NTNU
Norwegian University of
Science and Technology

# NTNU Cyborg - Teaching the Cyborg to recognize humans

Amund Froknestad

Master of Science in Cybernetics and Robotics
Submission date: June 2017
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

# Abstract

**Background -** Several types of depth cameras exist, utilizing different principles. The camera on the cyborg, a *Stereolabs ZED*, uses the embedded stereo method which is mechanically simple but computationally expensive. It is connected to a *NVidia Jetson TX1* development board, which works well but somewhat limits performance. There also exists several function on the cyborg which should be taken into account when creating new algorithms.
**Method -** Two separate methods of human recognition are presented, and some work is done in implementing the selected method. Some practical preparations were also necessary, as system requirements are found and relevant functionality of the Stereolabs Software Development Kit (SDK) is learned.
**Results -** The randomized decision forest method was deemed as currently not being usable, in spite of several advantages. The drawbacks are too noticeable, and so the region growing and Support Vector Machine (SVM) method is deemed preferable. While a basis for further work is made, implementation of the method is not finished.
**Conclusion -** This thesis presents interesting methods for human recognition, as well as an overview of practical tasks that needed to be completed to facilitate implementation of the chosen method. Both methods are interesting, but the region growing and SVM method is deemed preferable.

# Sammendrag

**Bakgrunn -** Det finnes flere typer dybdekamera, som bruker forskjellige prinsipp. Kameraet på kyborgen, et *Stereolabs ZED*, bruker innebygd stereo-metoden som er mekanisk svært enkel, men krever mye prosesseringskraft. Det er koblet til et *NVidia Jetson TX1* utviklerbrett, som fungerer bra men begrenser ytelsesevnen. Det finnes også flere funksjoner på kyborgen som bør tas hensyn til når nye algoritmer lages.
**Metode -** To separate metoder for menneskegjenkjenning presenteres, og noe arbeid gjøres innen implementering av den valgte metoden. Enkelte praktiske forberedelser var også nødvendige, som at systemkrav blir funnet og relevant funksjonalitet fra Stereolabs sitt SDK læres.
**Resultater -** Metoden som bruker tilfeldige avgjørelsesskoger ble vurdert til å ikke være brukbar slik systemet står i dag, til tross for flere fordeler. Ulempene er for merkbare, så metoden som bruker voksende regioner i kombinasjon med en SVM ble vurdert som bedre egnet. Selv om et grunnlag for videre arbeid ble laget, er implementasjon av metoden ikke ferdig.
**Konklusjon -** Oppgaven presenterer interessante metoder for menneskegjenkjenning, samt en oversikt over hvilke praktiske utfordringer som måtte løses for at det skulle være mulig å implementere den valgte metoden. Begge metodene er interessante, men metoden som kombinerer voksende regioner med en SVM blir foretrukket.

# *Acknowledgements*

I would like to thank Martinius Knudsen, the coordinator of the robot development activity on the cyborg project, and Sverre Hendseth, my principal professor, for helping where they were able, for guidance, and for periodic “butt-kicking” when necessary.

I would also like to thank Annette Stahl for her help with finding useful sources, and for help with my research.

Jørgen Waløen, my office partner, has been helpful both with research and staying sane.

My family and close friends deserve a great many thanks for mental support and tolerating highly infrequent visits and talks.

Last but not least, my partner in life, Saskia, for supporting me, helping me stay sane and tolerating long days when time was of the essence.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

**NTNU** Norges Teknisk-Naturvitenskapelige Universitet

**ROS** Robot Operating System

**SDK** Software Development Kit

**TOF** Time-Of-Flight

**fps** Frames per Second

**LED** Light-Emitting Diode

**GPU** Graphics Processing Unit

**CPU** Central Processing Unit

**SVM** Support Vector Machine

**HOD** Histogram of Oriented Depths

Chapter 1

# Introduction

The work done in this thesis is a part of an ongoing project known as "The NTNU Cyborg", run by Norges Teknisk-Naturvitenskapelige Universitet (NTNU). This is a project with the goal of developing a cybernetic organism, or **Cyborg**. In this case, the specific aim is to enable communication between living nerve tissue and a robot, as described at the NTNU page of the project [5]. To this end, the project has been divided into three main areas; Biological neural networks, communication with the aforementioned networks and robotics. The robotics part of this involves a robot base, with modifications made by the various students working on the project, that serves as the body of the cyborg. This is the part of the cyborg that people will be able to see and directly interact with, and the work in this thesis falls under the robotics domain.

In addition to the long-term main goal of building a real cyborg, the project has a sub-goal of increasing public awareness of biotechnology and information technology. This will be achieved by letting the cyborg drive around in different areas of the NTNU campus. While travelling around the cyborg will be able to interact with people and act as a "social" robot. To achieve this, two main functions are used: A function that allows the cyborg to recognize certain gestures, and a function that follows people around. The gesture recognition function is the crux of all interaction, as it is used both for things like games that people can play with the cyborg, but also for the function that allows the cyborg to select who to follow. The necessary data for these functions come from a depth camera, as having access to actual distances and suchlike in addition to simple images is massively useful. The depth images enable the cyborg to actually follow people, as well as helping with precision of gesture recognition.

The specific combination of hardware to make the depth camera work also seems to be popular among online robotics communities, as they are reasonably priced and Linux compatible. This means that any work done in this thesis and project may also benefit thousands of other robotics hobbyists.

## 1.1 Problem

As mentioned in the previous section, the cyborg already has code that uses data from depth images to let people interact with it. This is possible because the cyborg originally used a *Microsoft Kinect 2.0* that generated the necessary data. This camera, however, requires a Windows machine to work, and the robot base runs Linux. Therefore, it was connected to a laptop running a server, which processed the image data and sent them to the cyborg using a wireless network. This was both cumbersome and limiting, given the limited battery time of the laptop, so the camera was switched out. The problem that arose from this switch is that when the work on this thesis was started, the replacement camera was so new to the market that no software for human recognition using this camera existed. This means that completely new software has to be researched and made for the purposes of the cyborg project, as it would otherwise be completely non-interactive.

To reiterate, the problem to be solved is this: There exists a fair amount of functions on the cyborg that are currently not usable, due to the camera switch. The goal of this thesis is to attempt to find a method for human recognition, using the *Stereolabs ZED* depth camera, that will allow these functions to be used again. The focus is first and foremost on teaching the camera to recognize humans, and on communicating the position of recognized humans to the cyborg.

## 1.2 Motivation

The motivation for this thesis was to research and make the software necessary for the cyborg to again be able to recognize humans in its field of view. This software could then be built upon to both restore any missing functionality and improve existing designs.

