NTNU
Kunnskap for en bedre verden

Department of Engineering Cybernetics

TTK4551 - Engineering Cybernetics, Specialization Project

Position Measurement for Quadcopter

Author:
Jonas Bjerke

December, 2021

Preface

This report describes the details of a specialization project written during the fall semester of 2021 for the Department of Engineering Cybernetics at Norwegian University of Science and Technology. Specializing in a project is mandatory for all students and is expected to be a preparation for the master thesis the following semester. I got the opportunity to contribute to an interesting broad project, that consists of autonomous robots such as a quadcopter, and even virtual mapping.

The learning process has been very valuable to me. I have had the opportunity to learn more about the powerful microcontroller nRF52840 DK, and how to control it using the API named Segger Embedded Studio. I have also become very familiar with the ultrasonic sensor HC-SR04. The process from start to finish have shown to be challenging. From knowing barely nothing at all, I have learned a lot which will come in handy when continuing the project next semester.

During the semester, some of the challenges I have encountered have been solved with the help of fellow students who were working on similar projects, or with the help of my supervisor, Tor Onshus. They have gotten me through both simple and more complex issues. A big thanks to him and the students.

Abstract

The aim of this study is to find a suitable positioning measurement tool for an indoor quadcopter equipped with cameras. This is required for the quadcopter to detect the distance to what it is photographing. This same procedure is also used for robots on the ground. They are both connected to a mutual server that combines the pictures into a virtual map of the area. This server automatically tells the robots where to go next such that the map becomes complete.

A necessity for the quadcopter when it comes to automatically map a room, is the position. Usually, a GPS is used, however, the issue is that this option is not suitable as the mapping should be able to take place inside or underground. Alternatively, one could use the IMU inside of the quadcopter. This is a device that ensures a steady flight. The issue is the accessibility of this data, as well as calculating the position relative to its origin, giving it a potential error. In this case, using sensors might be the best option, which is what the results are based upon.

For this project the most ideal sensor is a supersonic sensor that uses sound and its reflections to measure distance. It is cheap, efficient and is optimal for the required distance. However, there are certain scenarios that might occur where this type of sensor struggles to acquire measurements. Some examples are angled surfaces, where the reflected sound does not return to the sensor, or very soft surfaces, where the sound is dampened or absorbed, thus interpreting the surroundings as open space. These scenarios might be avoided by adding sensors pointing in more directions, or even adding a variety of sensors working together.

This report examines how ultrasonic sensors can be used to measure the position of a quadcopter, such that attached cameras can take pictures and send them to an external server that consistently knows how far away the pictures were taken, giving the opportunity to virtually map a room.

Summary

The main objective for this project was to find a way to measure position for a quadcopter. The quadcopter will be using this position when taking pictures of its surroundings. Together the pictures and position data will be transferred to a server where they will be combined, creating a virtual map of of the area. The accuracy of the position determines how accurate the virtual map becomes.

Early on in the project it was decided that a sensor had to be used for the task. The IMU was considered for the job, but is best used to determine the position relative to the itself. It is also more complicated to reach as it is already part of the quadcopters hardware. Choosing the type of sensor needed some comparisons. An ultrasonic sensor was decided to best fit the task of measuring distance. It has the required range and accuracy, can measure through particulate matter such as smoke and dust and is also very cheap. The particular sensor used was a HC-SR04.

A microcontroller was then used to to control the sensor. The controller is a nRF52 which supports communication over Bluetooth Low Energy. This is a requirement such that the pictures and position can be transferred to the server via this communication platform.

To get the sensor to work it needed instructions from the controller. This software was developed by an API named Segger Embedded Studio. The code that was written implemented a timer that was used to measure the time it took for the sound waves to return after being set in motion by the sensor. The code than proceeded to use this time to get the distance.

The sensor then needed to be tested to ensure it would give the correct data during flights. The distance provided by the sensor needed to match the actual location of the sensor. An advanced motion capture system called OptiTrack was used to collect the position of the sensor. The position measured from both systems could then be compared to further evaluate if an ultrasonic sensor could be the solution to the original problem. Some special scenarios were also designed, to test the general use of the sensor.

Conclusion

This report tested how an ultrasonic sensor would perform as a positioning estimate tool for a quadcopter. When comparing the distance data from the sensor to the actual distance, it proved to be fairly precise. The distance was not entirely correct at all times, but this could be due to several minor errors. The greatest potential error could be that the data is compared with a slight offset in time, giving the illusion that the sensor provides incorrect results. To ensure that this is not the case, new tests have to be conducted.

However, even if the sensor would be exact, there are cases where it fails to provide good results due to the use of sound waves. Very soft surfaces absorbs the sound, but such soft surfaces are rarely found in rooms or other areas.

The refresh rate of the sensor did also turn out to be an issue. It only logged a total of 3 measurements per second. This is not optimal even though it provided decent results.

All factors considered, using an ultrasonic sensor on the quadcopter would give a good position estimate. To strengthen this conclusion, more testing have to be done. These tests would need a faster refresh rate and identical overlap of the collected data from the sensor and motion capture system. Other sensor types should also be tested, such as sensors that uses light instead of sound.

I am eager to continue this work on my master thesis next semester.

Table of Contents

1 Introduction

Imagine sending an army of autonomous vehicles on the ground supported by flying drones into an unknown area. After a while the shape of the room begin to appear on the screen you are looking at. The floor and walls starts showing, together with irregularities such as holes in the floor or objects inside the room. In the end, the entire structure of the area is known without having stepped inside.

