Towards indoor and outdoor accurate flying using AutoQuad

University:
University of Southern Denmark

Author:
Mathias Mikkel Neerup

Bachelor Thesis

In coorperation with:
Jussi Hermansen
Info@viacopter.eu
Cottagevej 4
3300 Frederiksvaerk

Supervisor:
Kjeld Jensen
kjen@mmmi.sdu.dk
Maersk Mc-Kinney Moeller
University of Southern Denmark

June 1, 2016

Mathias: : A bachelors thesis (15 ECTS) report from a single student should have a length no more than 35 pages plus maximum 10 pages of appendices.

Mathias: : Ja, men gør det som elektronisk bilag (pdf) og beskriv i rapporten, at den ligger der. Marker det evt. dine bidrag med farve, så det er meget let at se.

Todo list

Abstract

Until now drones keeps getting bigger and larger to carry bigger batteries with more capacity and to lift heavier payloads. This leads to drones getting less efficient, less responsive and gets more dangerous. Instead it has become popular to make drones smaller and increase the number of drones needed to solve a task.

(Materials & methods)
This thesis describes how to make three drones follow a leader drone with a preprogrammed path as an example of drones cooperating. A Linux PC running MarkerLocator tracks each drones position and wirelessly transmits, using Xbee, the drones positions to all drones.The position of each drone is spoofed into the drone using the CAN-bus and thereby overwriting the onboard GPS. An outdoor test has been made using the onboard GPS to test the leader-follower algorithm in a bigger scale. A small PCB has been developed and mounted on each drone to route packages from the Xbee module to the CAN-bus of the drone and to measure the local altitude of the drone using a ultrasonic sensor. The PCB carries an AT90CAN128 as microcontroller which build-in CAN support making it obsolete to carry an external USB CAN-controller.

(Results -¿ discussion)
The accuracy of the vision based localisation is measured using a laser pointer pointing out the drones 2D position on the floor making it possible to measure the variance of the drones position. The leader-follower algorithm was also tested outside using the onboard GPS. The performance of the leader-follower algorithm is measured and discussed using plots that reveals the distance between the drones.

(Conclusion -¿ perspective)
It is shown that it is possible to implement the leader-follower algorithm using a vision and ultrasonic based positioning system. The distance between all drones when flying indoor was +/- 10 cm which is less than the maximum accepted error. It was possible to add 5 follower-drones without editing the code showing it is a generic system The system can further be used to indoor testing of navigation algorithms and explore the many possibilities drones has to offer. If drones at some point needs to fly indoor to help eg. Mobile robots navigating, vision might be a way to obtain a absolute indoor position for the drones.

Resumé

abstract på dansk

Reading Guide

This thesis contains three parts.

Chapter 1 gives background information about the purpose of this project. It starts giving an overview of the topic and ends up focusing on why it makes sense to make the project and what other universities have done. It further contains related work, problem statement, defined hypothesis and assumptions and limitations. A image of the final extension board can be seen to make it clear to the reader how the developed extension board and AutoQuad Ladybird drone looks

Chapter 2 describes materials and methods used in this thesis. It explains how and why the following design chooses where made concerning firmware as well as developed PCB. Each chapter starts with an introduction and ends with test results evaluating weather the choices made works or not.

Chapter 3,4,5 concerns showing, analyzing and discussing results. Evaluation of the design choices made throughout chapter 2 with respect to the hypothesis is summarized in the conclusion.

On figure 1 the different parts used in this thesis are highlighted.

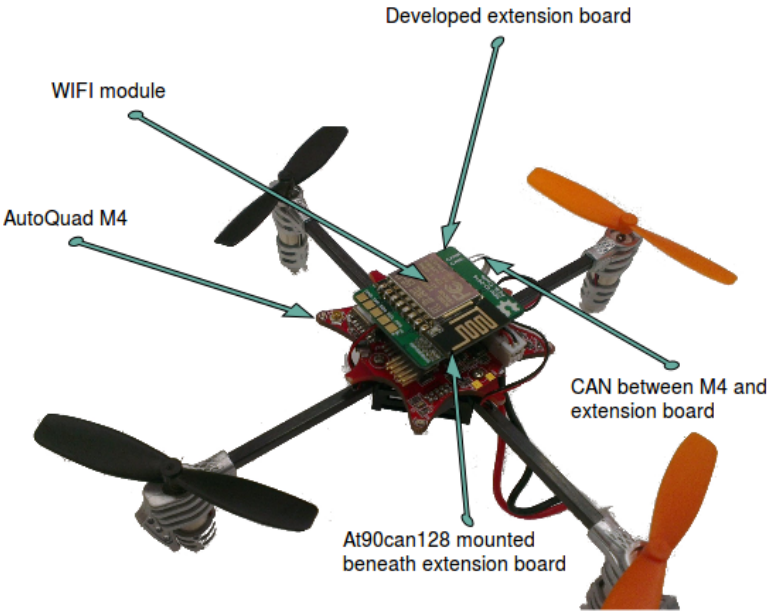


Figure 1 *Ladybird M4 drone with developed extension board*

List of Abbreviations

AQ	AutoQuad Flight Pilot

Preface

Throughout the project period my classmates has been at great help to discuss and generate ideas.

Thanks to the people listed below.

- My supervisor Kjeld Jensen for providing solutions and help when needed.
- Developer of MarkerLocator Henrik Midtiby for helping me understand and adding functionality to his software.
- Carsten Albertsen for designing and soldering the extension-board.
- Friend Morten Albeck Nielsen for meeting once a week to help generating ideas, debugging, sparring and proof reading the thesis.
- Friends Mads Tilgaard Jensen, Eskild Andresen & Michael René Andersen for listening to technical issues, evaluating solutions and carrying out tests.
- My girlfriend Anna Riisberg Soerensen for proofreading the thesis and understanding the time required throughout this project period.

Table of Contents

Chapter 1

Introduction

This chapter will give the reader background information about UAVs and point out a problem SDU is currently facing in two ongoing projects. Further more it will describe the focus of this project and how others have managed to solve it.

Mathias: Introduction: Indøres navigation http://innovationsfonden.dk/da/case/droner-rykker-indendoers-med-dansk-teknologi-0

Background

UAS is an emerging technology used in lots of different areas[1].

Especially quadroters also refereed to as multiroters have started to gain a lot of attention. This is, among other reasons, because it has become cheaper to produce the hardware needed to build a quadroters. Microcontrollers have become powerful enough in order to make it possible to implement control and navigation algorithms on a quadroters.[2]

A multiroter works by having three or more rotors mounted with equal distance from the centrum of the multiroter. Not all roters of the multiroter spins the same way in order to compensate for the torque generated by the multiroter. A multiroter is usually equipped with a flight controller, battery, three or more electronic speed controllers and rotors. The multirotor balances using its flight controller that controls the velocity and thereby the thrust of each rotor. Usually a rotor is brushless which means it has three phases that needs to be turned on and off, so called commutation, at the right time. The job of the Electric Speed Controller is to do the commutation depending on the speed sent from the flight controller.

One of the challenge concerning multirotors is the amount of energy they are capable of carrying. If the drone is equipped with a heavy camera or other kind of payload, then the flight time begins to decrease. By looking at the nature, one can see how small animals like ants and birds manage to cooperate and thereby build or move bigger things that they would not be able to do on their own. This way of small independent, decentralized units working together is called a swarm[1].
By making multirotors smaller, they get more efficient, their flight time increase and they get cheaper but of the cost of their ability to lift[3]. Therefore an idea would be to make small multirotors cooperate to solve more heavy and complex tasks.