## 1.3 Outline

To avoid "reinventing the wheel", a lot of time went into researching methods used by others for similar problems. This was to be expected, and so the research was prioritized first. This led to a large amount of time being spent on the method initially deemed most promising, the randomized forest algorithm. It was later found that there were several practical problems with implementing this method on the cyborg, and it was discarded in favour of a different approach. Although the randomized forest algorithm was, in the end, discarded, a thorough description is included. This is both due to the amount of time invested in it, and the fact that the method might still be interesting for future work.

In addition to the research into existing methods, some time was spent on the practical side of implementing this kind of algorithm. This included finding simple system requirements, such as any minimum Frames per Second (fps), some simple setup of the *NVidia Jetson TX1* development platform, and learning how the SDK that comes with the *Stereolabs ZED* camera works. All of this is described in chapter 3.

The randomized forest algorithm is described in chapter 4. The algorithm and planned implementation is explained, as well as a summary of why it was, eventually, deemed unfit for use.

The method that replaced the randomized forest algorithm is described in chapter 5. This includes explaining the principles, and how the finished code works.

Chapter 2

# Background

The aim for this chapter is to bring the reader up to speed on important concepts that affect the rest of this text. This will include an explanation of the general field of depth vision, what kind of depth vision the *Stereolabs ZED* camera uses, some specifications of the camera and *NVidia Jetson TX1* development platform, and a basic explanation of the existing functions of the cyborg that rely on depth vision.

## 2.1 Depth vision

Depth vision is, in this text, used to describe the field of work that uses some sort of camera or visual sensor to find the distance from the sensor to the objects in its field of view. As robots are used more and more in different areas, the importance of this field increases. In a completely controlled situation, for example a sealed room in a production line, vision might not be needed, but in a robot driving around in a hospital it is vital to safety. Thus, a considerable body of work has been built, with sensors and algorithms that fit a wide range of uses. Each of these have their own benefits and drawbacks, and the three most common methods will be explored later in this chapter. A shared trait of all of them, however, is that they at the very least return some sort of depth map. This depth map can then be decoded as needed, and used to find the distance of items from the camera. This depth map is what is used as a basis for any functionality the user wants.

### 2.1.1 Time of Flight

Cameras using the Time-Of-Flight (TOF) method of depth vision use a known, modulated source of light that is projected onto the work area, and observe the reflected light. The observed light is used to find the phase difference from the projected light, and this difference is used to calculate the distance from the camera. The modulated light usually comes from a laser or an Light-Emitting Diode (LED), and is outside of the wavelength that humans can see[7, 1].

This method has two main advantages, speed and compactness. The principle makes for computationally cheap equations, giving good quality depth images at a low cost to processing power. The compactness comes from the simplicity of the design: It only needs a source of light and a camera, and they can be placed close to each other without negative effects.

The drawbacks to this method are mainly due to the core principle, as any reason for phase difference other than distance will throw the calculations off. Reflective sources and background light are the main sources of this kind of noise, so the method works best in controlled environments.

### 2.1.2 Structured Light

The Structured Light method relies on sending out a pattern of light, usually not visible. This light is then registered by a camera, and used as a basis for depth calculation. The principle relies on the deformation of the projected light is used to calculate depth. The reflected pattern is compared to the known pattern that was projected, and depth is calculated based on the differences between the two[6].

The main advantages of this method are that it is fairly inexpensive computationally, and can provide fairly precise depth data. Thus, it is a fast and precise method, as well as providing normal images of reasonable quality if needed.

Its drawbacks are fairly similar to those of the TOF method, as reflective surfaces and changing background light can add distortion to the reflected image that is not accounted for. One additional flaw is that objects moving quickly may not yield accurate measurements, as the object may move between frames, making precise calculations more difficult.

### 2.1.3 Embedded Stereo

The embedded stereo method of depth vision is the one used by the camera on the cyborg, so it will be described in more detail than the others. This method is very simple in principle, as it is directly based on the way humans see. The Embedded Stereo method uses two cameras, mounted some known, horizontal distance from each other, and creates a map of the differences between the two images that are returned. The intensity of difference is then extracted from the map, and the distance from the camera setup calculated from it. Simply put, the closer an object is to the camera, the more intense the difference between the images will be. The specific equations needed will vary based on the specific cameras used, so that the software is the most important part of the product. The principle of this method also means that the resolution of the cameras used is much more important for precision than for the other methods mentioned. This is mentioned here rather than in the paragraphs on advantages or drawbacks as it can be both. On the one hand, it means that this method is easier to customize based on application than the others, as it is possible to prioritize whether precision or performance is more important in the specific case. On the other hand, this means that in systems where performance is important due to system constraints, precision will suffer. It also means that a reasonably powerful Graphics Processing Unit (GPU) is essential to both performance and precision, severely limiting what hardware can be used with the cameras.

This principle can actually be experienced in a simple experiment without any equipment; If you point at some object and close one eye at a time, the visual displacement from when both eyes are open is directly related to how far the object is from you. It should be quickly noted that if this experiment is tried there will only be a noticeable displacement when one user-specific eye is closed, but this is simply due to how our brains process images from our eyes, and does not affect the principle used by the camera. In addition to being the only method that can be demonstrated without a camera, the Embedded Stereo method differs from the other common methods in that it doesn't require its own source of light.

The single biggest advantage of this method as compared to the other common methods is the lower sensitivity to noise from background noise. As no light is emitted by the camera, background light contributes an almost negligible amount of noise, and reflective surfaces should not matter too much when it comes to difference between the cameras. An added, less important advantage is image quality. Given that image resolution greatly impacts precision, the cameras can be used to get high quality colour images for other images if this is necessary.

This method of depth vision has two major drawbacks, both entirely due to the underlying principle used. The first one is the simple fact that while this method is mechanically extremely simple, only requiring two cameras, it is computationally expensive. As previously eluded to, this means that a fairly powerful GPU is necessary. This has led to a paradox wherein depth sensors using this method are some of the cheapest on the market, making them popular in student projects and other low-budget projects, while also requiring more expensive computers to be connected to. This, in turn, has led to many projects devoting precious time to trying to reduce the computational cost rather than the actual focus of the project. In addition to this drawback, the measurement error is a quadratic function of distance. This means that while sensors using this method are theoretically only limited by image resolution in their measurable distance, in practice the error will be so large that any measurements are useless after a given distance. The specifics of each sensor, such as cameras and algorithm, decide the growth factor of the error. This means that certain sensors will have far greater usable range than others, but after a while the problem will always be there. As can be seen in the descriptions of some sensors using this method [17, 23], the usable distance tends to be quite high, but the drawback is still notable, as it also means that even while in the usable range, precision will always diminish when objects get further away.