This is exactly what this project is part of. To make this happen, the robots needs to be connected to a server that tells them where to drive next. As the robots are making their way around the room, the server collects data and creates a virtual 2D map of the room using pictures and positions from the robots. As for now, this is all happening on ground level. The idea is to include a quadcopter as part of the robot army to map a new dimension in the height. To get an exact idea of what the room looks like, the position of the quadcopter is needed in the exact moment it takes the pictures.

The scope of work is to figure out the best way to consistently measure the distance from the quadcopter to its surroundings, such that pictures taken can correlate with this position.

2 Theory

2.1 Hardware

2.1.1 Quadcopter

Due to the quadcopter being a relatively new concept to this project, it has not yet been bought or created. One of the reasons is that the total weight of the equipment has not yet been determined. This weight consist of the microcontroller, sensors, cameras and maybe more equipment. As for this project, one important factor is the width of the quadcopter. This width will determine whether or not a sensors range is capable of detecting objects before the quadcopter collides with it.

2.1.2 Arduino

Arduino is an open source microcontroller board. Compared to the nRF52840 DK microcontroller which will be addressed in the following subsection, it is simpler to use due to its user-friendly programming language. For this project it was used as a tool to fast and efficiently test if the sensor was working. However, a more complex microcontroller is needed as the quadcopter requires to communicate with a server using bluetooth, which Arduino is not capable of without an external module.

2.1.3 nRF52840 DK

The nRF52840 DK is a multi-purpose board development kit. The microcontroller supports Bluetooth Low Energy, which allows it to communicate with other devices. This is vital as an external server needs a live feed of the readings done by the microcontroller. Compared to the Arduino, the nRF52 has a more complex structure allowing it to do more demanding and advanced tasks. It has several output and input pins allowing it to handle multiple instruments simultaneously, such as sensor readings.

2.1.4 Sensors

There are a variety of different distance sensors available for use. They vary in detection limit, update rate and accuracy. Three different type of sensors will be considered in this section.

2.1.4.1 IR sensors IR stands for infrared. These sensors uses LED technology and operates by emitting infrared light, which is in the wavelength above the visible spectrum. Once the light hits an object, it is reflected back to the sensor where a light detector interprets the signal, thus

providing the distance. They are able to measure very short distances, but when it comes to longer distances, they become more expensive.

2.1.4.2 LIDAR Lidar stands for Light Detection and Ranging. It uses a laser to detect distances. The method is similar to IR, but the wavelength is different. Due to the higher wavelength it can measure very long distances. It also has a very fast update rate. The downside is the cost and current draw.

2.1.4.3 Ultrasonic sensors Ultrasonic sensors produce sound waves that hits and reflects of the object. When the sound wave returns to the sensor, it is converted to distance using the speed of sound. It is capable of detecting distances from 0.02 meters up to 10 meters, depending on the sensor. It is also capable of looking through different medias such as smoke, dust, mist, vapor etc. Ultrasonic sensors can be found very cheap while still maintaining its qualities.

2.1.4.4 Comparison To find the most optimal sensor for the quadcopter it is necessary to consider its purpose. It is meant to map a room and does so by flying relatively close to the walls, floor and ceiling. The sensor will not have to detect very small distances because it is attached to the center of the quadcopter, and there is a limit to how close it can fly to the different objects. The same applies for long distances, as the camera takes pictures relatively close to the wall. LIDAR have the possibility to detect the required range, but is expensive. IR sensors could be a good choice but the distance is not optimal as IR is best for shorter distances. Ultrasonic sensors might seem to be the best option for this use. They have the desired range, works well in different medias and are cheap. If it is required to effectively detect the distance from the quadcopter, it is unnecessary to go for a more expensive alternative. Ultrasonic sensors are by far not flawless. If the surface to be detected is soft, the sound waves can be absorbed in the material, thus giving incorrect results or no result at all. Ultrasonic sensors might also have trouble detecting angled surfaces and it is a risk that the update rate is to slow. To settle for an ultrasonic sensor, further testing is required.

2.1.5 HC-SR04

The ultrasonic sensor used in this project is the HC-SR04 seen in figure 1. It is a very affordable, but well-functioning sensor. It requires a power supply of 5 voltage which can be supplied by the nRF52.

The sensor are equipped with four pins. The power supply, ground, Trig and Echo. The sensor operates by emitting a low-frequency sound when the Trig-pin is triggered. The sound travels until it hits an object, and bounces in the opposite direction back towards the sensor. The Echo-pin continuously observes until it receives the reflected sound [2].

Figure 1: Ultrasonic sensor HC-SR04

This module is able to detect distances from 2 cm up to 400 cm. This is sufficient if the quadcopter has a diameter of less than 800 cm, which is most likely the case. The update rate is 60 ms, which means it can provide approximately 17 measurements per second. If the sensor exceeds 15° relative to the surface that is being measured, it is not efficient as the sound waves wont reflect directly from the surface back to the sensor. The accuracy of the sensor is +- 3 mm when below 1 meter [1].

2.2 Software

2.2.1 Segger Embedded Studio

Segger Embedded Studio is a free IDE, which is short for Integrated Development Environment. Among other things, it allows the user to write code which can be uploaded to the nRF52840 microcontroller. The program was used for this project to get the microcontroller to correctly read the sensor data.

2.2.2 OptiTrack

OptiTrack is a motion capture system used in this project. It can track objects using 16 cameras. To track an object it needs to be mounted with small reflective spheres that are detected by the cameras. These spheres are then measured relative to the ground using an object with the same kind of spheres. The object can then be tracked with 120 FPS. This system was used to compare the actual position of the sensor with the data retrieved from the it.

