Aalborg University
Electronics and IT 5. semester

Digital and Analog Systems Interacting with the Surroundings

Semi-automatic Quadcopter Positioning

17. december 2015

AALBORG UNIVERSITET
STUDENTERRAPPORT

Aalborg University
School Of Information and Communications Technology
Department of Electronic Systems
Frederik Bajers Vej 7
9220 Aalborg Ø
Tlf.: 9940 9730
Telefax: 9940 9725
http://www.es.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Subject: Digital and Analog Systems Interacting with the Surroundings
Title: Semi automatic Quadcopter positioning
Project period: 10. September 2015 - 17. December 2015
Semester: 5. Semester (P5)
Project Group: 15gr515
Authors:

Jóhannus Kristmundsson

David Aleksander Stanislaw Syberg

Piotrek Wierciński

Regin Lindenskov Hansen

Supervisor: Kirsten Mølgård Nielsen

Physical Copies: 7
Number of Pages: 100
Attachments: One CD

Synopsis:

Bridges and hulls of ships have to be measured on a regular basis to ensure structural integrity. The theme of this project is utilizing a quadcopter to assist humans in making ultrasound measurements of ship hulls. This problem was analyzed and it was discovered that the quadcopter has to stay at a constant distance from the wall for some time, to do the measurement. Several measurement methods were analyzed and ultrasonic ranging was chosen. A system consisting of a remote control with a user interface for settings, a communication protocol with addressing and package types (among other properties), built on top of a storebought solution for the physical layer, sensors for distance and pitch, and controller algorithms for keeping a constant distance and direction to the ship hull, was designed and implemented. A sensor circuit was designed from measurement of quadcopter noise and transducer response in an anechoic chamber, but the design was unsuccessful in implementation, and another sensor was used. When testing the project as a whole the direction controller worked, but there was problems testing the distance controller, because of blind spots in the motion tracking system used, and quadcopter platform instability/drift.

Preface

This project is a semester-project on fifth semester of Electronics and IT at Aalborg University, and is thus somewhat limited in theme and contents because there are requirements that some theory must be included and some methods used. Read the project report with this in mind. But even though these requirements where kept in mind during the choice of project, the theory and use of methods arrived naturally during the work on this project, because as it turns out, they are useful. But this is also a reason for not dwelling too deep into for example the acoustic aspects of some of the things in this project, but instead focusing on signal processing.
We would like to thank Kirsten M. Nielsen, for supervising the project, and the staff at Aalborg University, for being very helpful, especially Henrik Schiøler for providing advice and equipment, and Jens Dalsgaard for providing an Arduino kernel.

Reading guide

The report is structured in four main parts. First an analysis of the problem and a look at possible solutions. Then a system specification. Then a in depth look at the design and implementation of that system. In the end are tests and some conclusions. Measurements that were made as a part of the project work, are described in appendices in the end of the report.
The acronyms are in the beginning of the report (but are also explained first time used). All programs, and measurement data, schematics, etc. is located on the CD, and is to be seen as a reference in case the reader wonders about something that was not described in the report.

Abbreviations

ADC Analog to Digital Converter

CMOS Complementary metal-oxide-semiconductor

CPU Central Processing Unit

DUT Device Under Test

EEPROM Electrically Erasable Programmable Read-Only Memory

FC Flight Controller

GOT Games on Tracks

GPS Global Positioning System

IC Integrated Circuit

IO Input-Output

ISR Interrupt Service Routine

LCD Liquid Crystal Display

LED Light-Emitting Diode

LIDAR Light Detection and Ranging

QC Quadcopter

PPM Pulse position modulation

RADAR Radio Detection and Ranging

RC Remote Controller

RF Radio Frequency

RMS Root Mean Square

RPM Revolutions per minute

SDK Software Development Kit

SONAR Sonic Navigation and Ranging

SNR Signal to noise ratio (usually in dB)

TOF Time of Flight

VCO Voltage Controlled Oscillator

Contents

Chapter 1

Introduction & problem statement

1.1 Introduction

This project revolves around the notion of using quadcopters as a tool. The popularity of quadcopters is on the rise due to the availability of quadcopters with many features, such as remaining in balance state, following GPS coordinates, and they also have small sizes and agile maneuverability. The main reason though, is that they are mechanically simple, and thus perfect for hobbyists and amateurs alike [1].
They are very simple in construction and replacing used components with new ones, is not a problem. Because of this they are increasingly used for many tasks other than just flying. A common one is for filming, as they are much cheaper than a news helicopter. They can be used to solve tasks in ”hard to reach” places, because they can fly, and they can lift quite a lot of equipment.
One of these tasks could be measuring thickness, force tensions and condition of many kind of materials in constructions that require a lot of climbing or large cranes to be measured, such as the hull of ships, bridge constructions (arches), tall pillars, chimneys, stanchion etc. That way people can save their time, safety and money. As for now many of those activities are done by humans who must have appropriate safety harness and be in a good physical condition. These jobs require education and experience, what's always followed by extra costs.
Also this is a good solution for repeated measurements as they could be done by programming a special 'route' for a drone along most exposed places. This would especially be valuable with bridges and ships where measurements have to be made with constant time period.
In order to meet the challenge a quadcopter must be very stable (remaining in one place during the time of measurement), have adequate size and possess a set of sensors, that would contain an ultrasound sensor (the standard technology for measuring the hull of a ship), and some sensors to get the quadcopters position, so a controller can make sure the quadcopter positions itself appropriately. It could also be semi-autonomous with possibility to control it in an easy way by a pilot. This could be done by using remote control connected wirelessly to the quadcopter. Also it would be even easier for the pilot to control the quadcopter if it possessed a camera, thus eliminating the need for the pilot to be within line of sight.
The important thing about this idea is the environment in which quadcopter would work. We must know what conditions are a danger to the machine, what and how it can affect

the measurement and under what circumstances it will be the most efficient.
Our project will focus on obtaining a satisfying behavior of the quadcopter to do measurements, without getting into measurement itself, to limit the scope of the project. Following parts will be a statement of the problem, and an analysis of it.

Chapter 2

Problem Statement

The problem of making ultrasound measurements on the hull of ships can be stated as:
How does one, with the help of an electronic system integrated into a quadcopter platform, measure metal thickness on the hull of ships?
There are several big parts to solving this problem. One is the measurement itself on the hull of a ship, and processing the data. The other is to mount this system on a quadcopter. The third is to position the quadcopter for making the measurement somewhat automatically (since it is hard to fly a quadcopter).
In order to limit the scope of this project to one semesters work, we choose to focus on the positioning of the quadcopter for making measurements, and will be using a finished quadcopter with a flight controller which keeps it level. This project will be about designing a remote control, a platform to receive those controls, and can take over for the pilot, to position the quadcopter for measurements. This means this project will contain:

- Measuring of distance from quadcopter to ship hull
- Automatic placement of the quadcopter for measurements using distance sensors as input, and outputting control commands to the quadcopter flight controller
- Remote controlling the quadcopter wirelessly and receiving sensordata

From this the problem statement can be rewritten:
How does one wirelessly control a quadcopter and position it for making ship hull thickness measurements automatically, using distance sensors for position determination?
This problem statement raises several question, which should be answered, before starting the work on the system prototype:

- What is the method of measuring hull thickness (to figure out what the requirements of positioning is)?
- How can the quadcopters position be determined?
- How will the quadcopter position itself for making measurements?
- How is the quadcopter controlled, how does it fly?

Chapter 3

Analysis & Specifications

3.1 Measurements on ships

Every fifth year all ships over 100 tons have to go though a classing, where it is determined if the ship is sea worthy. During this classing, the state of the hull of the ship will be checked. To check the state of the hull, the thickness of the steel must be measured.

3.1.1 Ultrasonic thickness measurements

NB! This part explains why it is necessary to touch the wall we want to measure with the sensor, thus explaining why the quadcopter needs to position itself at a certain distance to the wall. This project will not focus on actually measuring ship thickness, but rather on the positioning itself.
Ultrasonic thickness measurement is a method of performing measurements of the local thickness of a solid element, typically metal. There are a couple of different ways to measure the thickness, one of them will be explained here.

An ultrasonic wave will be emitted trough a delay line, into the material, which will be measured. When the wave is emitted, the receiver will pick up the signal. When the wave enters the material, and when the wave exits the material, the wave reflects back. This reflection will be picked up by a receiver in the same place as the transmitter. The measurement is based on the time taken from the entry echo to the return echo. The reason for the delay line is when a signal is transmitted, the receiver will pick up this signal. That means the material needs to be thick enough for the transmitted pulse to be finished transmitting, before the wave reflects of the inside of the material, because if it isn't finished, it is not possible to find the variable t.

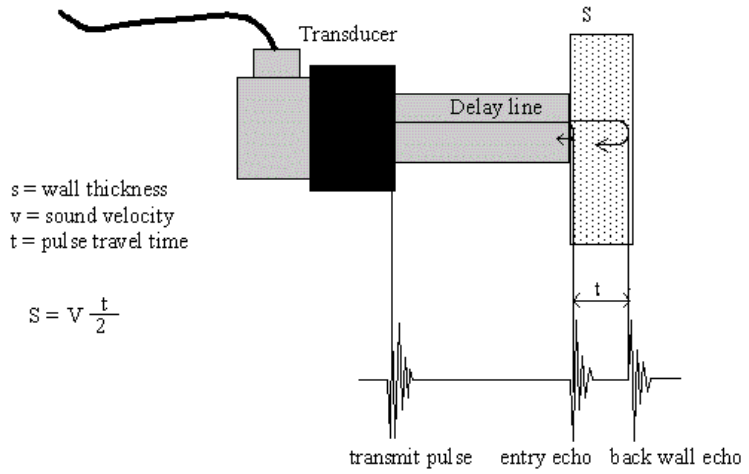


Figure 3.1: Thickness measurement on a solid measurement [2]

The ultrasonic wave is typically emitted from a piezoelectric cell. This piezoelectric cell, and the sensor for receiving the ultrasonic wave are typically mounted on the same head. It has been observed that ultrasonic waves travel through metals at a constant speed characteristic to a given alloy with minor variations due to other factors like temperature. Thus, given this information, one can calculate the length of the path traversed by the wave using this simple formula:

$$S = V \cdot \frac{t}{2} \tag{3.1}$$

Where:
S Is the thickness of the material. $[m]$
t Is the time from the entry echo to the return echo. $[s]$
V Is the sound velocity in the material. $[m/s]$

One of the issues of ultrasonic thickness measurement, is it is necessary to calibrate the sensor for every material. Specifically, to find the value for V, because is varies between different materials.
The reason the transmitter/receiver head has to have contact to the material which has to be measured, is because to much of the initial wave will be lost in reflections on the entry, and thus it will not be possible to measure the second reflection. To calculate the reflection fraction the acoustic impedance has to be known.
The acoustic impedance is defined as:

$$Z = d \cdot c \tag{3.2}$$

Where:
Z Is the acoustic impedance. $[\frac{m/s \cdot kg}{m^3}]$
d Is the density of the material. $[kg/m^3]$
c Is the speed of sound wave in the material $[m/s]$

In this case, we want to find the acoustic impedance of air and steel. The density of air is 1,225 $[kg/m^3]$, and the speed of sound through dry air at 20 degrees Celsius is 343 $[m/s]$ [3] and thus the acoustic impedance of air is :

$$Z_0 = 1,225[kg/m^3] \cdot 343[m/s] = 420 \qquad \left[\frac{m/s \cdot kg}{m^3}\right] \quad (3.3)$$

The density of steel is between 7750 and 8050 $[kg/m^3]$ [4], and the speed sound travels through steel is about 5920 $[m/s]$, and thus the acoustic impedance of steel is:

$$Z_L = 7750[kg/m^3] \cdot 5920[m/s] = 45,9E6 \qquad \left[\frac{m/s \cdot kg}{m^3}\right] \quad (3.4)$$

With this information, it is possible to calculate the reflection fraction of the transition from air to steel, the reflection co-efficient is the ratio of the intensity of the reflected wave to the incident wave.

$$R = \left(\frac{Z_L - Z_0}{Z_L + Z_0}\right) = \left(\frac{45,9E6 - 420}{45,9E6 + 420}\right) \approx 99,99982\% \quad (3.5)$$

Where:
Z_L Is the acoustic impedance of the material with will be measured.
Z_0 Is the acoustic impedance of the medium the wave has to go trough
R Is the reflection co-efficient

This means almost none of the intensity of the transmitted wave will have entered the medium, which has to be measured, and thus the intensity will not be large enough to measure the second reflected wave. This is why the sensor has to be flat on the measurement surface.
The equations for transmission and reflection of ultrasound intensity are independent of frequency for specular reflection. Therefore changing the transducer frequency does not alter the fraction of intensity transmitted/reflected at an interface.

3.1.2 Measurement positions on a ship

The first rule of steel thickness measurements on ships, is that there has to be at least one measurement on each plate. This is the very least which has to be done, but there are a couple of areas of interest, where more measurements have to be taken. It is less harmful if a hole comes on the tank of the ship, compared to other places. Because tanks are isolated from each other, and from the rest of the ship, and that means if a hole comes on the tank, water will just fill up the tank, and not leak into the rest of the ship. This makes the outside of tanks less important to check well. Under the engine room is also a place of interest, because if a piece of metal falls down to the keel of the vessel, it will cause the steel under it to corrode quicker. So to catch tools, screws and other things made of metal, before they make to much damage, more samples are taken under the engine room. There are also taken a few more samples around the rudder and propeller of the ship, but these places are often difficult to reach, because of the curvature around these places.
At the front, especially on the forepeak are also places, where more measurements will be taken. The reason for this, is the front is more often damaged when the ship docks, it is also the part which breaks the ice in cold climates.

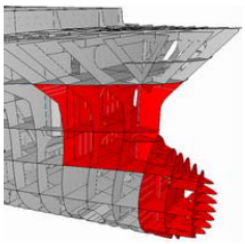


Figure 3.2: There is a larger chance for the forepeak to be damaged, compared to the rest of the ship [5]

The water wind streak is the place where the water at the air meet. This place typically is wet, but air can still access it. This makes this streak corrode quicker than on the rest of the ship.

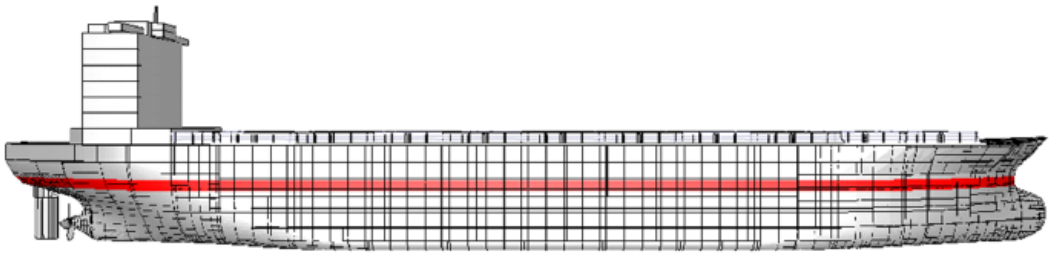


Figure 3.3: Wind and water streak[5]

An often used pattern is taking a lot of samples on the water wind streak, typically about one per 20 cm, and once per plate make a line up and down per plate.

3.2 What does it take to make flying measurements?

This section explores what is actually needed, for the quadcopter to make measurements on hard to reach places, in terms of flight and control. As the measurements are being made at hard to reach places the quadcopter must first navigate to the designated measurement area. Then it needs to approach the area, touch the sensor to the surface, stay still while the measurement is made, remove the sensor from the surface, and then fly to the next measurement area. If a series of measurements should be made (as is often the case), it could be convenient to be able to navigate the quadcopter in two dimensions. This means it would stay at a fixed distance from a wall, or so to say "follow" it.
The first part of flying to the wall can be accomplished with semi manual flight. This means the "pilot" decides in which direction the quadcopter should move, and the computer on board translates this into motor power, while keeping the balance of the quadcopter.
The second part of approaching the wall is more complicated. The quadcopter should approach the wall carefully, at a right angle. We don't want to slam into the wall, but carefully touch the sensor to the wall. This means the quadcopter computer needs to

know its distance to the wall, and its angle relative to the wall.

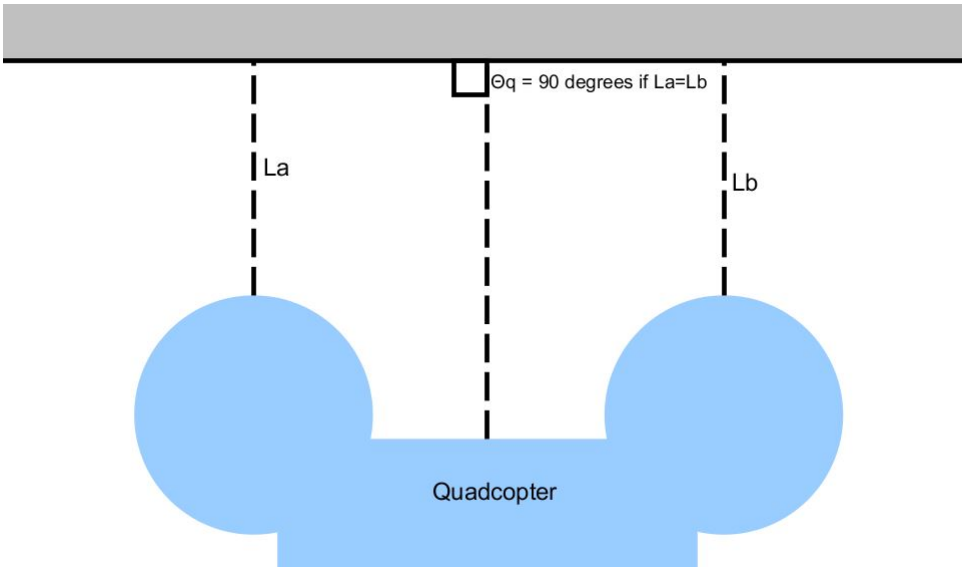


Figure 3.4: Example of measuring method using two distance sensors outputting l_a and l_b.

There are several ways of doing this. An obvious way would be to simply measure the distance in real time using one of several possible sensors. In order to keep a right angle it is necessary to have two sensors, one on each side (as indicated on the figure above). The device for measuring the thickness of the hull would then be placed on rod placed centrally and going out a distance from the quadcopter (to have a safety distance from the wall when flying). In order to ensure good contact, the measuring device mounting rod should be somewhat flexible.

3.2.1 The Rangefinder - Distance sensors

In order to measure the physical distance and angle between the quadcopter and the wall, two distance sensors are needed. There are many ways of ”sensing” a distance, aside from contact methods. The distance can be measured acoustically, by emitting an ultrasonic pulse, and counting the time until its reflection returns. This can also be done with an electromagnetic wave (radio). The measurement can also be made optically using a laser, and again measuring the time until the reflection returns (commonly referred to as time of flight TOF [6]). Measuring the distance with a laser and light sensor is in theory the same as with an radio transceiver, only the frequency is much higher (both are electromagnetic radiation and have the same propagation velocity, speed of light), although it is measured in very different ways in practice.
Other methods of measuring a distance include: *Stadiametric range finding* (measurement of a distance to an object of a known size by use of reticle in telescopic sights [7]), and *coincidence range finding* (measurement of a distance by use of the parallax principle, parallax being the displacement in an apparent position of an object when viewed along two different lines of sight [8]. This is how humans gain depth perception). Both of

these principles are impractical, especially for an electronic system of small size, and will therefore not be discussed further.

Laser range finding - LIDAR

LIDAR (from RADAR, but with light), is a method of measuring distance without contact, by illuminating some target, and analysing the reflected light. This is commonly done by measuring the time between emitting the light and receiving the reflection. There can of course be many reflections, but the interesting one is the "line of sight" reflection, which would also have the highest amplitude. Since the distance that is measured in this application is very short (down to centimeters) the receiver/detector should be very fast. A possible method is to emit light and let a capacitor charge until the reflection is received, as a way to measure the nanosecond TOF. Either way it requires sub nanosecond timing circuits. The distance D is then:

$$D = \frac{c \cdot t}{2} \qquad [m] \tag{3.6}$$

Where c is the speed of light, and t is the time measured (can be calculated from capacitor voltage) [9].

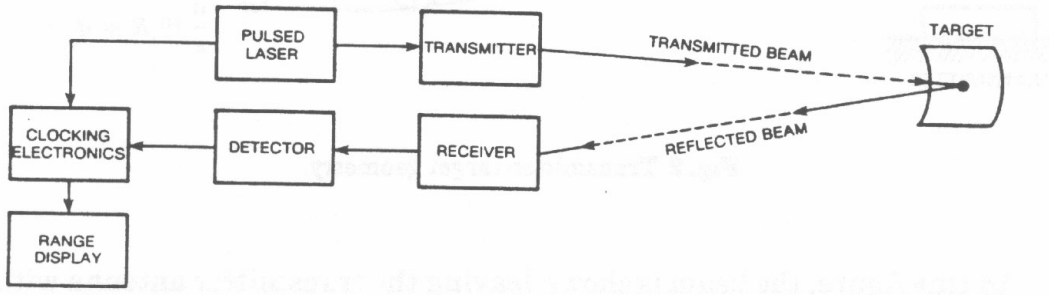


Figure 3.5: Block diagram of TOF measurement LIDAR. [10]

Another method is to current modulate the laser at some megahertz frequency (or higher), and then summing the emitted wave with the received (reflected) wave, and looking at the phase shift. From this we get:

$$t = \frac{\phi}{\omega} \quad [s] \quad \rightarrow \quad D = \frac{1}{2} \cdot \frac{c \cdot \phi}{\omega} \quad [m] \tag{3.7}$$

Where ϕ is the phase delay, and ω is the angular frequency. This can be rewritten to (by inserting the equation for wavelength):

$$D = \frac{\lambda}{4} \cdot (N + \Delta N) \qquad [m] \tag{3.8}$$

N is the integer number of wave half-cycles of the round-trip and ΔN the remaining fractional part, and λ is the wavelength of modulation.

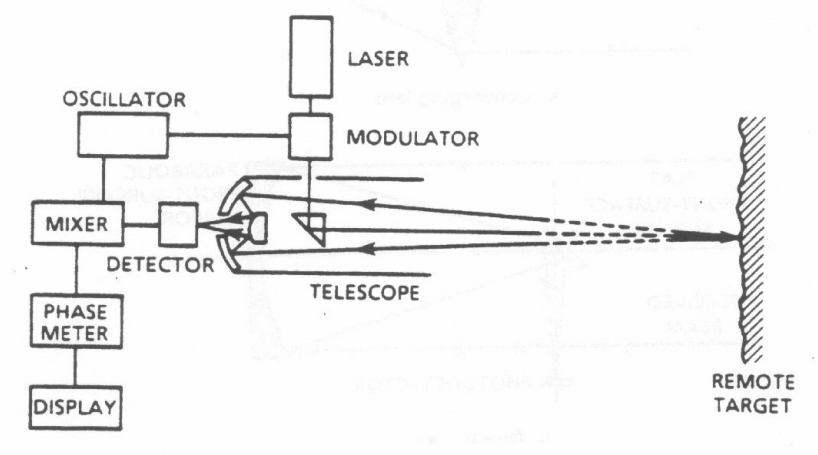


Figure 3.6: Blockdiagram of modulated LIDAR. This diagram shows a modulator after the laser, and some telescope. A simpler realization would be to modulate the current provided to the laser, and having the laser and detector with seperate optics. [10]

The disadvantages of this type of measurement are that it is sensitive to changes in ambient light (time of day) and temperature, depending on what kind of laser is used. But most important of all this technology requires a processor capable of very fast computation, and the hardware to current modulate the laser, and then phase detection, is quite complex. Another problem with laser is that in many cases it is not safe for the eyes. In order to avoid eye damage it should be kept at a low power level, and/or at a wavelength that does not damage the eyes as easily.

Radio range finding - RADAR

RADAR (Radio Detection and Ranging), used radio waves to determine range, velocity, angle and even size of objects [11]. RADAR is typically used for long ranges (compared to the distances to be measured by the quadcopter system), as aircraft altimeters, or to detect aircrafts, guided missiles, motor vehicles, et cetera.
In RADAR, some of the same techniques are used as in LIDAR. But other methods are also used. An interesting way to accomplish a distance measurement is by emitting a radio wave that has been frequency modulated, because frequency comparison was considerably more accurate with older electronics. By measuring the frequency of the returned signal and comparing it to the original, the difference is then translated to time when knowing the frequency modulation rate. This technique is however sensitive to the doppler effect, and uses much spectrum (which is scarse nowadays).
As with LIDAR, it is very hard to measure TOF at short distances, because of the wave propagating at the speed of light. If the frequency modulation scheme described above is used, the signal must be modulated very fast since light takes 3.33 nanoseconds to travel a meter. The modulation frequency should then be in the gigahertz's, and the carrier frequency should then be even higher. There are precision RADAR systems, like automotive RADAR operating at 77 Ghz [12].
Transceivers are nowadays produced as small CMOS ICs, using gallium arsenide semiconductor technology [13], making them much smaller than conventional RADAR. These are however very expensive and not really available to the consumer market.

Ultrasonic range finding - SONAR

The SONAR (Sonic Navigation and Ranging), or ultrasound rangefinders, work on many of the same principles as RADAR and LIDAR, except the transducers are speakers and microphones (or both), and they are transmitting sound waves, and receiving the reflection. So it is not electromagnetic waves, but pressure waves. Another huge difference is that the wave propagation is now the speed of sound, which is 340.29 m/s, compared to $3 \cdot 10^8$ m/s. This means that for an accuracy of 1 mm, we need to measure time intervals of 3.93 μS (which corresponds to approximately 250 kHz), and since most processors run in the megahertz clock speeds, it should not be a problem. Because of this, the most usual way to measure ultrasound distances is to emit a short pulse and then count the time until it returns, using a counter. Ultrasound transducers often come in the form of piezoelectric ceramic transducers, that work as both sender and receiver [14].
The advantages of ultrasonic range finding are: Relative low cost and complexity, low speed of wave propagation making it possible to measure with almost any microcomputer. In contrast to radio there are no real restrictions with respect to frequency use. There is also no interference, and only the noise made by scattering of the wave. Ultrasound is also relatively safe to use compared to laser. The UK Health Protection Agency recommend to limit exposure to 70 dB at 20 KHz and 100 dB at 25 KHz and above. Exposure to above 120 dB can cause hearing loss, 155 dB may cause harmful heating effects and 180 dB can be deadly [3].