Multirotors can be used in a wide range of applications. If the multirotor is equipped with a camera it can quickly provide an overview of a fence, wiring, lamps etc. Multiroters is usually used in applications outside which among other reasons, might be caused by the requirement of

[1] https://en.wikipedia.org/wiki/Swarm_intelligence

GPS signal being available. Most multicopters are capable of flying indoor however they often has little knowledge about where they are since they lack GPS indoor. It limits the applications multirotors can be used for indoor. A research project named UAWORLD has just started which will focus on doing indoor flying.[2]. They will focus on making multirotors robust and safe enough in order to let multirotors fly indoor. They are going to develop the system in cooperation with GamesOnTrack.com which delivers the indoor 3D localization system. GamesOnTrack's localization system uses beacons and triangulation to localize objects.[3] The are sure that drones will be used indoor at hotels, hospitals, schools and offices.

The GPS available outdoor has an accuracy of 3.5 meter [4] which might be adequate depending on the applications. In most flight controllers, the GPS is fused with onboard sensors like accelerometer and gyroscope however the positioning is still not adequate if the multirotor is supposed to fly close to an object within centimeters.

Even though GPS is usually available outdoor there might be places where no GPS signal is available eg. In forests or cities with high buildings. In indoor as well as outdoor applications where absolute positioning is required but GPS is not available alternative solutions such as radio triangulation, vision or totalstations could be used. However most of the multiroters available on the marked only supports using their build-in GPS which might be a limitation if GPS is not available but accurate flying is necessary.

At the time of writing, SDU is collaborating with HCA airport to find anomalies in fences. *Airports are burdened by a number of required inspection tasks to maintain a high level of safety and security. Some of these tasks may advantageously be performed by a drone rather than the current manual labour, and this will save significant resources. In this project they are targeting a specific need for frequent inspection of the fence surrounding the airport shown in figure 1.1a. The inspection concerns fence holes or similar anomalies. We hypothesize that a drone is capable of unsupervised autonomous inspecting the airport fence detecting small holes down to a radius of 5 cm.* [5]

(a) *Part of the fence surrounding HCA Airport.*

(b) *Drone spreading ladybirds and gall midges to avoid spreading pesticides in organic crops*

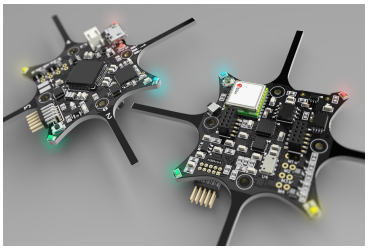
In order to avoid flying into the fence and to fly accurate enough for the camera to film, it is required to have an vertical accuracy of 2-3 cm [6]

[2] http://innovationsfonden.dk/da/case/droner-rykker-indendoers-med-dansk-teknologi-0
[3] More information is not available online
[4] http://www.gps.gov/systems/gps/performance/accuracy/
[5] Application sent to Energi Fyns Udviklingsfond which has been granted, Ansøgning 2015-01-30 Energi Fyns Udviklingsfond-1.pdf on USB
[6] Henrik Egemose Schmidt - Drone inspection of fences

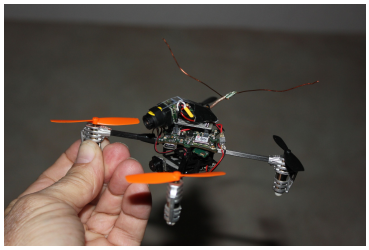
In another project, SDU is working together with Ecobotix ApS, Aarhus Univercity and EWH BioProduction ApS, to avoid the need of pesticides in organic crops. Pesticides are used to kill pests from the crows. When using pesticides a health risks exists for those who eat the crops. The use of pesticides also reduce the amount of diversity in nature since it is also killing beneficial organisms and potentially harm other animals in the food chain like birds. If nothing is used the crops might die, and that is very expensive to the farmer. The project has been granted 8.356.126 kr. from GUDP [7]
They intend to use multirotors to spread bugs like ladybirds and gall midges, over the crops to eat pests as shown in fugure 1.1b.
It is required the drones fly approximately 1 meter above ground at 1.5 meter/sec. It is therefore necessary to use an RTK-GPS in order to avoid hitting the ground in case of bumps in the fields. [8]

Many different flight controllers exists on the marked. SDU UAS has decided to use AutoQuad since, among other reasons, the code is Open Source [9] and the multiroters show reliable and steady flying. An AutoQuad multiroter delivered by ViaCopter[10] usually comes with with the AutoQuad flight controller M4, Electronic Speed Controllers(ESC32) and rotors. One of ViaCopters smallest drones is delivered without Electronic Speed Controllers since H-bridges used to control the velocity of the motors are build into the M4 flight controller. ViaCopter can deliver drones in different sizes but their flight controller is the same. This means that code developed for ViaCopters Ladybird can be deployed on a large drone and it will continue to work[11]. However AutoQuad does not support any other source of absolute positioning other than using its onboard GPS.

Because of the research-projects at SDU, there is a need for being able



(a) *M4 flight pilot*



(b) *Ladybird with mounted M4*

(c) *Eduquad with M4 + ESC32*

Figure 1.2 *Pictures of AutoQuad hardware*

[7] http://naturerhverv.dk/tvaergaaende/gudp/gudp-projekter/2015/oekodrone-skal-sprede-mariehoens-i-stedet-for-pesticider/ last visited 18-04-2016

[8] Jensen, K.; Larsen, R.; Laursen M.S.; Neerup, M.M.; Skriver, M. and Jørgensen, R.N. Towards UAV contour flight over agricultural fields using RTK-GNSS and a Digital Height Model. Accepted for oral presentation at CIGR-AgEng June 2016.

[9] Quatos, which is their control algorithm is not Open Source.

[10] https://viacopter.eu/

[11] Assumed the developer did not break important lowlevel tasks

1.1 Problem Statement

The current AutoQuad version does not support any other source of global positioning than the onboard GPS and thereby does not support accurate flying better than GPS and if GPS is not available, there i is no alternatives supported. For AutoQuad to be used in applications requiring high accuracy or where GPS is unavailable this functionality is needed.

1.2 Related Work

In order to find relevant research about drones flying indoor, a few search phrases was conducted. The following keywords were used to create different phases: Indoor, environment, swarm, localization, AutoQuad, quadrotor, mini UAV, test facilities, ETH, accurate, RTK, totalstation. Based on the keywords a few papers was deemed relevant to the project and has been combined in order to give the reader an overview within the field of this project.

Developers of AutoQuad did previously try to implement RTKLib [12] in AQ. Unfortunately they did manage to make it work as expected [13].

One of the big players within the field of indoor navigation and controlling multiple drones accurately is the university ETHzûrich and their Institute for Dynamic Systems and Modelling [14]. They have developed a test flying area they call "Flying Machine area" which provides facilities for doing prototype testing of new control algorithms [4]. The FMAs dimensions is 10*10*10m and provides nets to protect people and mattresses to protect the drone if a crash. The FMA has further been developed into a mobile installation to be used in demonstrations in Europe and North America. One of their demonstrations where used in a TED video about multiroters and their capabilities [15]. The multiroter usually used in the FMA is Ascending Technologies' Hummingbird with custom wireless communication and electronics.
They have build it as a module design in order to be easy to replace parts of their system by simulations and to make it scalable. One of their modules is a copilot that implements an accident handler in case of user-code crashing or sending invalid commands to the drones. They are using UDP multicast packets as communication between ground computation and flying objects. The use of UDP multicast packets between modules since to makes the system more simple but also to avoid the need of buffers to handle unsuccessful transmissions and retransmissions.
In order to detect the quadroters they are using a commercial motion capture system. Three reflective markers is mounted on each flying object in order to obtain attitude and position. They are using three cameras to reduce the risk of false positive even though two cameras would be enough to get a flying objects 6D position.