3 Development

3.1 Setting up the hardware

Before starting to develop software, the sensor needed to be connected to the microcontroller. This was done as shown in figure 2. The red cable provides the sensor with 5 volts. The purple cable is ground, while the green and blue cables are connected to the output and input pins, hence echo and trig on the sensor.

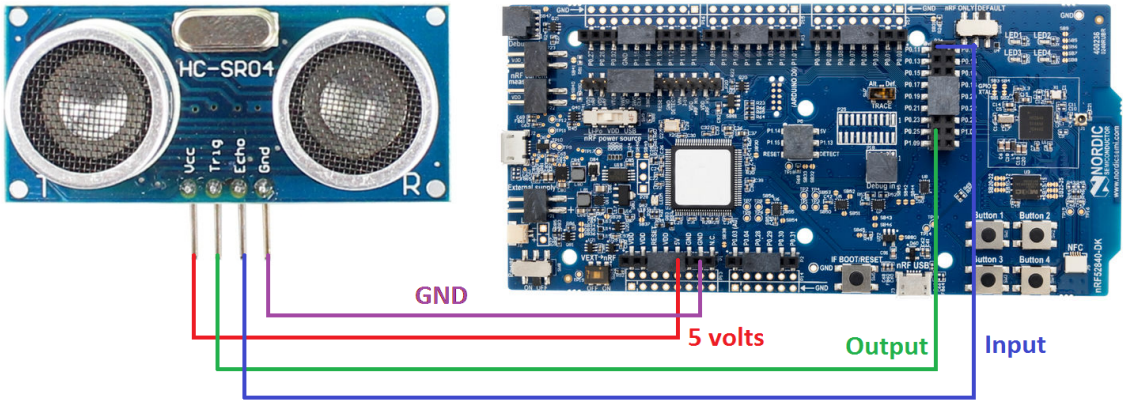

Figure 2: Ultrasonic sensor HC-SR04 connected to the microcontroller nRF52 through 4 cables.

When the physical layout of of the system was done, the software was remaining to get the sensor working. The software is vital as it tells the sensor how to function, and how the different signals should be interpreted.

3.2 Testing with Arduino

To avoid future confusion while debugging, the sensor was first tested using an Arduino. The reason being that it provides a simple way of getting the sensor to function with just a few lines of code. This code can be seen in Appendix A.1. The layout of the components looks more or less identical to figure 2, only that the nRF52 was replaced by an Arduino. The result of this minor test was successful. The serial monitor on the computer printed a distance in centimeters, which was cross checked with the help of a tape measure and wall. This concluded that the HC-SR04 sensor worked as it should, and that the implementation using nRF52 could begin.

3.3 Setting up the code

The code was written in Segger Embedded Studio. An example project specifically meant for nRF52 was provided by Nordic Semiconductors such that all the files needed for the nRF52 was configured and ready. To further get the proper working code for the system, the main.c file

needed to be modified. The full code are located in Appendix A.2. The fundamentals will be simply briefed in the two following subsections.

3.3.1 Converting sound waves to distance

As mentioned in subsection 2.1.5, the ultrasonic sensor operates by emitting sound waves and waits for the echo. If the time for this to happen is measured, one could use that time to calculate the distance. Equation 1 shows how this is done.

$$\text{Distance} = time \cdot 340\frac{m}{s} \cdot \frac{1}{2} \tag{1}$$

In the equation, time is the duration it took for the sound to travel to the object and back. This is the reason for multiplying by 1/2. Without this factor, the resulting distance would be the distance to the object summed with the distance back to the sensor. The speed of sound is set to be 340 meters per second. This equation was implemented in the code to extract the distance, and is the foundation for how ultrasonic sensors work.

3.3.2 Timer

To measure how long it takes for the emitted sound to return to the sensor a timer was needed. This timer starts as soon as a pulse is sent out and stops as soon as it is received again. The time can then be calculated using the speed of sound as mentioned in the above section. Getting this timer in Segger required more than simply importing a timer which can be done in other programming languages, such as e.g. Python.

The code to implement the timer was found online [4]. To get it to work it needed libraries which was not yet in the program and had to be implemented as well. These libraries and configuration files was also found online in another tutorial [3]. As the instructions on these websites was not completely related to the task, some configurations had to be done as well.

After the time-consuming task of creating the code, it could be uploaded to the nRF52, which could be used to control the sensor. Using a debug log function in Segger the distance was continuously shown live on the screen. This distance data was also collected by Segger into a separate file which later was used as comparison.

4 Testing

As mentioned in 3.2 - Testing with Arduino, the ultrasonic sensor did indeed communicate the correct distance. However, using a tape measure with static measurements is not sufficient when the sensor is to be used on a moving quadcopter. To get a more genuine comparison, OptiTrack was used. This system can simulate a moving object in space while tracking the position at all times, thus providing a precise and sufficient comparison.

4.1 Setting up the test environment

From section 2.2.2, OptiTrack requires three small reflective spheres to properly detect and follow an object. The sensor is the object in this scenario due to the measured distance being relative to the sensor itself. The challenge was therefore to make OptiTrack see the sensor as this object. The solution was an improvised rigid body containing the microcontroller and sensor together with the three spheres seen in figure 3.

