes, to see where the sensor was. It sometimes helped, especially when only one box was visible."

Observing the scene from above In total, 4 subjects mentioned the top-down approach (rotating the view to a top-down angle). One participant reported that it helped see the depth better. Another reported having thought of it, but did not use it because it would take more time. Two reported they came to think about the top-down approach only after they had completed the test.

The non-rotators 16 of the 31 subjects (~50%) did not rotate the view at all, or did not find it helpful after trying. Many said it did not improve their overview, as the default view was sufficient. Below, some of the answers are quoted (all translated from Norwegian):

- "Yes, once. I tried to rotate to compare how it looked from the side."
- "I rotated the view one time, hoping it would help me see the depth between two boxes, where the one was behind the other. It did not help, I did not gain any information."

5.3.4 Difficulties with the traditional method

This section summarizes common issues the users experienced using the traditional method during the experiments.

> "If two boxes were close, but one was missing, it was hard to tell which it was." (Translated)

The above quote addresses a common issue and was reported by multiple subjects. It appears in scenes where groups of two cubes are placed close, making it difficult to tell which of the two cubes is visible in the 3D model.

> "It was hard to identify the cubes when you see the same pattern between the boxes in the point cloud and several places in the physical scene." (Translated)

> "It was difficult, especially if there are few items visible in the point cloud." (Translated)

The above quote confirms that the intended design worked, as some of the scenes were designed to be difficult in this manner. In total, 4 subjects reported this specific issue.
"I was not sure if I chose correctly" (Translated)

Lack of perspective 8 participants said they had difficulties from the lack of perspective. The following quotes addresses the issue well:

> "It was very difficult to get perspective, since I don't know where the tray is in the scene, and I don't know the distances from the cubes to the camera." (Translated)

> "Because we only saw the points it was difficult to relate it to reality." (Translated)

Lack of depth perception 8 subjects reported that they had trouble seeing the depth in the 3D model. The grid in the 3D model makes it possible to see the relative depth difference between the cubes, but without deeper knowledge about the grid size it does not show the actual depths in relation to the physical scene.

Difficulties with few visible cubes 5 participants said few visible cubes made the task much harder.

> "With only 1 visible box in the 3D model, I had no relation to compare with." (Translated)

5.3.5 Solving the tasks with augmented reality

The feedback on how the participants solved the tasks with augmented reality was quite short, compared to the traditional method. Most subjects answered with a single sentence on the form: "I looked where the pink dots where, and read the number on the cube.". Most of the subjects hesitated on this question as they were not sure how to explain it because they thought the procedure was very simple.

> "It was pretty straight forward, I saw which of them that were illuminated." (Translated)

> "I looked at the real world, and searched for places where there was pink, that's where the robot was looking." (Translated)

5.3.6 Difficulties with the augmented reality system

Compared to the issues reported with the traditional method, the following issues are of a more technical matter.

Resolution issues 16 out of the 31 participants reported low resolution in the goggles as an issue. Most of them said it was of low seriousness, since they only had to lean closer to read the numbers. One of the subjects reported it was hard because of visual impairment. The participants with glasses did not wear them during the experiment, as they do not fit well inside the goggles.

Unstable point cloud Another issue reported was that the point cloud disappeared when the subjects leaned towards the cubes to see the numbers more clearly. The system is dependent on keeping the robot and its marker within the field of view to function properly. If a participant leans closer, but fails to keep the marker within the frame, the point cloud disappears until the marker is back in the frame. 7 participants reported this as an issue. The following quote shows this well:

> "It was hard to lean closer to see better, because it made the points disappear." (Translated)

Noise and point intensity 4 of the subjects reported that they sometimes were confused by noise in the scene. There is some random noise, observed as individual points, or small groups, in the scenes.

> "I assumed the noise that was around could be a goal, so I had to lean closer to see better." (Translated)

Another subject said that some of the cube surfaces were more illuminated in pink than others, causing confusion. The reason for this is the angle of the cubes, as the sensor senses surfaces at certain angles better than others.

5.3.7 Previous experience

13 of the 31 participants said they had previous experience that they thought might have affected their performance. Most of the 13 had used virtual or augmented reality goggles before. One reported experience with *rviz* (the 3D program), and another said that the their academic background in robotics might have helped.

Chapter 6

Discussion

This chapter put the findings from the experiments into perspective.

6.1 The study

This section takes a deeper look at how the study was designed, and which consequences the design choices might have had.

6.1.1 Thoughts on the study design

This section evaluates the design of the study, and discusses what could have been done differently.