## 2.2 Stereolabs ZED

The *Stereolabs ZED* depth camera is the camera used on the cyborg. As briefly mentioned in the previous section, if uses the Embedded Stereo method for generating depth images. As can be seen in the table of specifications, table 2.1, the camera in itself can deliver depth images of very high resolution, with a theoretical usable range of 0.7-20 meters. The benchmark figures for performance, however, will not be attainable in this project. This is due to the hardware that the camera is connected to, a *NVidia Jetson TX1* development board. While the combination of camera and development board is popular enough that the camera came with software specifically made for the development board, it is simply

not powerful enough for the kind of performance that the benchmark references. Another curious feature of this camera can be seen in the available resolutions listed in the table; while the standard for the so-called VGA quality of images is 640 by 480 pixels, the lowest quality available here is 672x376. This is referred to as VGA quality in all software and code provided by the manufacturer, although it does not match up to the standard size. The unusually large width at this resolution is due to a combination of the wide field of view provided by the camera, and the need for precision. Since depth is calculated based on the difference between the left and right pictures, extra width in the image is needed at such low resolutions to compensate. At higher resolutions the extra width isn't needed, and so the images are of standard proportions. This unusual scaling of proportions means that some extra work is needed if one tries to convert software normally used for other cameras, but this is hardly an insurmountable obstacle.

The SDK has certain system requirements that need to be met, as seen in table 2.2. The compute capability that is mentioned in this table is a figure used by NVidia that denotes the general specifications and available features of a NVidia GPU [11][13]. The compute capability of a given NVidia GPU can be found on their site, by searching for the relevant GPU.

| Stereolabs ZED | |
|---|---|
| Resolution | 672x376 - 2208x1242 (adjustable) |
| Image frequency | 672x376: 100 fps |
| (Benchmark) | 1280x720: 60 fps |
| | 1920x1080: 30 fps |
| | 2208x1242: 15 fps |
| Field of View | $110°H, 54°V$ |
| Range | 0.7-20 meters |
| Power usage | 1.9W |
| Connection | USB 3.0 |
| Platform | Dual-core 2.3 Ghz or faster |
| OS | Windows 7, 8, 10 |
| | Linux |

**Table 2.1:** Specification of Stereolabs ZED depth camera [23]

| | |
|---|---|
| Processor | Dual-core 2.3Ghz or faster |
| RAM | 4 GB or more |
| GPU | NVidia GPU, compute capability > 2.0 |

**Table 2.2:** SDK system requirements [23]

## 2.3 NVidia Jetson TX1

The *NVidia Jetson TX1* development platform is a development board from NVidia, sporting a powerful GPU and Central Processing Unit (CPU), in a compact package. As can be seen in the table below, table 2.3, it surpasses all system requirements posed by the SDK that comes with the *Stereolabs ZED* camera. This should come as no surprise, as Stereolabs published a specific SDK for this development board and its older sibling, the *TK1* [22]. The development board is mounted in a case on the cyborg, as can be seen in figure 8.1 in the appendix. The development board runs Linux, and is able to communicate with the robot base using either a wireless connection or an ethernet cable.

| NVidia Jetson TX1 | |
|---|---|
| GPU | NVIDIA Maxwell GPU, 256 NVIDIA CUDA Cores<br>Computational capability 5.3 [12] |
| CPU | Quad-core ARM Cortex-A57 MPCore Processor<br>Max frequency per core is 1.91 GHz [9] |
| RAM | 4 GB |
| Connections | USB 3.0, USB 2.0, HDMI and others |

**Table 2.3:** Relevant specifications of NVidia Jetson TX1 [14]

## 2.4 Existing functions

This section will be dedicated to a short summary of the important parts of the old functions on the cyborg that were made with the old *Microsoft Kinect 2.0* in mind. These functions were made to ensure the interactivity of the cyborg, but as the depth camera has been changed, they are currently unavailable. Given that it is desirable that these functions are restored to working order, their general construction should be taken into account when creating new code.

The most basic function is vital to all other functions that try to read gestures, as it is the labeling function [19]. When in use, this code is a part of the start-up cycle of the cyborg. The function sorts through the list of body part locations sent by the camera, and labels the ones that are useful by name. While this function greatly improves code quality and the simplicity of making new code, it also needs to be completely reconstructed to suit the actual list of body parts that is being sent.

The other functions are a set of gesture-testers[20]. These are simple functions that simply check the available data for predefined gestures. They accomplish this by checking the relative positions of a useful set of body parts against each other, where the body parts checked in each gesture has to be defined by the programmer. The list that is referenced in [20] defines several gestures that are then used by the cyborg. In addition to providing some gestures that are already made, the list can be used as a guideline for "construction" of a gesture when new ones are made in the future.

Chapter 3

# Practical preparation

This chapter will be dedicated to the practical work that needed to be done, no matter which algorithms were to be used later.

## 3.1 Requirements and simple setup

This section will be a short summary of the simple setup work done at the start of the practical part of the project, as well as a short summary of the requirements imposed upon further work with the camera.

The system setup work mostly consisted of simple jobs that were nevertheless required to be able to get any work done on the *NVidia Jetson TX1* development board. Since the development board is mounted on the robot base but, unlike the other external components mounted on the base, not powered by the base, power was supplied by a normal electrical outlet. This is only mentioned because it meant that the cyborg was now stationary. Since the depth camera requires both electrical and processing power from the development board to run, it could not be used unless the board was connected to power. This, in turn, meant that not only can the effect of movement on the camera not be tested, but the locations used for testing was also extremely limited. This is a temporary problem, as the development board will be powered by the robot base in the future, but it severely limited the possibility of testing and generating data sets in the current conditions. Smaller jobs, like finding a screen, mouse and keyboard, and setting the language of the keyboard to match the physical one, were also done, but are simple enough to warrant no further explanation.

The least flexible requirement for any function utilizing the depth images is in this case the fps. As can be seen in previous work done on the cyborg [25], motion tracking becomes too imprecise to be of use when receiving less than 10 fps. In general, the higher the amount of fps, the better all forms of motion tracking work, but 10 fps is the lowest acceptable limit. In tests it was found that when running the software provided by Stereolabs, it is best to use the lowest available image resolution (672x376 pixels). On the *NVidia Jetson TX1* development board, this resulted in a steady stream of images at around

at least 35 fps, while the next step up (1280x720) only managed around 15 fps. While both of these figures are satisfactory, they are achieved when very little additional computation is done. Given the computational costs that come with any kind of image classification problem, it is extremely unlikely that it is possible to achieve an acceptable, stable level of fps in any higher resolution than the lowest. This has the unfortunate effect of limiting the attainable precision of the depth calculations, as well as lowering the maximum usable range of the camera. Very little can be done to circumvent this drawback, however, so in all future works done in this thesis the resolution was kept at 672x376 pixels.