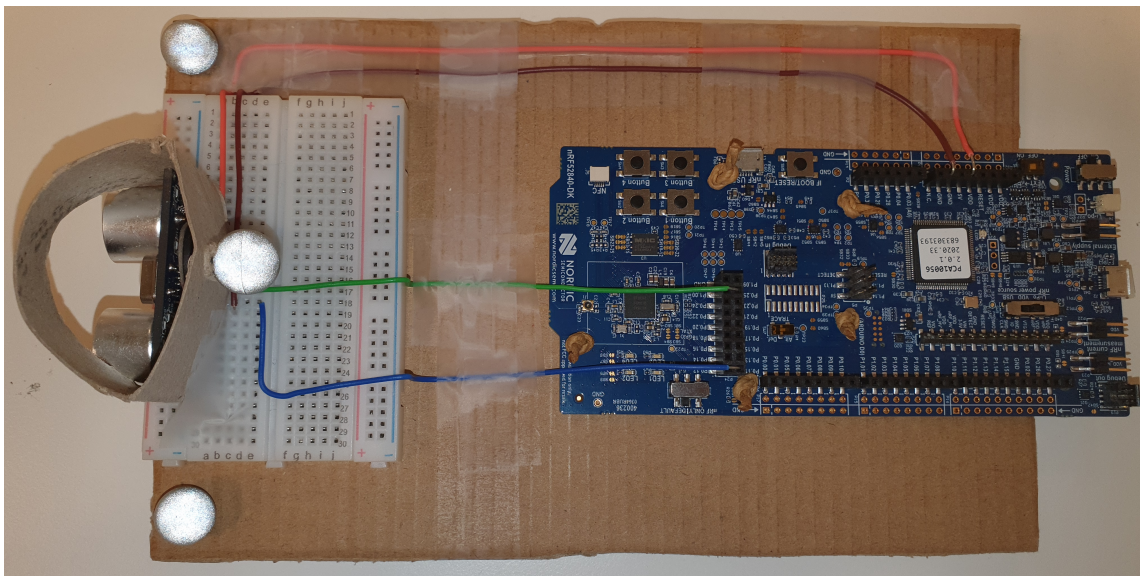

Figure 3: Assembly containing the functioning sensor together with OptiTrack spheres

The three spheres are located to the left in the assembly, surrounding the sensor. The sphere in the middle, slightly to the right, is elevated higher than the sensor, and the two other spheres on the edges lies slightly lower than the sensor. The reason for the placement comes from the triangulation point caused by the three spheres. To get precise data, this point had to imitate the exact position of the sensor.

The board could now be connected to a laptop via USB, providing sensor data as well as real-time position from OptiTrack.

4.2 Different scenarios

The assembly shown in figure 3 was used to conduct the following experiments, together with OptiTrack. The following subsections are only explaining how the tests was organized as well as showing the results. These results will be discussed in section 6.

4.2.1 Ordinary surfaces

The first test conducted was to check how the sensor behaved for a simple maneuver. The simulated flight was of a hovering quadcopter that slowly ascends for a few seconds, before descending. The sensor is pointing straight down. The floor was made a reference point for OptiTrack using a rigid body that comes along with the kit, such that both the sensor and OptiTrack calculated the distance down to the floor. Figure 4 shows the corresponding results from this test, where OptiTrack can be interpreted as the actual height, shown in orange.

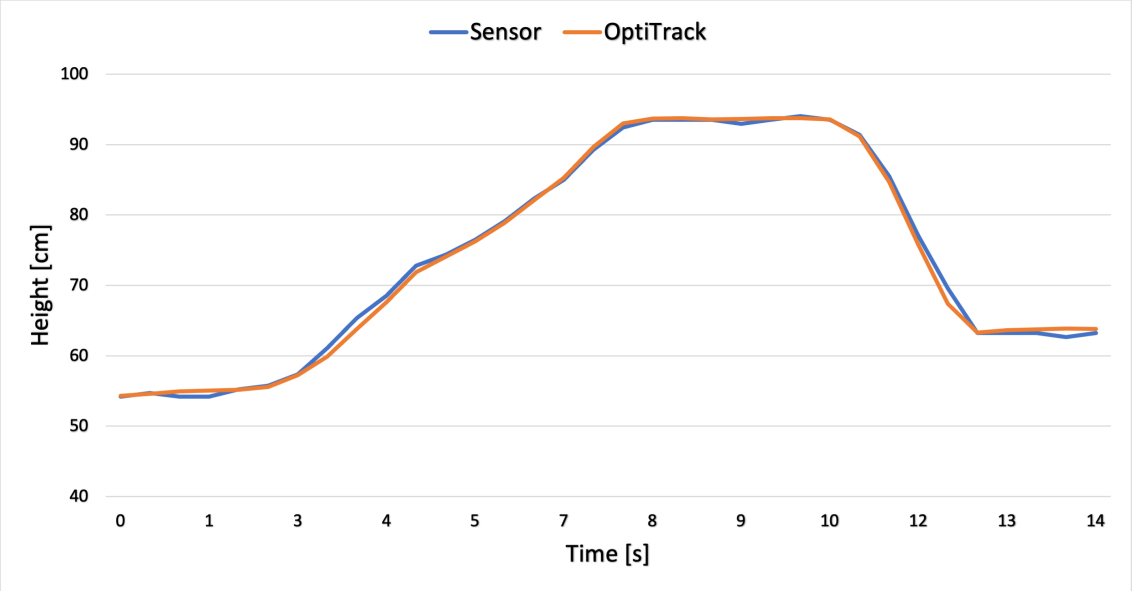

Figure 4: Comparing measurements from sensor and OptiTrack

To challenge the sensor and data from the first test, a second test was run to check how the performance was during a more aggressive hovering of the quadcopter, that is where it is quickly alternating between flying up and down. Figure 5 shows how the sensor performed compared to OptiTrack for such a flight.

This test simulated a flight lasting 33 seconds. The data shows slightly more difference between the sensor and OptiTrack compared to the results from the first test.