Time stress Before the main experiment started, the users were told "You should state your answer as fast as possible. You have 30 seconds for each task.". This was done to improve their performance, as discussed in section 2.1.1, under *Medicine*, where users under moderate stress were found to perform better. A study on HCI, on menu selection systems, found that time stress actually decreased the users performance, both accuracy and efficiency[59]. Since these two studies found the opposite result, this study might have seen better performances if the participants were told to take their time. It would probably not have affected the measured difference between augmented reality and traditional methods, since both methods were subject to the time stress effect.

Interpretation of the user's accuracy As discussed, there are different ways of measuring the accuracy. Although the original plan was to record the number of misclassifications in a scene, the data was transformed into a binary measure, error, or no error. It is important to notice that the way the experiment was designed, made it possible for a subject to misclassify cubes without it being recorded. This can happen if for example two cubes are visible, and the subject classifies the first as the second, and the second as the first. This is a possible weakness in the experiment, but was not believed to have a significant impact on the results.

Accuracy resolution There was a limited resolution on accuracy, between 0 and 5 errors per subject. The scores on were fairly high, making the score distribution very skewed, as most of the subjects had 0 errors with AR, and 1 error in the traditional method. This suggests that the tasks were too easy, as a less skewed distribution would have been better.

Correlation between questionnaire data and error severity Observing that the error severity was much worse in the traditional way might further confirm the users' description of their technique in solving the tasks. The users described their technique for solving the tasks with the traditional method as very dependent on the other cubes. They first found one cube, and then used that information to identify the others. This suggests that multiple misclassifications easily can occur if the first cube is misclassified.

Users limited to the chair Whether to let the users move around as they pleased or to limit them to the chair was a big topic during the design phase. As seen in the results from the questionnaire, more than 50% of the participants used the centerline to identify the cubes with the traditional method, see section 5.3.3. Since the users were not allowed to move from the chair, understanding where the actual centerline coming out of the physical sensor might have been harder than in a real use case, where the users are able to move around and look at the system from above. The choice in limiting them to the chair was done to limit the amount of factors in the experiment. It was hypothesized that some users would move around, gaining an advantage above the more discrete users, not leaving the chair. This decision might have come in favor to augmented reality, since its way of solving tasks is so simple, compared to the traditional method.

6.1.2 A deeper analysis of the scenes and cubes

From the main experiment results, especially in accuracy, scene 4 produced more errors than scene 5. Recall that the scenes were numerated according to difficulty, in ascending order. This is also visible in the efficiency scores, where the two scenes come out very close. These two findings might confirm that the fourth scene in fact is harder than the fifth.

This can be reasoned in multiple factors. As assumed before the experiment, and supported by the questionnaire, the positioning of the cubes are crucial for the difficulty of the scene. Another possible factor is confusion between the cubes. It was observed that some participants mixed cube 4 and cube 1, this most likely happened because of the similarity of the two numbers combined with low resolution. Additionally, as the numbers were hand written, and were not consistent in all the scenes.

This was not investigated further, as it was not prioritized due to the fact that it could only have affected the augmented reality system negatively.

The conclusion would therefore not likely have been affected. If such an inconsistence exists in e.g. one of the scenes, a new statistical analysis can be conducted, excluding the data from that specific scene.

Issues with the augmented reality system The issues the participants experienced with the augmented reality system can be said to be prototype-specific, as both low resolution and an unstable system will not likely be an issue in a future system.

6.1.3 Bias in the study

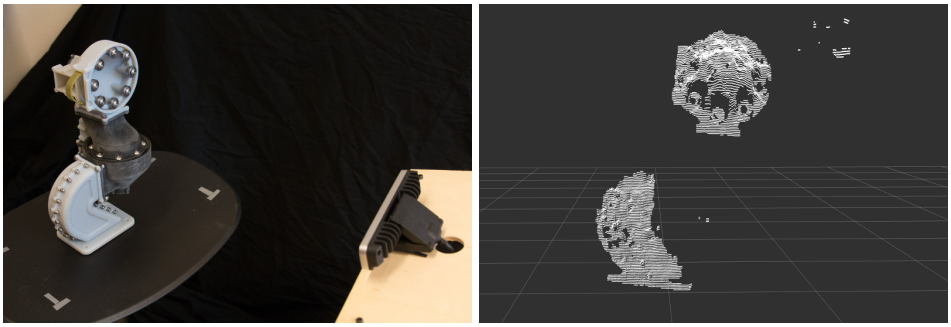
Running a study alone does not come without the risk of bias. Both the main experiment and the questionnaire were at risk of bias. As discussed in 2.6.2, in the paragraph about errors, the researcher is at risk of biasing the participants in the study. This can happen both intentionally or unintentionally. In the study, the experiment leader did his best to not to say anything else than what was scripted, as this can easily bias the subject. Also, the experiment environment was kept as equal and clean as possible to avoid environmental bias. From the quantitative data in figure 5.12, we observed that all subjects preferred the augmented reality system over the traditional one. This is a suggestive type of question on the form "Do you like the amazing system I created?". Since all the subjects agreed that AR was best, there was reason to suspect bias. This could have been avoided if an independent person had conducted the experiment, instead of the creator of the system. Such bias is technically possible in the main experiment as well, as subjects can intentionally fail with the traditional method, to *help* the researcher. However, the main experiment was conducted in a controlled manner, so we have little reason to believe it was biased enough to affect the conclusions.