Precision and range
There are several aspects to the precision and range of ultrasonic rangefinders. Precision is largely circuit dependent. Firstly there is the timer resolution, which directly corresponds to measurement resolution. Unless an analog phase detector is used. Then the resolution of the measurement corresponds to the resolution of the analog to digital converter after the phase detector.
The temperature also has an effect on the measurement, since the speed of sound is temperature (and humidity) dependent. Since we assume to make measurements in dry weather the humidity effect is disregarded. Since the temperature has a slow change rate (relative to quadcopter movement) this can also be disregarded, as the purpose is to keep a constant distance from the wall. When approaching there will be an error anyway due to the uncertainty zone. The uncertainty zone is the area close to the sensor where it is impossible to measure the distance because of transmitter reverberations, and the directivity of the sensors. Reverberations occur in the transmitting transducer (especially at the resonant frequency), giving the pulse transmitted a certain minimum length additional to the 25 μS which is 40 kHz period. During this time the receiver must be disabled to prevent a false range measurement. This affects the uncertainty zone, making it larger.
The maximum range of the sensor is decided by receiver sensitivity and SNR (Signal to Noise Ratio), because the maximum transmit power is limited (due to safety of the operators ears). But it is possible to enhance the directivity of the transducer and filter out noise, improving range. A ”horn” could be added to make the transmitter more directional, and shield the receiver from unwanted echoes and noise.

3.2.2 Choice of distance sensor technology

It is chosen to work with ultrasound as range finding technology since the short distance makes for very short measurement times if using RADAR or LIDAR, due to the speed of light. Another reason is price, availability and simplicity of ultrasonic transducers.

It is chosen to use a transceiver of the type that measures time of flight, because of its precision, and simplicity. The circuit for this must be designed.

3.3 How does the quadcopter work?

Quadcopter is an aircraft that uses four motors with four propellers to fly. Two motors move clockwise(M1 and M3 in the figure below and two counter clockwise M2 and M4). This is to stabilize it so it does not spin [15]. In our project we use the "X" configuration of the quadcopter. The representation of which can be seen on figure below.

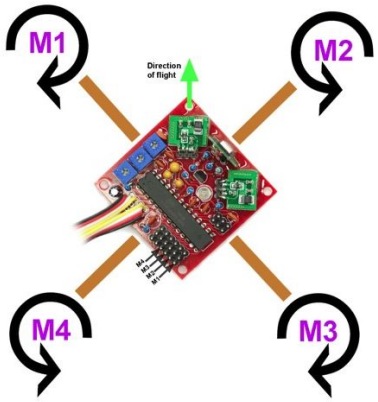


Figure 3.7: X configuration of the Quadcopter [16]

Torque of each motor is canceled by the motor rotating in opposite direction, that is why quadcopter does not need a stabilizing rotor like helicopters. The zero angular acceleration about the yaw axis can be obtained if all the rotors will spin at the same angular velocity. In the hovering state RPM of all four rotors is the same which allows it to keep stable altitude in the air. In order to move in the air quadcopter uses variable thrust between the four motors. To vertically ascent or descent we Increase or decrease RPM of all of the rotors simultaneously.
In order to make it easier to understand the maneuvers described below it is good to look at the picture attached below when reading about them.

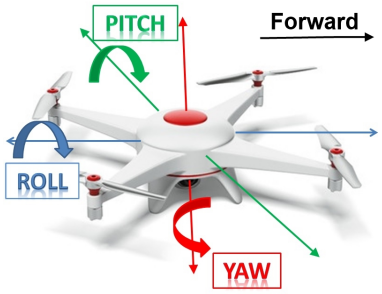


Figure 3.8: Visual representation of described maneuvers (the right side being the front, thus pitch being the tilt back and forth)[17].

To pitch forward we can make RPM of the rear motors(M3 and M4 on the first figure) greater than the RPM(Revolutions per minute) of the front motors(M1 and M2). Oppositely, when RPM of the rear rotors will be higher, the quadcopter will pitch backwards. In order to roll left or right we can make the RPM on the one side greater than on another(for example increasing the RPM of the rotors on the left side - M1 and M4- and decreasing the RPM on the right side-M3 and M2 would result in right roll) Another manouvre we can use is yaw, To make the quadcopter rotate in yaw direction we can increase or decrease the RPM of the opposing rotors, what will result in increasing or decreasing torque in that pairs direction. To illustrate it by example, if we increase the RPM of the rotors M4 and M2 while decreasing the RPM of M1 and M3 we will obtain right yaw. Opposite(increasing the RPM of M1 and M3 while decreasing the RPM of M2 and M4) will result in left yaw.

3.3.1 Further considerations

In order to keep a constant distance from a wall, it would be enough to control pitch, providing that the quadcopter is stable in yaw and roll direction, however it might prove difficult for the pilot to control the two other directions, since movements in these directions also affects the quadcopters perceived distance. It could be beneficial to also control yaw automatically, thus enabling the pilot to only move along the wall at a fixed distance (up, down and to the sides) using throttle and roll.
Since the quadcopter must tilt forwards or backwards to move to and from the wall, it is apparant that attaching distance sensors to the quadcopter pointing forwards will result in large distance errors when it is adjusting distance. To counter this it could be benificial to have a pitch sensor (either a three axis compass, a gyroscope, or an accelerometer). This measurement could then be used to compensate for the pitch change during distance adjustment. This will also help prevent the control loop from crashing the quadcopter (when moving away the distance sensor will look up, and the distance will be too long, so it moves closer by tilting forwards, again getting a too long measurement, and then it might continue forwards until it hits the wall).
The ”pitch compensation” could be done like follows (pseudo code):

```
static inline int pitch_compensate(int distance){
float pitch = get_pitch_angle();
pitch = fabs(pitch);
```

```
//to not get negative (since a triangle cannot have negative angle)
comp_factor = cos(pitch); // from cos = adj/hyp
pitchcompdist = distance * comp_factor;
return pitchcompdist;}
```

The new value will be shorter the larger the angle until 90 degrees where it will be zero. This should give an appropriate when pitching forwards or backwards.

Chapter 4

System Description

4.1 System Description

The system is divided into two major parts: One of the parts will be on the quadcopter, and the other one will be in the hands of the pilot as a remote control. The remote control will be responsible for reading the controls, sending the information to the quadcopter, and providing feedback to the operator from the quadcopter (status, etc.).
The part on the quadcopter will be responsible for sending the information received from the ground station, to the Pixhawk flight controller, and the Pixhawk, will in turn, control the engines. Mounted on the front of the quadcopter will also be two distance sensors, which are also part of the project.

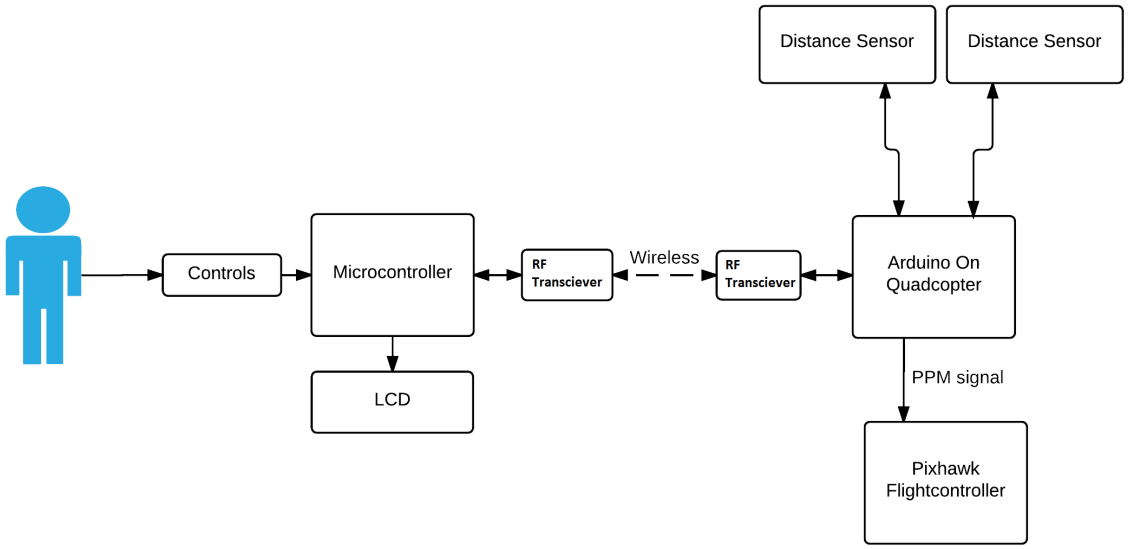


Figure 4.1: Blockdiagram of proposed system. The two main parts communicate with a RF transceiver.

Operation: An operator will use the controller to fly the quadcopter up close to the surface, which has to be measured (hull of ship). When within range of the hull, the pilot

will press a button to enable the controller, which will then take over the controls for yaw and pitch. This regulating algorithm will make the quadcopter stay flush to the surface, at a preset distance. Then the pilot can position the quadcopter and make measurements. The requirements for each of the parts are:

4.1.1 Remote Control

The controller will consist of a number of parts. These include an Atmel microprocessor, a LCD, some analog potentiometers for controls, RF transceiver, an emergency stop button, a buzzer and an encoder (for the settings menu). The main functions of this subsystem should be:

- Receive user inputs for yaw, pitch, throttle, roll.
- Receive user inputs for engaging control algorithm, emergency stop, etc.
- Provide the user with information about status of controls and quadcopter.
- Provide user with ability to adjust some settings (communication address, protocol, trim values for potentiometer adjustment, control algorithm gains, etc.
- Send the controls etc. to the quadcopter, and receive status and measurement wirelessly.

4.1.2 Network

This part of the system is the protocol which handles the communication between the controller and the quadcopter, and will have to be defined. The physical platform is two RF transceivers, on top of which the protocol will be implemented. There will be some differences between the quadcopter node and the remote control node, since information is mostly going in one direction. This should conform to following specifications:

- Real time control by user (latency less than 100 ms).
- Communication both ways without collision.
- Error detection.
- Ensure that the correct device is receiving the message.
- Should be connection oriented.
- Should take up little processor time.

4.1.3 Quadcopter subsystem

This subsystem must handle several things: Receiving controls from the user through the remote control, measure distance to wall, send status back to the remote control, run the automation algorithm (see below). This part will consist of a microcontroller, an RF transceiver, two distance sensor circuits, and he quadcopter itself (including its Pixhawk flight controller). The specifications are as follows:

- Sending control signals to Pixhawk flight controller with a low usage of processor time.

- Receive controls from controller, send status.
- Measure distance with ultrasonic transducers.
- Regulate the distance and yaw angle from the wall using distance measurements.
- Should indicate to the user its status (with LED's).

Regulator

This is a part of the quadcopter system, and will enable the quadcopter to stay at a fixed distance from the wall. It will be implemented as an algorithm that takes in the reference distance and sensor data, and outputs a part of the signal that goes to the Pixhawk and controls the engines (the yaw and pitch part).

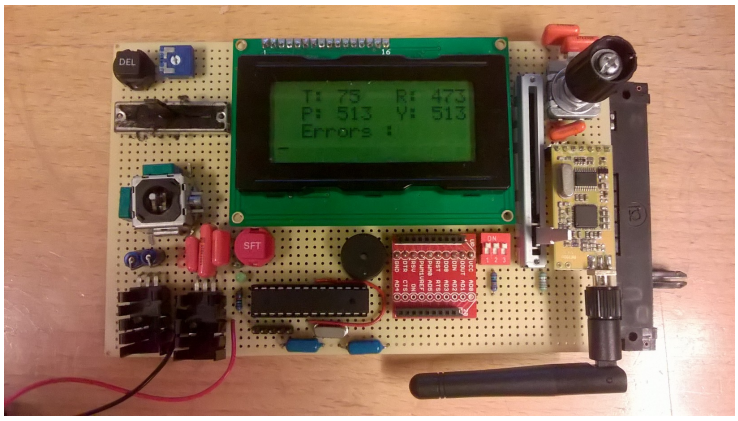
- Should be capable of overshoot less than 10 %
- Should be capable of Settling time of maximum 3 s
- Should be capable of rise time of 1 s

In the next parts of the report the design of these systems will be described. Each subsystem will also be tested by itself for functionality during the design (just to confirm it works). The system will be tested more in depth in the test section.

Chapter 5

Remote Control

5.1 Remote Control Of Quadcopter



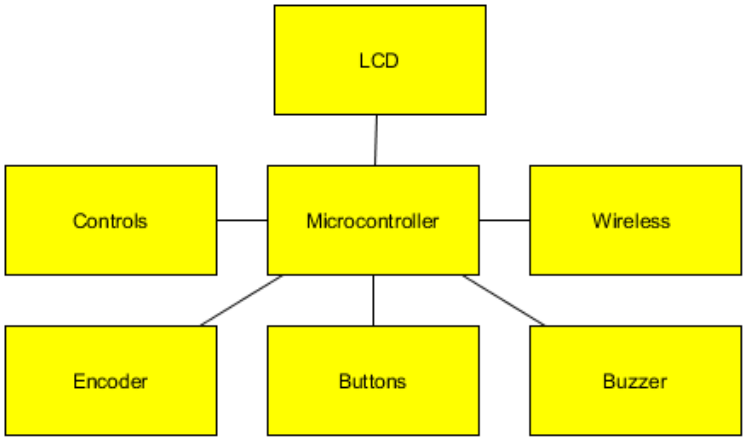


Figure 5.1: Block diagram of Quadcopter

To control the quadcopter is a remote control necessary. It was decided to design and build Remote Controller from a scratch as well as a communication protocol to know

exactly how our system reacts to inputs from the user. Another advantage of doing this, is it will be possible to add extra functionality. The operator should be able to change the values that can be send by the communication protocol. It should be easy and intuitive in usage and enable fast and precise change of quadcopter's position with ability to quickly cut the power in case of emergency situation and giving the user info about its status.

5.1.1 Functionality and Requirements

Apart from the main function, which is receiving user's input for throttle, pitch, roll and yaw and sending it to Arduino on quadcopter, RC should also be able to: respond to input for establishing the connection, attach to the wall and detach after finishing. Furthermore, it must react to emergency input to safely land the quadcopter in case something fails and respond to step input, which is required for a modelling part.
Additional features enable to change RC address, quadcopter address and save them to EEPROM. Furthermore in order to compensate for drifting of the quadcopter while flying we added trimmers for pitch, roll and yaw, that are also saved to EEPROM.
Besides that our RC will provide user with a distance from the wall and battery level. That will be displayed on LCD.

5.1.2 Software

On the remote controller, it was necessary to use Jens Dalsgaard's scheduler, which is an Arduino Uno scheduler. This makes it possible to run several tasks in parallel, which is controls, UI and Networking.

UI

GUI has been done using LCD. It displays information for user and allows him to change some settings. Moving across the graphic interface user uses rotary encoder. This is visible to him as a position of a hyphen on the LCD. After the user starts the RC he sees main menu that provides him with controls value. From there he can go to menu by pressing the encoder. Also in any other case, if user wants to choose a feature or enter a setting it is done by moving the hyphen to specific location and pressing the encoder. While in the menu it is possible to choose between Status, Setup and Trimmers. Status provides us with possibility to change the distance from the wall printed in cm. Setup allows to search for the quadcopter address and change it together with Remote controller and Emergency address that is used to stop the quad copter. In 'Trimmers' user can adjust pitch, roll and yaw so that quadcopter won't drift.
The part with moving across UI was was implemented using boolean variables that were changing their values each time we enter or exit a setting/feature.

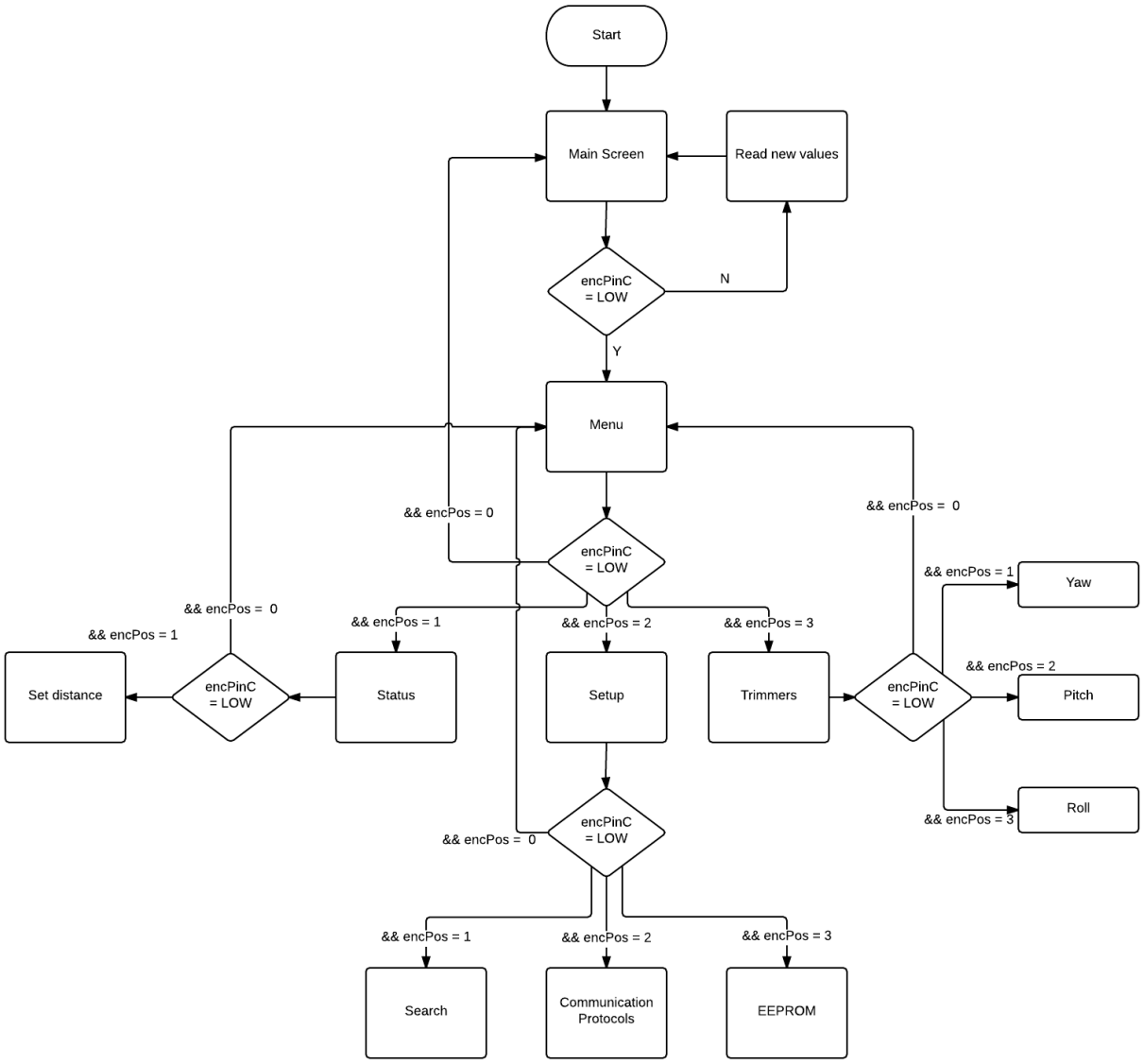


Figure 5.2: Block diagram for UI

Controls

Receiving user's input is done by implementing four potentiometers (throttle, yaw, roll and pitch). We measure the voltage with the built in ADC on the microcontroller. The reference on the ADC is 5 volts, and the resolution is 10 bits (from 0 to 1023). On pitch, roll and yaw controls was it nessercary to implement a dead zone in the middle, because of how the potentiometer is built. It was due to the fact, that after changing one of the control and trying to put it back to the middle, the values from the ADC weren't always the same. The obtained numbers are then packed and send in packages to arduino on quadcopter, and thereafter sent to the Pixhawk.

Emergency button

In case of emergency will the emergency button be pressed. Pressing the emergency button will order the controller to compose and transmit a package to Arduino on the quadcopter that orders the Pixhawk to change its flight mode to landing. That affects in constant pulling down the throttle until the engines stop. Also after the button is pressed, none controls will be used until the Arduino on the quadcopter is reset.

Controller enabling button

Pressing the button for enabling controller will make the Arduino on the quadcopter to take over the pitch and yaw controls, using a Proportional and Proportional-Derivative controller provided with distance from the wall measured by ultrasonic sensors and comparing it with reference distance set by the user. Roll and throttle will continually be controlled by the operator.

5.1.3 Hardware on the Remote Controller

The remote controller has been implemented on a perfboard, and assembled by hand. The remote controller consists of following items:

- Atmel atmega328P microprocessor
- Encoder
- APC220
- LCD
- Buzzer
- Emergency button
- Controller button
- Controls, implemented as potentiometers

Also there are a few miscellaneous components, like regulators, resistors and capacitors.

5.2 Multitasking

Multitasking is dividing a program into tasks that operate independently of one another. KRNL is used to be able to do multitask programming with an Arduino UNO. KRNL, which is an Arduino library made by Jens Dalsgaard Nielsen(hereafter known as JDN), is a preemptive real-time kernel. For the Arduino DUE there's a library called Scheduler, which allows the DUE to run multiple functions at the same time, without tasks interrupting each other. The Scheduler is a cooperative, or non-preemptive, multitasker. By sharing the CPU time between the tasks it can perform parallel programming by allocating to each task a portion of the available CPU time. Contrary to the Scheduler the KRNL is a real-time preemptive multitasker which stops the execution of a running program by the hardware interrupt system, allowing another program to access the CPU. By assigning priorities to the processes/tasks, a task with a higher priority can interrupt a task with a lower priority, which guarantees CPU time for tasks that, for example,

are getting external incoming data. So you have two kinds of processes; those that are waiting for input/output, higher prioritization, and those that are constantly running, lower prioritization, fully utilizing the CPU [18]. Below is shown an example illustrating three modes of processing with 3 tasks at hand; reading the mail, answering the phone and answering the door.

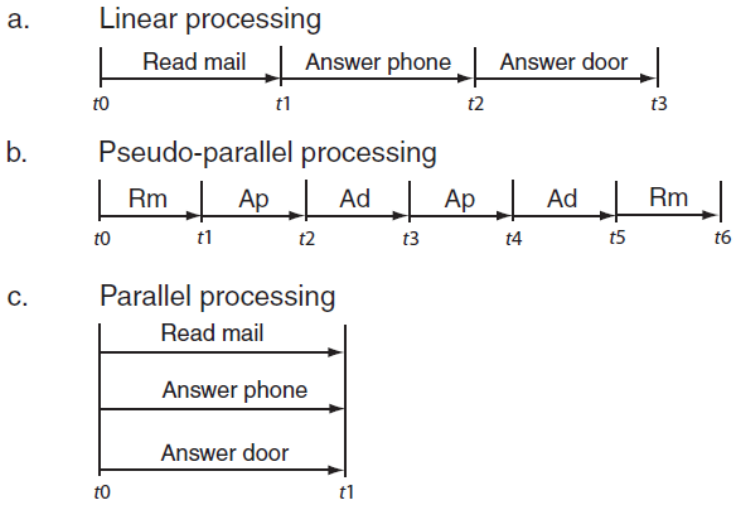


Figure 5.3: Three modes of processing.[18]

The pseudo-parallel is simply like the linear processing, but the tasks switch fast, even during execution. This is the way our system is processing.

5.2.1 Multitasking on the Remote Controller

In this project, JDN's KRNL is used in the remote controller. On the remote controller are three tasks, these are responsible for the following things: Sending Controls, User interface and Incoming messages. Two of these tasks will run periodically, and the last will run when the first two are not running. This will give a CPU utilization of 100%

Sending controls

This task will read the values from the controls on the controller. If any of these values have changed, they will be compiled into a package, and transmitted to the quadcopter. It has been decided to give this task the highest priority, as it will need to run atleast once every 50 ms. This will very likely interrupt the other lower priority tasks.

Incoming messages

This task will check if any messages have been received on the controller. If a message is intended for the controller, it will act upon it. This task it not very demanding, as very few messages are intended for the remote controller. This task has been given middle priority, because of it's quick execution time.

User Interface

Relative to the other tasks, this task takes up a lot of CPU time. This is because, even when it's not running, it will check if the encoder has been moved. The reason the CPU

has to check if it has moved, is because of the design choice of not connecting the encoder to an interrupt pin. This task has been given the lowest priority, because it will barely run while flying, and that's when the other two task run the most.

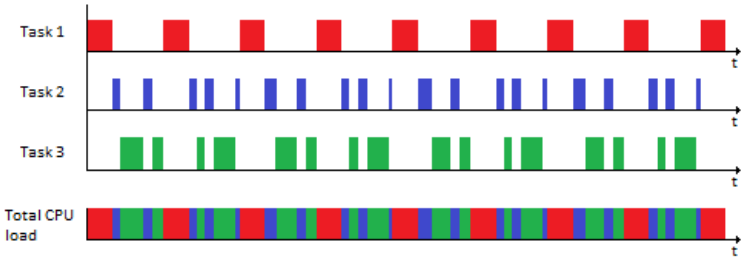


Figure 5.4: Time sharing on the CPU on the RC while transmitting [18]

Chapter 6

Communication Link

6.1 Communication Link

In order for the remote control to send information to the quadcopter, a protocol is needed. This part is a description of how the communication protocol is designed, and how it is implemented on the two micro controllers. The protocol is build on top of a physical layer which in this project is provided by a system called APC220, but it could also be implemented with different wireless technologies. This protocol features some of the same functionality as the radio module does, for example it will allow addressing and error detection. The reason for the duplicated functionally, is because this protocol is made in a way, so it can run on different physical layers, and also many APCs with the same address would be able to work using our addressing.

APC220

The APC220 is a cheap two way radio module. This module has forward error correction, and can service more than 100 channels. It's working frequency is between 431 Mhz and 476 Mhz, and will send with a output power of 20 mW. This radio module will allow a maximum airspeed of 19200 bps.

6.1.1 Overview of the protocol

This protocol will be used to communicate between the quadcopter, remote control, and other relevant equipment, is a connectionless packed based protocol. It's assumed that all devices on the network run a compatible version of the protocol. Every packet has this basic structure:

Datastructure on packages	
Byte No	Type
1	Address of reciver
2	Type and length
3 - 17	Data
18	Parity byte

Each device has at least one address, and any device will only act on packets intended for aforementioned device.