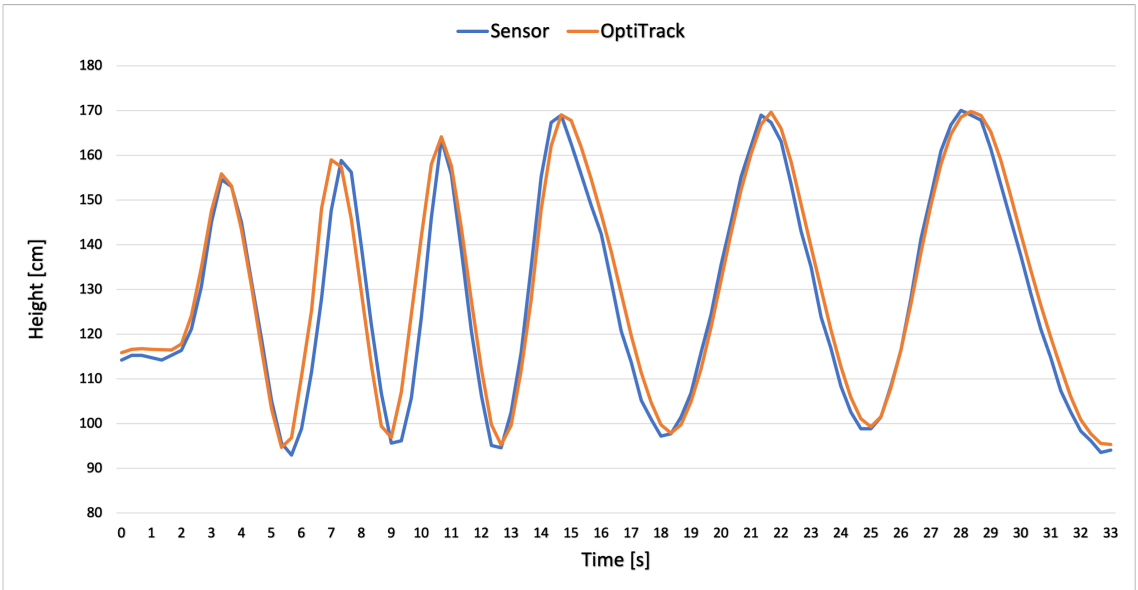

Figure 5: Comparing measurements from sensor and OptiTrack

4.2.2 Soft surface

From section 2.1.4.4, the flaws of ultrasonic sensors were mentioned. Whereas one of them was that soft surfaces could absorb sound, thus stopping the sensor from receiving reflected sound. If the case is that the sensor still functions in this specific environment, only with reduced quality, it might still be an good option. To test this, a mattress made out of foam rubber was placed against the wall, shown in figure 6.

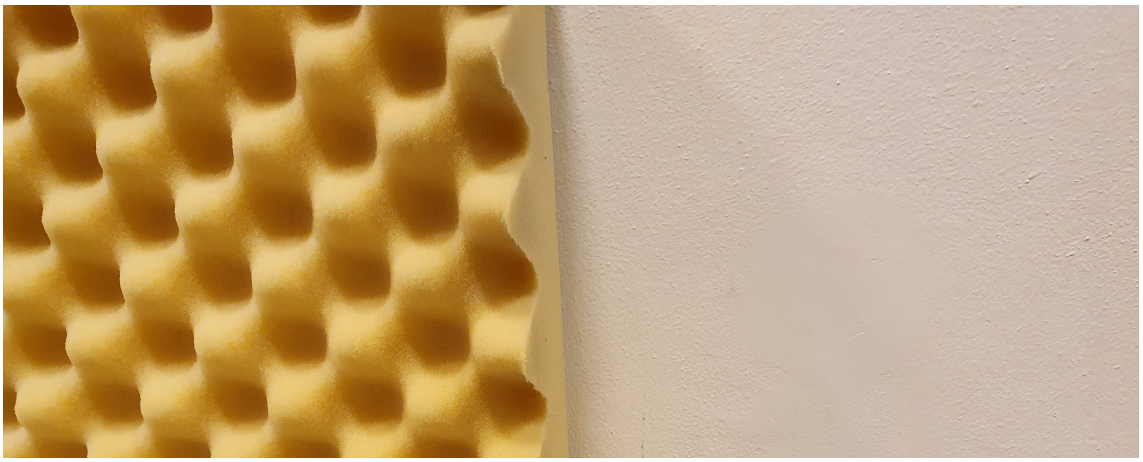

Figure 6: Soft surface to the left, hard surface to the right

The sensor was then activated while pointing at the wall. From this position it was slowly moved parallel with the wall towards the mattress. When reaching the mattress the live log went from continuously producing distance data, to not producing anything at all. However, the same test was conducted again, but with a thin sheet over the mattress. This thin sheet was sufficient to make the sound reflect back to the sensor.

4.2.3 Hanging obstacles

When the quadcopter is autonomously flying inside a room, it might encounter hanging obstacles. These obstacles can for example be wires supporting a lamp, or threads supporting hanging decorations. In this test a hanging net was used to check if small tiny objects could be detected by the sensor. The setup is seen in figure 7.

Figure 7: Sensor pointing towards a net and a rigid surface

The distance from the sensor to the wall to the right is 70 cm, and the net is hanging at a distance of 40 cm from the sensor. Figure 8 shows how the sensor performed in such a situation. It varies between detecting both the net and the wall, resulting in a very uneven graph.