Unfair setup? In the traditional tests, the users had to take the goggles off to be able to see the scene. This operation took approximately half a second, and may have introduced bias in the study, by making the traditional efficiency scores marginally worse. However, the augmented reality system was unstable and had low visual quality, compared to the completely stable traditional method. This suggests that the bias would have little effect compared to the effect of an improved prototype.

Removed edge cases Recall that the edge cases, where none or all the cubes were visible, was excluded[1]. This design choice might have affected the results, but it is hard to tell in which direction. Although the edge cases were believed to be much easier to solve in the traditional method than the normal cases[2], they confused the subject, as seen in the pilot study. This

[1] If the program produced an edge case, it started over, until the subset of visible cubes was not an edge case.

[2] What makes the traditional method hard compared to ar, is that the cubes have to be identified by looking at the relative positions of the visible cubes. If edge cases were



(a) Part (b) Traditional method

(c) Augmented reality

Figure 6.1: Part and its point cloud. Traditional visualization method compared to augmented reality.

suggests that the edge cases could have improved the accuracy, especially in the traditional method, but reduced the efficiency in both methods.

6.2 Visualization in augmented reality vs. virtual reality

During the implementation phase, visualization of the point clouds was tested in both VR and AR. This section discusses some major differences. A part consisting of different materials is pictured in figure 6.1. The part's point cloud is visualized with both the traditional method (rviz), and with augmented reality. Visualizing the point cloud in VR would looks very similar to the traditional method, except for a few important differences.

Visualization in virtual reality The first working point cloud visualization in Unity was in virtual reality. This was an interesting experience, as it was much easier to understand the point cloud in 3D, as opposed to with traditional methods (on a monitor in a 3D visualization program), because of the depth perception offered by stereo vision. Additionally, navigation in the point cloud is intuitive, since the view is controlled by the user's movements. This suggests that a user without experience can handle the navigation better in VR than with traditional methods.

included, the subject could solve the task by simply counting the visible cubes.

Visualization in augmented reality Augmented reality is not isolated in a virtual world, and offers a link with the real world. As we have seen in the implemented system, users can see the relation between the real world and how the robot sees it, because the two information sources overlap. In contrast to virtual reality, the system does not offer depth perception, because the augmented reality is presented on a 2D screen, in virtual reality. As covered in the background chapter, there are other AR platforms that can give augmentation in 3D, such as Microsoft's HoloLens. HTC Vive is not a platform for augmented reality, but it worked as a prototype.

6.3 Augmented reality as visualization tool

The augmented reality system was tested on users to investigate how it performed compared to traditional methods. The user study tested the understanding of the robot's sensor data. As seen in the previous chapter, the results were very clear, there was no doubt that augmented reality was superior to traditional methods, it was much faster, and it produced ten times fewer errors. It is however important to remember that the experiment does not cover all types of understanding, as only the robot's point cloud, relative to the robot and its environment was tested. In the implementation chapter, understanding was defined as follows:

> The ability to create a mental picture of the robot's situation, that is its position, state, and view of the world around it.

Perspective The results from the questionnaire support one of the main issues with the traditional methods for sensor visualization, part of the motivation for the thesis. A quote from one of the subjects, after being asked if he/she experienced difficulties with the traditional method, follows.

> "It was very difficult to get perspective, since I don't know where the tray is in the scene, and I don't know the distances from the cubes to the camera." (Translated).

This is spot on what the augmented reality system solves, giving perspective. In the AR system, the cubes, tray, robot, and its sensor are visible, and presented in the user's point of view.