The other requirement is more of a guideline than a strict requirement, as it is derived from the knowledge of old functions described in section 2.4. Based on the structure of these functions, and the assumption that it is desirable to restore their usability, it can be seen that any function that identifies humans should have an end goal of returning a list of useful limbs. This requirement does not greatly affect the work done in this paper, as the main goal was always to actually recognize humans before worrying about specific body details. The requirement is still deemed worthy of being mentioned, as all research performed was done with this requirement as a consideration.

## 3.2 Working with the Stereolabs SDK

This section will be dedicated to summarizing the features and peculiarities of the Stereolabs SDK that need to be understood to do this kind of work. Some of the work done here is based on sample code provided by Stereolabs, all of which is written in C++. As a result of this, all work done in this section is also done in C++.

### 3.2.1 Grabbing depth data

Given that all work done in this paper is based on the depth images provided by the *Stereolabs ZED* depth camera, it is essential to be able to actually grab this data in real-time. This requires finding the format used by the data stream, selecting the correct part, and storing as necessary. The following is based on a version of available sample code, modified to suit the needs of the cyborg.

Since the desired result is grabbing data from a stream constantly and in real time, a buffer is needed. A struct is constructed, consisting of pointers of the expected data types and corresponding mutexes. In addition to the pointers and mutexes, integers denoting the width and height of the picture, as well as the number of channels that image data can arrive in, are in the struct. The reason pointers are used is that a data stream consisting of matrices is expected. Given that C++ is used, any function attempting to return an array or a matrix will return a pointer to the initial position of the array or matrix. The mutexes are used to avoid data corruption, and the integers will be used in matrix construction later.

Next, a grabbing function is defined. This function is responsible for the actual act of grabbing the data. In this function, pointers of the required data types are defined, and built-in functions from Stereolabs are used. The function enters a while loop that waits for a stop signal. For every time step while no stop signal is received, data is retrieved. For depth data, a pointer of type float is returned when the relevant function is called. This

data is copied into the buffer using the standard C++ function memcpy, and the loop starts over.

Lastly, the actual main function is defined. The desired video quality is chosen, and corresponding width and height of the image is found. This is also where the buffer is created. Storage space for the buffer is allocated, and the integer values are set as equal to the previously retrieved width and height. Memory is allocated in the appropriate size for the depth image, also based on retrieved width and height. After this is done, a suitable matrix is created. This is done through use of OpenCV[15], an open source library of computer vision. This is partially done to ensure that received data is saved in the correct structure, and partially because OpenCV also features tools that make it very simple to display the grabbed image. This simplifies the task of testing to see if the depth grabbing is being performed correctly. After the OpenCV matrix is created, the grabbing function is run by a grabbing thread, allowing it to run in parallel with the main function. A while loop is entered, where for every time step, buffer data is copied to the OpenCV matrix. For testing purposes, the data is then displayed. The OpenCV matrix is now being constantly updated, and the value of any given pixel can be found. Thus, a grabbing thread that returns a constantly updated matrix containing the depth data is complete.

### 3.2.2 Precision of depth calculations

As mentioned in section 3.1, precision has had to suffer for the sake of performance due to the available hardware. Given that future work depends on the measured distance, a rough idea of how precise the measurements can be expected to be is necessary. Given that the cyborg could not be moved, a test setup was made in the cyborg office, as can be seen in figure 3.1 and figure 8.2 in the appendix. In this experiment, chairs were placed in front of the cyborg, in distances measured with a standard tape measure. Using the ZED depth viewer that comes with the camera, the computed distance of some pixel on the chair was retrieved, and compared to the measured distance. In addition to finding the depth error this way, another test was performed at the same time. This test was to ascertain the relation between pixel value in the depth image that was grabbed, and distance. The experiment gave two interesting results: Firstly, that pixel value in the grabbed image was the calculated absolute distance from the camera, and secondly, that the discrepancy between calculated and measured length was far from stable.

Firstly, the results of the error testing will be discussed. In general, the calculated distance tended to be slightly larger than measured distance. As expected, the further away an object was from the camera, the larger the error would be. At distances of about a meter, the error tended to be approximately 5 centimeters, at three meters it averaged about 20 centimeters, and at 6.5 meters it was around 40. These values were found when using the lowest resolution, and precision improved markedly when using higher resolution. While this means that objects far away have a reasonably significant error, this is still not a considerable problem to the NTNU Cyborg. When moving, distance to a target will most likely be constantly updated, so that when the cyborg is close enough for the error to be really significant, the error has been reduced enough that it shouldn't matter. What is more important to this project, on the other hand, is variance. Even when both the cyborg and target object is standing still, in unchanging light, the calculated distance can vary a fair amount. This variance follows the same trend as the error, as it grows with distance. The

biggest problem, however, is the inconsistency. Even at a distance as low as a meter, the computed distance can vary by as much as ten centimeters from one frame to the next. The error variance does not grow as quickly as pure error, as even at 6.5 meters it was rarely more than 20 centimeters. The effect of the variance in this project is thankfully mainly restricted to adding a need for a certain amount of filtering. This filtering mostly manifests in a need to use some discretion when creating functions, so that they will work even if depth values vary more than would normally be reasonable.

The error variance mentioned in the previous paragraph also affects the testing of pixel value. Especially at closer distances from the camera, the difference between the actual calculated depth and the calculated absolute distance can often be less than the error variance. This means that there is very little reason to use unnecessary computing power on computing actual depth instead of absolute distance, so absolute distance is used as depth in the project. The error caused by this simplification becomes more of a problem when the target object is farther from the camera, but in the testing environment the error was negligible up to a distance of about four meters. After this point, the error could be accounted for by a simple subtraction, so in all experiments done in the process of writing this paper is done with this simple method.