The first byte contains the address. The second byte is divided into two parts. The first 4 bits specify the type of data, witch this packet contains, and the last four bits specify how many bytes are until the parity byte. The length and parity byte are used to check for errors under transmission.

6.1.2 Addressing

The addressing on this protocol is rather simple. All devices have at least one address. If a packet has received, and the address doesn't match its own address, the packet will be disregarded. The address for the individual unit, is stored on the build in EEPROM. During start up, the device will ping it's own address, to check if it's in use. If it's in use the device will increment its address by one, and try again until it finds a available address. The address may also be changed by sending a "Setting agreement package".

A reason a unit may want to have more than one address, is in case an emergency message has to be sent. The quadcopter is implemented with two addresses, one is for reliable data transfer, and the other is for high priority messages. If a message is received on the emergency channel all error checking is disabled. The controller will also be disabled, and the controls and receive tasks will get the highest priority. This will make the operator take over all controls, and be able to land the quadcopter safely. If a message is received on the standard channel, the packet might be processed at a later time, because of the lower priority.

6.1.3 Type

The next byte in the packet specifies what the contents of the packet is. There are four bits available for this, and that limits the number of types to 16.
These types have been defined to be:

Type	**Meaning**	**Lenght**
0	Controls - Yaw	2
1	Controls - Throttle, Pitch and Roll	6
2	Controls - Pitch and Roll	4
3	Controls - Throttle	2
4	Undefined	-
5	Undefined	-
6	Undefined	-
7	Sensor Data	1 - 4
8	Emergency Stop	4
9	Battery Level	1 - 2
A	Change Controller Reference	1
B	Toggle Controller	1
C	Ping / Echo	2
E	Nullify setting agreement	1
F	Setting Agreement	3 - 6

The first four types are dedicated for controls. The reason the controls are paired in such a way, is because it's very unlikely that all the controls will be changed at the same time. So to save CPU and radio time, it has been decided that most of the time, only either throttle or pitch and roll will be changed, and thus only that will be sent. The reason

it's not possible to only send either pitch or roll individually, is because both numbers are read from the same joystick, and thus it's very difficult to only change one, and not the other. A control package will be sent when the controls have changed.
The 0100 packet will be sent once in a while, and will act like a "heartbeat" packet, to keep the quadcopter "alive". The time between these packets will be described in the congestion control section. If a heartbeat packet is not received for some time, it's assumed that the remote control malfunctioned, or was turned off. This will decrease the throttle steadily to 0, to make the quadcopter stop.

6.1.4 Error detection

A parity bit is a bit added to the end of a string of binary code that indicates whether the number of bits in the string with the value one is even or odd.
The way error detection is implemented in our protocol, is in the form of a parity byte. The parity byte is made up by eight parity bits, each made by performing the XOR operation on every eight bit before it this package, this is done eight times, once for every colon. The length in the second byte in the message specifies where the parity byte is.
This is an example of a throttle package, where the throttle is at position 1000(DEC), and how the parity bit is made:

Number	Type	Data [Bin]	[Dec]	
1	Address	0100 0000	100	
2	Type/Lenght	0011 0010	3, 2	
3	D1	0000 0011	3	0
4	D2	1110 1000	232	1
5	Parity	1101 1011	219	2

The first bit in the parity byte is composed of this binary string "0001"(Shown in red). Since this string has a odd number of ones, the parity bit is 1. This is called even parity.
The way this is implemented, is when the sender composes the package, it saves the data in a char[uint8] array. And for every byte saved in the array, the type/length byte increments. One of the problems with this protocol is, that the length of the data must be less than 16 bytes.
When the contents of the package is written in the array, this function will be executed, this function makes the parity byte, the value this function returns is added to the end of the package, and the package will be sent.

```
char Parity_byte(int lenght)
{
  char temp_byte = DT_SND[0] ^ DT_SND[1];
  for (int i = 2; i > lenght; i++)
  {
    temp_byte = temp_byte ^ DT_SND[i];
  }
  return(temp_byte);
}
```

The "^ " is a byte wise XOR operator. The first two bytes XOR'ed together make a temporary byte, witch all following byes will be XOR'ed to.

When the receiver gets the message, the same piece of code runs. If the parity byte made by the receiver, is the same as the transmitter sent, will it be assumed that the package was received unharmed, and can be used. If the parity bytes don't match, the package is discarded, and a error message will be sent back to the original transmitter. One of the weaknesses of this protocol, is that the transmitter's address is not in the package. This is because it will require an extra byte to be sent with every package, and instead all error messages will be sent to a address determined in a package called a "Setting agreement".

6.1.5 Processing of received packages

After a packet is received, it has to be processed. The first byte is read from the serial buffer, and states the intended receiver. If the package is not intended for this receiver, it will be disregarded. If the package is intended, the next byte will be read. This byte contains the type and length of the package. To get the two variables out of one byte, the following code will be run. This piece of the program reads the variable, and shifts it to the right four times and saves the value as the type, thus only leaving the first 4 bits in the original byte. The length is found by shifting the part of the message containing the 'type' out of existence, and then shifting the 'length' back where it's supposed to be.

```
type_length = Serial.read();
char type = type_length >> 4;
char length = type_length << 4;
length = length >> 4;
```

The variable 'type' is used in a switch, and each case can handle a curtain type of data. This handling means that the data will be saved, but on acted upon yet. After the data has been handled, the XOR operation be used on all previous bytes, this value will be compared to the byte at the address; length. If these bytes are the same, the message will be acted upon. This means using the data earlier stored. If the bytes don't match, the message will be disregarded. This is a flow diagram showing the whole process:

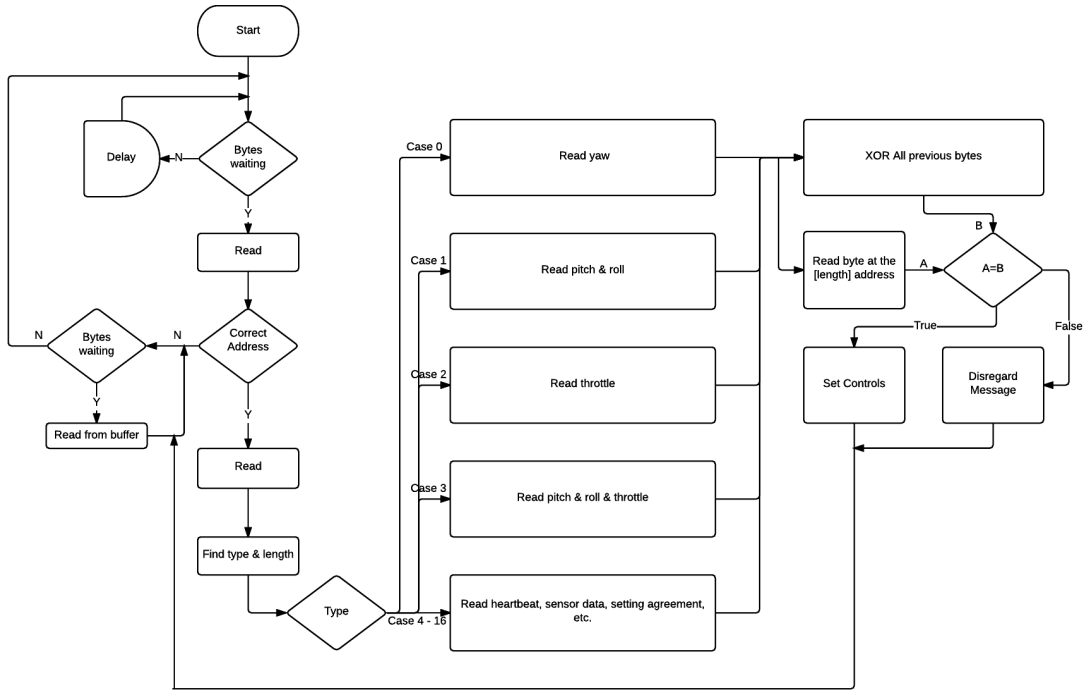


Figure 6.1: Flow diagram of message handler

The function of the red box is to empty the serial buffer, if there's anything in it.

6.1.6 Other functions

There are a few less significant functions in this protocol, these will be described here.

Setting agreement

A setting agreement, is a way for two devices to exchange settings between them. This is implemented as a special type of packet. This packet contains information, like what address to send errors to, what address should be used for the emergency channel. Before proposing a change of address, a ping will be sent to this address, to prevent a conflict
This package will also contain some values for the controller, like proportional, integral and derivative gain.
The settings in a Setting agreement will only be implemented if the receiver sends an acknowledgement plus a status package.

Status

A status package contains information about the state of the current system. This information may be sensor data or battery level.

Ping

The purpose of a ping package is to check if a any device on the network has a specific address. It may also be used for measuring round trip time on the network. A ping

packet contains the receiver and transmitter address. If a device receives a ping package, an echo will be sent back to the initial transmitter, with the address it sent from.

Search

The search function sends a ping out to all addresses on the network. If it receives a echo, it will inform the operator with it's address.

Over congestion

If the read buffer on the receiver overflows, a message will be sent to the transmitter. This message will increase the pause between messages transmitted from the transmitter. It will also increase how much the controls have to change before the package has to be resent.

Error in parity

If the parity witch is sent with the message doesn't match the one made by the receiver, an error message will be sent. This error message will do the same as the over congestion message, except if it's an important message. If the message is broken, the original transmitter will resend the package, after a random amount of time.

6.1.7 Congestion control

Overloading the system, which causes delay and loss that degrades the performance, is called congestion. Congestion control (algorithms) are used to prevent a congestion collapse from happening. Specifically the Traffic Throttling is used in this case, where the sender controls/adjusts the transmission to send as much data as the receiver can handle. The receiver's (called network in the figure below) aim is to operate just before the onset of congestion. When congestion is about to occur the receiver tells the sender to throttle down the transmission and slow down (and after an interval the sender will increase the transmission again, until the receiver tells the sender to throttle down again). This operating point is sometimes referred to as "congestion avoidance" in contrast to the situation where the network has become congested.

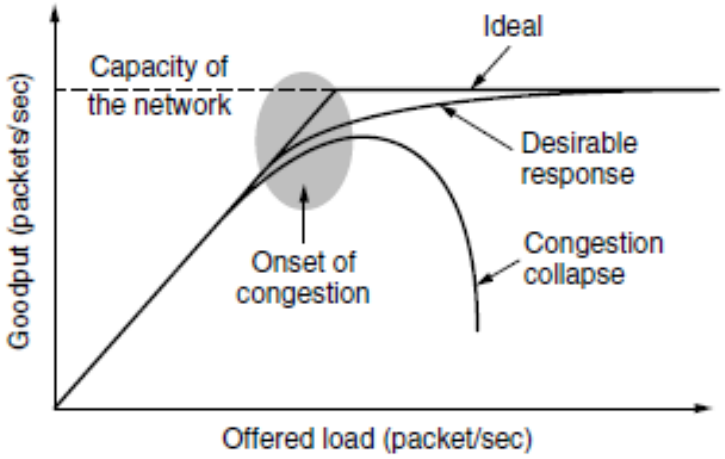


Figure 6.2: Performance drops acutely with too much traffic. [19]

The Traffic Throttling is one of the faster congestion controls, as shown in the figure below.

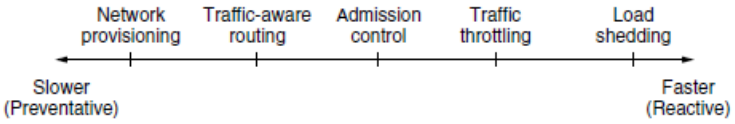


Figure 6.3: Different approaches of congestion control in timescales.[19]

Traffic Throttling is implemented by decreasing the time between each heartbeat package and decreasing how much the controls have to change before the package has to be resent.

Chapter 7

Quadcopter

7.1 Quadcopter systems

The quadcopter is the main part of this project. It has to fly toward the hull of a ship, then stay at a constant distance to it while making a measurement (again, the measurement of ship hull thickness is omitted in this project, only the flight is designed). In order to do this it needs some different tasks accomplished, like measuring the distance to the hull, flying accordingly, and receiving controls from the pilot. This is handled by a number of subsystems.
This part is about those subsystems that will be integrated on the quadcopter. These are:

- Microcontroller with: control algorithms, communication protocol (described previous section), PPM generation, sensor algorithms.
- Sensors: 2 x Distance and pitch.
- RF module.
- Connector to Pixhawk flight-controller (FC).

The quadcopter itself was assembled from parts, and includes: four engines, four propellers, four engine controllers, a power distributor, battery, Pixhawk flight-controller, frame. Mounts for sensors were 3D printed, and a board for the components was made, which can be plugged into the Arduino Due board. The microcontroller and peripherals are powered from the main battery which is a LiPo, and is the same one that powers the engines and pixhawk.

The Arduino Due was chosen as the main microcontroller on the quadcopter, because of its fast processor, and its many IO ports. It has the SAM3X8E ARM architecture (A simple reduced instruction set computer, RISC) CPU, which is 32-bit, and uses a 84 MHz clock, making it rather fast compared to the Uno which is 8-bit and has a 16 MHz clock.

7.1.1 Multitasking on QuadCopter

On the quadcopter are three main tasks, these are: Networking, Controller and PPM generator. These tasks are much more demanding than on the remote controller, and that's why a much more powerful micro controller has be chosen. One disadvantage of the micro controller which was chosen, is that it cannot run JDN's KRNL. Instead a mix of a built in library called Scheduler, and some predetermined timing, a custom multitasking program has been developed.

PPM Generator

This task has to run every 20 ms, failure to do so, will cause the quadcopter to crash. The runtime of this task varies a lot, and is all depended on what controls are sent to the Pixhawk. The minimum execution time of the PPM task is 6 ms and maximum is 9 ms.

Networking

This task is similar to the Incoming Messages task in the remote controller. This task is responsible for handling any packages received from the remote control. It is quite quick to execute, and thus barely affects the rest of the scheduling.

Controller

This task runs when the controller is on. The task is responsible for reading the sensor values, apply a filter to them, compare them to the reference and calculate a value to output. It takes some time for the distance sensor to measure the distance, the maximum time is 10 ms each, thus making the execution time quite long.

Execution Schedule

When the controller is on, it is necessary to execute the code according to a predetermined schedule, to ensure stability. This schedule revolves around the PPM task, as it's the most important. When waiting for the first distance sensor to return the distance, the PPM signal will be made. Then the second distance sensor will be turn on, and while we are waiting for a response; network, Pitch compensator and the controller will calculate new values for Yaw and Pitch. All this is done, while the second sensor is measuring, after it's finished, the PPM signal will be made again, and thus the code loops.

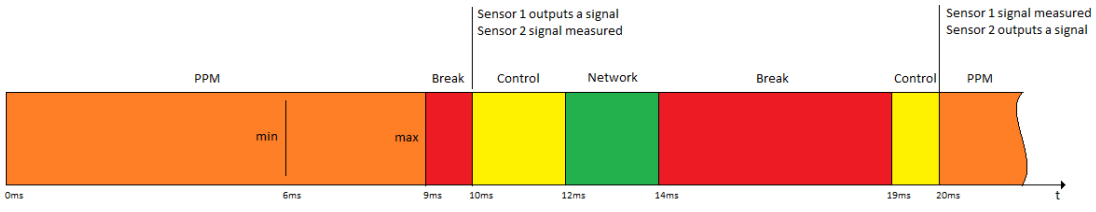


Figure 7.1: Timing diagram for quadcopter while controller is running

In the next section, the design of the quadcopter subsystems will be described, starting with the PPM signal generation, and then the sensors, and finally the control algorithms.

7.2 Pulse Position Modulation

PPM (pulse position modulation) is the type of modulation that we use to send the message from Arduino on Quadrocopter to Pixhawk flightcontroller. This type of modulation works by sending pulses (electrical/electromagnetic/optical) to communicate simple data in our case the control values from the remote control (both set by the pilot, or by the control algorithm). First pulse is the reference pulse, from it the rest of the pulses are set relatively to the pulse occurring before, placing them proportionally to change in the controls. This is done by decoding received messages from the RC and translating them into time domain. To make it easier to visualize this principle we can look at the figure below.

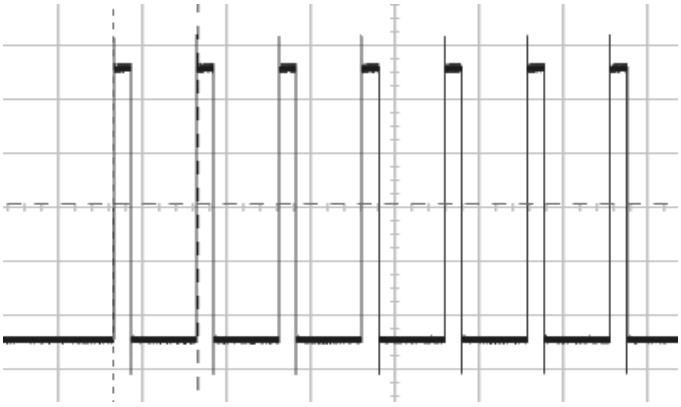


Figure 7.2: PPM signal given to Pixhawk FC, with all controls at zero. Picture from Agilent Oscilloscope.

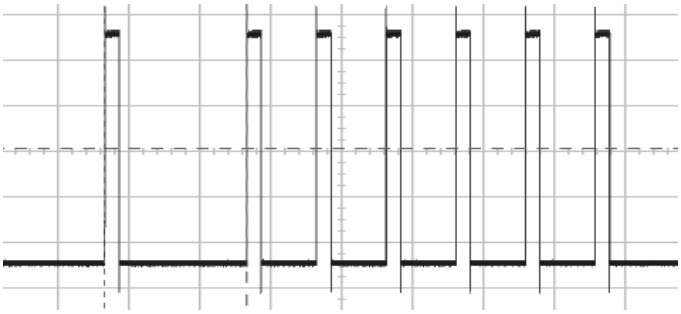


Figure 7.3: PPM signal given to Pixhawk FC, with first control value set to maximum. Picture from Agilent Oscilloscope.

One of the main advantages of the PPM modulation is that it communicates only simple commands and is good to use in lightweight applications, because of the low system requirements. Since we need to send only very simple commands it is suitable for our purposes.
In the Arduino Due on the quadcopter, the values are received by the communication interface as integers (or set by the controller algorithm). The first pulse is then set for ”pulselength” time, and then there is a loop that sets the rest of the pulses. This loop

waits for ”framelength - pulselength + controlvalue”, and then sets the next pulse. This will decide the pulse position, and at the same time ”move” the next pulses by the same amount. This means the total length of the PPM signal will be longer the higher the control values. This has to be repeated every 20 ms, and runs in its own scheduler loop.

7.3 Distance Sensor

In the analysis it was chosen to make the distance sensor using ultrasonic range finding, and using the transducers MA40S4-S/R sender and receiver (S and R). We shall now design a system around these that can make the measurement. This system should include a microcomputer which initiates the measurement, counts the time, and calculates the distance. In order to interface the microcomputer with the transducers, some circuitry is necessary.

7.3.1 Specifications

In order to design the distance sensor, it is necessary to know some parameters: The transmitter sensitivity and frequency response, the spectrum and magnitude of the engine noise, the decay time of the transmitter when sending a pulse at the operating frequency. The last one is about the short range of the distance sensor. It will take some time for the ultrasonic speaker to stop vibrating, and until it stops we cannot measure echo because we will be receiving the signal directly from the speaker. Therefore we have to know how long to turn of the receiver after while transmitting, and after transmission has stopped. Only after this interval can we detect the echo.
NB!These measurements are described in to measurement reports in the appendix 1 and 2, but the main results are summarized here:

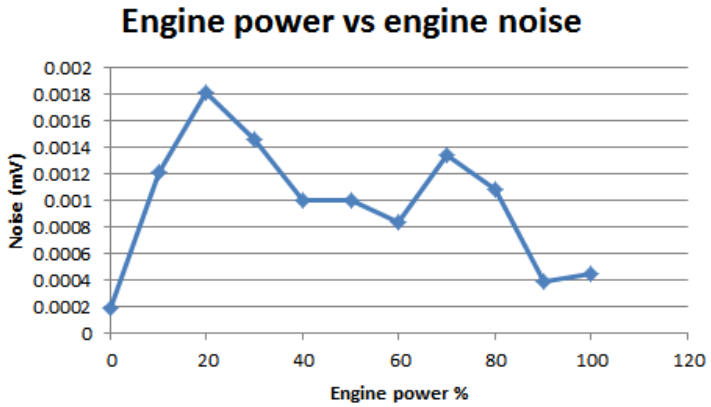


Figure 7.4: Engine noise at 40 kHz vs engine level as seen by sensor circuit.

From this measurement the echo detection threshold can be set. The spectrum of the noise was also measured:

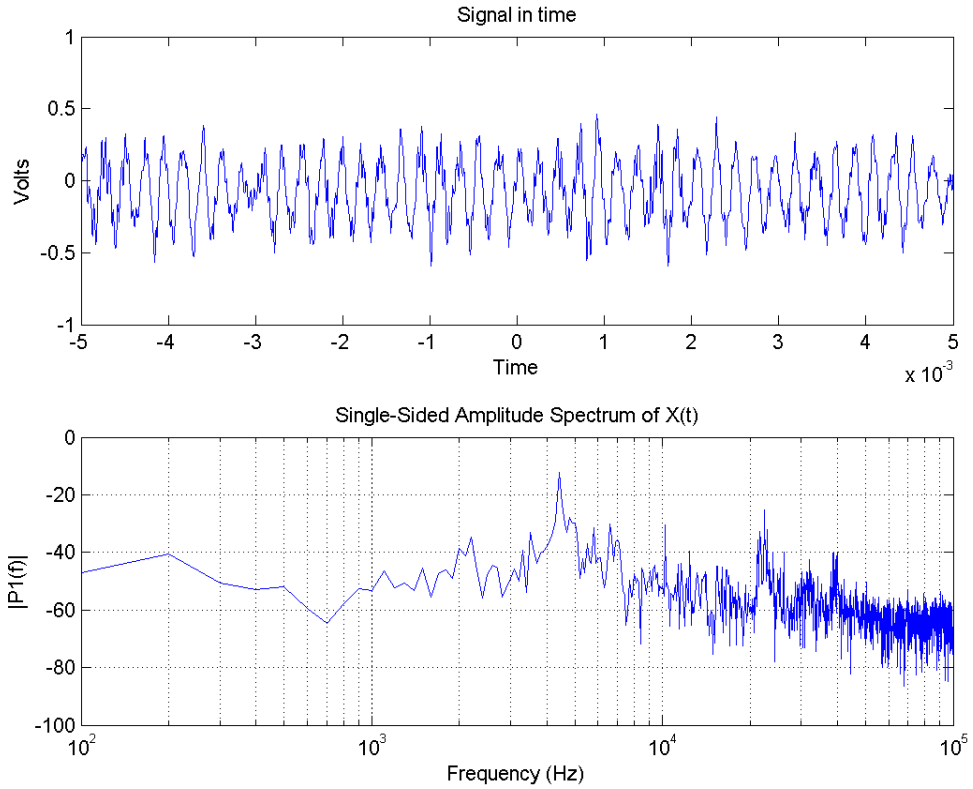


Figure 7.5: Recorded engine noise spectrum.

From this measurement it was decided that a filter will improve the receiver significantly.

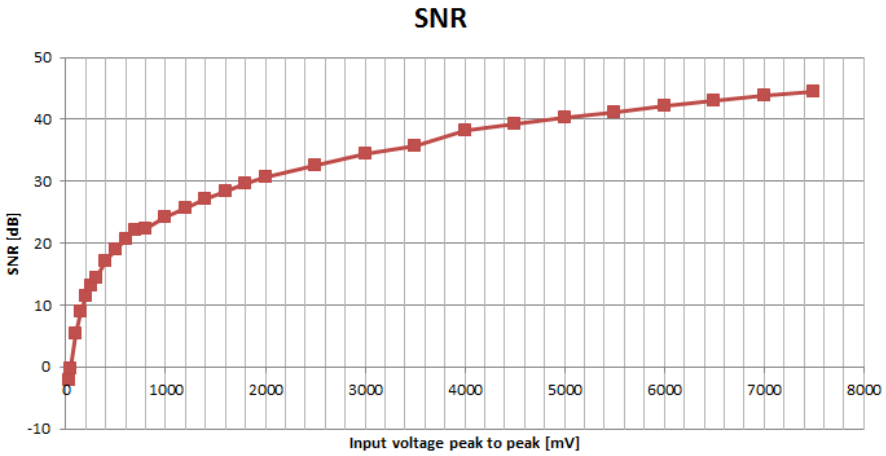


Figure 7.6: Signal to engine noise ratio for different transmitting levels.

From this measurement a sufficient transmission voltage can be chosen.

From the characteristics of the transducers that were measured in the anechoic chamber, and the measured engine noise, some specifications can be put forth:

- The operating frequency should be 40 kHz (This is the transducer response center frequency)
- The transmitter circuit should supply the ultrasonic speaker with 40 kHz wave of at least 2.5 V peak to peak and maximum 5 V, square or sine
- The receiving circuit should have a gain of 60 dB at 40 kHz, and a response characteristic of a BP filter with center frequency 40 kHz, and bandwidth of 2 kHz (to reduce noise).
- The receiving circuit should be able to detect the echo and output a logic HIGH to an arduino microcomputer
- The transmitter should be controllable by microcomputer, and the blanking time for receiving the echo should be around 1.2 ms (to eliminate false positive). This can be tuned in later.
- The receiver should be able to handle an SNR of 32.5 dB without false triggering.
- The arduino microcomputer should be able to measure time with a reasonable speed (CPU clock must be much faster than signal frequency), and the distance measurement should be made as efficient as possible.