What did the study test? The point clouds should not be interpreted as the raw sensor data, as they represent the robot's understanding of its surroundings. Autonomous robots can make decisions based on information gained from processing their raw sensor data. This is the type of information investigated in the study, by testing the users' ability to relate the robot's view to their own. However, the experiment can be said to be somewhat shallow, since it only tested a specific case, in a very limited environment.

Applying the results to a more general case Although we cannot prove the system is effective in a more general area than the one tested, it is reasonable to believe it will. We have seen the system's effectiveness in strengthening the link between human and robot. This link can be applied to a wide range of use cases, and a stronger link can increase the level of situation awareness in humans.

Chapter 7

Conclusion and future work

This chapter summarizes the findings from the thesis and arrives at a conclusion.

7.1 Conclusion

This thesis has presented an augmented reality system for visualization of robotic sensor data. The system was implemented on the virtual reality platform HTC Vive. The chosen sensor data was 3D point clouds from a depth camera, connected to a robot running ROS. The system can easily be applied to other sensors and robots. A user study was conducted and showed that the system had a significant positive effect on the users' understanding of the sensor data, compared to traditional methods. In light of the investigation conducted and the results from the study, we conclude that augmented reality can improve a user's understanding of a robot system.

7.2 Future work

The second goal of this thesis was to test the system's effectiveness in improving the user's understanding of sensor data. This section discusses what can be done in the future to further investigate this topic.

7.2.1 Applying the system to a real robot

As seen in figure 7.1, the robot prototype was attached to the live quadrupedal robot, *Dyret*[55]. As the image shows, the user of the augmented reality system is able to see the robot's perception of the human in front of it.

Visualization of internal state There are a lot of other interesting work that can be done to raise the observer's understanding. Information such as the robot's battery status, navigation goal, planned path, and control loop could be visualized to provide the observer with valuable picture of the robot's situation.

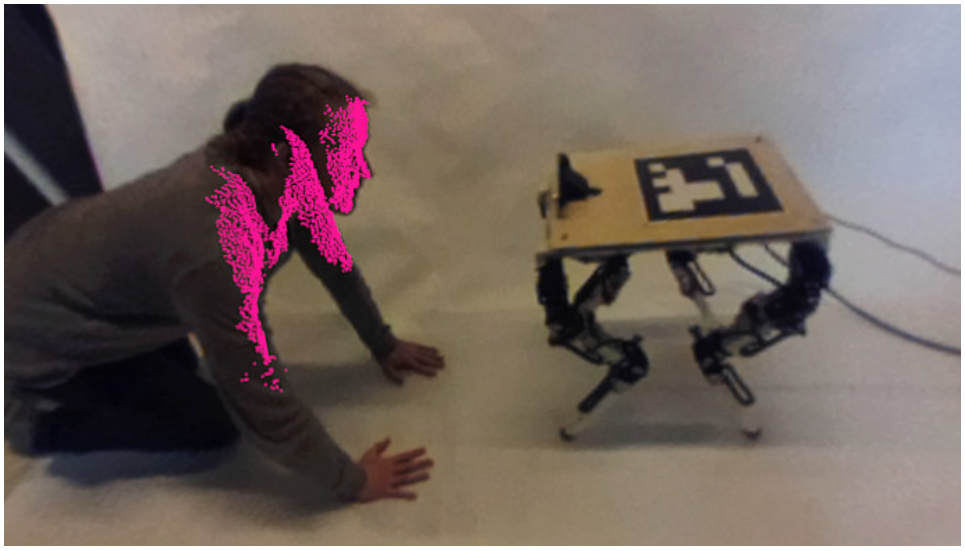


Figure 7.1: Human and robot, seen through the augmented reality system. The user of the system understands exactly what the robot sees. Note that the image is from a continuous video feed, and the user is free to move around and inspect the scene from other angles.

7.2.2 Visualization of decision

The robot's decisions could be visualized by showing the different factors that are used in the decision process in a combined fashion. This concept can be applied to different use cases.

Debugging The first case is in a debugging scenario, where a quadrupedal robot for example has failed to pass an obstacle, and fallen over. The researcher could go back to the time of the event, and see the robot's decision process, discovering that the robot classified the obstacle as passable, but did not change its gait, causing it to trip and fall. This type of visualization could be valuable in robot development, and can be investigated in the future.

Real-time supervision and control Another case is in real-time situations, where a robot engineer observes a robot solving a task. He or she can inspect how the robot processes the data from its sensors in combination with its internal state, learning how the factors and chosen parameters affect the decision making. For example, a robot navigating through an obstacle track could have its path planning visualized, by projecting the possible paths in front of it, and the following selection process.