[5] proposes a more simplistic approaches to do indoor navigation. They use bluetooth 4 to communicate between their multiroter(Rolling Spider) and an android phone which controls the multiroter. They have mounted a camera on the ceiling to detect the target and the flying multiroter. By doing background subtraction they can detect where the drone is in the frame by subtracting the background from each frame [6]. By doing a convolution sum, the targets can be

[12] http://www.rtklib.com
[13] Jussi Hermansen, owner of http://ViaCopter.eu
[14] http://www.idsc.ethz.ch/research-dandrea/research-projects/aerial-construction.html
[15] https://www.ted.com/talks/raffaello_d_andrea_the_astounding_athletic_power_of_quadcopters

located. By analyzing the pixels around the location of the multiroter, they can get the heading.

[7] proposes a framework to accelerate the process of prototyping multiroters behaviors. Their framework is designed for a swarm of drones to fly in a environment with obstacles. One of their design requirements is, that the framework should be highly decoupled from the application the researcher is testing in order to speed up the development process. Position estimates is obtain by using onboard IMU and optic flow. To avoid expensive motion capture systems they have used markers that can easily be recognized by cameras mounted on the drones to get a absolute 3D estimate. Each obstacle got a ArUco-marker [**?**] that can be detected by the front camera mounted on the multiroter.
They have decided to use ROS as middleware to provide generic interfaces between the modules used in their framework. Different multiroters can be used as long they use the same interface. Communication between multiroters and ground station(if used) is done using WIFI.

The most common type of indoor localization is using vision where the camera is either mounted in the environment or where the drone is equipped with cameras to obtain position estimates.
[8] uses a different approach where they use a *Robot Sensor Network* to map the environment. The idea is that each drone can either be a beacon or explorer. Each drone alternates between these two states. Beacons stays still below the ceiling without moving while explorer flies around to unknown locations. Beacons emit IR light in order to triangulate beacons position. Beacons detecting unexplored locations calls for explorer that will become beacons and so forth until the environment is mapped. To synchronize the beacons 2.4 ghz WIFI is used. When the environment is mapped, graph searching algorithms can be used to find a path through the environment.

1.3 Hypothesis

If each drone's 2D position is obtained using indoor localization and sent into each drone using CAN at the same frequency as an onboard GPS, then it is possible for at least 3 drones to follow a leader drone with a preprogrammed flight path and keep a distance of 50 cm within plus minus 10 cm to the leader and its neighbours.

1.4 Aim of project

The aim of this project is to test the hypothesis by making an indoor flying environment to make 3 multirotors follow a leader multirotor. Since a need from SDU has arisen of sending RTK GPS coordinates into AutoQuad, this will also be used to test and to contribute to a scientific paper.

Chapter 2

Materials and methods

This chapter concerns the most important parts about how this project has been implemented. It has been split into several sections.

2.1 System architecture

This section describes the over all system architecture indoor as well as outdoor. The section goes in details about the components used, how they are connected and describe the information flow through the system. The design choices and implementation details will be covered later in this chapter

Since this project concerns injecting positions from a RPi outdoor and the extension-board[1] indoor, the system architecture description has been split into two.

2.1.1 Indoor

Figure 2.1 and 2.2 shows the components used, how they are connected and how the information flows indoor and outdoor, respectively.

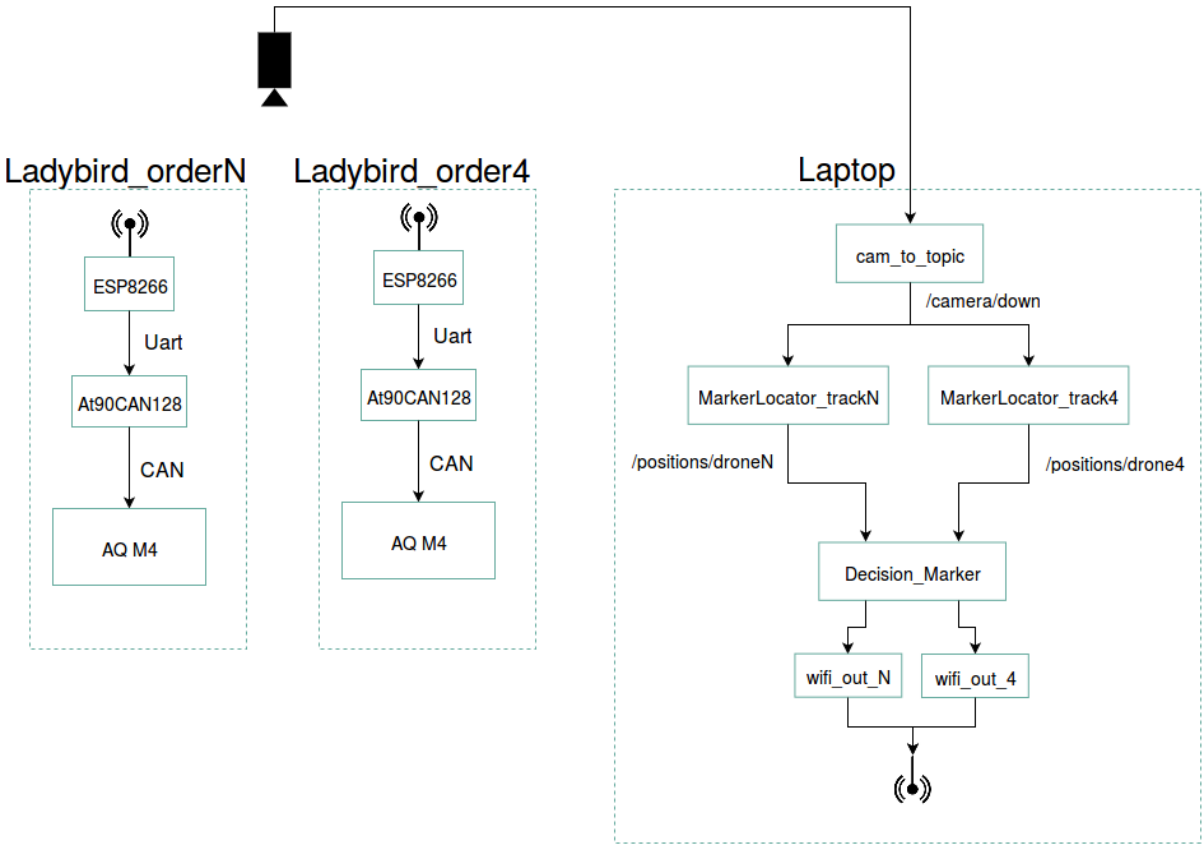


Figure 2.1 *Diagram shows information flow when using indoor flying. Server detects the drones position, sends it through the ROS nodes and transmits the GPS positions using WIFI to the drones. Each drone receives its position and sends it using CAN into M4 flight controller*

Indoor When used indoor, vision[2] is used to obtain the position of the drones with respect to the camera.

A Logiteh C930[3] was mounted below the ceiling to detect the drones when flying.

[1] **??**

[2] Described in section 2.5

[3] The camera is used in other courses to track mobile robots so the author did not have any influence on which camera to use

ROS[4] is used as middle ware on the laptop as inter-process communication. By using ROS it is easier to debug since each node[5] can be isolated and debugged without everything has to be connected. ROS uses subscriber-publisher pattern which means one node can produce data, and one or more nodes can consume. Topics are used to "transport" data between the nodes. ROS has further more a backup-utility called rosbag that records data between one or more nodes. This comes in handy when testing and later on debug what happened.

The squares shown in the Laptop section in 2.1 shows the ROS nodes and how they are connected using topics. The first node, *cam_to_topics* reads directly from the webcam. Each time a frame is capture the image is published to /camera/down where the MarkerLocator_trackX[6] is subscribing. The MarkerLocator was customized to support receiving frames from a topic instead of the camera directly. By doing this, multiple instances of the MarkerLocators can be run in parallel without their performance influencing each other.