**Figure 3.1:** The depth experiment, as seen from the cyborg

### 3.2.3 Horizontal distance

In addition to finding the depth of target objects, the ability to find the horizontal position of the object is necessary for future work. When searching through the SDK provided by

Stereolabs I was not able to find any functionality that could return this position in any other format than pixels, so an experiment was necessary. The experiment consisted of positioning several objects at the same depth, with a known horizontal distance between them. This gave three points with known distance in meters. The distance in pixels was recorded, and the amount of pixels per horizontal meter at the given depth was calculated. This was checked with another control point, and recorded in table 3.1. The data in this table was then used to estimate a function that could calculate a denominator for a given distance from the object to the camera, and the pixel position in the horizontal plane could then be divided by this denominator to return a reasonable estimate of horizontal position.

| Distance from camera (meters) | Pixels per meter |
|---|---|
| 0.6 | 490 |
| 0.8 | 440 |
| 1.0 | 375 |
| 1.2 | 303 |
| 1.4 | 245 |
| 1.6 | 218 |
| 1.8 | 188 |
| 2.0 | 175 |
| 2.2 | 155 |
| 2.4 | 142 |
| 2.6 | 127 |
| 2.8 | 115 |
| 3.0 | 110 |
| 3.4 | 96 |
| 4.0 | 85 |
| 5.0 | 75 |
| 6.0 | 55 |

**Table 3.1:** Relation between distance from camera and pixels/meter

After some experimentation, this function was the result:
If object is closer than 1.3 meters:
$Denominator = 682 - 320 * depth$
If object is further away than 1.3 meters:
$Denominator = 1/(0.00025 + 0.003 * depth)$
This function results in graph 3.2, with error graph 8.3 which can be found in the appendix. Given what is known about the error from the camera, this is deemed acceptable.

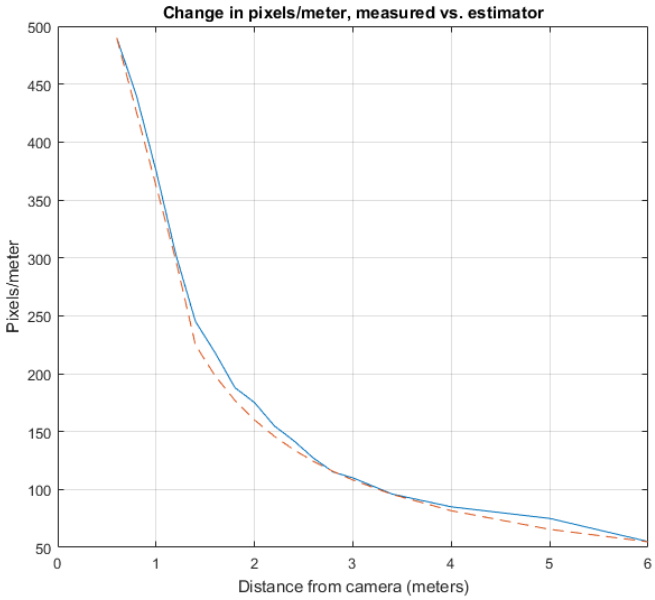


**Figure 3.2:** Change in pixels/meter, measured vs. estimator. Estimator is dashed line

Chapter 4

# Random forest method

In this chapter an interesting approach to recognizing humans in depth images will be presented. The method seen here is heavily based on existing papers [18] [10] that outline solutions to similar problems. This method is particularly interesting as it not only recognizes humans, but also identifies and labels body parts. For systems such as the NTNU Cyborg, this is extremely useful if it works.

## 4.1 Preparation for the classifier

Some work needs to be done before the depth images can be fed into the random forest classifier. This includes simple things, like some background subtraction, and more complicated procedures.

Background subtraction would here be used to try to reduce both the data that needs to be analyzed, and the chance of false positives. In areas with varied backgrounds, such as the halls where the cyborg will be expected to operate, there is a risk of false positives due to clutter. This refers to the fact that sufficiently cluttered backgrounds may add up to something that an algorithm could mistake for a human, as seen in previous work with the cyborg[25]. The simplest method of achieving this would be to ignore any part of the picture that is further from the cyborg than some set distance. In the case of the cyborg this should not pose too great a risk of information loss, as it is reasonable to assume that any human that wants to contact the cyborg will not be very far away from it.

As is explained in the paper by Shotton et al.[18], another vital part of the proposed method is body part recognition. This means that to achieve any precision in recognizing body parts, an object recognition algorithm has to be trained. This would be done by creating a 3D surface model of a generic human, dividing it into some number of body parts, and training the algorithm on these 3D parts. The amount of body parts chosen would depend on the application, as a greater number of parts would, up to a limit, mean better precision at the cost of performance. The inferred body parts could then be reprojected into world space, and a proposal for skeletal position could be found. A random forest

algorithm would then be trained to use this object recognition algorithm to evaluate each pixel in a depth image, and a probable human body would be returned.

## 4.2 Randomized decision forest

Random decision forests, also known as simply random forests, are a machine learning method for classification and similar tasks. As the focus of this paper is the classification aspect, this section will focus on only the relevant parts of the method. They operate by constructing several decision trees, the number of which is defined by the programmer, based on extensive sets of training data. They then return the mode of the classes of the individual trees, mode here meaning the value that appears most often in a set of data. One of the main advantages of creating random decision forests rather than simple decision trees is overfitting, or the tendency of decision trees to describe random error or noise rather than the useful data. This is corrected for by random forests, leading to potentially much better precision.

### 4.2.1 Decision trees

Decision trees are, in short, simple representations for classifying examples. They are used extensively in fields such as statistics and machine learning, because they are quite fast while being reasonably simple to construct. In the case of classification trees, each leaf (the end-points of the tree, as seen in figure 4.1) represents a class label, and each branch (the paths leading from node to node) represents the features that the classes following a branch have in common. For example, if the values 7 and 15 are sent through the decision tree in figure 4.1, their first branch is shared, but not the second one. Classification trees also, by necessity, have a finite set of leaves, since any value sent through the tree will be classified as belonging to a known class.

Decision trees can be handmade, as in the example figure, but they can also be generated by known datasets. To achieve this, the most common strategy is the top-down induction of decision trees [16]. This process entails splitting the dataset into subsets based on a test of attribute values, and repeating this split on each subsequent subset. This recursive partitioning continues until the splitting no longer improves the predictions, or the subset of a node has the same value as the target. These decision trees can then be used to classify new data.

As mentioned, this method is quite extensively used, because of their simplicity, flexibility and speed. However, as also mentioned, they have issues. Decision trees tend to be less accurate than other approaches, and overfitting is a problem. In cases like body part classification, this means that while the core principle is very useful, it needs modification.