In a teleoperation situation, the robot operator could get a clearer picture of the robot's situation, and thus be able to control it in a better fashion. Another example is a semi-autonomous robot that has encountered a situation where it needs assistance from a human to proceed. The different

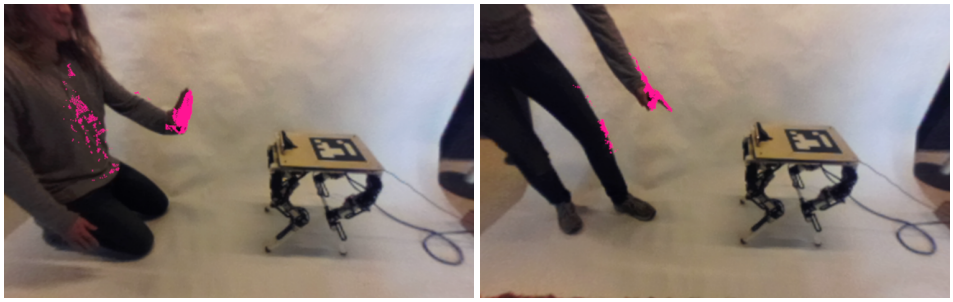


Figure 7.2: Human and robot interacting, seen through the augmented reality system.

options and possible trade offs, detected by the robot, could be visualized for the operator, giving a higher situation awareness, and thus making the his or her decisions simpler. At this point in time and near future, robots still need assistance from humans. Future investigation of this subject can thus lead to a better symbiosis between the robot and the operator.

Prediction Prediction is an important field in robotics, where the robot attempts to predict what will happen in the near future. This process can be complicated, in which multiple factors, like current sensor data, knowledge about the environment, and previous experience, are combined to make a prediction. The prediction model could be visualized to show different outcomes, and their perceived causes. Visualizing this allows an engineer to get a clearer picture of the process and thus create a better prediction model, making this a good candidate for future work.

7.2.3 Human-robot interaction

An example future use case for the system is in an Urban Search and Rescue situation, where a team of humans and robots work alongside to rescue victims trapped in the debris. The robots can enter small entrances and dangerous areas, unavailable for the rescuers, providing them with the ability to see through material, to observe the robots' locations and view, in the search for signs of life.

Figure 7.2 shows the interaction between the quadrupedal robot *Dyret*[55] and a human, controlling it. There is a lot of related work that can be done in the field of HRI. Since augmented reality can strengthen the link between humans and robots, it is reasonable to believe that applications where humans work alongside robots can see improvements in the coming time.

7.3 Future development

This section covers technical details in the current system that should be improved for achieving better results. Discussed topics are the efficiency,

stability, and the visual quality of the system.

7.3.1 Efficiency

Efficiency is an important topic in this project, because of the importance of stable frame rate in a mixed reality system. Below are some things that can be done to improve the system.

Triangles instead of 130.560 points The system draws all the points in the point cloud as a single point. This is unnecessary many to represent a surface, which is what the point cloud represents. By drawing shapes, like triangles, the number of required points could be decreased greatly, which would increase the system's efficiency.

Alignment of the point cloud in the image The most efficient way of making the point cloud align with the image is by adjusting the rotation and translation vectors received from the *ArUco* function *estimatePoseSingleMarkes*, so the transformation accounts for the offset (both rotation and translation) from the *ArUco* marker to the sensor.

In this project, for simplicity, another less efficient method is applied. The way this project achieves alignment is by transforming each point in the point cloud into the *ArUco* coordinate system. This is a computational heavy operation with a complexity of O(n), where n is the number of points in the cloud. In a typical scene there are about twenty to seventy thousand points, and this is calculated for each frame in the video feed.

7.3.2 Stability

As of today, the point cloud is only visible when the *ArUco* marker is detected in the frame. This means that if the marker gets partially covered by for instance a hand, the point cloud disappears. The marker used in the experiments is 20x20 centimeters. The thought behind a large marker was to make it easy to track in the low resolution and highly distorted images from the Vive's front facing camera. One of the most important drawbacks with a large marker is therefore the criteria to have clear line of sight to the entirety of the marker at all times. A possible solution is to use multiple, smaller markers[54]. This way, as long as at least one marker is visible, the AR headset's pose is known. The markers should also be placed in different angles around the sensor, to cover more viewing angles. In a robot use case, markers should be placed in such a way that there is always at least one visible marker. The downside of adding more markers, and especially on different planes, is that each marker's precise position and rotation, relative to the sensor, must be known.