The position detected from each instance of the MarkerLocator is published to */positions/droneN* whereN is the order of the marker and there by used as identifier of the drone. A *Decision_Marker*-node is subscribing to the positions topics. Since ROS provides this modularity the Decision_Marker-node can easily be replaced if another behavior of the swarm is required. The Decision_Marker-node shown was designed to control both nodes at once, however if one think of implementing a swam algorithm, it might be more suitable to make a Decision_Marker for each drone. The outputs of the Decision_Marker are positions of each drone in latitude/longitude.
The *wifi_out_N* nodes will pack the latitude/longitude, append CRC and send the frames to the right drone using wifi[7].

When a drones receives a frame it is packed into a SLIP packet and sent to the At90CAN128 using UART where the data of the packet is un-SLIP'ed and the data[8] is verified using CRC. The positioning-data is then packet as CAN-messages [9] and sent to the M4 flight controller.

[4]http://www.ros.org/
[5]Process in ROS terminology
[6]The MarkerLocator is descriped in section 2.5
[7]Described in section **??**
[8]Described in section **??**
[9]Described in section **??**

2.1.2 Outdoor

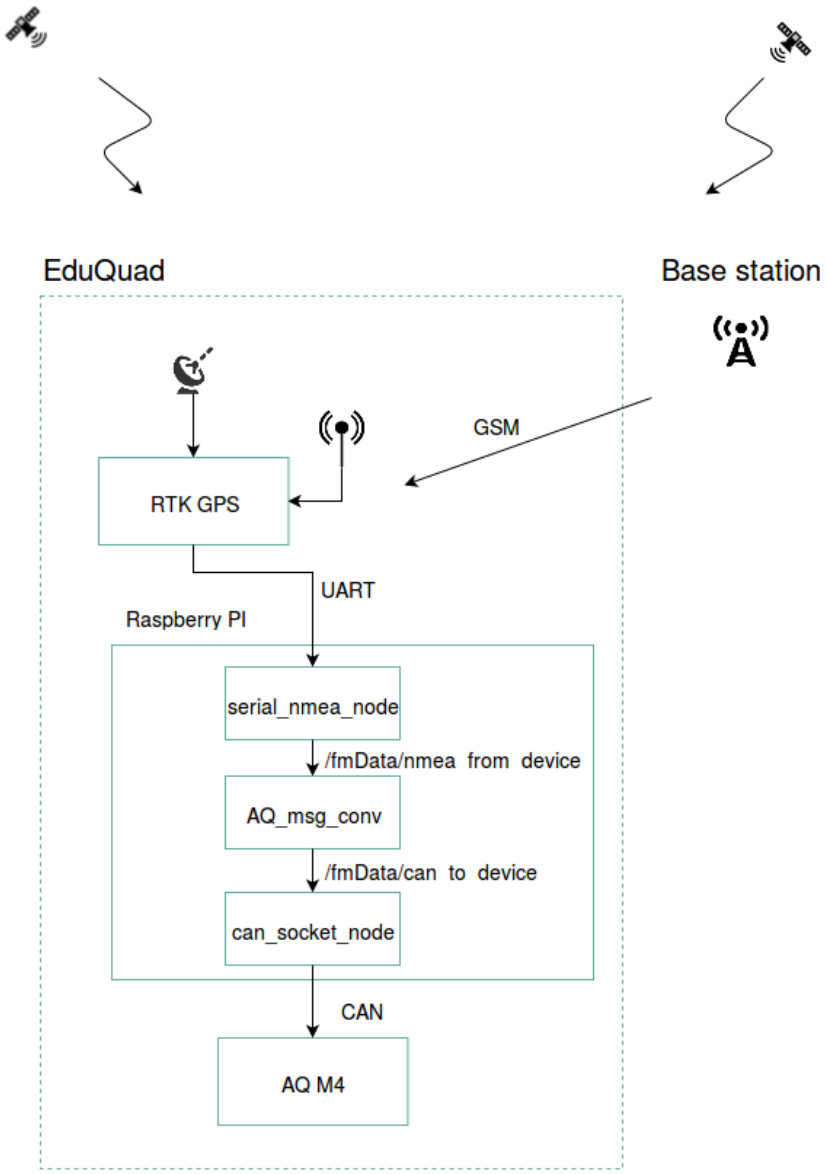


Figure 2.2 *Diagram shows information flow when using indoor flying. Server detects the drones position, sends it through the ROS nodes and transmits the GPS positions using WIFI to the drones. Each drone receives its position and sends it using CAN into M4 flight controller*

The Raspberry Pi is also running ROS since it uses two components from Frobomind[10].

- serial_nmea_node is used to read and write from the serial port. Further more it decodes the NMEA string sent by the GPS/RTK-GPS.

- Can_socket_node is used to communicate with the CAN-adapter[11]

[10]Field robot architecture created and maintained by Kjeld Jensen
[11]PEAK-CAN Adapter has been used in this project

When a GPGGA/GPRMC[12] is received from the GPS/RTK-GPS its read by the *serial_nmea_node* node. It is then forwared to the *AQ_msg_conv* nodes which simply converts the content of the GPGGA/GPRMC to CAN-messages received by AQ M4 flight controller.

Even though the positioning is shown using vision and RTK-GPS, other sources of positioning can be used as well. The same code is running on the AQ M4 whether vision or RTK-GPS is used, so if other positioning systems should be used one should implement the CAN protocol shown in appendix5.1. If using CAN is not possible a RPi and CAN-adapter can be used as intermediate, then the *serial_nmea_node/AQ_msg_conv* should be replaced.

[12]Decribed in section **??**

2.2 AutoQuad extension board

This secion will go in depth with the extension-board created as an addon to the AQ M4 board. The extension-board was developed to act as a bridge between the PC and the CAN-bus using wireless communication.

Wireless communication module

An important part of the hardware is the wireless communication used between the PC and the drones. The wireless communication module has severel requirements it needs to fulfill in order to make the whole system work as expected. A comparison table has been made in table 2.1 to find the wireless communication module that is best suited for the task.
The following wireless modules where considered and compared.

- **ESP8266**

ESP8266 is a generel purpose 32 bit SOC with integrated WIFI 802.11 b/g/n support and buildin TCP/IP stack. It can be setup its own access point or it can connect to an existing wireless network. It runs at 80MHz and can be flashed with a custom firmware. The SOC is sold as modules with different pinouts and features such as extra flash memory [13] and different antennas. The chip has been on the marked for about two years and costs approximately 7$. It has been widely used in DIY-projects due to its low price and because it requires a minimum of network knowledge to get up and running.[14] When the SOC is shipped, it comes with a preloaded firmware which either accepts AT commands or LUA scripting depending on the version of the module. These simple programming interfaces makes it quick and easy to interface the cheap.
This leads to a large community where most of the problems have been found and solved already. Arduino has been ported to ESP8266 which makes it even easier to get it up and running. Their official Arduino GitHub has 2125 commits on their master branch at the time of writing[15]

- **EMW3165**

EMW3165 is a SOC much like the ESP8266 supporting 802.11 b/g/n WIFI with buildin TCP/IP stack. As with ESP8266 it supports setting up an access point aswell as connecting to an existing network. It has a Cortex-M4 μC which run at 100MHz. It supports custom firmware and can be aswell be bought as different modules with different pinouts and antennas. It differentiates itself from the ESP8266 by its higher frequency its 5 volts compatible pins [16] which makes it easier to connect other hardware which run 5 volt without the need of a logic level shifter. It has been on the marked for only one year and costs approximately 9$. Since it is a newer board than ESP8266 it has not been used in the same number of applications and thereby has a smaller community behind[17]. Their most active GitHub has 147 commits on their master branch at the time of writing[18].