Another important parameter is the ”sampling time” or how often the distance is measured. The strategy is to make the distance measurement take as little time as possible, and then afterwards, when designing the control system, it can be chosen how often the function to measure distance should be called.
From these specifications the subsystem can be designed:

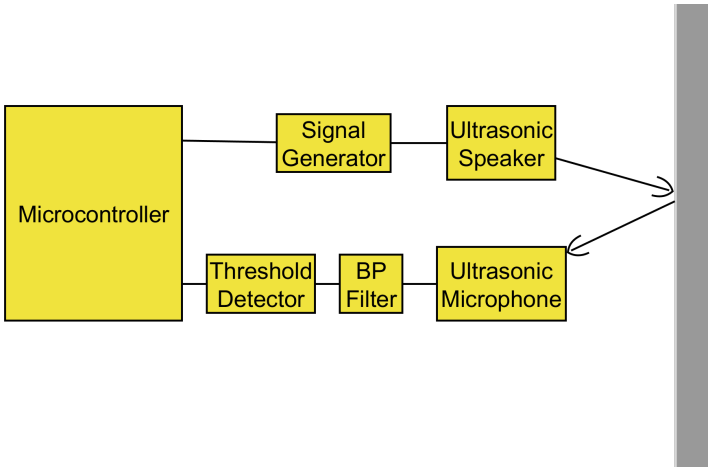


Figure 7.7: Blockdiagram of distance measurement circuit (computing algorithm not included). The gray block represents the wall. The lines represent signal path, and the arrows are the ultrasonic sound waves.

The reason of the analog bandpass filter is that the ADC in the Arduino microcomputer has a sampling frequency of 15 kHz, which is too slow, and even if an external ADC was used, the microcomputers resources are too scarse to handle the samples (in case a digital filter should be implemented). The speed of the microcomputer is also the reason an external oscillator circuit is used for signal generation. The arduino cannot generate a reliable 40 kHz signal with 50% duty cycle.

7.3.2 Hardware

The hardware part of the distance sensor contains the transmitter circuit (40 kHz generator), and the receiver circuit (filter, amplifier, detector).

40 kHz Generator

It was found that, for the prototype, the easiest and smallest option is to use a VCO (voltage controlled oscillator) IC, and then use a transistor to turn off output until it is needed (by sinking the signal into ground instead of the speaker). The frequency is chosen to be 40 kHz, and duty cycle is set to 50%, by choice of resistor and capacitor, and voltage on pin 5 (see circuit).

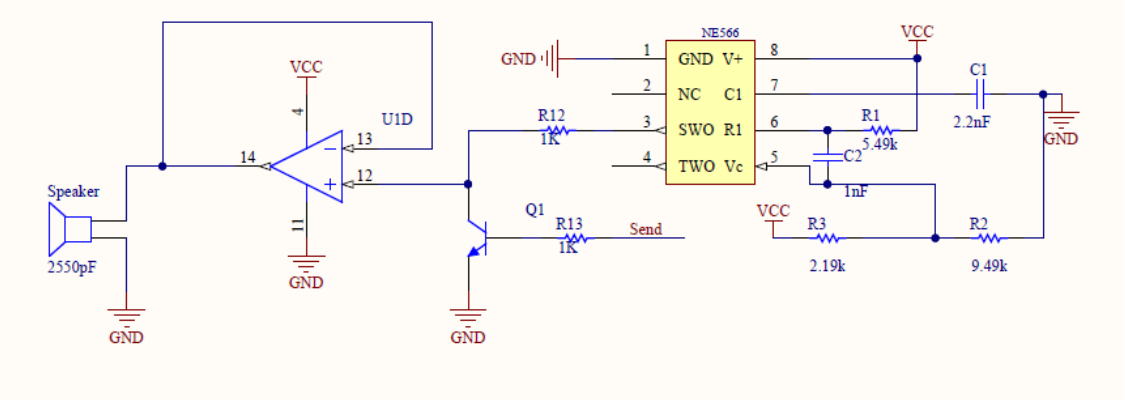


Figure 7.8: VCO circuit.

The buffer in the circuit is there mainly because there was a spare opamp (from a chip with 4 in it), and we were unsure how the VCO would handle the capacitive load of the speaker. A resistor was added in parallel to the speaker later, because the opamp had difficulty driving it, this is due to the fact that a speaker has a capacitive load.

Receiving Filter and amplifier

The filter for the receiver should be a bandpass filter with 40 kHz center frequency, and reasonably low bandwidth. The bandwidth is chosen from the knowledge that the MA40S4S has some variance in center frequency from production, and thus the receiving filter should accommodate that (to avoid having to manually tune it. Also the order of the filter should be reasonable, so that the filter can be implemented in a single stage biquad circuit, thus saving space.
So the requirements are:

- Center frequency f_c should be 40 kHz

- $-3dB$ Bandwidth should be 2 kHz
- Order should be 2 (max)

From this a filter will be designed:

Bandpass filter

A bandpass biquad is described with a transfer function of he form:

$$H_{BP}(s) = \pm \frac{G \cdot b_1 \cdot s}{s^2 + b_1 \cdot s + b_0} \tag{7.1}$$

Where G is the DC gain (the level at the max of the filter, which would be 0dB for a passive filter), and b's are coefficients. The transfer function is also sometimes shown as:

$$H_{BP}(s) = \pm \frac{G \cdot \frac{\omega_0}{Q} \cdot s}{s^2 + \frac{\omega_0}{Q} \cdot s + \omega_0^2} \tag{7.2}$$

Q is defined as the inverse of the fractional bandwidth and should thus be:

$$Q = \left(\frac{2000}{40000}\right)^{-1} = 20 \tag{7.3}$$

If we choose a gain of 1, and insert it in the transfer function we get:

$$H_{BP}(s) = \frac{12566.37062 \cdot s}{s^2 + 12566.37062 \cdot s + 6.316546817e10} \tag{7.4}$$

Plotting this in Matlab gives:

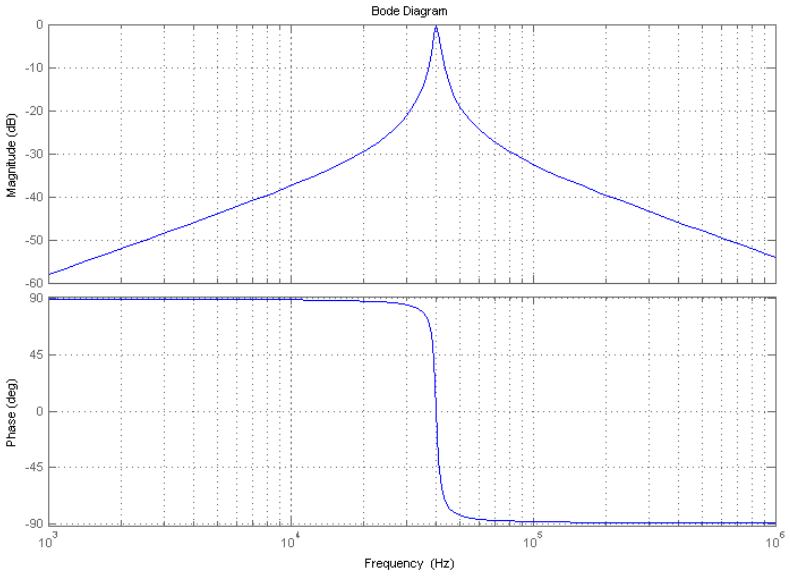


Figure 7.9: Bodeplot of filter characteristic in Matlab.

This fits with the specifications perfectly. It is possible to add gain to the filter, which would mean we need less gain in the next stage (amplification stage).
It seems reasonably stable. It has a zero in origo, and a conjugated pole pair, which are far enough in the negative half pane of the real axis.

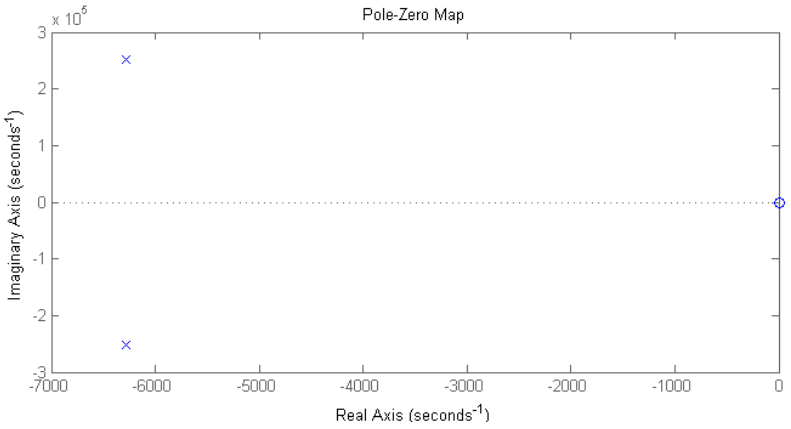


Figure 7.10: Pole Zero plot of filter transfer function.

Implementation

The Filter implementation we chose is an infinite gain multiple feedback filter because it is generally less affected by component tolerances.

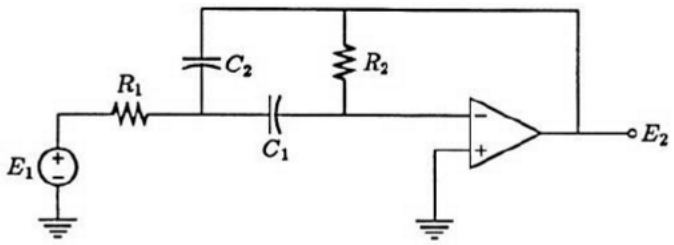


Figure 7.11: Filter topology. Source: Kendell Su

The transfer function for this filter is:

$$\frac{V_o}{V_i} = -\frac{\frac{1}{R_1 \cdot C_1} \cdot s}{s^2 + \left(\frac{1}{R_2 \cdot C_1} + \frac{1}{R_2 \cdot C_2}\right) \cdot s + \frac{1}{R_1 \cdot R_2 \cdot C_1 \cdot C_2}} \quad \rightarrow \tag{7.5}$$

$$\omega_0 = \frac{1}{\sqrt{R_1 \cdot R_2 \cdot C_1 \cdot C_2}} \tag{7.6}$$

$$Q = \frac{\sqrt{\frac{R_2}{R_1}}}{\sqrt{\frac{C_2}{C_1}} + \sqrt{\frac{C_1}{C_2}}} \qquad G = \frac{R_2 \cdot C_1}{R_1 \cdot (C_1 + C_2)} \tag{7.7}$$

If we choose standard capacitors of reasonable size, and if we choose $C_1 = C_2$ it can be seen from the expression for Q, that the denominator will be 2 (its maximum) and it will be easier to attain high Q. We wont have to have unrealistic resistor values, since the Q is also decided from the ratio between the resistors. The resistors will then be:

$$\omega_0 = \frac{1}{\sqrt{R_1 \cdot R_2 \cdot C^2}} \leftarrow R_1 = \frac{3.270957632e8}{R_2} \tag{7.8}$$

From the expression for Q we get R_2 to be 723431.6 Ω, making $R_1 = 452.145\Omega$. With these values the DC gain will be (since $C_1 = C_2$):

$$G_{dc} = 20 \cdot log_{10}\left(\frac{R_2}{2 \cdot R_1}\right) = 58dB \tag{7.9}$$

This can be reduced by splitting R_1 into two resistors forming a voltage divider, but it is not deemed necessary, due to the need to amplify the signal.
The closest E96 values for the resistors are 453 Ω and 715 $k\Omega$

Simulation in LT spice

The circuit is simulated with the E96 resistor values to ensure performance:

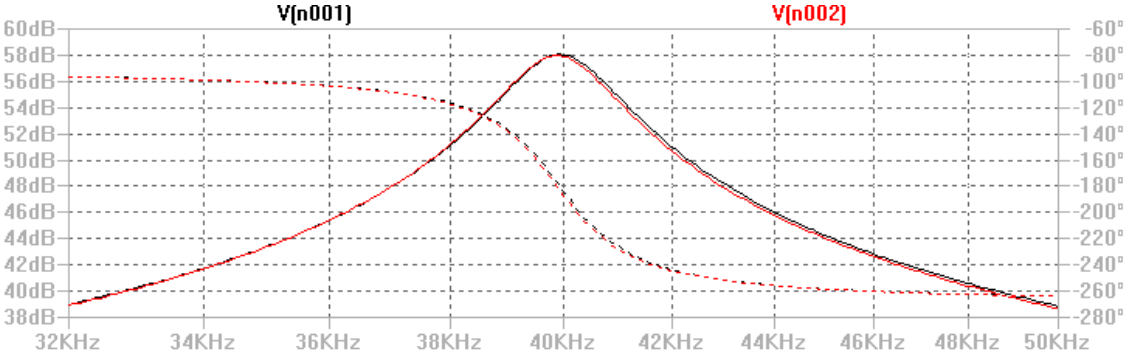


Figure 7.12: Lt spice simulation of filter response and phase. The red line is E96 value filter, and the black is with ideal values. Circuit simulated with ideal opamp.

As it is shown in the figure, the E96 values do not change the filter significantly.

Measurement of response with analyzer

The filter was implemented on breadboard and tested with an analyzer NI-4461. The analyzer sends out a sine wave for each measurement frequency, and then does an FFT of the signal it gets back, and compares it to the signal it sent out (which is also fed back to the machine through a cable).

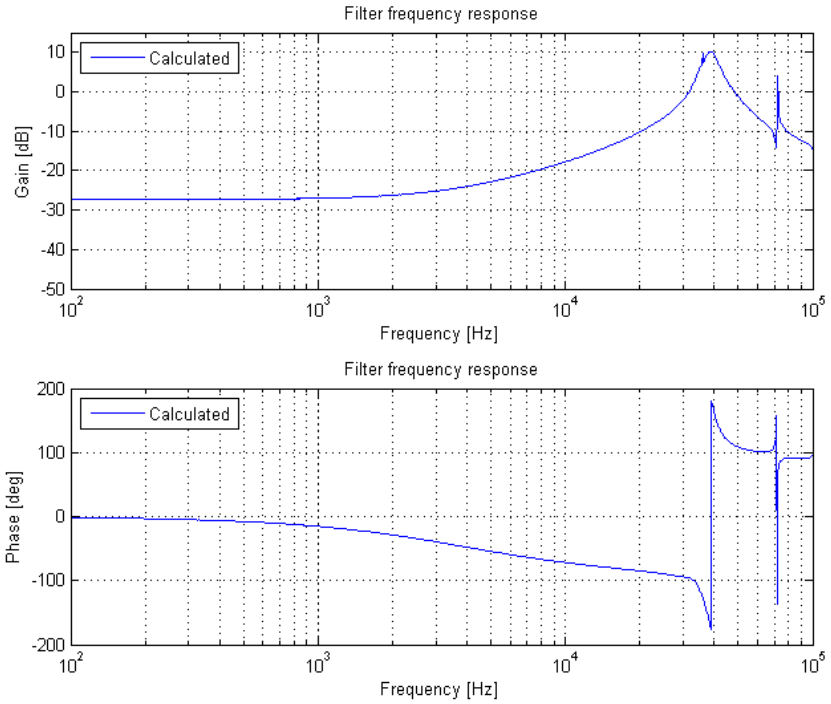


Figure 7.13: Measurement of filter response. Plotted with matlab

The filter behaves as expected, and the design is deemed a success. It was chosen not to measure the distortion with the analyzer since it does not really matter for this application, and since the analyzer mostly measures noise as distortion (at these signal levels).

Echo detector

The echo is detected using a threshold detector. This is to ensure that the circuit does not trigger on the engine noise etc. The output of this circuit will be switching between 0 and 5 volts when there is an echo (which the arduino can pick up). The threshold detector is designed with an opamp.

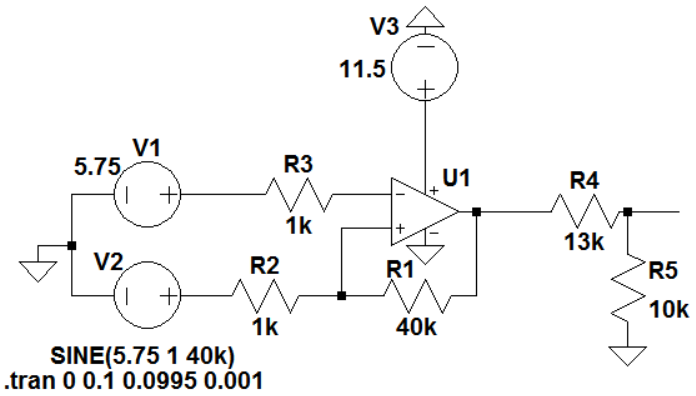


Figure 7.14: Threshold detector circuit.

If we remove R_1 the output will switch from ground to Vcc (or from ground to 5 after the voltage divider) when the signal V2 crosses the reference V1. When R_1 is added, there will be some hysteresis in the circuit, meaning the signal has to be a certain amount above V1 before it switches.

7.3.3 Distance measuring algorithm

Distance will be measured by measuring the time from the speaker starts transmitting the echo, until the echo has returned. In the mean time the microphone should not listen to the signal it gets directly from the speaker, but only the echo. This is done by turning it off for a small period of time (in reality just a delay).

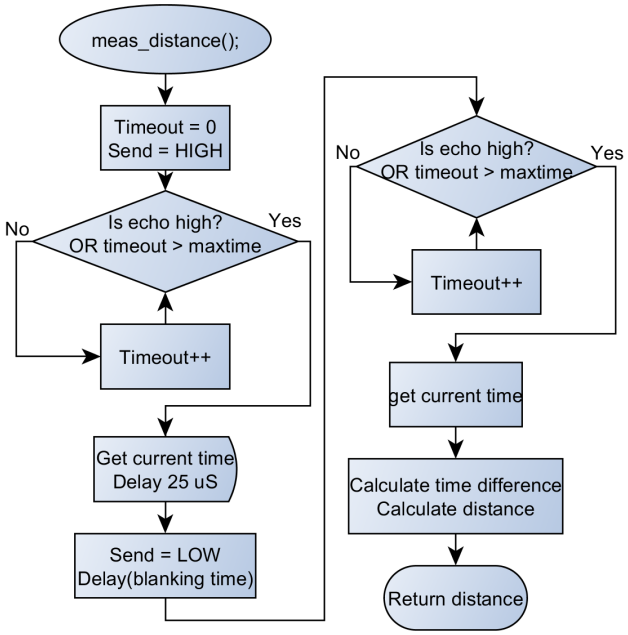


Figure 7.15: Algorithm for measuring distance.

Change of algorithm due to circuit failure

We made 3 iterations of the design with little success. After the construction of the final distance sensor it went through a preliminary test which revealed that the transmitting circuit worked (was just very slightly off frequency), and the receiving filter worked, but there was some problem with the amplification stage afterwards. After some measurements it was concluded that the opamp was at fault. Due to lack of time, it was decided to use a finished sensor circuit instead, which works in a similar way but is based in a microprocessor. The HC-SR04 sensor breakout board.

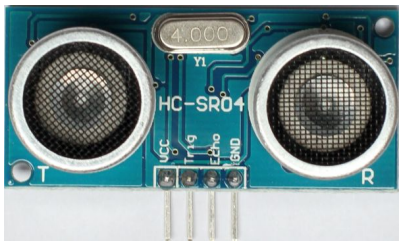


Figure 7.16: Sensor alternative. HC-SR04

This sensor works in a slightly different manor. You send a 10 microsecond pulse to the trig pin, and then wait until echo pin goes high. It will go high when the sensor is done sending the ultrasonic pulse train. Then when the echo returns the echo pin goes low again. So the time that should be measured is while the echo pin is high. This is usually done with an algorithm that is similar to the one used by our own sensor. But in order to save CPU time, it was chosen to implement it using interrupts:

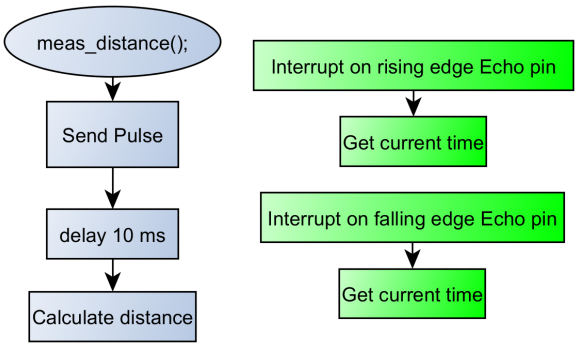


Figure 7.17: Blue part is the function called (the delay frees up the cpu for other tasks on the scheduler), the green part is ISR.

This way the loop will also be a constant time each time, because even if it receives the interrupt before, it will still wait the 10 ms.

Range limit

Because of the PPM task running on the scheduler, the program has to by cyclic at 20 ms, giving a maximum of 10 ms for each sensor. In 10 ms sound can travel 3.4 meters

(if the speed of sound is 340 m/s), making the maximum measurable distance 1.7 meters because the sound has to echo back. This is enough for the application since it has to stay close to the hull of the ship anyway.

7.3.4 Pitch Compensation

The pitch compensation (described in quadcopter description), is achieved using the "adxl 345" 3 axis accelerometer from Analog Devices. The sensor is accessed over the I2C bus using the wire.h library in arduino. The adxl345.h library is then used to get the number for each direction (x,y,z). The code is:

```
acc.read(&Xg, &Yg, &Zg); //from the adxl345 lib
pitch_angle = atan2(Xg, sqrt(Yg*Yg + Zg*Zg);
//Take length of yz vector
//atan2(x,y) takes the angle in radians between the positive x axis
// and the point given by (x,y)
```

This is then used as described in section "How does the quadcopter work?" in the problem analysis.

7.4 Modelling

In order to keep the quadcopter in one place during the measurement we must be able to control its distance and angle from the wall. That means controlling pitch (moving forwards and backwards) and yaw (turning around z axis). Our system will take the measurements from distance and pitch sensor, and from those values calculate a control output in the pitch and yaw direction. It will be up to the user to control the other direction. For this to happen it is required to model a behaviour of the quadcopter while moving in those directions. That will deliver us two models, two transfer functions.

7.4.1 Quadcopter model

In order to design controllers for regulating the distance and angle, two models of the system must be made. This also makes it possible to simulate the design, and estimate performance/stability. The system in this case is the quadcopter. The input to the quadcopter (which is including a PixHawk flight controller) is our PPM signal to the PixHawk FC. The output from the system is the quadcopters distance from the wall and the angle towrds it, which is what we want to regulate. The product will actually be a cascaded system, where the inner loop is the Pixhawk's three PID controllers that stabilize the quadcopter, and whose inputs is the values in the PPM signal, and whose output is quadcopter's engines power . The PixHawk has an accelerometer and a compass which measures its heading, but not its position. The position of the quadcopter is measured in the outer loop (distance from wall and angle). An accelerometer in the outer loop was necessary since access to the pixhawks measurements is non-existent.

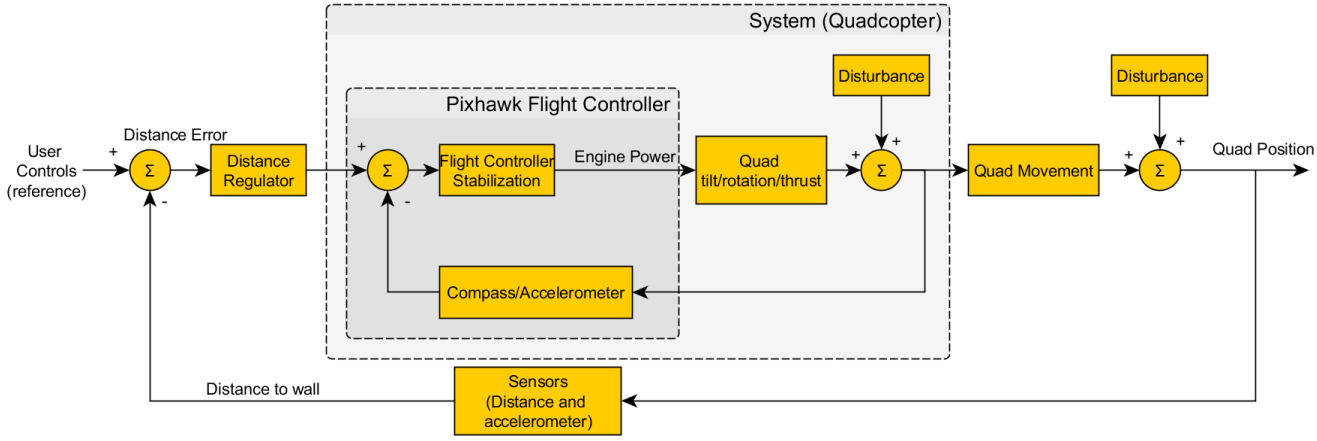


Figure 7.18: Cascaded loop representation of quadcopter regulators.

The part that has been grouped under the name ”System (Quadcopter)” is the part that will be modeled experimentally using step response. This is because of its complexity. It would be very difficult and time consuming to model theoretically.

The step response

The step response is used to describe a systems dynamics. It shows such properties as rise time, overshoot, steady state error, settling time, oscillation frequency (if present), etc. From these properties a first or second order transfer function can be made. The input step function is zero until the time zero, where it instantly switches to one. In the Laplace domain it can be described as $u(s) = \frac{1}{s}$, which is an integration. The step response is thus (for first order):

$$Y(s) = G(s)u(s) = \frac{k}{\tau s + 1} \cdot \frac{1}{s} = \frac{k}{(\tau s + 1)s} \tag{7.10}$$

Where τ is the time constant of the system, and k is the gain. It should then be possible to get the transfer function by multiplying by s (differentiating in the Laplace domain).

Measurement of step response

The measurement of step response of the system is described in appendix. Quick summary: The measurements were done using both Vicon motion tracking cameras, and also with an ultrasound + RF location tracking system called Games on Track (GOT). GOT proved ineffective for this kind of measurement(low sample rate, lots of noise) and it was decided to use Vicon instead. The results from the measurements with Vicon motion capture cameras:

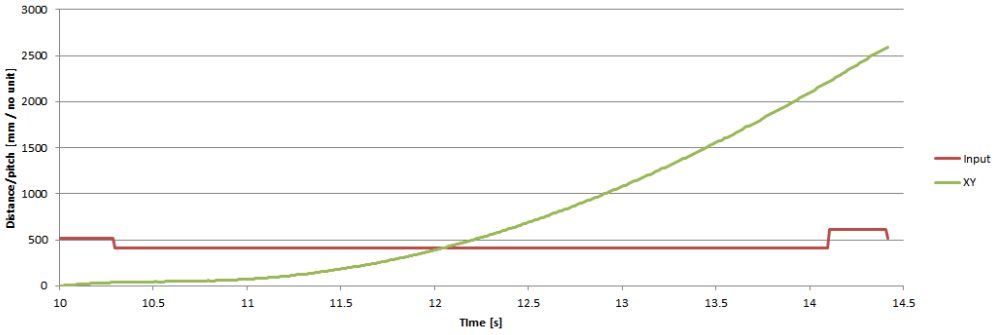


Figure 7.19: Step in pitch using Vicon to track position.