### 4.2.2 Bootstrap aggregating to random forests

To compensate for the known problems with decision trees, a technique known as bootstrap aggregating, or bagging, is utilized. This method takes a training set with appropriate responses, repeatedly selects a random sample with replacement of the training set, and fits trees to the samples. After the required amount of trees is sampled, predictions for new

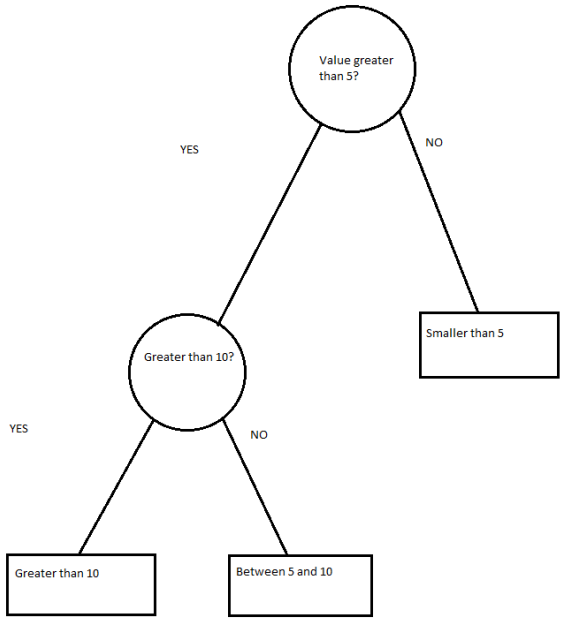


**Figure 4.1:** A simple decision tree sorting a value into one of three categories

data are made assigning one vote to each tree. The votes are tallied, and the class with the majority of the votes is assigned as the true class. In short, instead of creating one tree based on all data in a data set, multiple trees each based on parts of the data set are created and used in conjunction.

To create random forests, the bagging technique is modified slightly. When the decision trees are trained, a random subset of the features are selected instead of trying to train for all of them. Basically, we introduce another layer of randomness to try avoid certain features being prioritized too much. In ordinary bootstrap samples, strong predictors will tend to be chosen often, so other predictors may be ignored. Thus, a set of decision trees is created, instead of one, and the subsets are chosen in a random fashion, to avoid trees that focus on certain features more than they should. This means that the common problem of overfitting is avoided, which in turn means that much larger data sets can be used for training. This means that the algorithm is far more precise than normal decision trees, while being similarly fast.

### 4.2.3 Some available tools

The popularity of randomized decision forests has led to several tools becoming available. The most commonly known are the Scikit package for Python, and the 'random forest' package in R. Both of these supply flexible functions that greatly simplify the creation of randomized forests. More generalized machine learning software like Caffe[2] and TensorFlow[24] are also freely available, and can be utilized for the same purpose.

## 4.3 Incompatibility with *Stereolabs ZED*

As can be seen in the previous sections and the paper by Shotton et al[18], this method seems to be a fast and fairly accurate method of human recognition. However, there are some issues when attempting to use it with the *Stereolabs ZED*. The main problem is the amount of training data required for good precision. As seen in [18], several hundred thousand images of reasonable variance are needed. In the mentioned paper this is solved by a combination of synthetic data generation and the large sets of motion capture images available for academic use[4]. Synthetic data generation is outside the scope of this paper, and sorting through and labelling images from the publicly available data sets would be extremely time consuming, though possible.

In addition to this, there are scaling problems. There are the minor problems, like the fact that pixel values in the data sets is on a completely different scale to the pixel values provided by the camera. A bigger scaling problem is image size; Any algorithm trained on the available data sets will be expecting images to be 640x480 pixels, when they will be 672x376.

Another factor is the expected error. In general, the available datasets will have been produced with a version of the *Microsoft Kinect*. In the operational range, this will mean smaller errors on average, meaning that coefficients that are a result of training on the datasets might not work with the data from the *Stereolabs ZED* camera.

There is also the factor of training time needed. Randomized trees seems to require fairly powerful computers and long hours to be trained enough to be precise. The kind of computing power needed for this is not available on the cyborg, so a computer for training would have to be localized for it to be possible.

The last big drawback is the fact that a complete 3D model of a human would have to be generated, and a completely separate object recognition algorithm trained on it. While all of these things are certainly possible, they are well outside the scope of this thesis.

In summary, there are several problems of varying severity. These problems add up to enough work and uncertainty that the method has to be scrapped in this case.

Chapter 5

# Region growing and Support Vector Machine method

This chapter describes an fairly adaptable approach to human recognition in depth images. It is based on a similar method as proposed by Choi et al.[3], but modified to fit the cyborg. As is explained in the mentioned paper, this algorithm requires roughly two and a half thousand training images, and is quite fast. Their implementation averaged approximately 33ms per processed frame on a single core of an Intel Core i5 processor, which equates to about 30 fps. Given the performance information provided, it is reasonable to assume that the *NVidia Jetson TX1* would be able to perform similarly well. This means that even when factoring in the actual image grabbing, it would be expected that the minimum requirement of 10 fps should be met. As was also mentioned, this method requires far fewer training images than the randomized forest method. This means that it is possible to at least create a basic training set with the cyborg as is, so that training is possible. It should be noted that any data set created in such a manner will lack some of the variety that is to be preferred, and as of such is not advisable in the long run. It would, however, be good enough for testing.

## 5.1 Image Segmentation

This step is where a set regions in the image is obtained. The first thing that is done to achieve this is to subsample the image. Subsampling is done both to filter out some noise, and to cut down on computational cost. The subsampling is done by dividing the picture into smaller n*n regions, and selecting $\alpha$ random points within each region. For an image of height h and width w, this results in a new matrix of height $h/n$ and width $w/n$. To be able to store necessary information throughout the algorithm, a matrix PV is created to store the depth and horizontal position of a given point, and another matrix PP is created to store the pixel coordinates in the original picture.

The $\alpha$ points are then sorted based on depth. This was achieved by creating a cus-

tom struct containing pixel coordinate and depth value, and sorting with a simple sorting function. The sorting function takes a list of the custom structs, extracts depth value, and returns a list of sorted structs. This means that it is possible to sort based on depth, while remembering the actual coordinate of the pixels being sorted.

The median value of the list is then selected. If the depth at the median is valid, horizontal position is also calculated, and the values are stored in matrix PV. The coordinates are stored in matrix PP, so that for any point in PP, depth value and horizontal position is at the same coordinates in PV.

The next step is to find connected regions. A simple region growing algorithm was selected for this, mainly due to the relative simplicity of implementation. The biggest drawback of region growing algorithms is their speed, but the heavy subsampling done on the image should go some way to rectifying this.

To start the region growing, two empty matrices R and V are created. R is the same size as the original image, and will be used to denote the regions when the algorithm is finished. Regions will be tracked by integer value, so that all pixels belonging to the first region are of value 1, the second region has value 2, and so on. V is of similar dimensions as the PP and PV matrices, and will be used to indicate whether a point in PP and PV has not yet been visited, has been visited but isn't a part of a region, or has been visited and added to a region.

The algorithm then goes through each point in PP, checking neighbours in the four principal directions. If a point has already been added to a region, it is skipped. Likewise, if a neighbour has already been visited by a point in the current region and deemed to not belong in the region, it will be skipped. If the neighbour has not been visited, difference between the current point and the neighbour is calculated, and if it is within limits, the neighbour is added to the region. This process is repeated until all points have been visited and assigned to a region. The matrix R is then a set of all the regions found in the image, and is returned by the function.