- **nRF51822**

[13] https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266/open-source-hardware
[14] http://www.esp8266.com/ - 43.000 posts in forum
[15] https://github.com/esp8266/Arduino
[16] https://hackadaycom.files.wordpress.com/2015/07/emw3165.pdf
[17] http://www.emw3165.com/ - 200 posts in forum
[18] https://github.com/SmartArduino/WiFiMCU

nRF51822 is also a SOC, but it is using Bluetooth instead of WIFI. The nRF51822 μC is implementing BLE which is a power efficient way of sending and receiving data. The chip supports broadcasting which could be used in this project. The μC can be bought as a standalone component or mounted on modules as the two other μCs. Different modules offers different types of antenna connectors or buildin antenna on the PCB. It has not been possible to find an Arduino ported firmware that supports this μC. To write a firmware for the μC it has to be done using Nordic Semiconductor's proprietary SDK.

- **XBee**

XBee is a module that implements the Zbee standard. The Xbee modules work as a wireless serial connection. The Xbee modules supports mesh networking which means the modules by themself figure out which module is closest and makes the connection. This idea makes sense in this application since there will be multiple drones and one computer. If one drone gets too far from the PC, it can just connect to one of the other drones closer to the PC.
The Xbee solution is ready to use and requires a minimum of programming to get up and running. The modules also support GPIO for digital in and output and analog input.

Product	**Size**	**Weight**	**Price**	**Documentation**	**Range**	**Score**
ESP8266	24x16mm[19] {7}	1.5g[20] {8}	7.5$[21] {9}	Great community {9}	?	33
EMW3165	32x16mm[22] {6}	5g[23] {2}	9$[24] {8}	Less available {3}	?	19
nRF51822	18x10mm[25] {8}	3g[26] {4}	7.5$ [27] {9}	Ok documented {2}	30 M[28] {9}	32
XBee	24.38x27.61mm [29] {3}	3g [30] {4}	25$[31] {4}	Lots of DIY {9}	91 M[32] {9}	29

Table 2.1 *Comparisontable used to compare different wireless communication modules*

The products compared in 2.1 are chosen to have approximately same specs such as onboard antenna and no need for extra components. The ESP8266 module based on the highest score but must of all because it is use in a wide range of applications and has a lot of activity on their official Arduino github. The weight of the module was measured after the decision was made. If it was heavier than the other modules it would still have been used due to the large number of projects using this module. In case help was needed to make the module work, plenty of projects is available online and they have an active forum[33]

[19] https://www.mikrocontroller.net/attachment/243558/fcc_11.pdf last visited 26 Maj
[20] Measured by author on chemistry weight
[21] http://www.seeedstudio.com/depot/ESP8266-based-WiFi-module-SPI-supported-p-2486.html last visited 26 Maj
[22] https://hackadaycom.files.wordpress.com/2015/07/emw3165.pdf last visited 26 Maj
[23] http://www.seeedstudio.com/depot/EMW3165-CortexM4-based-WiFi-SoC-Module-p-2488.html last visited 26 Maj
[24] http://www.seeedstudio.com/depot/EMW3165-CortexM4-based-WiFi-SoC-Module-p-2488.html
[25] http://www.fanstel.com/Product/bluenor.htmllast visited 26 Maj
[26] http://www.seeedstudio.com/depot/MDBT40P%C2%A0%C2%A0nRF51822%C2%A0based%C2%A0BLE%C2%A0module-p-2503.html last visited 26 Maj
[27] http://www.seeedstudio.com/item_detail.html?p_id=2503 last visited 26 Maj
[28] https://dl.dropboxusercontent.com/u/54939426/Fanstel_BT600.pdf last visited 26 Maj
[29] http://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4#specifications
[30] http://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4#specifications last visited 26 Maj
[31] https://www.sparkfun.com/products/11215 last visited 26 Maj
[32] https://www.sparkfun.com/pages/xbee_guide
[33] http://www.esp8266.com/ last visited 26 Maj

Microcontroller

The list below shows the requirement for the microcontroller on the extension-board.

- **CAN-controller** to communicate with the AQ M4 board.
- **2 UART** to communicate with the ESP8266-module and with a PC for debugging purpose.
- **Small package** such as QFN64 (9*9mm)

The AT90CAN128 was chosen since the author and SDU had experience programming and using that microcontroller. It meets the requirements as well since it has built-in CAN controller, two UARTS and is available in QFN64 package. Kjeld Jensen has been in front of the development of the Frobomind-controller[34], which uses the AT90CAN128 microcontroller. Since the Frobomind-controller was available the author could start developing the firmware while the PCB for the ladybirds were made.

Block Schematic

The block schematic shown in figure 2.3 was created by the author. It was then given to Carsten Albertsen who created the schematic and did rest of the creation of the PCB. Some time was spend debugging since there was an error in the ISP pins. Later on the author had to make a correction to the board since a pin(GPIO15) on the ESP8266 that had to be grounded in order to select booting from the flash.

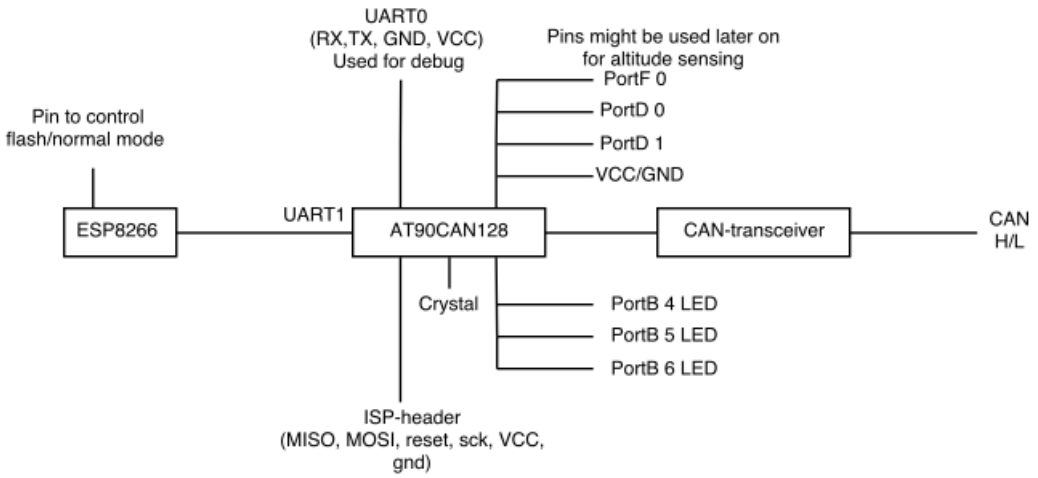


Figure 2.3 *Block schematic of the extension-board developed to AQ M4. It can be seen that the ESP8266 chip is connected to the At90CAN128 using UART1 and that the AtMega is connect to a can-transceiver which communicates with the AQ M4 board.*

[34] http://www.frobomind.org/index.php/FMCtrl:FroboMind_Controller last visited 22 Maj

Pins

A few pins where made available through solder pads for easy access if needed later on.

The following pins where available as solder pads:

- PortF 0 - Alternative function as ADC, channel 0
- PortD 0 - Alternative function as interrupt, INT0 and I2C, SCL
- PortD 1 - Alternative function as interrupt, INT1 and I2C, SDA

In case the onboard baromter is not accurate enough, an alternative distance could be used to measure the drones altitude with respect to the ground. PortF0 has been made available since some distance sensors give output as an analogue value. An example of such sensor is an Infrared proximity sensor.[35]
As an alternative type of distance sensor, a ultrasoinc could be used such as HCSR04. As output it gives a binary output with high-time proportional with the distance.[36]. To detect the high-time, one of PortD1/0 would be useful.

Which type of sensor suits best as a distance sensor to provice altitude information to the drone is ouf of the scope of this report. The PCB has just been made ready to different types of sensors.