On the other hand, when the marker completely leaves the frame, there are other techniques that can be used to still keep track of the position and orientation of the AR headset. *SLAM* (Simultaneous localization and mapping) is a computer vision algorithm that can keep track of the camera's movement between the frames in a video feed. Another option is to use an IMU (Inertial measurement unit). This is an external sensor that has to be mounted on the AR headset, and gives position updates based on its acceleration. A clear advantage over SLAM is the reduced need for processing power. However, SLAM is a computational heavy algorithm which might put too much load on a mobile system. Another option would be to use *Optical Flow*, as discussed in section 2.2.2. To improve the system's stability when it temporarily looses track of the marker, a Kalman filter can be applied to estimate the marker's position.

7.3.3 Visual quality

HTC Vive is a virtual reality platform, not designed for augmented reality. This resulted in a very low resolution AR video feed. Additionally, a better way to display the AR video feed in virtual reality would be on a curved screen, not a flat one, as suggested in the Unity tutorials[1]. Implementing the system on another platform, better suited for augmented reality, would likely have given much better results. Good candidate platforms are Microsoft's *HoloLens*[2], and Meta's *Meta 2 Developent Kit*[3].

Improving the accuracy in the point clouds Testing the augmented reality system showed that the point cloud sometimes did not properly line up with the real world, as discussed in section 4.1.2, under *Debugging the inaccurate point clouds*. An unanswered question from the implementation was whether using *OpenCV*'s fish-eye model, instead of the standard model, to undistort the image would resolve the problem, or not. This would probably not help, as the model is only better near the edges[4], and the deviations were not only present near the edges. Since the point clouds were first inspected in virtual reality, the inaccuracies were not discovered until the augmented reality system had been implemented. The issue was thus not resolved, since the deviations were small, and there was not budget for further investigation. This is thus a factor for improvement in the system.

X-ray vision Picture a natural work environment, i.e. a laboratory, where the robot engineer is observing the robot, wearing augmented reality glasses. Since the observer is free to move around, the sensor data (or other augmented information[5]) from the robot can get behind elements

[1] https://unity3d.com/learn/tutorials/topics/virtual-reality/user-interfaces-vr, accessed December, 2016

[2] https://www.microsoft.com/en-us/hololens

[3] https://www.metavision.com/

[4] http://docs.opencv.org/trunk/db/d58/group__calib3d__fisheye.html

[5] Augmented information can be battery status, navigation goals, state, and similar.

in the scene. The system's current configuration makes the augmented information always visible, making it work like *x-ray vision*. A question on whether this is desirable or not arises. This type of functionality can be valuable in some applications, for example in an Urban Search and Rescue situation, where the ability to look through material can be useful, as discussed in section 7.2.3. On the other hand, this information can very well be redundant and obscure objects of interest in the foreground.

No 3D, only 2D The inspiration to do augmented reality came from the HTC Vive's *room view*. This is a functionality that lets the user see through the camera on the front side of the goggles. The room view is monochrome, with very little detail, but, is more than good enough to navigate in the room, and untangle the headset's cable. The original plan was to use this functionality and draw the point clouds into it. Unfortunately, the *room view* feed was unavailable; the see through part of the augmented reality application therefore had to be created manually from the raw video feed from the camera. This video feed was, as explained in the implementation chapter, simply drawn one meter in front of user in virtual reality, creating a virtual, see through monitor. Ideally, a view similar to HTC's room view should have been implemented[6], to give full 3D experience, and thus using the virtual reality headset to its full extent. The increased complexity of this problem did not fit into the time available to solve all the challenges, included in this work.

[6]This could have been implemented by using stereo from motion. Comparing every frame with the previous frame, finding matching features, creating a disparity image, and thus a 3D model of the room.

Bibliography

[1] J. R. Echard Freund, "Projective virual reality: Bridging the gap between virtual reality and robotics," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, vol. 15(3), pp. 411–422, 1999.

[2] F. K. Paul Milgram, "A taxonomy of mixed reality visual displays," *IEICE Transactions on Information Systems*, vol. E77-D, 1994.

[3] P. G. de Barros and R. W. Lindeman, "Multi-sensory urban search-and-rescue robotics: Improving the operator's omni-directional perception," *Frontiers in Robotics and AI*, vol. 1, p. 14, 2014.

[4] K. Haluck, "Computers and virtual reality for surgical education in the 21st century," *Arch Surg*, vol. 135, pp. 786–792, 2000.

[5] F. S. Lupien, Maheu, "The effects of stress and stress hormones on human cognition: Implications for the field of brain and cognition," *Brain and Cognition*, vol. 65, pp. 209–237, 2007.