It looks like the position change is linear after 2 seconds, which suggests the relationship between input and output (the transfer function) is a linear relationship with a delay. In other words, it looks like an integrated first order system. The step response would then be:

$$Y(s) = \frac{k}{(\tau s + 1)s^2} \tag{7.11}$$

Making the systems transfer function:

$$G_p(s) = \frac{k}{(\tau s + 1)s} \tag{7.12}$$

To validate this model, a step is applied (using matlab step function), and it is plotted in the time domain, next to the original:

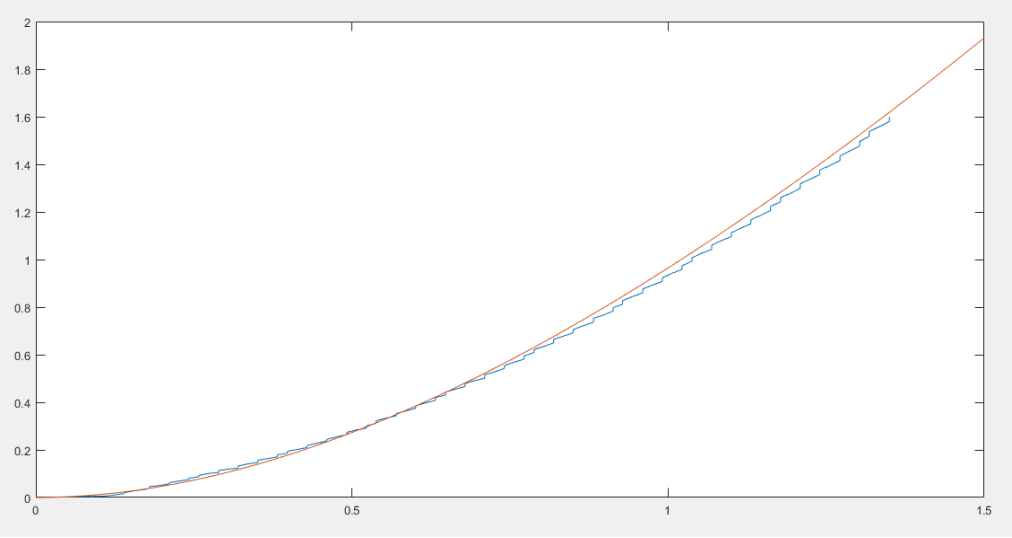


Figure 7.20: Plot of measurement data next to the step response of G(s). Measured data (blue), simulated (red).

It is possible that later the plots would have more difference between one another but we weren't able to make longer steps to check that because of the limitations of Motion

Capture lab sizes.
From the graph the time constant is extracted. The time is the time it takes a step response to reach 63.2% (from $1 - 1/e$), of the steady state value. For this step response graph it is about 1.2 seconds. The gain of the system is the steady state value, which is 1,51 m. The transfer function is thus:

$$G_p(s) = \frac{3}{(1.2s + 1)s} \tag{7.13}$$

As you can see in the figure, the model fits the measurement data.
Since our step input was 100 "units of pitch" we have to divide the transfer function's gain by 100, therefore the final transfer function is thus:

$$G_p(s) = \frac{0.03}{(1.2s + 1)s} \tag{7.14}$$

Yaw transfer function

A model in the Yaw direction was also made using the measurement in Yaw direction which was made on the same occasion:

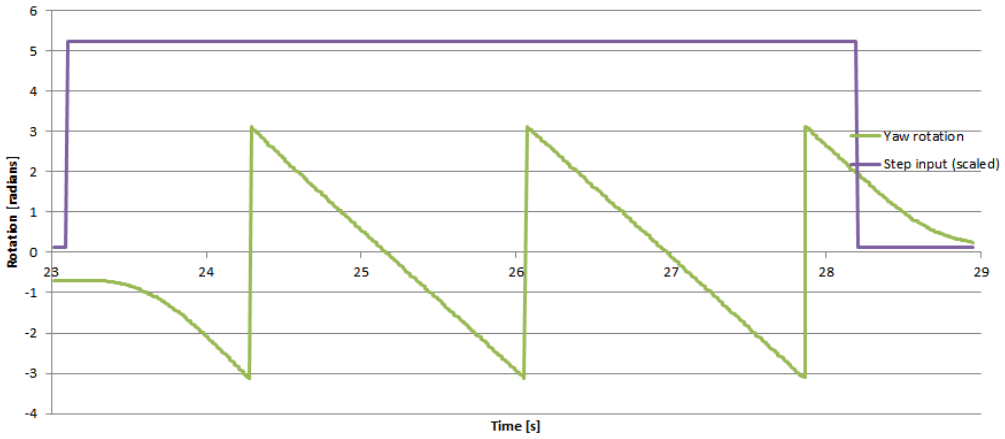


Figure 7.21: Yaw step. The polarity of the data is due to the rotation direction, and the clipping is caused by the sensor only outputting from $-\pi$ to π

The model in the yaw direction will be similar, since we have similar input output relation. The only difference is that the output is rotational and not translational, but this will not affect the model, only the unit. The time constant is read to: 0.6, and the gain is 5.2, making the transfer function:

$$G_y(s) = \frac{5.2}{(0.6s + 1)s} \tag{7.15}$$

To validate this model, the step response of the transfer function is plottet next to the measured data:

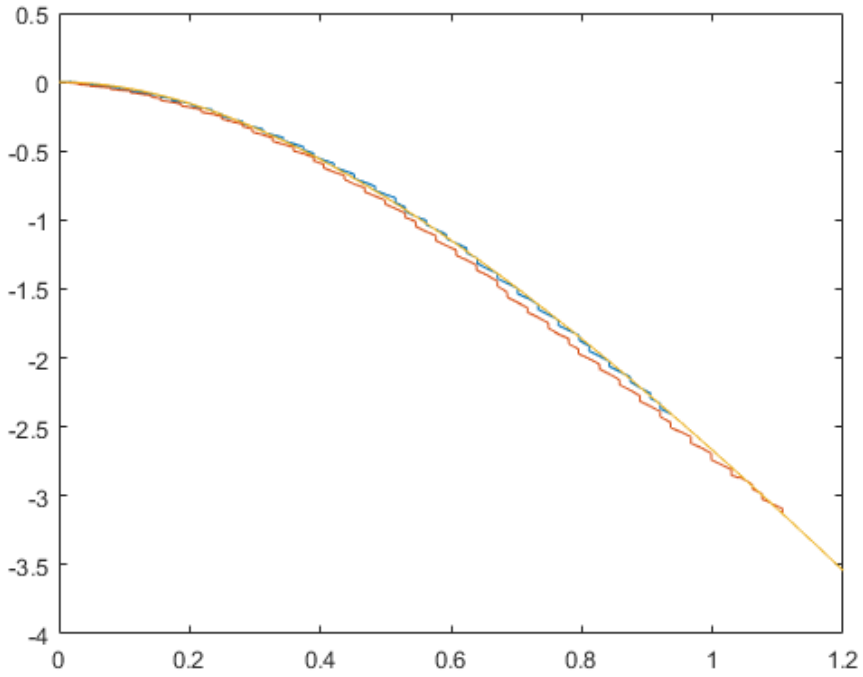


Figure 7.22: Step response of quadcopter in yaw direction as measured (blue and red), and step response of yaw model transferfunction (yellow).

This model fits sufficiently to the measured data.
Here again, the same as in obtaining pitch model we have to take into account that our step input was 500 "units of yaw", so the gain of the transfer function has to be divided by 500, that will result in:

$$G_y(s) = \frac{0.0104}{(0.6s+1)s} \tag{7.16}$$

These two transfer functions can now stand for the model of the system, and be used to estimate a controller, and simulate the system in its entirety.

7.4.2 Sensor model

The delay of the distance sensor can be modeled as a low-pass filter. This comes from the (0,1) Padé approximant of the root locus representation of a time delay, which is also called a first order lag [20].

$$e^{-T_d s} \approx \frac{1}{1+T_d s} \tag{7.17}$$

This can be rewritten to:

$$G(s) = \frac{\frac{1}{T_d}}{s+\frac{1}{T_d}} \tag{7.18}$$

It can be seen that this is the transfer function for a first order lowpass filter, since $\omega_c = \frac{2\pi}{T_d}$. This also tells us that, by this model, the system will be "slower" if the sensor delay is larger, due to the low-pass characteristic.
The delay itself is the time between the distance is something, until the sensor has measured it and it is fed into the controller. Because of the scheduling, the period time at which the controller and sensors updates is the delay, which is 0.02 ms, making the model transfer function ($\frac{1}{0.02} = 50$):

$$G(s) = \frac{50}{s + 50} \tag{7.19}$$

We will design the controller with this as a sensor model, although in reality it would be possible to sample faster when closer to the wall, but this will also take up more CPU time. It is estimated that a sample rate of 50 Hz is enough for our system.

7.5 Controlling

7.5.1 Controlling Quadcopter

To make the quadcopter that meets our requirements we need to design and apply controllers for our system that will keep the quadcopter in certain distance from the wall also in order to measure the distance correctly we must control an angle from the wall, since the sensors are mounted on the front of quadcopter the angle needs to remain at 0 radians for an accurate measurement. While moving towards and away from the wall we used an accelerometer so our measurements of distance are compensated and correct even if the quadcopter isn't level. Block diagrams for both systems are showed below.

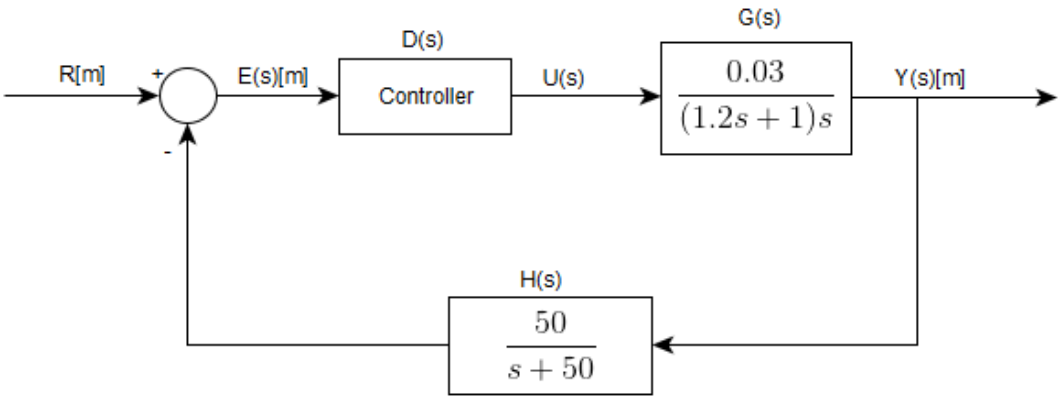


Figure 7.23: Block diagram for pitch control

Reference for this system can be changed by an operator from 20 cm to 170 cm. This is due to the fact of functionality limitations of distance sensors.

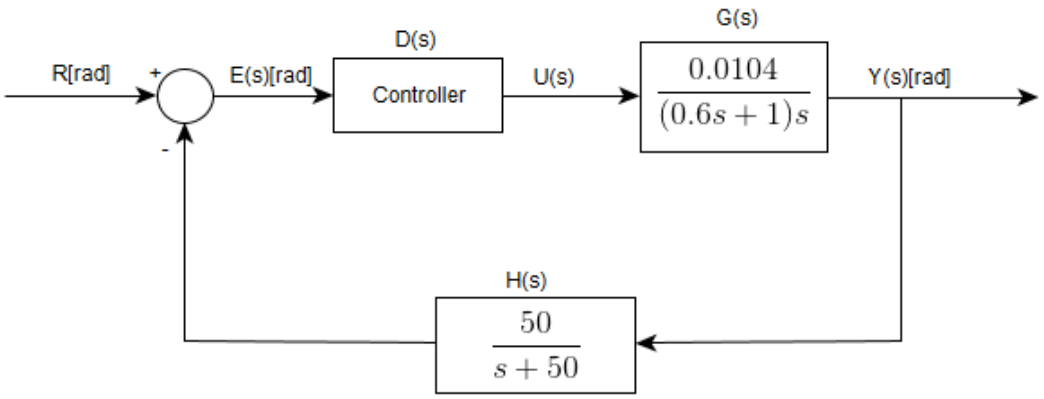


Figure 7.24: Block diagram for yaw control

Yaw reference should always be the same and equal to $\pi/2$ rad. Although, it is possible to change it between -30 degrees and 30 degrees. This limitations are due to the fact that after tilting quadcopter from perpendicular direction to the wall by this value, amplitude of sent sound wave from transmitter isn't high enough for the receiver.
Requirements for the controllers:

- Overshoot of maximum 15%
- Settling time of maximum 3s
- Rising time between 1s and 2s

7.5.2 Designing of controllers

To design controllers we must know how our systems behave without them. Using bode plots in Matlab we discovered gain and phase margin along with crossover frequency. Based on that we will design controllers in order for them to accomplish our requirements.
We will divide this control part in two parts, each for one of the controllers for clarity.

Pitch controller

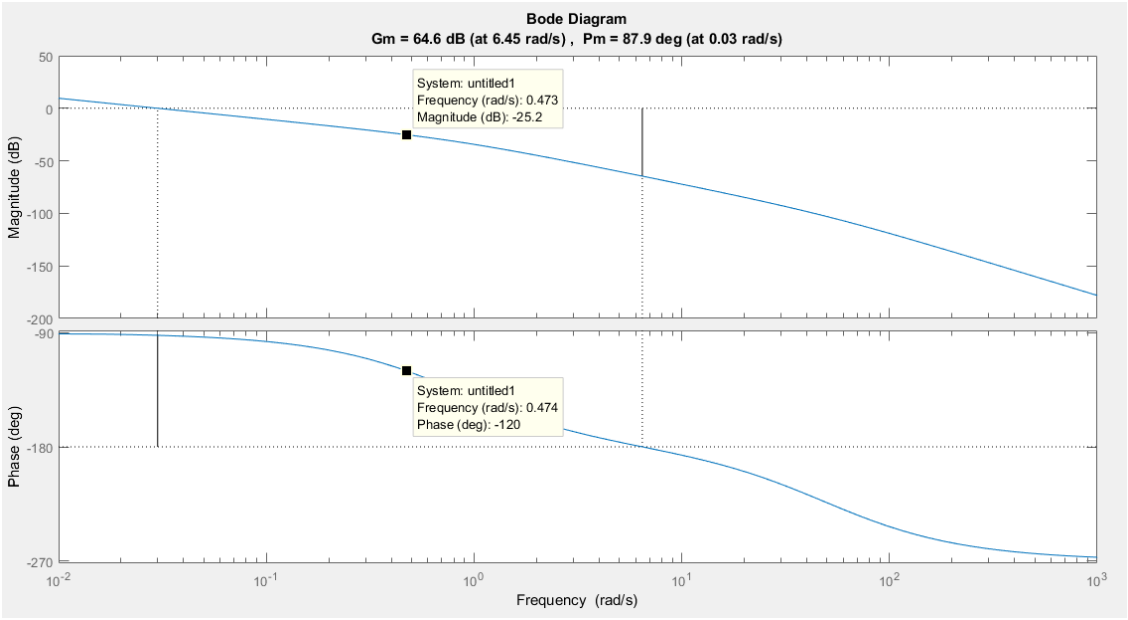


Figure 7.25: Bode plots with marked PM and GM for pitch control

$$x = 10^{-(\frac{25.2}{20})} = 18.2 \tag{7.20}$$

The bode plots show system without any controller. We can see that a P controller with the value of Kp=18.2 is enough for our system (with PM of 60 degrees) to accomplish set distance while being stable. This is although only true in theory. In fact we discovered that it will affect with too little change in controls to overcome drifting, thus it won't let our quadcopter to obtain set distance. Therefore we decided to add a higher gain. Consequence of that is the change in stability, that means we have to add also a D-part of controller to gain enough PM.
For the system to accomplish our requirement with the overshoot of 15% we need the PM of 60 degrees. That resulted in experimentally obtaining a D regulator with a zero in 2.22 rad/s. The effect of adding the D regulator can be seen on the picture below.

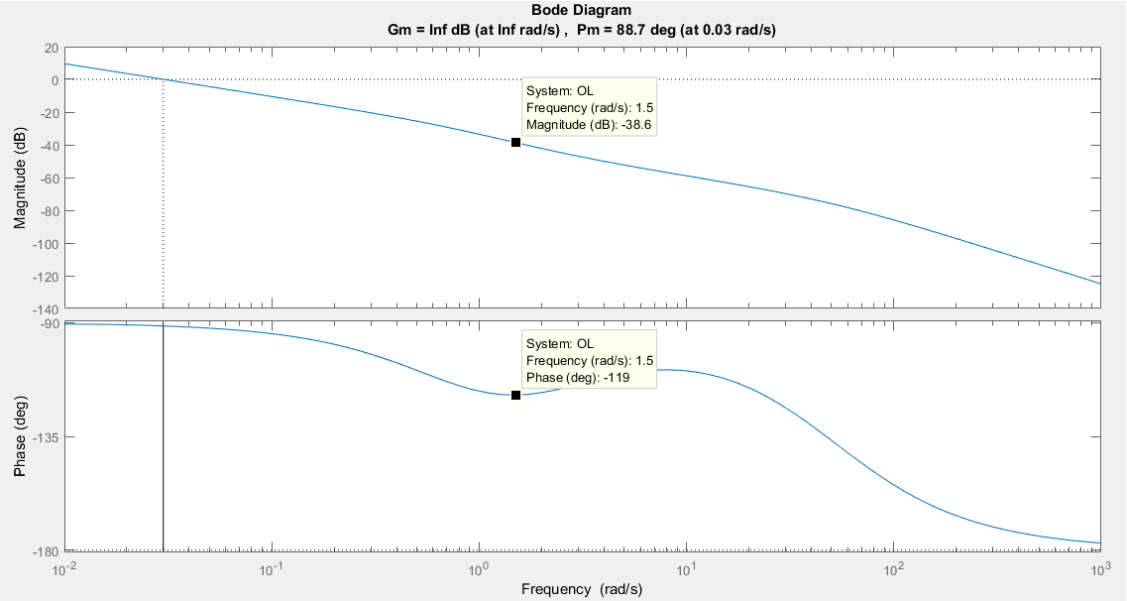


Figure 7.26: Bode plots with marked PM and GM for pitch control

$$x = 10^{(\frac{38.6}{20})} = 85.1 \tag{7.21}$$

Now we can add a gain of 85.1. That will result in PM of 61.4 degrees. It can be seen on the plots below:

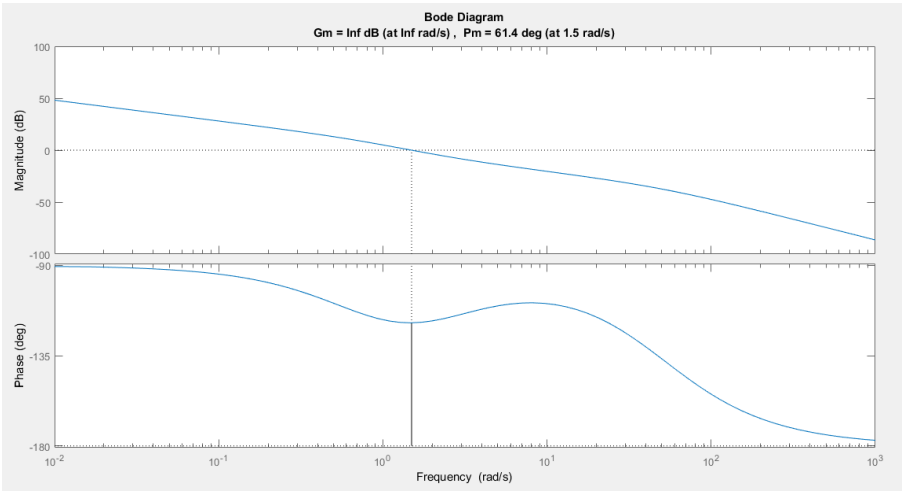


Figure 7.27: .

Transfer function for the PD controller is stated as:

$$D(s) = Kp(1 + Td \cdot s) \tag{7.22}$$

Putting our data into equation:

$$D(s) = 85.1(1 + 1/2.22s) \tag{7.23}$$

Final transfer function for the controller:

$$D(s) = 85.1 \cdot (1 + 0.45s), Kp = 85.1, Kd = 38.3 \tag{7.24}$$

Yaw controller

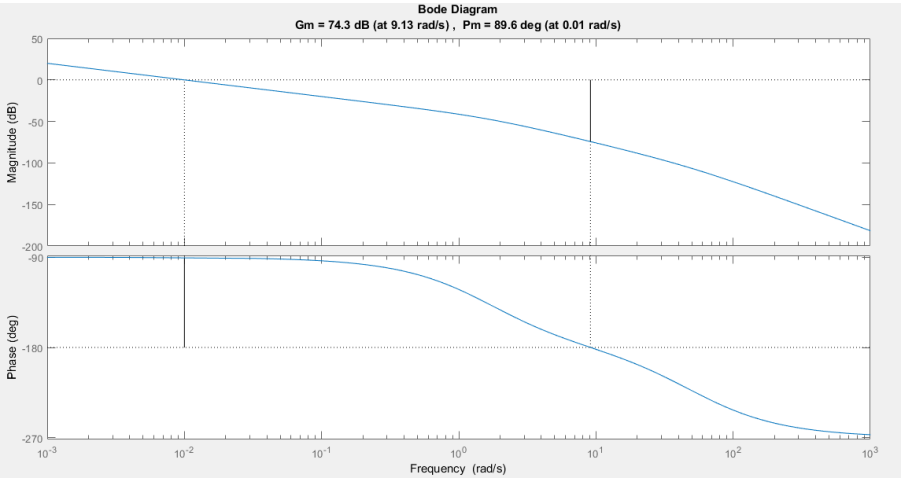


Figure 7.28: Bode plots with marked PM and GM for yaw control

Bode plots show that we only need to use a P controller. We are adjusting the gain keeping in mind that we have to stick to the 45 degrees margin, this is because the yaw system doesn't need that much of a phase margin, the quadcopter isn't drifting that much in the yaw axis. Instead of keeping PM that we don't really need, we decided to get faster response.

$$x = 10^{-(\frac{46.8}{20})} = 218 \tag{7.25}$$

Transfer function for the P controller is simply:

$$D(s) = Kp \tag{7.26}$$

Final transfer function for the controller

$$D(s) = 218 \tag{7.27}$$

Applying controller results in the GM of 27.5 dB and PM of 44.7 degrees.

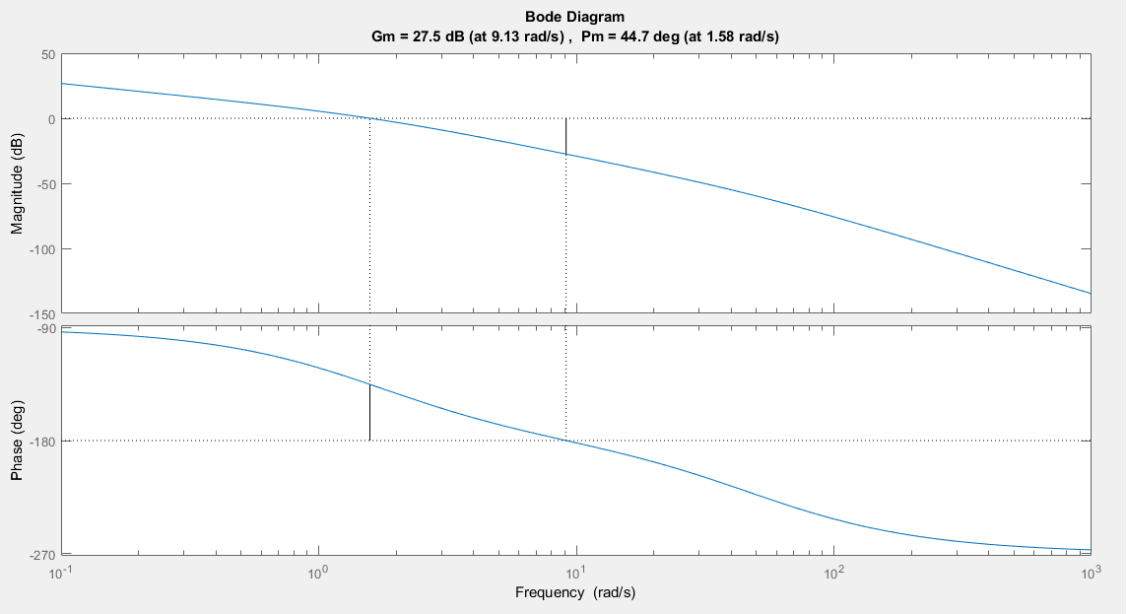


Figure 7.29: Bode plot for a system with applied controller

7.5.3 Simulation

Both systems were simulated in Simulink using block diagram. These include transfer functions for pitch and yaw, controllers and transfer functions of distance sensors. We also added saturation for the pitch system. This is because at the high pitch values quadcopter is tilting too much from the wall thus giving wrong measurements and obtaining too much speed. This is because amplitude of reflected sound wave from the wall isn't high enough and also our pitch compensator only work within ±20 degrees. Saturation is not needed for the yaw system since we've already put restrictions for the maximum values of angles. Then step input was applied with maximum values for both systems: 1.5m for pitch and 1.05 radians for yaw.

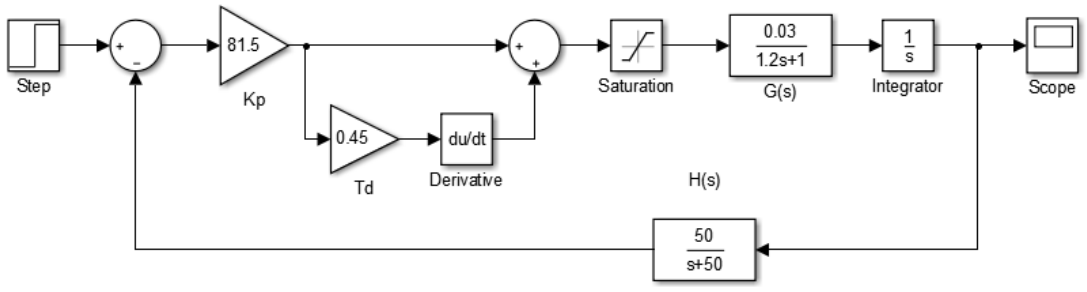


Figure 7.30: Block diagram for pitch system. In this diagram a saturation was added to set limits for the controller's output.