## 5.2 Region Filtering

This step takes the matrix R containing all the regions, discards or merges regions that are deemed unlikely to be humans, and returns a set of candidates that are likely to be distinct objects.

The first thing done in order to achieve this is calculating mean depth, height and width in meters and center point of each region. Mean depth is calculated by finding the mean depth of all points in the region. Height and width is calculated by finding points at the relevant edges, and subtracting the real values as necessary. The center point of the region is defined as the point that has the vertical position of the midpoint between top and bottom, horizontal position of the midpoint between left and right, and depth equal to mean depth.

Parametric heuristic rules specifying valid ranges for height, width and mean depth are found by measuring the ranges of positive examples in the test set. The regions are then checked against the rules, and any region that fails to satisfy them is deemed invalid.

The rule check is performed in order of size of region, so that regions that satisfy the heuristic rules, but are deemed too small, can be merged with a bigger region if it is

deemed sensible. If some larger region exists within reasonable real distance, and within reasonable difference in depth, the smaller region is added to the larger one. The mean position of the region is updated, with the new mean being the mean position of the merged regions, weighted based on size. Since the last step of the classification requires each region to be of fixed size, they are copied into an array called a candidate image. The regions are placed so that they are central in the candidate image, and the entire bounding box of the region fits into it. This means that the resulting candidate image is composed of only either depth pixels, if a pixel was copied from the region, or undefined pixels. The set of candidate images is then returned for use in the classification.

## 5.3 Classification

In this step, the set of candidate images is run through a classifier to determine which ones contain a human. The Histogram of Oriented Depths (HOD) descriptor described in Spinello et al.[21] is used to this end. The depth pixels in this case are the real depth, so the HOD descriptor can be calculated directly from the pixel values of the candidate images. As the name implies, the HOD descriptor is a Histogram of Oriented Depths, where the gradient between an undefined value and any other value is always zero. Signed gradients are used, given that the sign of gradients between metric depths matters. All objects in an image are always closer to the camera than their background, unless they are occluded. Simply put, a HOD descriptor uses subdivisions of an image, computing descriptors for each subdivision, and collects the oriented depth gradients into one-dimensional histograms. These histograms can then be used to train other classifiers, in this case a SVM.

The final classification step consists of using a SVM to classify the descriptors of each region to decide if a region contains a human or not. SVMs are classifiers that, if given labeled training data, outputs an optimal hyperplane which categorizes new examples. Given that OpenCV is already used in the project, the OpenCV SVM functions are good alternatives[8]. This library includes functions that enable both training and testing of SVMs, and so seems ideal for the project.

The classifier is then run on a training set of depth images, and the candidates saved and labeled as positive or negative. These candidates can also be used for deriving the heuristic rules mentioned in the region filtering section.

Chapter 6

# Discussion

This chapter will present and discuss the results of the work done in this thesis. In the case of this thesis, the results are mainly on the usability of the presented methods.

## 6.1 Randomized decision forest

As discussed in the chapter describing the random forest method, it is a promising method that nevertheless is deemed unfit for implementation on the cyborg. As discussed in section 4.3, there are several problems with the prospect of implementing this method on the cyborg. These include the difficulty of obtaining relevant training data, the time and processing power needed for training, and the need to create a usable 3D model of a human body and use it to train another, separate object recognition algorithm. While the speed and precision of the method make it very tempting, these issues add up to enough work and uncertainty of feasibility that the method is deemed currently not usable. This may change in the future, and if ways to overcome them are found, the method is very well suited to the task at hand.

## 6.2 Region growing and Support Vector Machine

This method is an attempt at taking a method that is known to work with the *Microsoft Kinect 2.0* depth camera, and modifying it to work on the cyborg. This method is fast and flexible, and requires far smaller traing sets than the random forest method. Thus, it seems ideal for the cyborg. Its greatest downside is that it does not identify separate body parts, but rather the whole person. While this seems to make the initial implementation of the algorithm simpler, it does mean that further work will be needed in the future. Due to time constraints, the implementation was not finished, leading to a lack of testing and optimization. While this is disappointing, the work that was done should help kickstart future attempts at implementation, as a basis exists.

Chapter 7

# Conclusion

This thesis presents two interesting methods for human recognition in depth images, as well as an overview of practical tasks that needed to be done for these methods to be implementable. The methods are presented as completely as possible without getting too detailed, and feasibility of use on the cyborg is explored.

The conclusion is that it is possible to use both methods on the cyborg, but the method using region growth and a SVM is recommended. Very few issues were found with this methods, and while a complete implementation was not achieved, a good basis was created so that it should be possible in the future.

Chapter 8

# Future work

In this chapter, potential work to be completed in the future is summarized quickly.

## 8.1 Implementation of human recognition

While this thesis discussed two possible methods for implementation of human recognition, no implementation was completed. The most obvious next step would then be to use the basis that is created here, both in code and research, to complete the implementation of at least one of them. Given the work already done, continuing the implementation of the method based on a SVM would make the most sense. This work would also include testing and optimizing the chosen algorithm, as no such work has currently been done. This could feasibly lead to some reworking of the algorithm, which might be necessary to achieve the kind of performance that is needed.

## 8.2 Skeleton construction

In addition to finalizing the implementation of human recognition, some form of limb recognition is needed. This needs to be completed in order to restore the gesture recognition software to a usable state, and in turn make the cyborg interactive again. In the process of researching for this paper several papers on skeleton construction based on an image containing a person was found, so it is not a new field. It would still be a considerable amount of work, but it needs to be done at some point in the future.

## 8.3 ROS

As the camera and development board is today, no data is transferred from the *NVidia Jetson TX1* to the robot base. Given that a state machine using Robot Operating System (ROS) exists on the base, a ROS node should also be set up on the development board.

This would enable communication, so that the necessary depth data can be sent to the robot base. This should not be too hard, given that such nodes already exist on the robot base, as well as several tutorials for exactly this kind of situation existing. It would therefore be recommended that this work is done as an added feature by someone doing other work on the cyborg.

# Bibliography


[1] Jan Aue, Dirk Langer, and Bernhard Müller-Bessler. *Efficient segmentation of 3D LIDAR point clouds handling partial occlusion*. Apr. 2017. URL: http://tinyurl.com/hzkcdxn (visited on 04/13/2017).

[2] *Caffe\Deep Learning Framework*. June 2017. URL: http://caffe.berkeleyvision.org/ (visited on 06/05/2017).

[3] Benjamin Choi et al. "Fast Human Detection for Indoor Mobile Robots Using Depth Images". In: *Proceedings of IEEE International Conference on Robotics and Automation*. Pittsburgh, PA, Aug. 2013.