[6] Y. Sato, M. Nakamoto, Y. Tamaki, T. Sasama, I. Sakita, Y. Nakajima, M. Monden, and S. Tamura, "Image guidance of breast cancer surgery using 3-D ultrasound images and augmented reality visualization," *Medical Imaging, IEEE Transactions on*, vol. 17, no. 5, pp. 681–693, 1998.

[7] L. Lamport, "A survey of augmented reality technologies, applications and limitations," *The International Journal of Virtual Reality*, vol. 9(2), pp. 1–20, 2010.

[8] H. L. Pryor, I. Thomas A. Furness, and I. Erik Viirre, "The virtual retinal display: A new display technology using scanned laser light," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 42, no. 22, pp. 1570–1574, 1998.

[9] Y. Genc, M. Tuceryan, and N. Navab, "Practical solutions for calibration of optical see-through devices," in *Proceedings - International Symposium on Mixed and Augmented Reality, ISMAR 2002*, pp. 169–175, 2002.

[10] H. Kaufmann, K. Steinbugl, A. Dunser, and J. Gluck, "General training of spatial abilities by geometry education in augmented reality. [References]," *Annual Review of CyberTherapy and Telemedicine*, pp. 65–76, 2005.

[11] O. Bimber and R. Raskar, *Spatial Augmented Reality Merging Real and Virtual Worlds*, vol. 6. 2005.

[12] J. Underkoffler, B. Ullmer, and H. Ishii, "Emancipated pixels: real-world graphics in the luminous room," *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 385–392, 1999.

[13] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. W. Picard, and A. Pentland, "Augmented reality through wearable computing," *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 386–398, 1997.

[14] A. Acquisti, R. Gross, and F. Stutzman, "Faces of facebook: Privacy in the age of augmented reality," *BlackHat USA*, no. 2, pp. 1–20, 2011.

[15] "Aruco: a minimal library for augmented reality applications based on opencv," 2016. [Online; accessed 17-October-2016].

[16] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[17] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[18] A. Ryberg, A.-K. Christiansson, B. Lennartson, and K. Eriksson, *Camera modelling and calibration-with applications*. INTECH Open Access Publisher, 2008.

[19] P. Sicard and M. D. Levine, "An Approach to an Expert Robot Welding System," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, no. 2, pp. 204–222, 1988.

[20] S. Jorg, J. Langwald, J. Stelter, G. Hirzinger, and C. Natale, "Flexible robot-assembly using a multi-sensory approach," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 4, no. April, pp. 3687–3694, 2000.

[21] G. H. Ballantyne, "Robotic surgery, telerobotic surgery, telepresence, and telementoring: Review of early clinical results," 2002.

[22] J. Casper and R. R. Murphy, "Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 33, no. 3, pp. 367–385, 2003.

[23] A. Davids, "Urban search and rescue robots: From tragedy to technology," *IEEE Intelligent Systems and Their Applications*, vol. 17, no. 2, pp. 81–83, 2002.

[24] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, vol. 23. 2004.

[25] R. Chatila, S. Lacroix, T. Simeon, and M. Herrb, "Planetary exploration by a mobile robot: Mission teleprogramming and autonomous navigation," *Autonomous Robots*, vol. 2, no. 4, pp. 333–344, 1995.

[26] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. 2005.

[27] G. Michalos, P. Karagiannis, S. Makris, Ö. Tokçalar, and G. Chryssolouris, "Augmented Reality (AR) Applications for Supporting Human-robot Interactive Cooperation," in *Procedia CIRP*, vol. 41, pp. 370–375, 2016.

[28] S. Tadokoro, H. Kitano, T. Takahashi, I. Noda, H. Matsubara, a. Shinjoh, T. Koto, K. Takeuchi, T. Takahashi, F. Matsuno, M. Hatayama, J. Nobe, and S. Shimada, "The {RoboCup}-{Rescue} project: a robotic approach to the disaster mitigation problem," *{IEEE} {Conference} on robotics and automation ({ICRA})*, vol. 4, pp. 4089–4094, 2000.

[29] P. Milgram, S. Zhai, D. Drascic, and J. Grodski, "Applications of augmented reality for human-robot communication," *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, vol. 3, no. C, pp. 1467–1472, 1993.

[30] J. W. S. Chong, S. K. Ong, A. Y. C. Nee, and K. Youcef-Youmi, "Robot programming using augmented reality: An interactive method for planning collision-free paths," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 3, pp. 689–701, 2009.