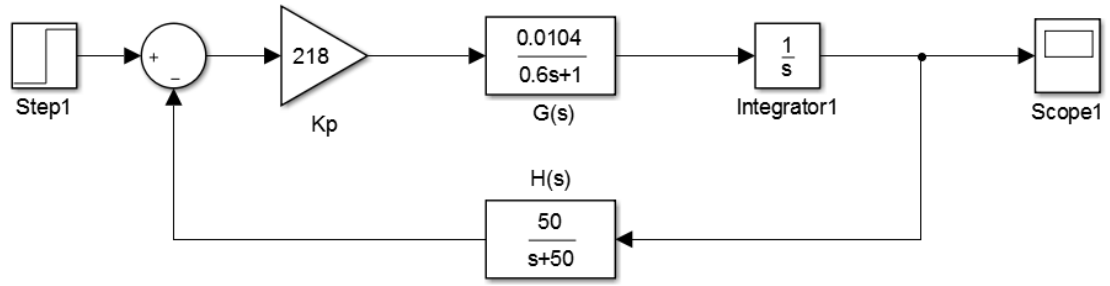


Figure 7.31: Block diagram for yaw system

We must also remember that there are limits of how much we can change the angle from the wall. This occurs due to the fact that after crossing the limit, transmiter's sound wave reflected from the wall is not coming back to the receiver.
Using Simulink we get the step responses and controller's output.

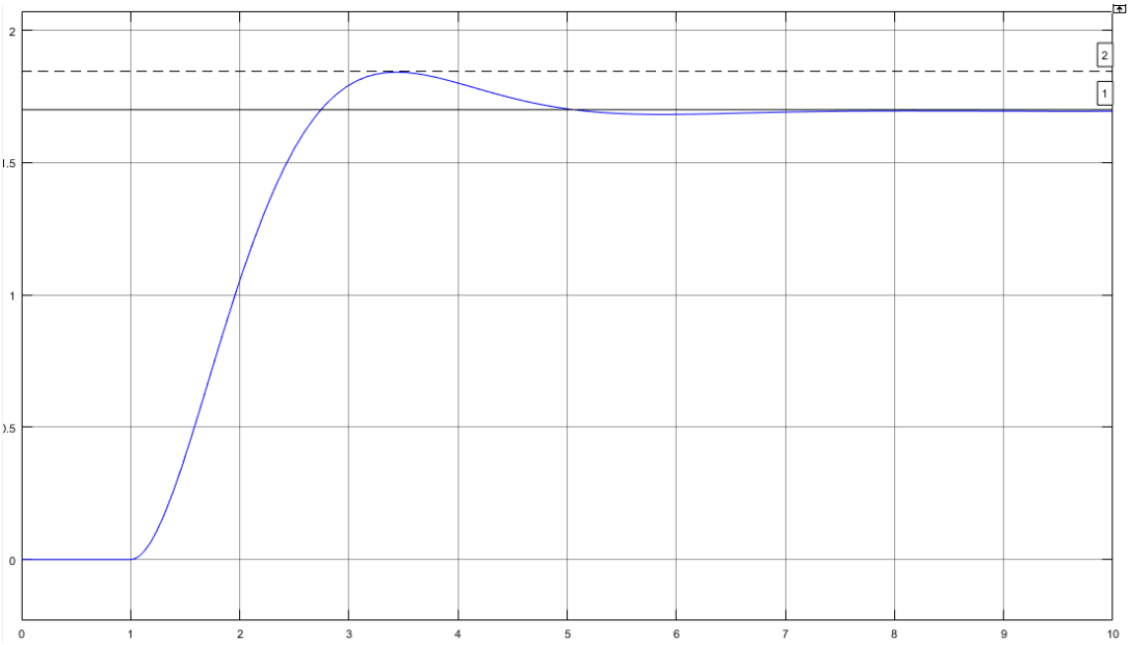


Figure 7.32: Step response pitch system with the highest step input. This is because we want to see if the clipping occurs. The overshoot reaches 1.63 and the steady value is 1.5.

Overshoot is calculated:

$$Mp = \frac{1.63 - 1.5}{1.5} \cdot 100 = 8.7\% \tag{7.28}$$

Plot also shows us a settling time of approximately 7 s.
Also we wanted to check whether the output from the controller clips, that could be caused by using the saturation of +/-200 units of pitch(approximately 20%).

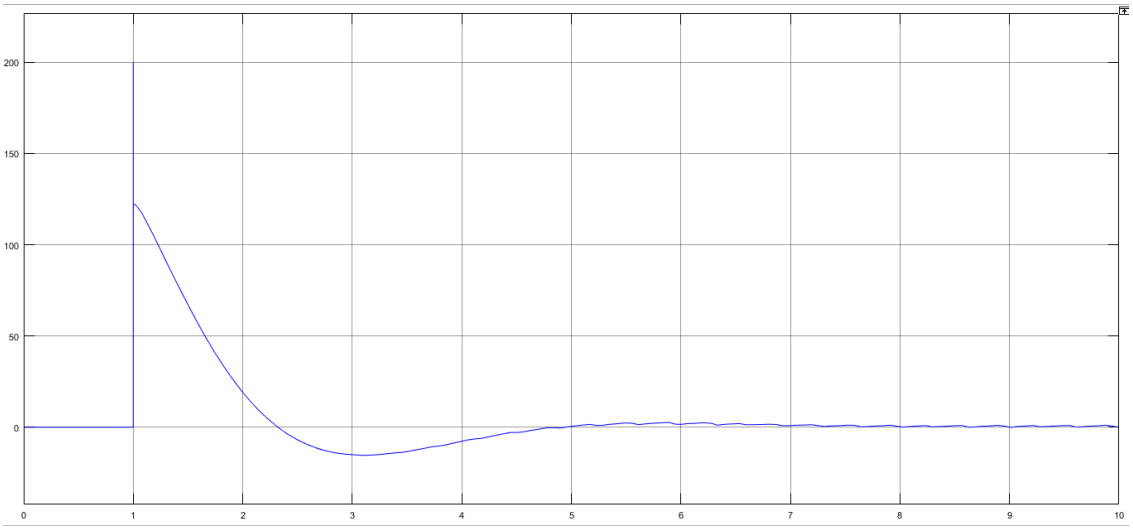


Figure 7.33: Pitch controller's output diagram for pitch system.

As we can see there is no clipping that means there's no need to lower the gain or increase saturation.

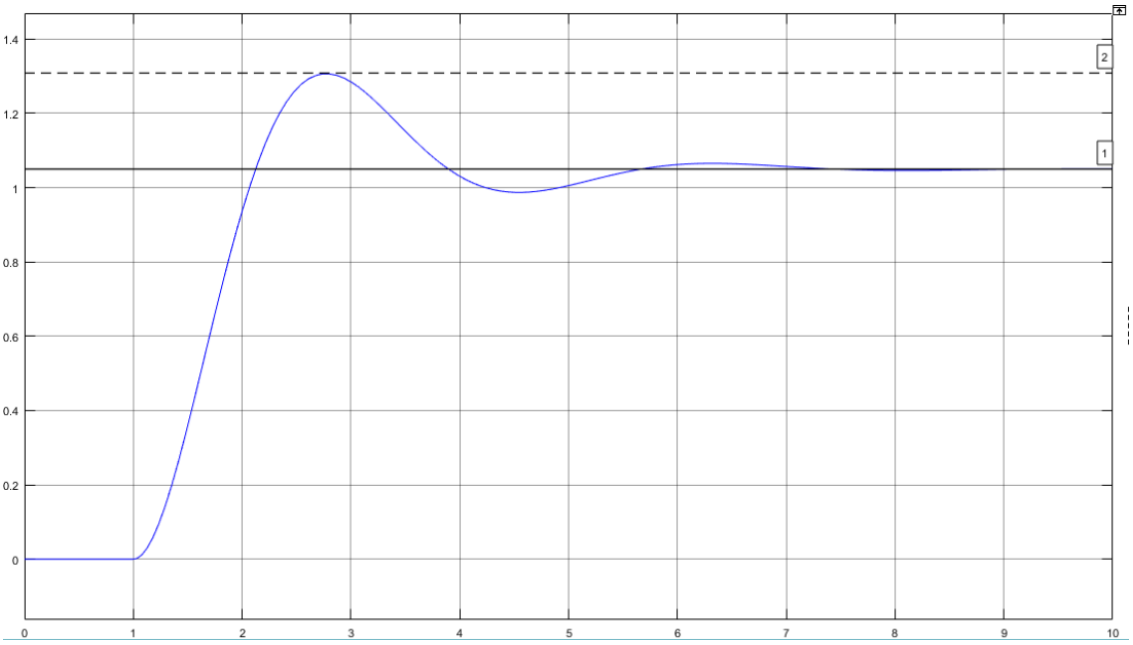


Figure 7.34: Step response for yaw system. The overshoot reaches 1.3 and the steady value is 1.05.

Overshoot is calculated:

$$Mp = \frac{1.3 - 1.05}{1.05} \cdot 100 = 23.8\% \tag{7.29}$$

Step response shows us a settling time of approximately 6.5s.

Step responses' results for our systems didn't meet requirements but we decided to use this controllers either way to see how the will behave in reality. This decision comes from knowing that in fact we don't really control the speed of engines, we only control the input to pixhawk. Besides, in the yaw controller fast response was more important than the overshoot and having longer settling time in simulation is not that relevant as stable system.

7.5.4 Implementation of Controllers

The output from the controllers was implemented as followed:

```
derivative_dist = (old_distance - mdistance)/(ctrl_timer - micros());

raw_pitch = (Kp_pitch * mdistance) + (Kp_pitch*Td_pitch * derivative_dist);
raw_yaw = Kp_yaw * deltaAngle;
```

Where 'mdistance' variable is a distance changed from millimetres received from the sensor to meters and 'old distance' is the distance measured previously. 'ctrl timer - micros()' defines how much time has passed since previous measurement. These are the values that we use to obtain a derivative part for the pitch controller. 'deltaAngle' is a variable that indicates the difference between last measured angle and the present one. Obtained values from that part of code are then added to controls values that are received by the quadcopter and used to set it at the right angle and distance.

Chapter 8

Testing, Verification & Conclusion

8.1 Conformance and performance testing of system

The system that has been designed will be tested to verify its functionality (conformance with requirements), and to find out what is the performance of this system. Conformance tests can be made as a part of the performance test. For example, if the quadcopter can receive data from the remote control, we know it meets that requirement, and this can also be confirmed during a test of the links stability, latency and speed.
The tests that will be made are:

- Test of sensors for precision and range.
- Test of basic distance control functionality
- Test of quadcopter distance control algorithm response with step input.
- Test of communication link and latency.

8.2 Measurement of sensor precision, range and delay

This will be a performance test, of the distance sensors range and precision.

8.2.1 Distance from wall with straight angle

This measurement was conducted by placing the distance sensors on a stand, in the same configuration as they will be placed on the quadcopter (distance between them and direction). This stand was then moved closer to the wall in 5 cm increments at first, starting from 1.5 meters, and within a meter at 3 cm increments. The wall that was chosen for the test is a flat, painted, brick wall, with no objects in the way. The measurement is read from a serial terminal.

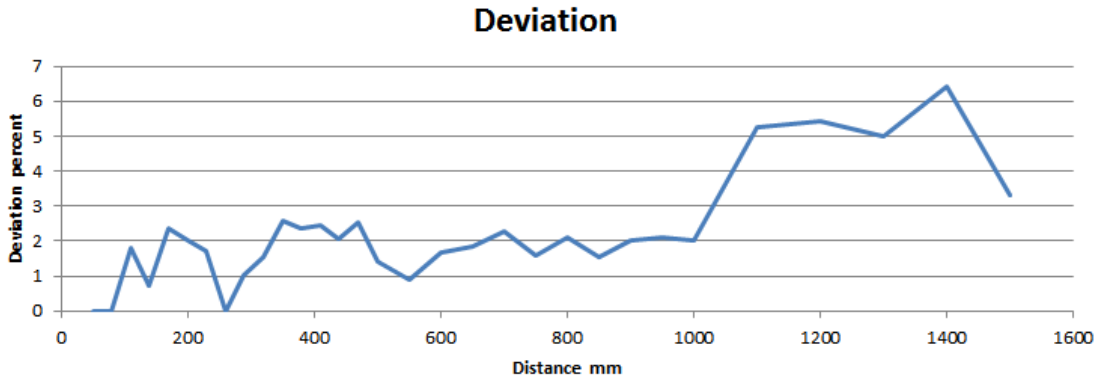


Figure 8.1: Deviation percentage

As it can be seen in the figure, the sensors are most precise within one meter distance. It was not possible to measure the distance within 25 mm, because of the error being too large. At 0 cm the distance sensor shows 44 mm, which must be due to some timeout built into it.

8.2.2 Distance sensor integrated into system

Because of the very real risk of crashing the quadcopter with a bad controller, a preliminary test was made without flight. The purpose of this test was to confirm that the sensor values will still be good when integrated into the quadcopter, and with engines running and propellers.
In order to make this test repeatable, a crude test setup was built, with a DC motor pulling a cart on which the quadcopter is placed. This then pulls the cart towards the wall from 2 meters at a constant speed. It is possible to measure 2 meters, because the program which is used only has the sensors and some value print outs, but not the PPM generation etc. Otherwise the limit would be 1.7, as mentioned earlier.

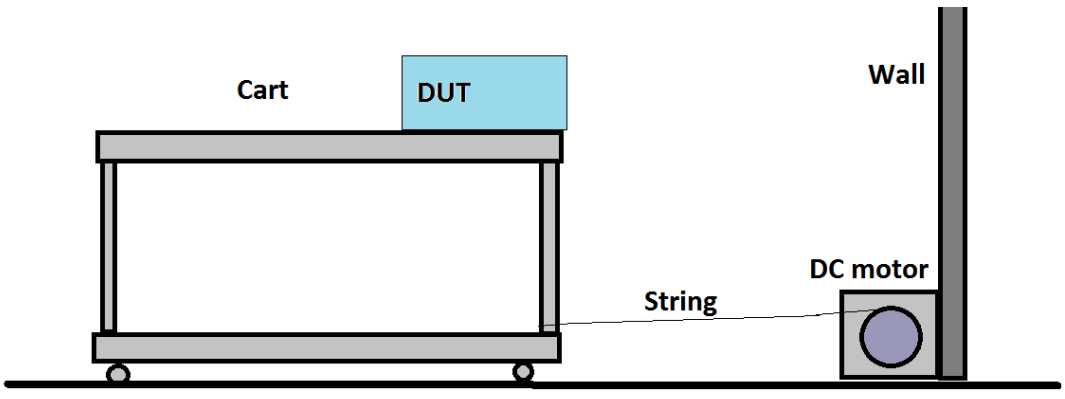


Figure 8.2: Crude drawing of test setup. The DUT is the Quadcopter.

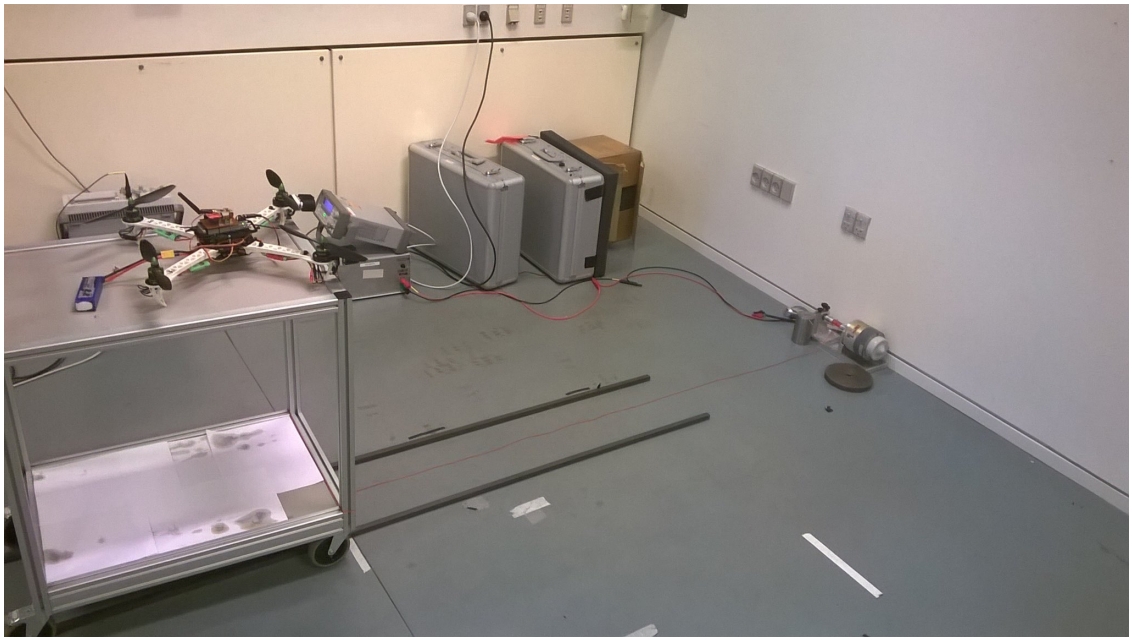


Figure 8.3: Photo of test setup.

The test is run from Matlab, with a function generator starting the engine which is controlled over a LAN socket which is created in Matlab, using the instruments library. The values for controller output and measured distance is captured with an APC220 radio transceiver plugged into a USB port, which Matlab then reads as a COM port. A timestamp is created with 0 at the start of the test. A total of 9 different tests were made, with and without engines, propellers, etc.

Results

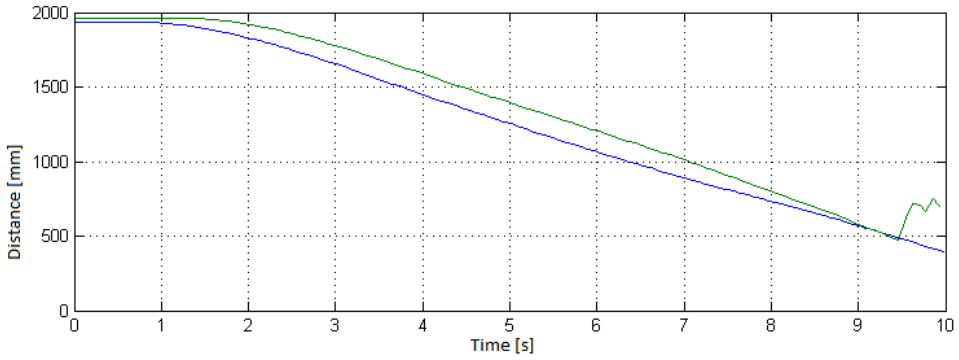


Figure 8.4: Reference measurement without engines or propellers. Blue is without pitch compensation and green is with. The error in the end is when the cart crashes into the wall.

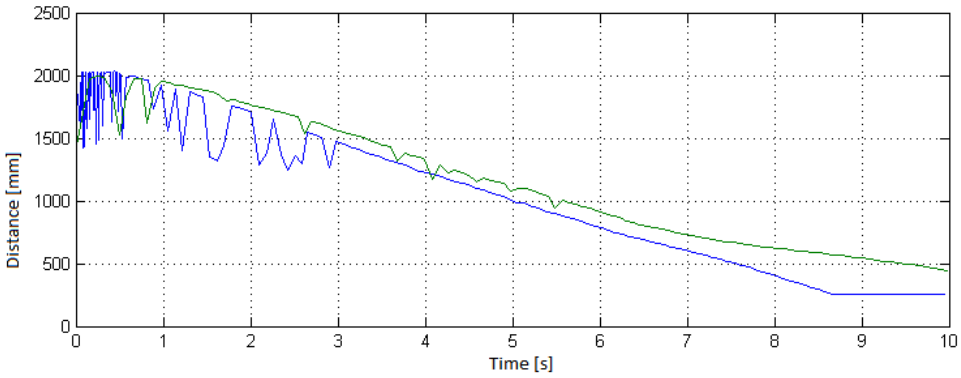


Figure 8.5: Measurement with engines on, without propellers. Blue is without pitch compensation and green is with.

From the rapid change in distance that was measured it is obvious that there is some kind of disturbance. This was found to be mainly the noise from the engines causing vibrations in the quadcopter, which are acoustically coupled to the receiver. It could also be in part caused by a magnetic coupling between the engine and sensor circuit, or the noise made by the engines getting recorded by the sensors microphone.

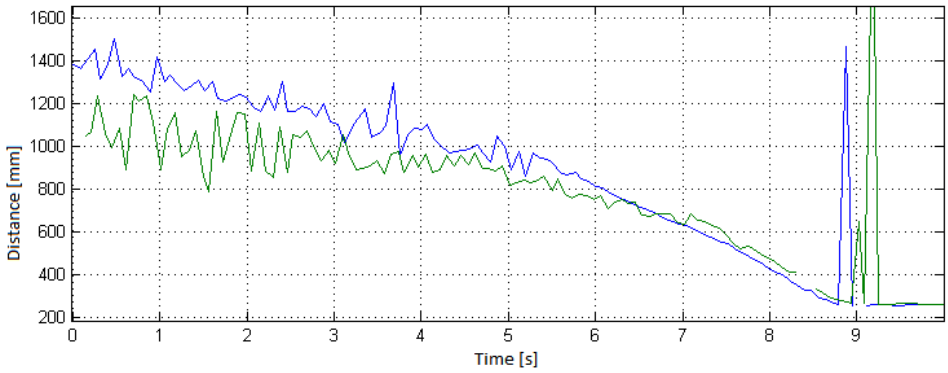


Figure 8.6: Measurement with engines and propellers. Blue is low power, green is at half power.

The propellers lower the signal to noise ratio so much that the range is limited. It only functions well within 800 mm of the wall. It could also be propellers obstructing the sensor, or the thrust disturbing it. It is seen that the frequency of the noise is approximately 4 Hz.

Adding of filter and extra padding

In order to reduce the effects of the engine and propellers a crude digital low pass filter was added to the distance sensor, using only one previous sample (first order).

$$y[n] = a \cdot y[n-1] + b \cdot x[n] \tag{8.1}$$

Where $y[n]$ is the filtered signal, and $x[n]$ is the input to the filter. This is an Infinite-Impulse Response filter, whose tranfer function is [21]:

$$H(z) = \frac{b}{1 - aZ^{-1}} \tag{8.2}$$

Where a and b are filter coefficients. If we want a cutoff frequency of 4 Hz and we have a sampling frequency of 50 Hz, those coefficients will roughly be 0.6 and 0.4, making the transfer function:

$$H(z) = \frac{0.4}{1 - 0.6Z^{-1}} \tag{8.3}$$

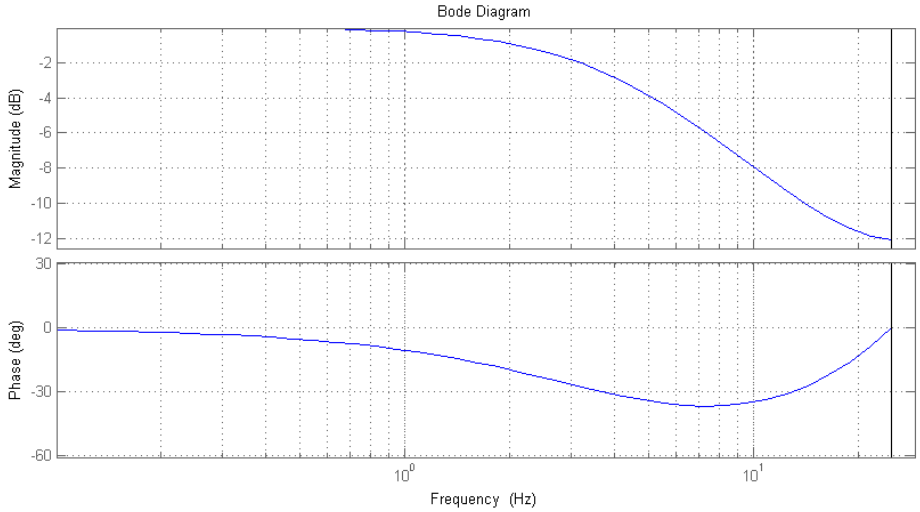


Figure 8.7: Bode plots of the digital filter. Black line signifies nyquist frequency.

The result with the filter was this:

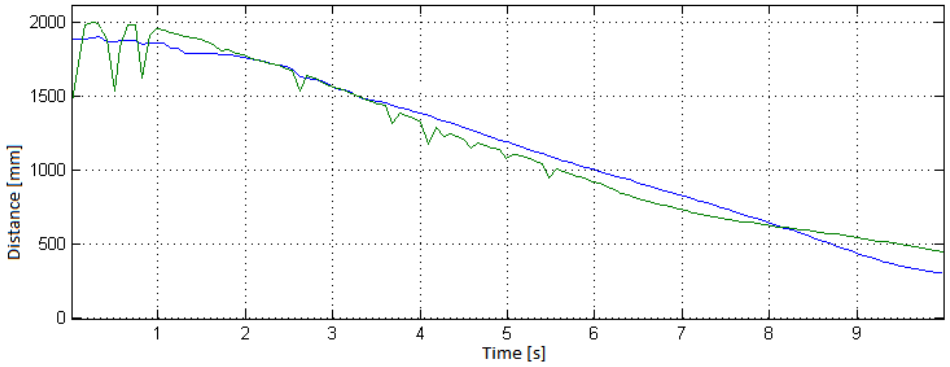


Figure 8.8: Measurement with filter (blue), plotted against measurement without filter (green). This was without propellers

It removes the fluctuations, but it is still bad when the propellers are added. To counter act this, a dampening layer (foam) was added between the sensor and the sensor mount, to increase the signal to noise ratio.

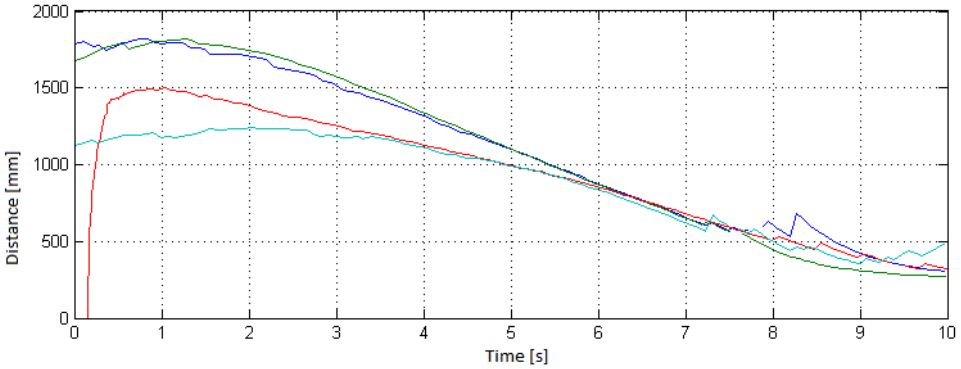


Figure 8.9: Measurement with engines, propellers, and extra padding at low power (blue) and medium (green), and without extra padding at low (red), and medium (Cyan).