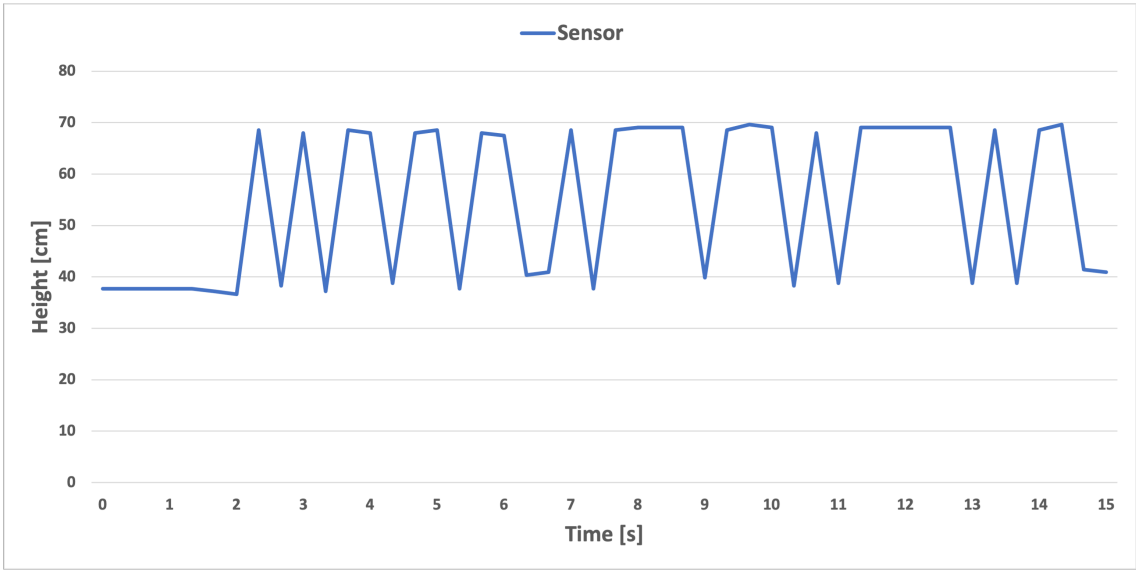

Figure 8: Sensor detection of multiple objects

5 Sources of Error

Due to a shortage of time and resources, the test environment was not optimal. Precision and quality are slightly affected due to this. The following errors can either be eliminated or minimized, as will be mentioned in Chapter 7 - Further Work.

5.1 Human error

OptiTrack measures the distance above the ground from a perspective of a rigid body. This body is placed with the help of a bubble level that ensures correct measurements. However, for the sensor this is not the case. When taking measurements with the ultrasonic sensor, it is essential that it points straight to the surface it is measuring. A change in angle will alter the distance measured. Equation 2 shows how an angle of 10° at a distance of 2 meters can give an error of 2 centimeters. Depending on the desired accuracy, this should be taken into account.

$$\text{Angled distance} = \frac{Distance}{Angle} = \frac{2}{\cos 10} \approx 2.031 \tag{2}$$

Another source of human error is when it comes to recording the data. As OptiTrack and Segger runs on different systems, the recordings needs to be started simultaneously by hand on each computer. This is not a major issue as the start times of the data can be altered at a later time. However, the need for altering can be difficult to notice when the difference is small.

5.2 OptiTrack

OptiTrack is a highly precise tool, but as all electronic systems it can have its flaws. At the beginning of every testing session it is calibrated to avoid potential changes of the cameras from last time. This ensures that the system is as precise as it can be. There is not much to do other than trust what this system is capable of.

5.3 Time reference

When combining the first data into graphs, a more complicated error was noticed. The trend shown in figure 9 continued to repeat itself at every test in varying styles. Some parts of the graphs overlapped while others appear to be shifted. An example of this shift can be seen in figure 9 from 70 to 140 seconds into the test.

This shift derives from the fact that the sensor-data is not matching up with the times from OptiTrack. The reason being that the nRF52 software was not implemented with a time-tracking system for each value measured. One could also argue that the sensor is not functioning, as it fails

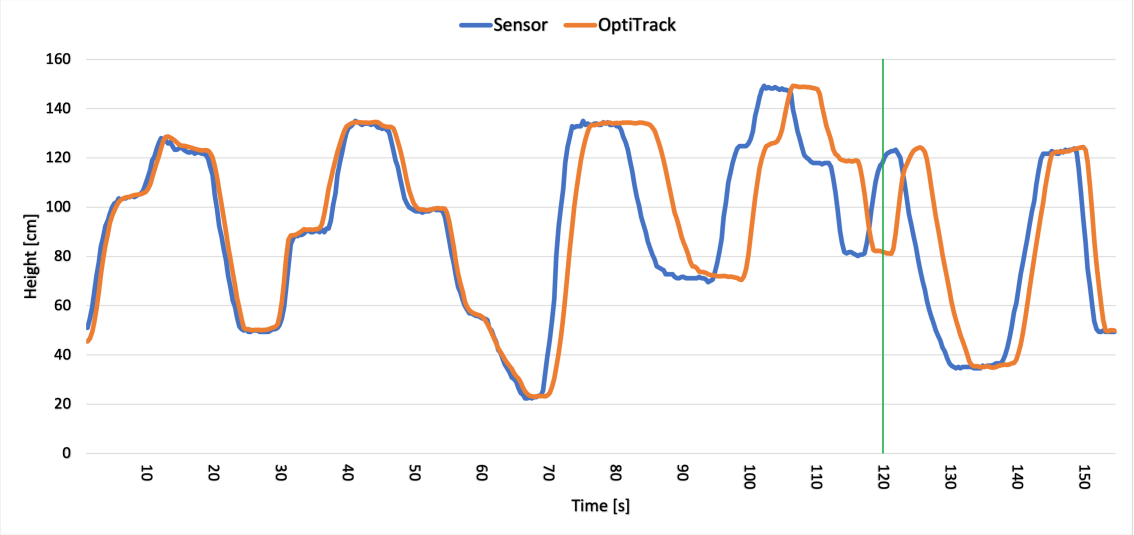

Figure 9: Example of a shifted graph.