[4] *CMU Graphics Lab Motion Capture Database*. June 2017. URL: http://mocap.cs.cmu.edu/ (visited on 06/04/2017).

[5] NTNU Cyborg. *The NTNU Cyborg*. URL: https://www.ntnu.edu/cyborg/about (visited on 05/30/2017).

[6] Guido Gerig. *Structured Lighting*. Apr. 2017. URL: http://www.sci.utah.edu/~gerig/CS6320-S2012/Materials/CS6320-CV-S2012-StructuredLight.pdf (visited on 04/14/2017).

[7] Larry Li (Texas Instruments). *Time-of-Flight Camera – An Introduction*. Apr. 2017. URL: http://www.ti.com/lit/wp/sloa190b/sloa190b.pdf (visited on 04/13/2017).

[8] *Introduction to Support Vector Machines*. June 2017. URL: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html (visited on 06/05/2017).

[9] Michael Larabel. *Everything You Need To Know About The NVIDIA Jetson TX1 Performance*. June 2017. URL: http://www.phoronix.com/scan.php?page=article&item=nvidia-jtx1-perf&num=1 (visited on 06/03/2017).

[10] Vincent Lepetit, Pascal Lagger, and Pascal Fua. "Randomized Trees for Real-Time Keypoint Recognition". In: *CVPR '05 Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02* (June 2005).

[11] NVidia. *CUDA FAQ*. June 2017. URL: https://developer.nvidia.com/cuda-faq (visited on 06/03/2017).

[12] NVidia. *CUDA GPUs*. June 2017. URL: https://developer.nvidia.com/cuda-gpus (visited on 06/03/2017).

[13] NVidia. *Programming Guide:: CUDA Toolkit Documentation*. June 2017. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities (visited on 06/03/2017).

[14] NVidia. *Unleash Your Potential with the Jetson TX1 Developer Kit*. June 2017. URL: https://developer.nvidia.com/embedded/buy/jetson-tx1-devkit (visited on 06/03/2017).

[15] *OpenCV Home page*. June 2017. URL: http://opencv.org/ (visited on 06/04/2017).

[16] J. R. Quinlan. "Induction of Decision Trees". In: *Mach. Learn.* 1.1 (Mar. 1986), pp. 81–106. ISSN: 0885-6125. DOI: 10.1023/A:1022643204877. URL: http://dx.doi.org/10.1023/A:1022643204877.

[17] Carnegie Robotics. *Carnegie Robotics Multisense S7*. June 2017. URL: http://carnegierobotics.com/multisense-s7/ (visited on 06/02/2017).

[18] Jamie Shotton et al. "Real-Time Human Pose Recognition in Parts from Single Depth Images". In: *Communications of the ACM* (Jan. 2013).

[19] *Source code for body part labeling*. June 2017. URL: https://github.com/thentnucyborg/OLD_cyborg_ros_k2_client/blob/master/src/startBody.cpp (visited on 06/03/2017).

[20] *Source code for gestures*. June 2017. URL: https://github.com/thentnucyborg/OLD_cyborg_ros_simon_says/blob/master/src/gestures.cpp (visited on 06/03/2017).

[21] Luciano Spinello and Kai O. Arras. "People Detection in RGB-D Data". In: *Proceedings of IROS 2011* (Sept. 2011), pp. 3838–3843.

[22] Stereolabs. *Developer Center*. June 2017. URL: https://www.stereolabs.com/developers/ (visited on 06/03/2017).

[23] Stereolabs. *Stereolabs ZED Specification*. June 2017. URL: https://www.stereolabs.com/zed/specs/ (visited on 06/03/2017).

[24] *Tensorflow*. June 2017. URL: https://www.tensorflow.org/ (visited on 06/05/2017).

[25] Tom Erik Tysse et al. "Følging og Interaksjon". In: *NTNU EiT* (May 2016).

# Appendix

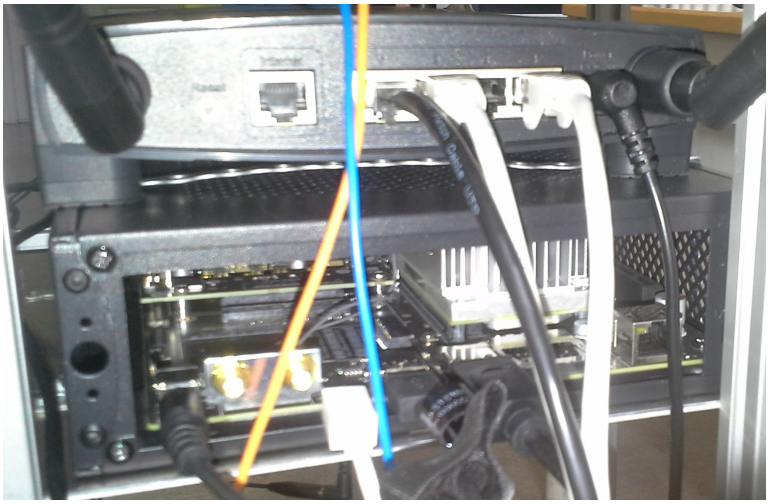

**Figure 8.1:** The Jetson TX1 development board, as mounted in the cyborg

**Figure 8.2:** The depth experiment setup, with the cyborg

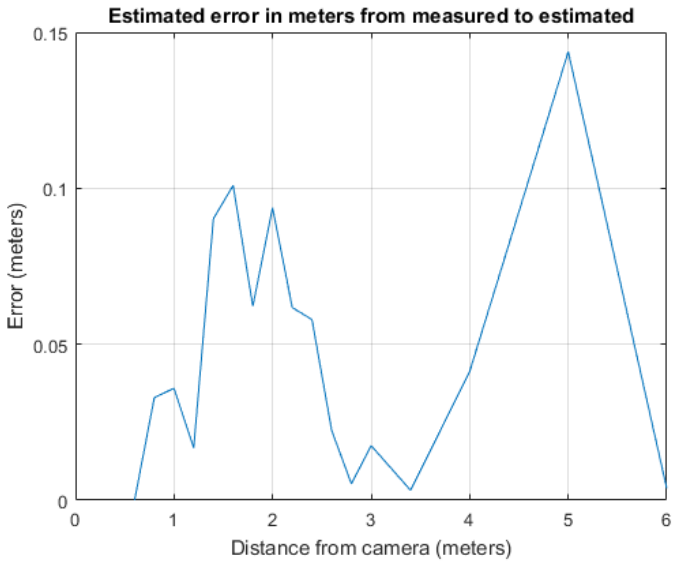


**Figure 8.3:** Error estimate in meters for the chosen function for horizontal position