This improved the distance sensors performance considerably. It is now reasonably usable for controlling distance, although it still has an error. However we will improve performance even more by moving the sensors further away from the engines by mounting them on stands. These stands also make it easier to take off, because they eliminate the "ground effect", which occurs when there is something underneath the air stream from the propellers within one propeller length [22].

8.2.3 The effect of the filter, on the controller

Because the filter was added to the control loop, it changes the model because the transfer function has to be added after the sensor in the feedback loop.

Pitch controller

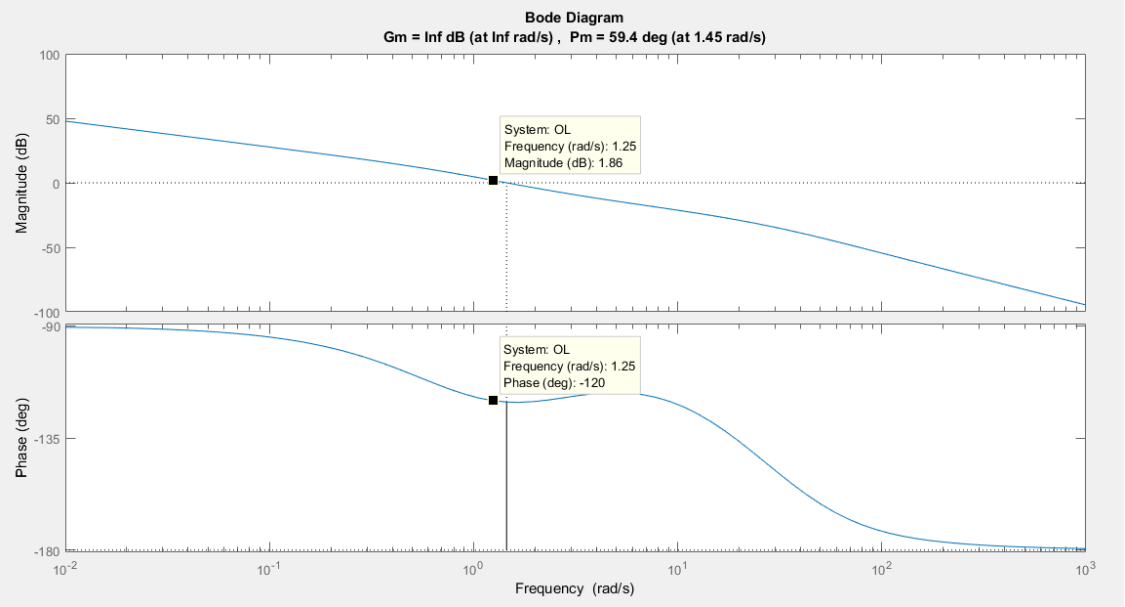


Figure 8.10: Bode plot of the open loop pitch system after adding the filter with old controller.

To compensate for adding the filter into the pitch system we need to lower the gain of the controller. That was done to keep PM at least at the 60 degrees. The result is visible on the plots below.

$$x = 10^{(\frac{1.9}{20})} = 1.2581.5/1.25 = 65 \tag{8.4}$$

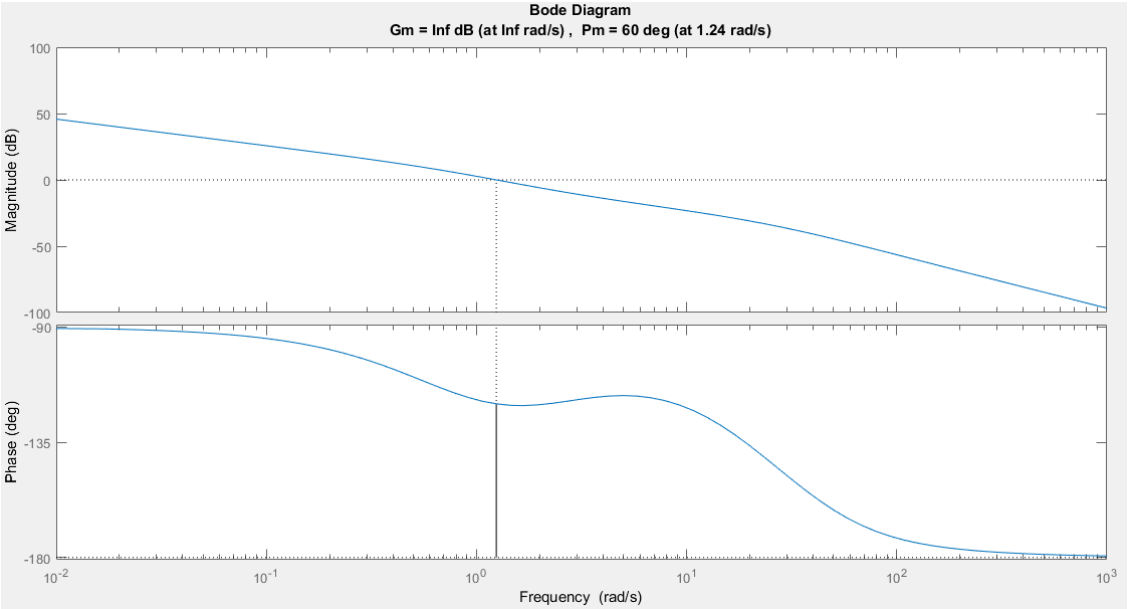


Figure 8.11: Bode plot of the open loop pitch system after adding the filter with new controller.

Final transfer function for the pitch controller:

$$D(s) = 65(1 + 0.45s) \tag{8.5}$$

Yaw controller

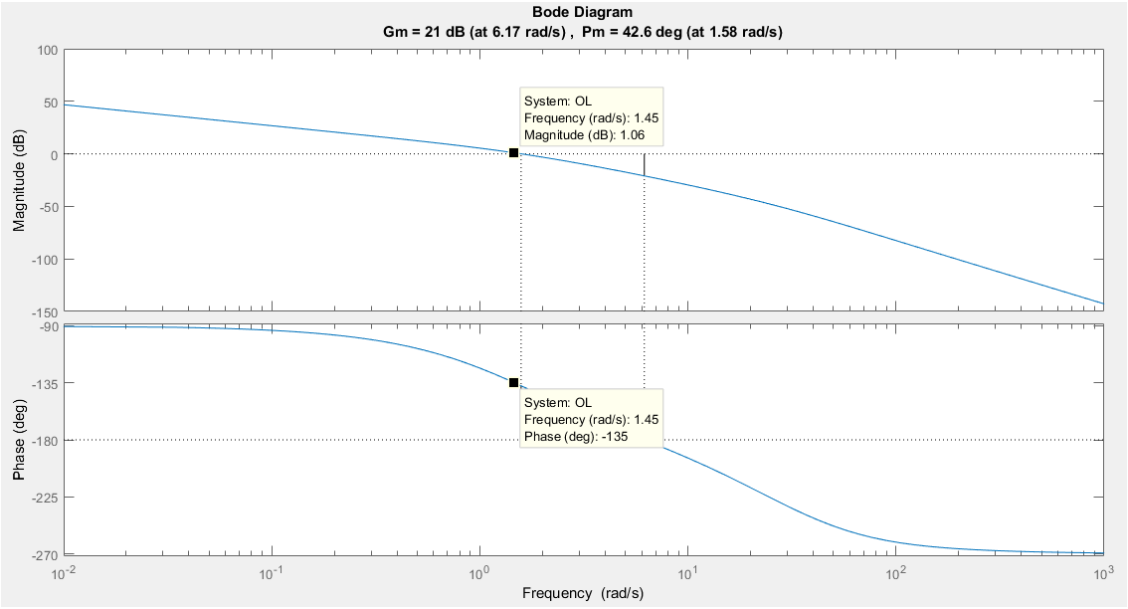


Figure 8.12: Bode plot of the open loop yaw system after adding the filter with old controller.

$$x = 10^{(\frac{1.06}{20})} = 1.13218/1.13 = 193 \tag{8.6}$$

In yaw controller adding the filter also required lowering the gain to Kp=193. The result can be seen on the picture below.

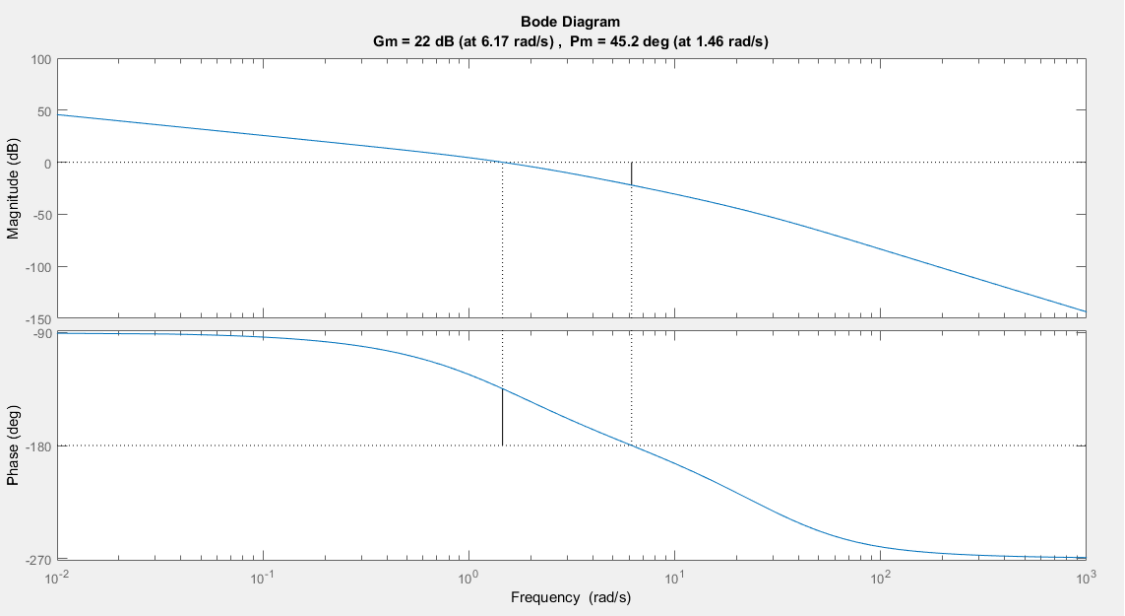


Figure 8.13: Bode plot of the open loop yaw system after adding the filter with new controller.

Final transfer function for the yaw controller:

$$D(s) = 194 \tag{8.7}$$

8.3 Functionality of quadcopter distance controller

In order to confirm that the controller outputs the correct values to the pixhawk FC, a preliminary test was made without flight. During the test of distance sensors, the controller was running. The output was captured and looks like this:

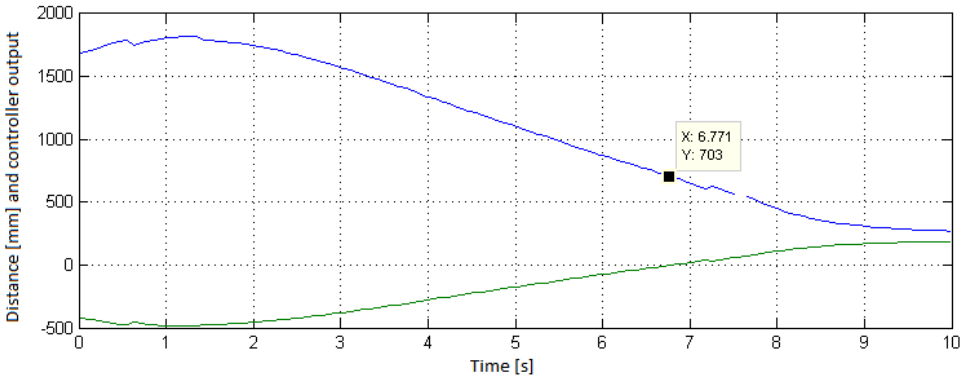


Figure 8.14: Test with controller output (green) and distance sensor output (blue). Reference distance set to 700 mm (marked).

At the reference distance the value changes to positive, because it wants to pitch up, to move away from the wall. This is the expected output, and the test is a success.

8.4 Step response of distance controller

This test was made to confirm that the whole control loop works, and that it performs as designed. At first it was attempted to do this test flying, but after several failed attempts, it was decided to mount the quadcopter on a test platform that was borrowed. This test platform was constructed prevent the quadcopter from crashing, while minimizing the dampening it adds to the system.

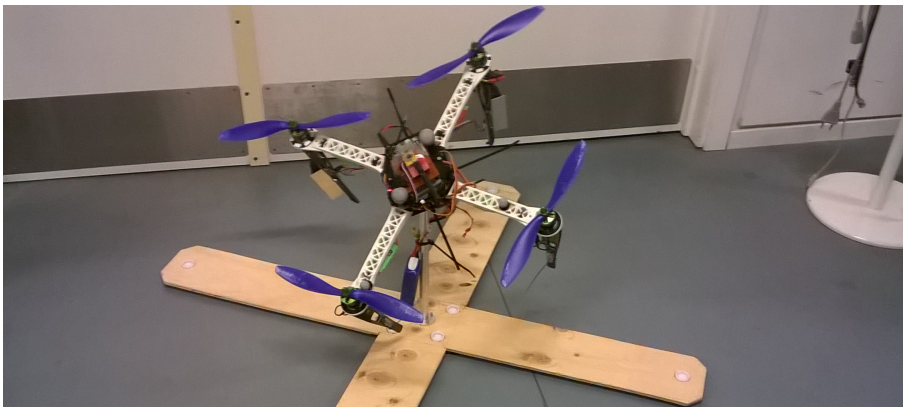


Figure 8.15: Quadcopter mounted on test rig.

The test rig consists of a platform with wheels mounted underneath, and a rod going up to a ball joint, on which the quadcopter is mounted.

8.4.1 Yaw controller step response

The first step response that was made, was made in the yaw direction. This controller is more advanced than the other, since the angle is calculated from the difference between

the two pitch sensors. The step response was made by setting the reference to -15 degrees, and then programming a step of 45 degrees (ending at +30) to a button on the remote control. The quadcopter is mounted with motion tracking balls, and the Vicon system is set up. A matlab script was used to capture the rotation of the quadcopter. Time stamps and a serial port was added to the script so that it can receive the step response command through an APC220 set up to listen. The quadcopter was then started, and the controller enabled, making the quadcopter snap to -15 degrees. Then the step is applied, and the quad rotates to the wanted position. This was implemented in radians.

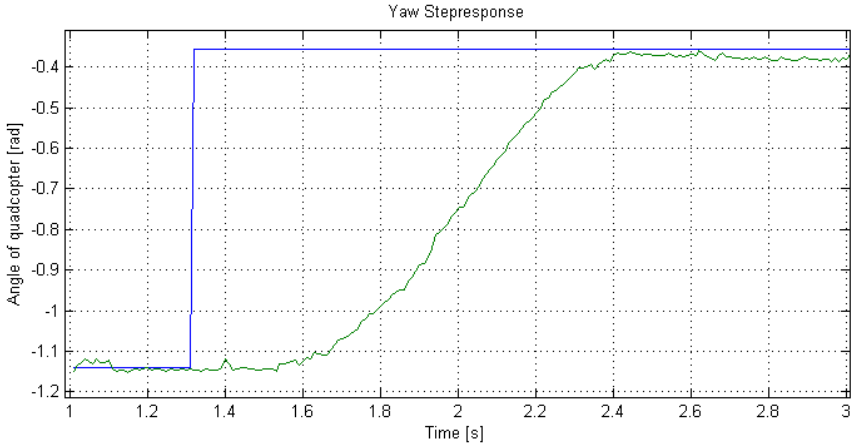


Figure 8.16: Result from yaw step response (green) and step input (blue).

The quadcopter does what it is supposed to.

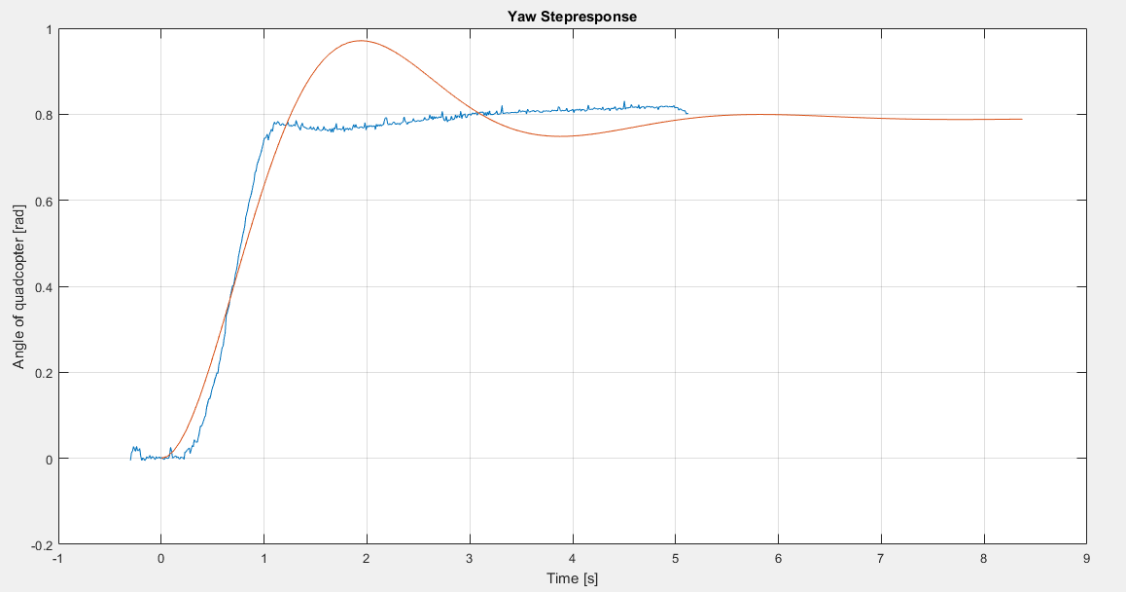


Figure 8.17: Comparison of step respones of the yaw system between real(blue) and simulated(red). Lack of overshoot in the real one may be due to the fact we mentioned earlier in the 'Simulation' part. We only control the input to the controllers inside pixhawk, not the speed of engines themselves.

8.4.2 Pitch controller step response

This step response was made flying close to the wall and then enabling controller with a pre-set reference distance. This was first tried on the platform, but the dampening in that direction from the platform was too great. The quadcopter did not move, and only slightly tilted. We tried to raise the gain, but only with ridiculous gain it moved, and even then it needed a bit of help to start moving with platform, however we spotted the tilting when it wasn't at the reference distance which sugest that our controller was designed correctly. The test was then tried flying but no data could be captured, due to several problems. First of all it was almost impossible to fly the quadcopter steady enough to do the test. It was constantly drifting in different directions. This is thought to be a problem with the Pixhawk FC, which has a bad sensor. It goes out of calibration fast, and sometimes even changes values for pitch/angle etc. while stationary. This drift is greater than the controller output, thus making the quadcoper behave erratically. Flight was tried with a greater proportional gain, to combat the drift, but this made it oscillate quite badly and the test had to be aborted quickly to avoid crash.
Another problem was that the quadcopter has to stay within sensor range of the wall, but the Vicon system has a "blind" area close to the wall, making any measurements unusable since most datapoints will be zero (when it cannot find the quadcopter). Due to these circumstances the test was dropped.

8.5 Test of communication link

In order to make sure the communication link works as expected, the latency was tested. This was done by measuring the time between sending a ping, and receiving the echo. The test was run in 10 iterations:

Table 8.1: Results

Test iteration	Latency [ms]
1	25
2	19,8
3	10,3
4	17,6
5	28
6	29
7	29,7
8	26,8
9	17,2
10	13,9
Average	21,73

On average the latency was 21.73 ms, which is certainly less than 100 ms, which was the requirement. The rest of the requirements were not tested because they are met by design.

8.6 Conclusions

The problem we wanted to solve with this project was, how to make flying measurements of hull thickness on a ship using a quadcopter. Specifically we wanted to design a system that can position a quadcopter for making measurements on ship hull. During the work with this project, the problem was analyzed, and certain design requirements were discovered, of those the most essential are that the quadcopter has to stay at a constant distance from the wall, and be pointing at it. From this a product idea was formulated, and with this idea in mind, the system was designed and implemented.
A remote control was designed, with the addition of a UI menu for editing parameters, and buttons for special functions. This proved a successful design strategy since it didn't only enable us to completely customize how the controls work, what extra functions to put in, but it also made it possible to monitor what the quadcopter is receiving, which was useful for testing. This enabled us to know exactly when the steps were applied etc. The communication protocol also worked out very nice. It was designed in a way that it was easily expandable (adding more package types with different lengths is easy), and also has capability for controlling multiple quadcopters with the same remote, since the packages are addressed (although this is seldom necessary at the same time).

During the design of the distance sensor, some problems came to light. The design was simulated in LT spice, and it worked fine, but when implemented, problems arose. The circuit was implemented on breadboard, and on two different PCB's but since the problems could not be worked out in time it was decided to scrap it, and use a store bought sensor.
The next problem was to implement that sensor. Initially it proved to be very slow because the program was waiting for echo response. This was solved by implementing scheduling and interrupts into the program. The final challenge was the interference and noise caused by the engines and propellers. This was overcome by use of filters and vibration dampening.
Had there been more time, another iteration of the distance sensor could have been built with better noise rejection (filtering), and a variable gain amplifier to increase the range

of the sensor. For example the amplifier could increase gain with time to catch echoes that travel far.
The controller was designed in several iterations. This was mainly because the sampling rate of the sensor was changed several times during design, which changed the model, and because a filter was later added. The final version of the controller turned out to perform differently in reality than in simulation. This was mainly due to the model not being very precise. With limited lab space it was hard to make a good test of the quadcopters dynamics. But in spite of this two controllers was made, and the yaw controller worked. The final results show that the use of HC-SR04 ultrasonic sensors for distance measurement, on a quadcopter, is not ideal because of their susceptability to noise, and limited range. Had we had a larger budget, in both time and finances, radar or laser could be used, or some combination including also a GPS. It serves as a proof of concept, and shows that the idea is possible, and that applications for quadcopters like this could become a reality in the future.

Bibliography

[1] Quora. *What Makes The Quadcopter Design So Great For Small Drones?* Forbes, Dec 23, 2013. URL : http://www.forbes.com/sites/quora/2013/12/23/what-makes-the-quadcopter-design-so-great-for-small-drones/.

[2] Rolf Diederichs. *Automated In-Mould Ultrasonic Wall-Thickness Measurement (IMM).* NDTnet, 13. December 1995. URL : http://www.ndt.net/article/imm/imm_e.htm.

[3] Wikipedia. *Ultrasound.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Ultrasound.

[4] Wikipedia. *Steel.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Steel.

[5] Det Norske Veritas. *Guide For Ultrasonic Thickness Measurements Of Ships Classed With Det Norske Veritas No. 10.* Det Norske Veritas, April 2009.

[6] Wikipedia. *Rangefinder.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Rangefinder.

[7] Wikipedia. *Stadiametric Rangefinding.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Stadiametric_rangefinding.

[8] Wikipedia. *Coincidence Rangefinding.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Coincidence_rangefinder.

[9] Wikipedia. *Laser Rangefinding.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Laser_rangefinder.

[10] Unknown. *Module 6 Laser Distance Measurement.* pe2bz.philpem.me.uk, 30. november 2015. URL : http://pe2bz.philpem.me.uk/Lights/-%20Laser/Info-999-LaserCourse/C00-M06-LaserDistanceMeasurement/module6.htm.

[11] Wikipedia. *Radar Distance Measurement.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Radar#Distance_measurement.

[12] Fujitsu Press Release. *Fujitsu Laboratories Develops Lower-Cost Millimeter-Wave Radar for Automobiles.* Fujitsu, 8. October 2014. URL : http://www.fujitsu.com/global/about/resources/news/press-releases/2014/1008-01.html.

[13] Andrew Tarantola. *This Millimeter Wave Radar Will Give Everybody TSA Vision.* Gizmodo, 28. March 2013. URL : http://gizmodo.com/5992504/this-millimeter-wave-radar-will-give-everybody-tsa-vision.

[14] Wikipedia. *Ultrasonic transducer.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Ultrasonic_transducer.

[15] Wikipedia. *Quadcopter.* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Quadcopter.

[16] vk. *Posts by comunity (webpage).* 30. november 2015. URL : http://vk.com/wall-72244369?offset=140.

[17] Robert. *A few flight related terms.* NoRunway, Jun 28, 2014. URL : http://norunway.com/wp/flight-related-terms/.

[18] Jeri R. Hanly & Elliot B. Koffman. *Problem Solving and Program Design in C, 7th Edition.* Pearson, 2013.

[19] Simon Haykin. *Communication Systems, 5th Edition.* Wiley & Sons Inc., 2010.

[20] Abbas Emami-Naeini Gene F. FRanklin, J. David Powell. *Feedback Control of Dynamic Systems.* Pearson, 2015.

[21] Ove Andersen. *Digital Filters Part 1 slides.* Sept. 24 2015.

[22] Wikipedia. *Ground effect (aerodynamics).* Wikipedia.org, 30. november 2015. URL : https://en.wikipedia.org/wiki/Ground_effect_(aerodynamics).

Appendix A

Appendix

A.1 Measurement of engine noise

This measurement was made as a part of designing the ultrasonic distance sensor. The purpose is to find out the noise level created by the quadcopter engines at the operating frequency of the ultrasonic sensor. Keeping this noise level in mind the receiver should be calibrated not to trigger on the noise, as it would cause a false positive.

A.1.1 Theory

We wanted to measure sound pressure generated by the engines, that means the amplitude and frequency spectrum, especially around 40 kHz as it is the frequency at which ultrasonic receiver is most sensitive. As the noise was fluctuating a lot, to obtain best result and the most accurate values we used the RMS voltage. The measurements were made for different engine power levels starting from 0 and increasing by 10 % until obtaining full power. We expect to see an increase in noise generated by the engines with an increase in engine power.