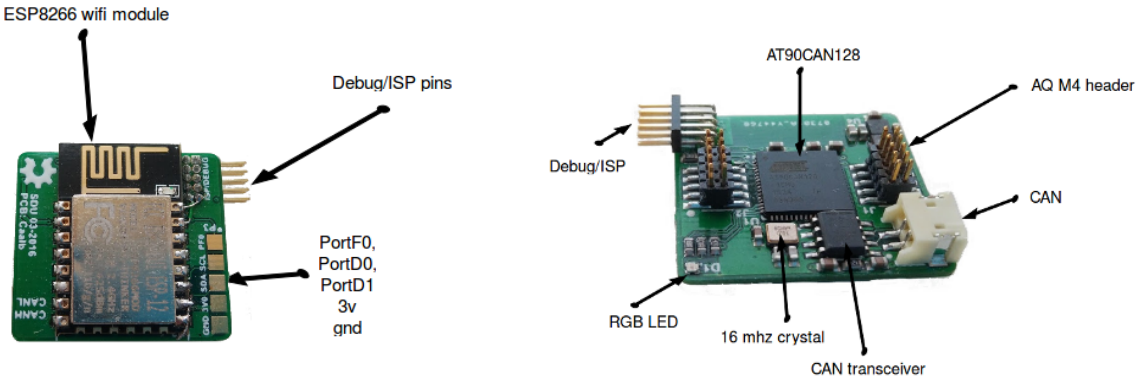
Debug/ISP

In the final schematic UART0 and ISP pins where combined in one pinheader for easy access through one cable.

Mathias: : Refer to image of board

To program the AtMega the ISP pins where required to be easy accessable. UART0 was made accessable to be used as debug and programming of the ESP8266 board. The idea was to setup the AtMega as UART passthrough from UART0 to UART1. Due to a mistake[37] in the final schematic, both UART0 and UART1 where made accessable trough the ISP/debug header. This ended up making it easier to program the ESP8266-board without using the AtMega as UART passthrough.

[35] http://www.sharpsma.com/webfm_send/1208
[36] http://www.micropik.com/PDF/HCSR04.pdf
[37] The wrong pair of MISO/MOSI pins where made available in the ISP-header. The correct pair of MISO/MOSI is also RXD0/TXD0 as alternative function



(a) *Top-view of the developed extension-board. If looking carefully, the ISP-fix can be seen between the ESP-8266 module and the Debug/ISP pins.*

(b) *Bottom-view of the developed-extension board. The two connects for the AQ M4 board can be seen pointing up. The CAN-connect is of same type mounted on the AQ M4 board.*

Figure 2.4 *Top and bottom view of the extension-board*

Test of WIFI range

A test of the wifi was conducted in order to test the range of the ESP8266 module. Test of the AtMega is done in section **??**

To make the test, the fimrware of the ESP8266 module had to be flashed to specify which access point it should connect to. How the networking was configured can be seen in section **??**.

The first test was conducted by pinging the module from the authors laptop 200 times, 10 hz with a packet size of the dataframe used in the application.[38]. The purpose of the test was to see how the latency behaves when the distance is increased and when it stops receiving or transmitting packets.

Figure **??** shows the results.

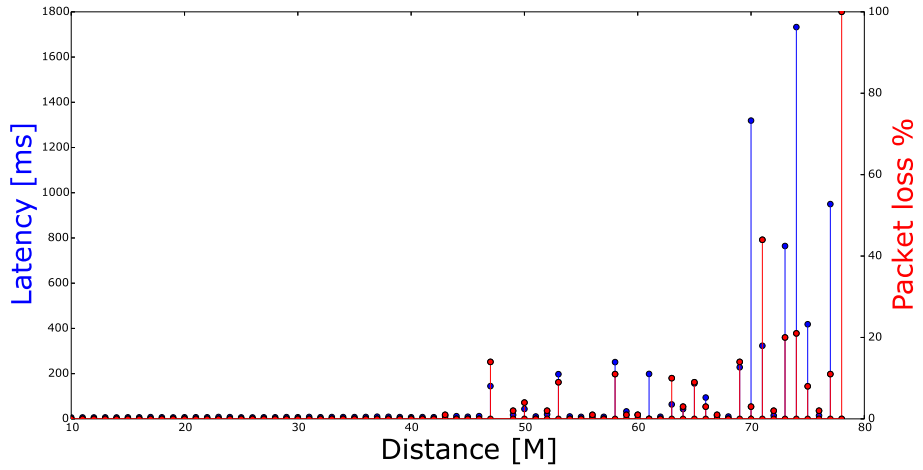


Figure 2.5 *Tasks diagram showing overview of the running tasks on the At90Can128*

[38]See more about the size and content in section **??**

Another test was done where the authors laptops sends datapackages to the ESP8266 module. The rests shows for how long a distance the module can be from the laptop and still receive valid packets. CRC-16 was used to validate the content of the packets. [39]

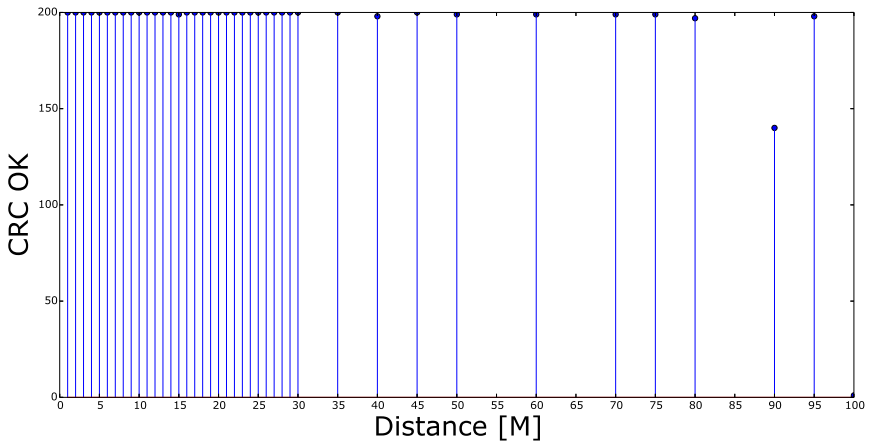


Figure 2.6 *Two modules were connected to one laptop which transmitted datapackets to both modules. Another laptop was connected to one extension-board at a time to check it received all of its packets without error*

A final test was made to see how the modules behave when two modules are receiving the packets. At 10 hz datapackets was sent to two modules at a distance of 10 meters.

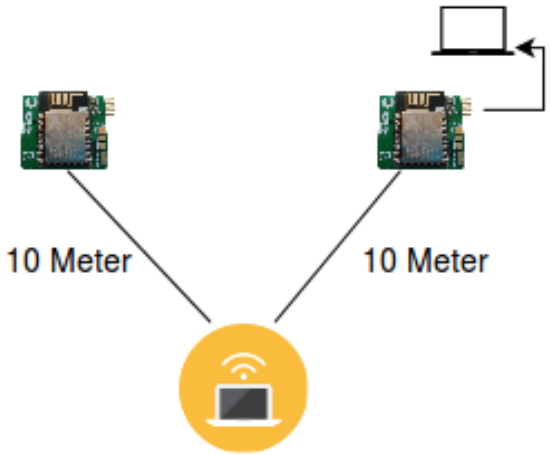


Figure 2.7

Both modules received their packets without CRC error.

It can be concluded that the chosen wifi modules, ESP8266 works well. At a distance of 100 meters it drops.... and that .. and it indicates that the communication works when more than one ESP8266 is connected to the laptop using WIFI.

[39] See more about this in section **??**

2.3 AutoQuad extension board firmware

In this chapter the development of the firmwares running on the extension board is described. Different types of schedulers for the At90CAN128 is discussed and a scheduler is chosen and developed with the required functionality. The communication between PC and M4 is further described. At the end of each section a test is conduced in order to show if the code works as expected

2.3.1 Scheduler

This capter concerns only the Atmega. The ESP module will be descriped in section **??**
In order to have a timing on the At90CAN128 a scheduler was chosen and implemented. The list below shows 3 different types of scheduler that was considered.