to provide satisfactory data over a longer period of time. To test that this was not the case, and that the data actually overlapped at each corresponding time, a simple test was conducted using a screen recorder. The screen recorder was started at the same time as OptiTrack while recording the values gathered from the sensor. Afterwards data and times was cross-checked, indeed proving that the graphs overlaps. An example of such a cross-check was at 120 seconds in figure 9. At this point in time it looks like the sensor is getting a height of 120 cm from the ground, while OptiTrack shows a height of around 80 cm. From the screen recorder, this was proved to not be entirely correct. At 120 seconds into the test, the sensor printed a value of 123.78 cm, while OptiTrack showed the sensor the be 123 cm from the ground. At this point of time, marked with a green vertical line in figure 9, it is the sensor data that shows the real height. This does not mean that OptiTrack provides wrong data, only that the collecting times does not match.

Another clear indicator that the data is wrong, is that from figure 9 the sensor appears to register the change in altitude before OptiTrack. This is highly unlikely as OptiTrack is an expensive and highly precise motion capture system, while the ultrasonic sensor is a very cheap and simple device.

The results shown in the previous chapter was also affected by this error. The figures from the two first test, figure 4 and 5, are in reality only the beginning of a much longer sequence of data. The reason for choosing only a small segment of a much longer data set was to reduce the time reference error. It seemed that the longer the set of data, the more the beginning of the graphs overlapped. This was also cross-checked with a screen recorder. The same scenario can be seen in figure 9, whereas the start of the figure seems to overlap more than the end. This was a quick fix to the issue, but is not an option for future tests, which will be addressed in chapter 7 - Further Work.

6 Discussion

The tests conducted in chapter 4 showed somewhat promising results. The data from the sensor are more or less overlapping the data from OptiTrack. This is a good indicator that the sensor is functioning properly. Its correctness varies slightly throughout the test. This can especially be seen from the second test in figure 5. Between 5 and 11 seconds one can notice what looks like a delay for the sensor. In this interval, the greatest deviation occurs at 7 seconds. At this moment the sensor is short by 20 cm compared to the OptiTrack plot. This means that at some point in time the sensor believes the quadcopter to be 20 cm closer to an object than it really is. However, this is only for a very short amount of time, less than 0.5 seconds.

Deciding wherever the results are good or bad, requires repeating the scope of this project:

The scope of work is to figure out the best way to consistently measure the distance from the quadcopter to its surroundings, such that pictures taken can correlate with this distance.

Taking this into consideration, a picture taken might in a worst case scenario correlate to a distance that is off by 20 cm. This will result in the virtual created map also being incorrect. However, due to the sources of error, the sensor could indeed turn out to be way more precise than what the results shows.

The third test confirmed that soft objects does indeed have a major impact on the sensors function. This is not optimal, but it could potentially be ignored. Most of the objects in a room is made out of hard materials such as wood, metal or plastic. If the sensor encounters a couch or bed, it is usually fitted with a sheet, thus being noticeable by the sensor. If very soft objects are to be accounted for, one would have to go for a different sensor, such as sensors based on light.

The last test showed how the sensor performed if it encountered a hanging object. The important part is that it noticed both the net and wall. With this information one could take pictures, and filter the data such that both objects could be mapped. This net was also rather dense. It is not tested on thinner nets, but this can also be considered a special scenario, as small hanging objects in the middle of a room is rare.

Ultrasonic sensors can, potentially with small error, provide the distance from a flying quadcopter. There are certain scenarios where it fails to provide results, but in general it works in most environments. Therefore, it could be used as a distance measurement tool to map a room. The precision however, can be improved by using more, or different sensors. A combination of sensors is also possible. This would create a highly precise system that would function in any environment.

7 Further Work

This last section addresses which way to move forward from this point on in the project. There are ways to improve the data, and project in general.

7.1 Update frequency

One issue that appeared during the testing was the rate at which the ultrasonic sensor provided data. As mentioned in subsection 2.1.5, the sensor is able to provide 17 measurements per second. When testing in reality it was counted to be roughly 20 measurements. This is a good thing, but the issue arose when using the nRF52 instead of the Arduino. During the testing, it only printed a depressing number of 3 measurements per second. That is 1 measurement per 330 millisecond. Exactly why this was the case was not solved. It might have something to do with the code for the nRF52 being complex. It provided enough data to finish the testing, but in a real time scenario this would have to be fixed for optimal performance.

7.2 Different sensor

The ultrasonic sensor proved to be effective and fit for the task in most scenarios, with minor errors. However, it needs more testing to check if these errors occur due to the sensor itself. It is most likely due to the update rate mentioned above combined with the sources of error. If it turns out to be less than ideal for providing distances, new sensors needs to be considered.

A sensor based on light, such as a LIDAR could do the trick. It is more expensive, but has better accuracy. It would also solve the problem of detecting soft objects and being able to detect steep edges. In return it would face different problems, such as measuring distance in smoke or dust filled areas. A combination of the two sensors would be ideal, but expensive. The mapping would then be able to function in almost all scenarios.

7.3 Improving the time similarity

To eliminate the error caused by time referencing as mentioned in 5.3, a proper timer should be implemented on the nRF52. By doing this the data from the sensor and OptiTrack could be compared at a very precise level, providing secure results.