[31] B. Giesler, T. Salb, P. Steinhaus, and R. Dillmann, "Using augmented reality to interact with an autonomous mobile platform," *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, pp. 1009–1014, 2004.

[32] F. Devernay, F. Mourgues, and E. Coste-Maniere, "Towards endoscopic augmented reality for robotically assisted minimally invasive cardiac surgery," in *International Workshop on Medical Imaging and Augmented Reality*, pp. 16–20, 2001.

[33] M. L. Andersen and H. F. Taylor, *Sociology: Understanding a Diverse Society, Updated*. Cengage Learning, 2007.

[34] L. Whitcomb, D. Yoerger, H. Singh, and J. Howland, "Advances in underwater robot vehicles for deep ocean exploration: Navigation, control, and survey operations," in *Navigation, Control and Survery Operations," in The Ninth International Symposium on Robotics Research*, Citeseer, 1999.

[35] M. A. Goodrich and A. C. Schultz, "Human-Robot Interaction: A Survey," *Foundations and Trends® in Human-Computer Interaction*, vol. 1, no. 3, pp. 203–275, 2007.

[36] M. R. Endsley, B. Bolté, and D. G. Jones, *Designing for Situation Awareness: An Approach to User-Centered Design*. 2003.

[37] D. Norman, *The design of everyday things*. New York: Basic Books, 1998.

[38] E. V. V. Rex Black, Dorothy Graham, *Foundations of Software testing*. 2012.

[39] J. Nielsen, *Usability Engineering*, vol. 44. 1993.

[40] Q. S. E. Q. R. Group, "Usability engineering." http://qse.ifs.tuwien.ac.at/courses/Usability/downloads_05/Usability_Engineering_20040920b.pdf, 2004. (Accessed on 04/10/2017).

[41] Y. Chen and H. Chi, "Use of tangible and augmented reality models in engineering graphics courses," *Journal of Professional Issues in Engineering Education and Practice*, vol. 137, no. 4, pp. 267–277, 2011.

[42] J. Lazar, J. H. J. Feng, and H. Hochheiser, *Research methods in human-computer interaction*. 2010.

[43] G. W. Oehlert, *A first course in design and analysis of experiments*. 2010.

[44] R. Rosenthal and R. L. Rosnow, *Essentials of behavioral research: Methods and data analysis*. McGraw-Hill Humanities Social, 2008.

[45] J. Rice, *Mathematical statistics and data analysis*. Nelson Education, 2006.

[46] C. G. Durbin, "How to come up with a good research question: framing the hypothesis," *Respiratory care*, vol. 49, no. 10, pp. 1195–1198, 2004.

[47] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Mg, "ROS: an open-source Robot Operating System," *ICRA*, vol. 3, p. 5, 2009.

[48] P. Saint-Andre, "RFC Standard 6455– The WebSocket Protocol," *RFC 6455 (Proposed Standard)*, pp. i –73, 2011.

[49] W. Goldstone, *Unity Game Development Essentials*, vol. 10. 2009.

[50] G. Bradski *et al.*, "The opencv library," *Doctor Dobbs Journal*, vol. 25, no. 11, pp. 120–126, 2000.

[51] T. Lee and T. Hollerer, "Handy ar: Markerless inspection of augmented reality objects using fingertip tracking," in *Wearable Computers, 2007 11th IEEE International Symposium on*, pp. 83–90, IEEE, 2007.

[52] C. Manresa, J. Varona, R. Mas, and F. J. Perales, "Hand Tracking and Gesture Recognition for Human-Computer Interaction," *Electronic Letters on Computer Vision and Image Analysis*, vol. 5, no. 3, pp. 96 – 104, 2005.

[53] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A Survey," *ACM Computing Surveys*, vol. 38, no. 4, pp. 13–es, 2006.

[54] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[55] T. F. Nygaard, J. Torresen, and K. Glette, "Multi-objective evolution of fast and stable gaits on a physical quadruped robotic platform,"

[56] "Presentation to ieee 802.3ab working group." http://www.ieee802.org/3/ab/public/feb98/ddmdix1.pdf. (Accessed on 04/07/2017).

[57] R. Durstenfeld, "Algorithm 235: random permutation," *Communications of the ACM*, vol. 7, no. 7, p. 420, 1964.

[58] B. D. Ripley, "Pattern Recognition and Neural Networks," *Analysis*, no. 1995, p. 403, 1996.

[59] D. F. Wallace, N. S. Anderson, and B. Shneiderman, "Time Stress Effects on Two Menu Selection Systems," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 31, no. 7, pp. 727–731, 1987.