- Real-time Operating System(RTOS) provides strict timing but at the cost of overhead. A RTOS runs task in a "parallel" environment. Each task runs in a loop and the RTOS scheduler will do context switching when needed. Using a RTOS requires mutexes and semaphores to protect shared resources which increases the complexity and amount of overhead

- Super-Loop provides no timing at all, but process data as fast as possible. It does not do any context switching and does not require mutexes or semaphores and and thereby takes no overhead.

- Run-To-Complete(RTCS) scheduler is a mix of the two other schedulers. It works by waiting for a tick generated by a hardware timer and then starts executing all tasks from the beginning. The order of the task matters if there is dependency between the tasks but also to make sure the task requiring the most precise timing is at the beginning of the list of the tasks. All tasks have to be finished executing before the next tick is giving in order to avoid timing is ruined.

The RTCS was chosen since it provides timing without the overhead created when doing context-switching and the need of mutex and semaphores. It further reduces the code-complexity.

Mathias: Scheduler: A little description about the scheduler implementation. Add tasks, and run the scheduler. Show how it waits for tick and the runs through all of the tasks that is in RUN state.

It was decided to implement the code in tasks to make a low coupling between the functionalities. This makes it easier to maintain and expand later of if needed. The task diagram shown in figure 2.8 shows the tasks and how they do intertask communication.

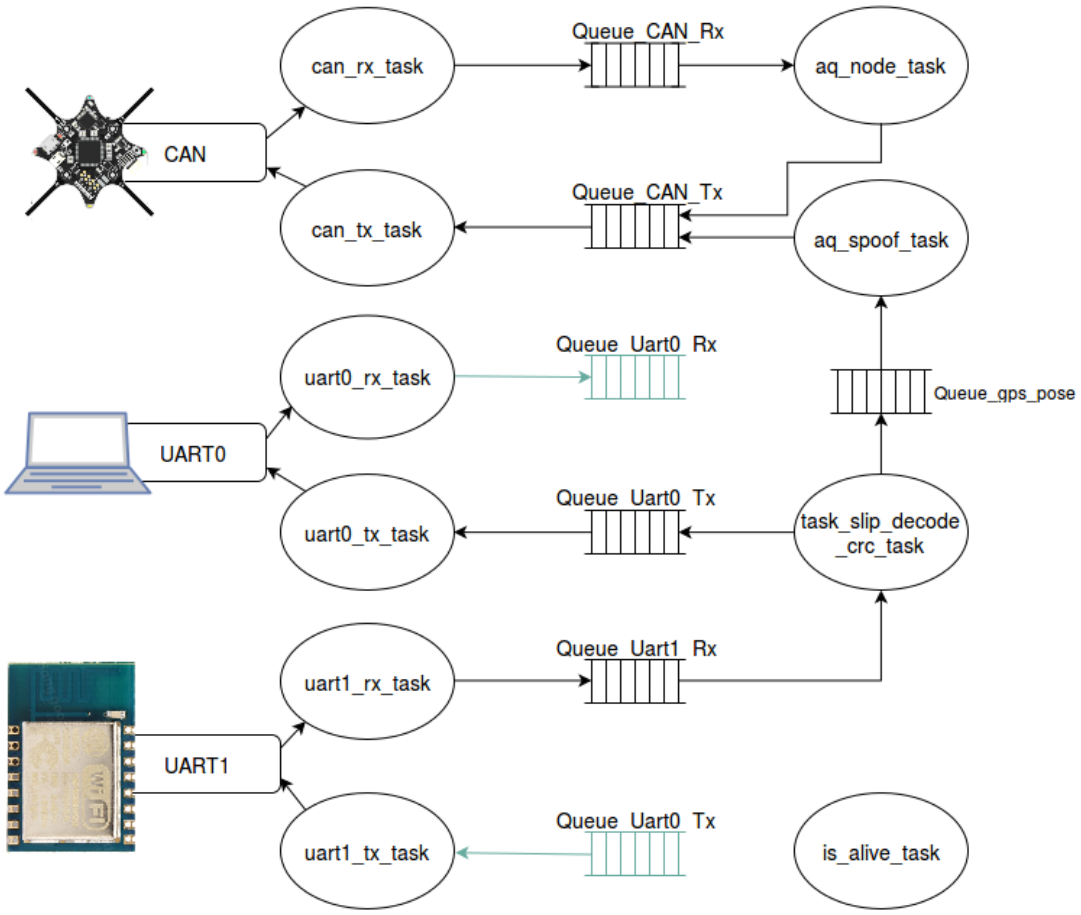


Figure 2.8 *Tasks diagram showing overview of the running tasks on the At90Can128*

A small description of each of the tasks is given below:

- **is_alive_task** Toggles the green led to make sure the scheduler is running. In case of stack overflow, memoryleak or any other error, the Atmega will freeze and it will easily be detectable since the green led stops blinking
- **uart0_tx_task** Responsible of sending characters in the uart0_tx queue.
- **uart0_rx_task** Responsible of checking if any characters in the uart-receive buffer is available. If any, put them into the uart0_rx queue
- **uart1_tx_task** Responsibile as uart_0_tx_task
- **uart1_rx_task** Responsibile as uart_0_rx_task
- **can_rx_task** Responsible of checking if a CAN-message is available in the MOB. If any put them into the CAN_rx_queue
- **can_tx_task** Responsible of transmitting messages available in the CAN_tx_queue
- **aq_node_task** Responsible of registering the node when AQ boots.
- **task_slip_decode_crc_task** Responsible of removing SLIP characters and run CRC
- **aq_spoof_task** Responsible of converting and transmitting received GPS messages from uart1.

Mathias: Scheduler: Add description of slip and crc task, maybe also spoof and node, not sure..

Test of RTCS timing

In order to test the timing of the scheduler, a led_task was written. The task can be seen in code 2.1

Listing 2.1 RTCS task used in timing test

```
1  void is_alive_task(uint8_t my_state){
2
3     // Write to UART0
4     UDR0 = my_state+'0';
5
6     switch(my_state){
7     case 0:
8        // Other off
9        INT_LED_ON_GREEN;
10       // Set next state
11        set_state( 1 );
12       break;
13    case 1:
14       // Other off
15       INT_LED_ON_RED;
16        set_state( 2 );
17       break;
18    case 2:
19       // Other off
20       INT_LED_ON_BLUE;
21        set_state( 0 );
22       break;
23    }
24    // Wait one second
25    wait( 1000 );
26 }
```

The test was done by writing the current state of the task to UART0.
A python script were made that measures the time between each character received. The script can be seen in code 2.2.

Listing 2.2 Python code used to measure time between received byte

```
1  # Imports
2  # Open serial port
3
4  t = time.time()
5  while True:
6      char = ser.read(1):
7      print time.time() - t, ","
8      t = time.time()
9
10 ser.close()
```

The output of the script were redirected to a file. After receiving 700 bytes the standard deviation and mean was calculated using matlab. The mean is 1.0089 sec with a standard deviation of 0.0042 sec.

Part of the variance is caused by the inaccuracy of the timing on the PC running the python code. If a more accurate measure was needed, a scope could be attached to the μC's GPIO. Each time the scheduler enters the task the GPIO should be set high, and when it exists the GPIO should be set low. The scopes at SDU is capable of telling the variance of the off signal. It can be concluded that the scheduler performs well.

2.3.2 Queues

Test of CAN and queues

In order to test CAN and queues, a task was written. The purpose of the task was to receive a CAN-message if any available from the queue and send the ID and data from the CAN message to a PC using UART. The task used can be seen in code 2.3.