7.4 BLE

The nRF52 is not yet connected to the server which will be receiving the photographs and distance data. This was neglected in this project as it is not needed to solve the related problem. The reason it still should be implemented at an early stage is because it will make testing easier. Instead of carrying around a laptop that communicates with the nRF52 through USB, it would be a whole lot simpler to only carry the nRF52 together with the sensor and a battery package.

7.5 Direction of sensors

Another aspect to be considered is the amount of sensors and direction in which they point. The quadcopter will be flying around a room taking pictures. If there are more than one camera pointing in different directions, there is also need for more than one sensor. How these sensors and cameras should be optimally placed could be figured out.

List of Figures

Bibliography

[1] barkhausen institut. *Research on Distance and Proximity Sensors*. URL: https://www.barkhauseninstitut.org/research/lab-1/our-blog/distance-sensor-research. (accessed: 17.09.2021).

[2] Rui Santos. *Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino*. URL: https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. (accessed: 17.09.2021).

[3] Einar Thorsrud. *nRF5 SDK Application Timer Tutorial*. URL: https://devzone.nordicsemi.com/guides/short-range-guides/b/software-development-kit/posts/application-timer-tutorial?pi864299341=2&pi628=2. (accessed: 03.10.2021).

[4] Mahesh Venkitachalam. *Talking to Ultrasonic Distance Sensor HC-SR04 using nRF51822*. URL: https://electronut.in/nrf51-hcsr04/. (accessed: 25.09.2021).

A Appendix - Code

A.1 Arduino

```
1 #define echoPin 2
2 #define trigPin 3
3
4 long duration;
5 int distance;
6
7 void setup() {
8   pinMode(trigPin, OUTPUT);
9   pinMode(echoPin, INPUT);
10   Serial.begin(9600);
11   Serial.println("HC-SR04 test");
12 }
13 void loop() {
14   digitalWrite(trigPin, LOW);
15   delayMicroseconds(2);
16   digitalWrite(trigPin, HIGH);
17   delayMicroseconds(10);
18   digitalWrite(trigPin, LOW);
19
20   duration = pulseIn(echoPin, HIGH);
21
22   distance = duration * 0.034 / 2;
23
24   Serial.print("Distance: ");
25   Serial.print(distance);
26   Serial.println(" cm");
27 }
```

A.2 nRF52840 DK

```
1 #include <stdbool.h>
2 #include <stdint.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include "nrf_delay.h"
7 #include "nrf_log.h"
8 #include "boards.h"
9 #include "nrf_gpio.h"
10 #include "app_util_platform.h"
11 #include "nordic_common.h"
12 #include "nrf.h"
13 #include "nrf_gpiote.h"
14 #include "nrf_drv_gpiote.h"
15 #include "nrf51_bitfields.h"
16 #include "app_timer.h"
17 //
18 #define pinTrig 25
19 #define pinEcho 11
20 #define DATA_SIZE 1000
21
22 static volatile uint32_t tCount = 0;
23
24 static volatile float countToUs = 1;
25
26 void start_timer(void)
27 {
28   NRF_TIMER1->MODE = TIMER_MODE_MODE_Timer;
29   NRF_TIMER1->TASKS_CLEAR = 1;
30
31   uint8_t prescaler = 0;
32   NRF_TIMER1->PRESCALER = prescaler;
33   NRF_TIMER1->BITMODE = TIMER_BITMODE_BITMODE_16Bit;
34
35   uint16_t comp1 = 500;
36
37   NRF_TIMER1->CC[1] = comp1;
```

```
38
39    countToUs = 0.0625*comp1*(1 << prescaler);
40
41    printf("timer tick = %f us\n", countToUs);
42
43    NRF_TIMER1->INTENSET =
44        (TIMER_INTENSET_COMPARE1_Enabled << TIMER_INTENSET_COMPARE1_Pos);
45
46    NRF_TIMER1->SHORTS = (TIMER_SHORTS_COMPARE1_CLEAR_Enabled <<
47                                TIMER_SHORTS_COMPARE1_CLEAR_Pos);
48
49    NVIC_EnableIRQ(TIMER1_IRQn);
50
51    NRF_TIMER1->TASKS_START = 1;
52 }
53
54 void TIMER1_IRQHandler(void)
55 {
56    if (NRF_TIMER1->EVENTS_COMPARE[1] &&
57          NRF_TIMER1->INTENSET & TIMER_INTENSET_COMPARE1_Msk) {
58
59       NRF_TIMER1->EVENTS_COMPARE[1] = 0;
60
61       tCount++;
62    }
63 }
64
65 bool getDistance(float* dist) {
66    nrf_gpio_pin_clear(pinTrig);
67    nrf_delay_us(20);
68    nrf_gpio_pin_set(pinTrig);
69    nrf_delay_us(12);
70    nrf_gpio_pin_clear(pinTrig);
71    //nrf_delay_us(20);
72
73    while(!nrf_gpio_pin_read(pinEcho));
74
75    tCount = 0;
76
```

```
77    while(nrf_gpio_pin_read(pinEcho));
78
79    float duration = countToUs*tCount;
80    float distance = duration*0.017;
81
82    if(distance < 400.0) {
83      *dist = distance;
84      return true;
85    } else {
86      return false;
87    }
88 }
89
90 int main() {
91      app_timer_init();
92      start_timer();
93
94      printf("starting...\n");
95
96      nrf_gpio_pin_dir_set(pinTrig, NRF_GPIO_PIN_DIR_OUTPUT);
97      nrf_gpio_pin_dir_set(pinEcho, NRF_GPIO_PIN_DIR_INPUT);
98      float dist;
99
100     while(1) {
101       if(getDistance(&dist)) {
102         printf("%f\n", dist);
103       }
104       //nrf_delay_ms(5);
105     }
106 }
```