A.1.2 Method

In order to achieve the most reliable results, we put the quadcopter into an anechoic chamber (Fredrik BajersB4 Large Anechoic Chamber) with a microphone B & K 4135 next to it with a distance of 5 cm pointing directly at the engine closest to it. The quadcopter itself is placed on a speaker stand.
The microphone was then plugged through the anechoic chamber connectors, to a measurement preamplifier B & K 2636. To this was also plugged an external filter B & K1617 BP with 1/3 octave bandwidth set to 40khz, so that the noise measured will primarily be the noise at distance measurement signal frequency (40 kHz).
50dB gain was applied on the preamp, for better ADC utilization on the oscilloscope that was used to note the values (Agilent 54621A) for different levels of engine power.

Table A.1: Equipment

Name	AUC number
BK1617 filter	8455
BK2636 measurement amplifier	8451
BK4135 Microphone	6551
Agilent 54621A oscilloscope	33869
Large Anechoic chamber B4	

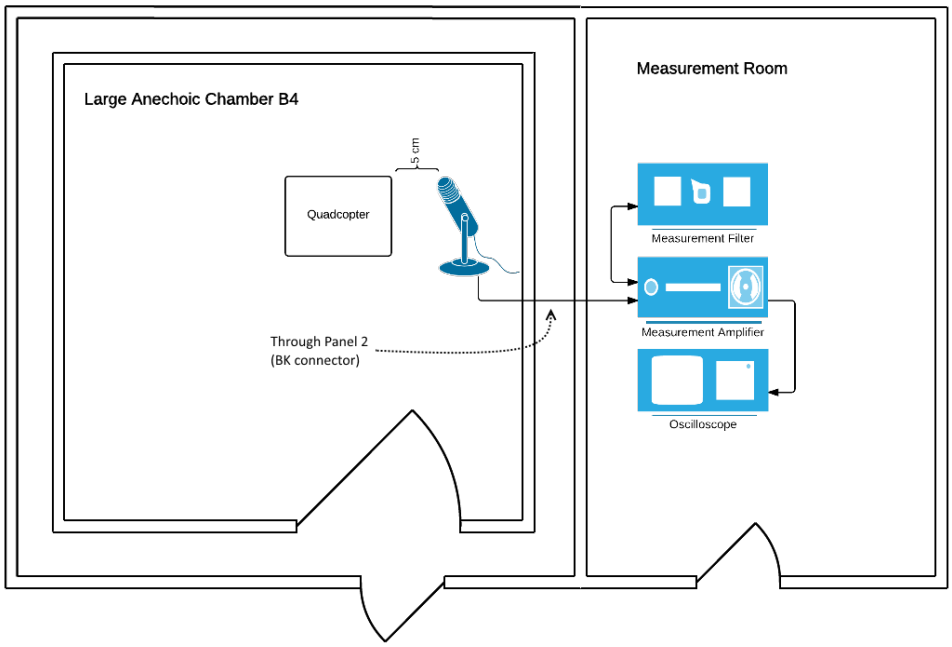


Figure A.1: Drawing of the measurement setup. Both quadcopter and microphone is on tripods.

A.1.3 Results

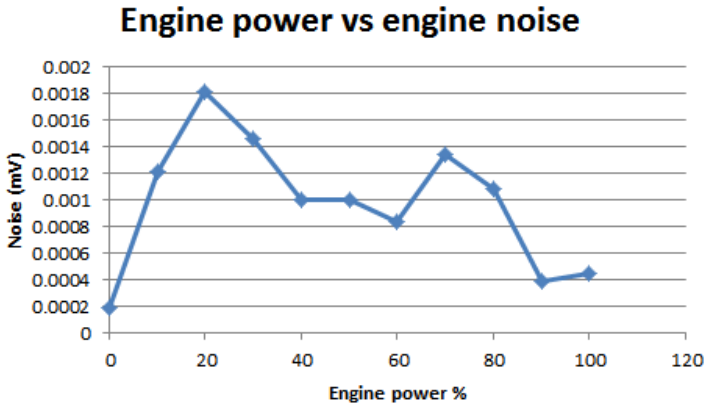


Figure A.2: Engine Noise in mV RMS (over 100 ms) recalculated to the sensitivity of the ultrasound microphone to be used in the circuit, plotted against engine power.

The voltages in this graph were recalculated to fit the voltages that the ultrasound microphone MA40S4S will see, by first converting to pressure with the measurement microphones sensitivity, and then back into volts using the MA40S4S sensitivity. This was done so the numbers can be compared with the received signal level (see appendix on measurement of MA40S4R ultrasound speaker).

Suprisingly the noise at 40kHz is not highest for highest engine power. It is actually highest for 20 percent engine power. It was observed that the audible noise was highest at 100% but, for 40kHz around 20 and 80 %, as seen in above figure.

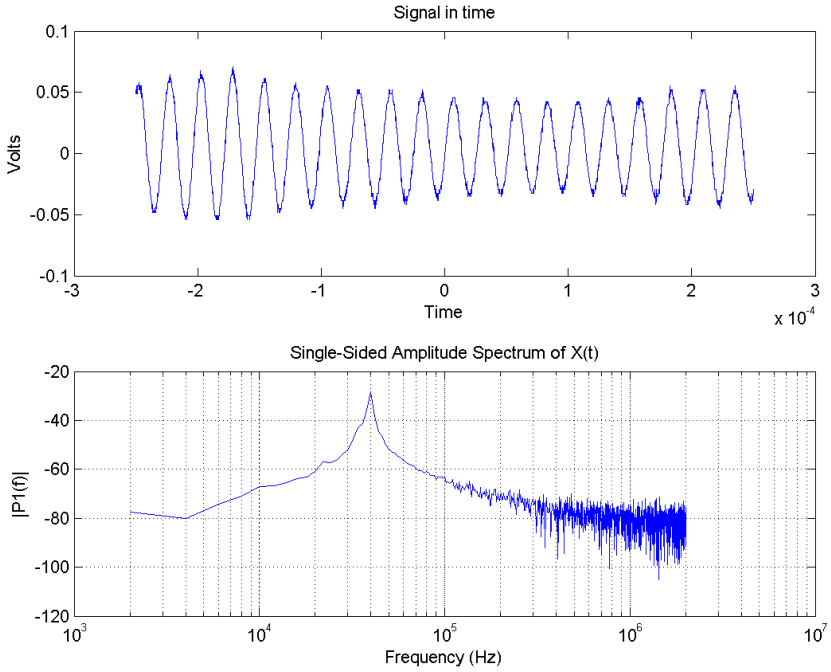


Figure A.3: Top: Time signal measured on oscilloscope (only 2000 samples), bottom: Frequency spectrum of signal using FFT. Signal was fed through a 40kHz Bandpass filter with a third octave bandwidth. The measurement amplifier was set to 50 dB gain, so levels are +50 dB.

As it is seen in the figure above, there is some noise at 40 kHz. This measurement is included as a reference to the first measurement where this filter is also applied.
An attempt was made to capture a broader spectrum of the noise, but the oscilloscope is limited by 2000 samples, which means some high frequency may be misrepresented, due to aliasing.

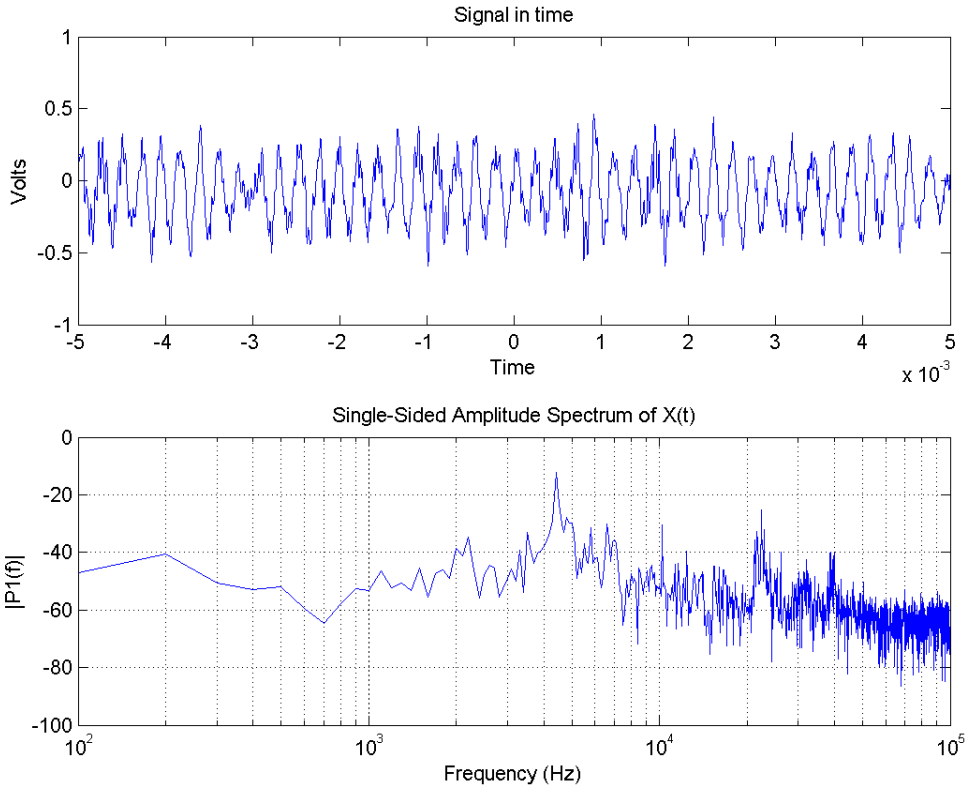


Figure A.4: Top: Time signal measured on oscilloscope (only 2000 samples), bottom: Frequency spectrum of signal using FFT. No filter on signal, but 50 dB gain.

This figure shows some significant spike in noise level at approximately 4.5 kHz. This must be filtered out at the receiver to avoid false positives.

Conclusion

There is some noise from the engines, especially at low engine power, which is when the quadcopter hovers. Therefore it is necessary to measure the MA40S4R ultrasonic speaker, to make sure the noise levels do not exceed signal levels. The SNR level is also important to know, to be able to set the sensitivity of the receiving circuitry. Therefore the MA40S4R will be measured also. The measurements were made without propellers, which could significantly alter the results. The propellers were dismounted for safety reasons, but it should be considered that the noise level might be higher when using the propellers.
The spectrum of the noise could not be measured conclusively due to there being no equipment for ultrasound in the acoustics department of Aalborg University. However the measurements show that there is noise outside of the signal frequency (40 kHz), which should ideally be filtered for better SNR.

A.2 Measurement of ultrasonic transmitter

For the project it was necessary to measure some parameters of the ultrasonic transmitter being used for the distance sensor circuit, which later turned out to be a failure. The important parameters are: frequency response (to see which frequency is best to use), and decay time, to see how fast the transmitter stops vibrating, and the signal levels, which tells us how much to amplify at the receiver. The decay time is used to set the detector so that it does not pick up the direct signal from the transmitter, but only the echo reflected from the wall. To do this the receiver is disabled for an amount of time while the transmitter is transmitting the signal and the decay time after the signal is off.

A.2.1 Method

In order to achieve the most reliable results, the measurements were done in the anechoic chamber with the same setup as in the previous test (engine noise).
To generate input we used Bang olufsen RC oscillator TG7, AUC 08495, which was connected directly to the MA40S4S ultrasonic speaker. To measure the output amplitude values we used Agilent 54621A Oscilloscope, as in the previous test.
To obtain the frequency spectrum of the transmitter we used a sine wave with an amplitude of 2.5 V peak to peak while changing the frequency of the signal. Then, for each frequency, we measured output from the microphone with 20dB gain in preamp.
For measuring transmitter sensitivity we used the same supply generator but this time using exactly 40khz frequency. Here the measurement amplifier is also set to 20 dB gain, but this is removed from the results before plotting.

A.2.2 Results

First the response of the ultrasonic speaker was measured.

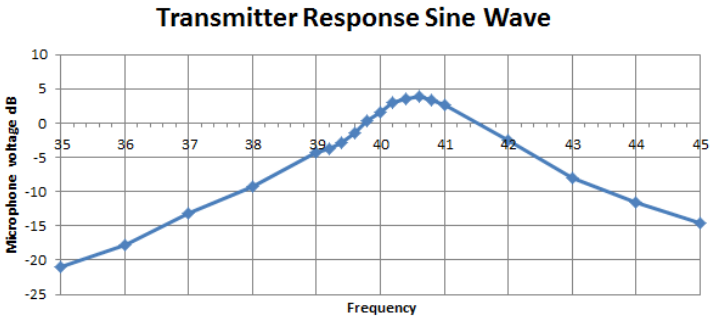


Figure A.5: Frequency response of the MA40S4S ultrasonic speaker.

Output level is from the measurement microphone, and was not converted since only the difference in level across frequency is interesting.

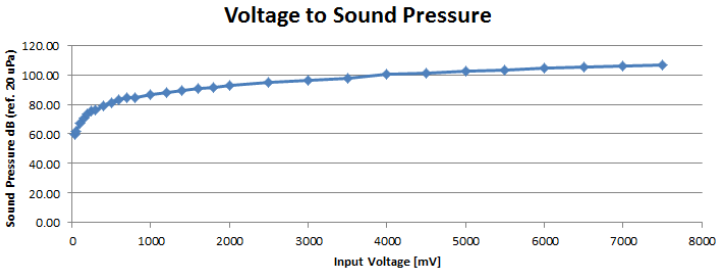


Figure A.6: Voltage to sound pressure ratio at 40 kHz.

This was made by calculating the voltage into pressure using the measurement microphones sensitivity, and then calculating dB referenced at 20 μPa which is the standard sound pressure reference.
This measurement tells us how far we can turn up the input voltage before we get dangerous sound pressure levels, which is about 100 dB at these frequency according to the UK Health Protection Agency[3].

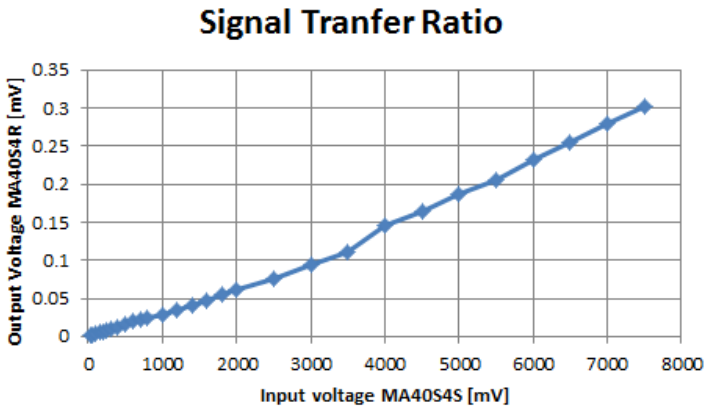


Figure A.7: Input voltage to output voltage at the MA40S4R ultrasonic microphone.

This was calculated using sensitivity of the MA40S4R from the datasheet.
According to this graph the maximum signal level we will receive when keeping at a safe transmit level, is around 0.1 mV, and this will decrease with distance. This means the signal should be amplified quite a lot (maybe 80 dB, which gives 1 volt).

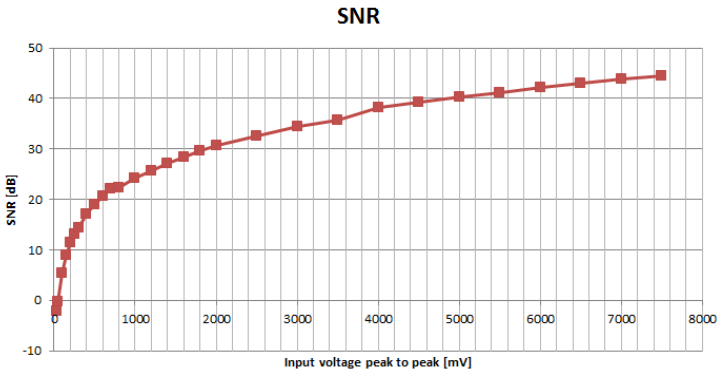


Figure A.8: Signal to noise ratio calculated from the noise levels in the measurement of engine noise.

At 2.5 volts peak to peak transmitter voltage, the SNR will be 32.5 dB.

Decay time

In order to not get a false positive in the start of the distance measurement it was measured how long it takes for signal to die out (this was in anechoic chamber, so only the echoes are from the microphone in front of the MA40S4R, but it does not stop vibrating immediately) The microphone was placed 5 cm from the transmitter.

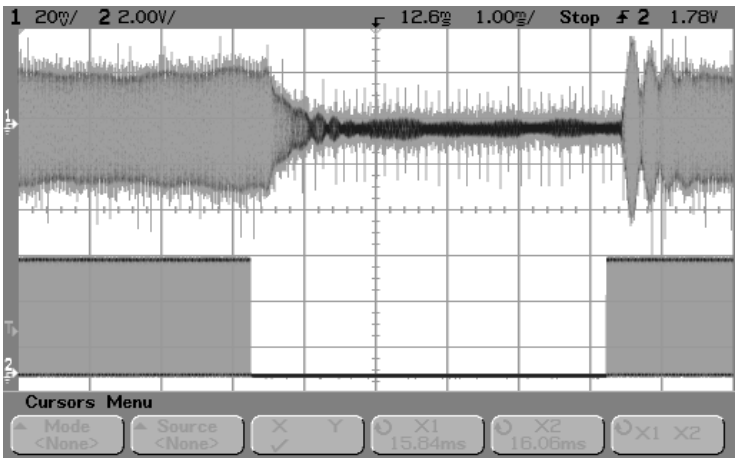


Figure A.9: Screencapture from oscilloscope showing a part of the decay time measurement. Top: Microphone signal, Bottom: Input signal. Notice the difference in time before the microphone receives the signal, and after the signal stops, before the microphone stops receiving it.

On the oscilloscope the time intervals were measured with cursers, and the result is: Delay before the signal is received: $236\mu s$, Decay time plus delay: $1488\mu s$. Decay time:

$$T_D = 1488 - 236 = 1252 \quad [\mu s] \quad \text{(A.1)}$$

It is chosen to wait for at least 1.2 ms before listening for echo. The short range limmit of the distance sensor will then be:

$$l_{limmit} = \frac{340.29 \cdot 0.0015}{2} = 0.20m \tag{A.2}$$

This is really far, and it must be tested how much the short range capabilities can be improved before the measurements become false, by adjusting this time. Another option is to somehow shield the receiver from the transmitter thus eliminating the need for a blanking time altogether. This could be done by putting cones on the transducers, thereby enhancing their directivity.

A.3 Measurement of Quadcopter step response

In order to be able to make a model of the system (the quadcopter), its behavioral parameters should be measured. The theory is that it is possible to estimate the first order transfer function from the step response alone. This part is described in the report, in this measurement report the step response itself will be measured.

A.3.1 Games on Track measurement

In order to get the position of the quadcopter over time a system called "Games On Track" (GOT) is used. This system consists of a "position sender", which is the entity whose position is tracket, "position satellite" which receives a signal from the sender, and with many of these the position of the sender can be triangulated. Then there is the position master, which is a unit that connects to all the satellites and senders, and a PC (to which it provides position data). The system uses a combination of ultrasound and RF technology.
With this system an SDK (written in C#) is included for a program for getting the position values to a COM port on the PC to which it is plugged (and also display the position on screen). This program had to be modified slightly to also output the current time, and output on the correct COM port. The values are then written serially to an Arduino Mega. To this Arduino, an APC220 (Radio transceiver) is also plugged, which is set to the same settings as the one the remote controller uses. This is used to listen to the control inputs. This enables us to see exactly when the step is applied, and the output (quadcopter movement) in the same timescale. To do this the Arduino Mega was loaded with a program that decodes the protocol, and converts the control values to ASCII. All these programs are included on the CD. From the Arduino Mega the input and output values are printed serially through USB to another PC, where the data is parsed to a comma separated value file.

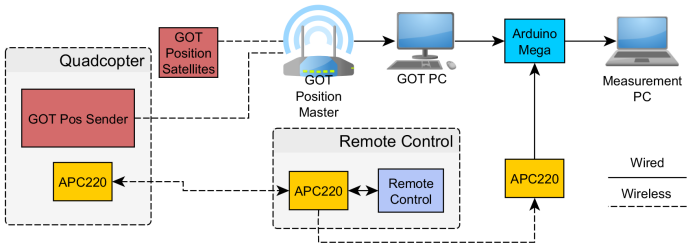


Figure A.10: Drawing of test setup.

The input unit step function is generated by the remote control. In the menu, a ”test signals” sub menu is added, which has an option called ”step input”. This function applies a pitch forward for a few seconds, and then it breaks the quad (to not hit the wall). The amount of pitch given for the response was decided experimentally to be 100 (the level value is 513, so the step forward is 413). The duration is chosen by the pilot. He can hold the step button for as long as possible before a crash is eminent.
The position is output as XYZ cartesian coordinates. The x and y coordinates are added with the minimum value, to avoid zero crossing. The length of the x y vector is then calculated, which is the position of the quadcopter in the direction it is moving (which is somewhat diagonal to the GOT coordinate system because there is more space in the diagonal of the room).

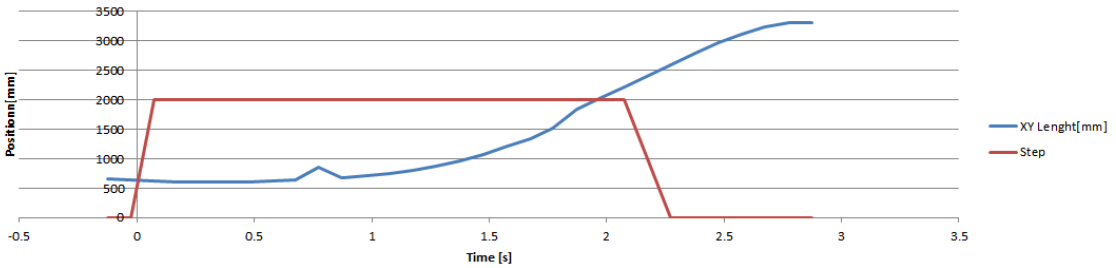


Figure A.11: Step from GOT system (step input has been scaled).

The position of the quadcopter changes as it accelerates forward, when the pitch is ”stepped”.

The results from GOT proved to be insufficient due to the low sampling rate, and the error samples. It was chosen to repeat the tests using the Vicon motion tracking system.

A.3.2 Vicon motion tracking measurement

The Vicon motion tracking system consists of some cameras mounted in different locations in the lab, and a piece of software on the PC. The cameras pick up marker balls, that have a special reflective surface. Five marker balls are placed on the quadcopter on different locations. These are then grouped as an object in the software, which then enables the system to track the objects position and rotation.

A Matlab script for getting the data is used. This software is modified to give out XYZ position, timestamp, and radian rotation in all three directions. To be able to get the control inputs in the same timescale, a serial port is opened to the same Arduino Mega as in the previous test. The control input data from the Arduino Mega is added to the same matrix as the other measurement data, and after the test has finished the matrix is saved as a comma separated data file.

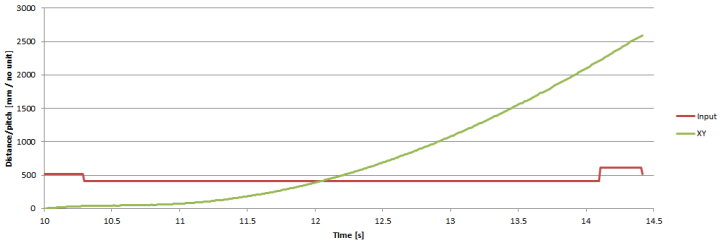


Figure A.12: Step from Vicon system, in pitch direction. This is the best step. The other measurements were not usable.

It is immediately apparent that the Vicon system is both more precise and has a higher sampling rate (100 Hz), than the GOT system. This is the most successful step of the three that were made.
A step was also made in the Yaw direction to determine a transfer function in that direction:

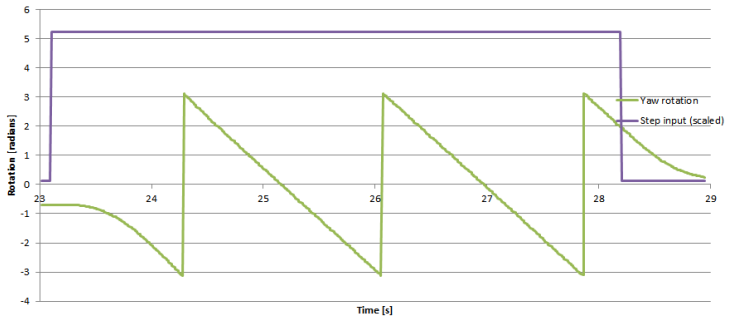


Figure A.13: Step from Vicon system, in yaw direction. The rotation overflows. Step input is only plotted to indicate the step time, but is not in radians.

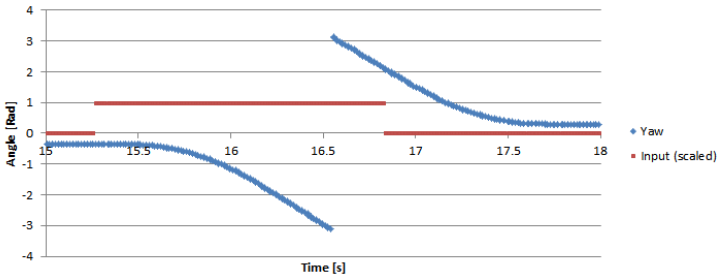


Figure A.14: Second step from Vicon system, in yaw direction.

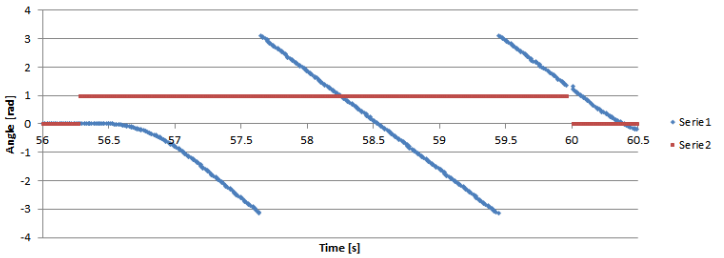


Figure A.15: Third step from Vicon system, in yaw direction.

These step responses will be used to determine the transfer functions that describe the quadcopter in the pitch and yaw direction, which is described in the report, in the control and automation part.