Listing 2.3 Snippet of task used to test CAN

```
CAN_frame frame;
if(QueueReceive(&Queue_CAN_Rx, &frame)) {
   char ch;
   /* ID out uart0 */
   for(int i = 3; i >=0; i--){
      ch = ((frame.id >> i*8) & 0x000000FF);
      QueueSend(&Queue_Uart0_Tx,&ch);
   }
   /* ID end*/

   /* MSG out uart0 */
   for(int i = (frame.dlc-1); i >=0; i--){
      ch = ((frame.msg >> i*8) & 0x00000000000000FF);
      QueueSend(&Queue_Uart0_Tx,&ch);
   }
   /* MSG end */

   /* End of line */
   ch = '\r';
   QueueSend(&Queue_Uart0_Tx,&ch);
   ch = '\n';
   QueueSend(&Queue_Uart0_Tx,&ch);
}
```

The command used to send CAN-messages can be seen in code 2.4.

Listing 2.4 Task used to test CAN

```
while true; do sleep 1; print $(date); cansend can0 1DEADBEF#FEDCBA9876543210; done
```

The command sends a CAN-message using a connected PEAK-CAN adapter. It sends a message each second and prints out the date to make it clear when it is sending a message. *1deadbef* is the 29 bit ID and *fedcba9876543210* is the 64 bit data.

The command used to show the received HEX values can be seen in code 2.5

Listing 2.5 Command used to get UART messages

```
cat /dev/ttyUSB0 | xxd -c 14
```

Cat reads from /dev/ttyUSB0 and pipes its output to xxd where it is shown in HEX.
-c is number of columns which is 14 due to 4 ID bytes, 8 data bytes and 2 as newline.

The result can be seen in figure **??**.

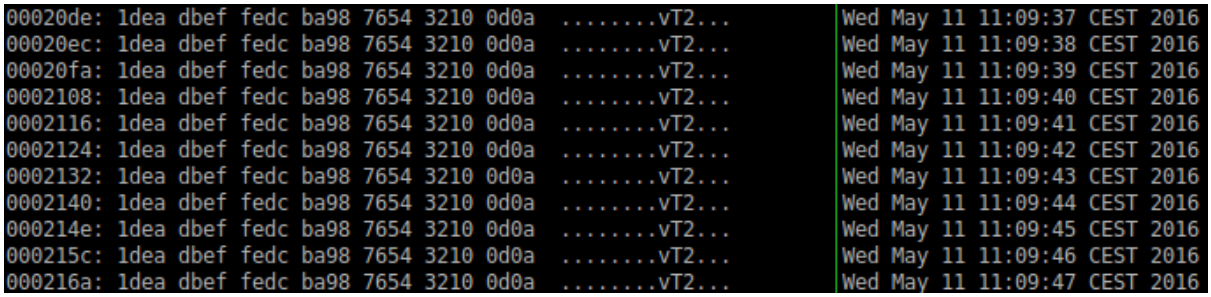


Figure 2.9 *Left shows output from command 2.5 and right shows command 2.4*

It can be seen that the received ID and data is *1deadbef* and *fedcba9876543210* respectively. *0d0a* is \r and \n respectively.

2.3.3 Wireless communication

Communication flow

Content	Lat	Lon	Height	DOP	CRC-16
Datatype	double	double	float	float	uint16_t
Bytes	31:24	23:16	15:8	7:4	3:0

Mathias: : Ændre 0:3 til 0:1 da en uint16_t kun tager 2 bytes og ikke 4

Table 2.2 *Message sent from PC to ESP8266*

2.4 AutoQuad M4 firmware

AutoQuad is the flight controller used in this project. The author was among the first students using AutoQuad on SDU, but the first student to make changes to the firmware and to communicate with AQ using CAN-bus. The AutoQuad firmware was not documented at all, so much time was spend reading through code and trying to figure out how it works. A great amount of time was also spend on more practical things like getting to know their Development Environment (CrossWorks for ARM)[40] in order to compile their firmware, waiting for Quatos[41] license and getting familiar with the flashing process of AutoQuad.
Much of the gained knowledge about AutoQuad and how to do debugging has been written down to share with other students. It can be found on the USB-key handed in *Appendix/SDU-UASAutoQuaddocumentation.pdf and SDU-UASAutoQuadsourcecodedocumentation.pdf.* Text marked with green is written by the author.

Since AutoQuad only supports proving GPS coordinates through a serial port[42] *another way had to be found and implemented. Sending GPS coordinates through the serial port would require the onboard GPS to be unmounted which seems like a bad solution. The CAN bus was chosen since it is already implemented in AutoQuad. However AutoQuad supports sensor inputs through CAN, GPS was not supported and there by had to be implemented.*

In order to send GPS coordinates to AQ a way in had to be found. A solution would be to unmount the onboard GPS, however that removes the functionality of the M4 board to fly normally without external hardware eg. A RPI. Since AutoQuad gets its coordinates from the onboard ublox M8Q [43] using ublox's ubx protocol, everything sent by eg. a RPI had to be encoded as ubx. Instead of making hardware changes to the M4 it was decided to use the already supported CAN-bus interface. The M4 board already uses the CAN-bus to send velocities to each of the ESC's on an EduQuad drone. AutoQuad default supports sending some sensor values using the CAN-bus, however GPS coordinates, DOPs and accuracies was not implemented. The only implementation was to receive the battery voltage and current usage and log it to an MicroSD-card. AutoQuads CAN protocol is described, implemented and tested in section **??**.

When all CAN-nodes has successfully registered, different node-initializing functions run. First a summary run which sends the number and types of the nodes to QGroundStation. [44]. It can thus be validated by the pilot that all the ESC's and CAN-sensors are detected. After the summary, a CAN-sensor initialization happens. The initialization of each CAN-sensor happens which is to send CAN_CMD_TELEM_* described in section **??** but more important also to create a callback. The callback will then be attached to the CAN-type which in this case is CAN-sensor[45] so when AutoQuad receives a sensor reading the callback will be invoked.

Default the callback fills in the sensorvalue in an array so the sensor value can be used in another task. The canID [46] is then used as index so the task needing the sensorvalue knows where to look since it knows which sensorsvalue it needs. Code 2.6 shows the callback[47]

[40] http://www.rowley.co.uk/arm/
[41] The controller, Quatos, used in AutoQuad is not Open Source so a license had to be bought by SDU
[42] Seen by inspection of the schematic to the M4-board done by the supervisor
[43] http://autoquad.org/wiki/wiki/m4-microcontroller/m4-gps-antenna-options/
[44] Graphical userinterface which provices generel information about the drone such as battery, attitude but also waypoint functionalities
[45] CAN-types are described in section **??**
[46] Described in section **??**
[47] https://github.com/bn999/autoquad/blob/master/onboard/canSensors.c

Listing 2.6 Callback invoked when a CAN-sensorvalue is received. It stores the value in an array indexed by canId

```
void canSensorsReceiveTelem(uint8_t canId, uint8_t doc, void *data) {
    canSensorsData.values[canId] = *(float *)data;

   /* Reception time of the message stored as well */
}
```

The callback is fairly simple and does not save the doc[48] which is needed in order to tell which value the CAN-GPS sensor is sending.

All of the CAN-GPS packet handling has been implemented in the callback function. A prettier solution might have been to create a task and use a semaphore to signal when there is a new GPS-packet available. Due to lack of time implementing everything was done in the callback.

Code 2.7 shows a pseudo code of how the GPS-packets is handled. In section **??** the CAN-GPS packets can be seen.

Listing 2.7 Modified callback invoked when a sensor-value is received. Shows how doc is used to tell which GPSpacket is received and when height is received the flags are set

```
canSensorsReceiveTelem(canId, doc, *data) {
   if doc == CAN_DOC_LAT then
      canData.latitude = data

   if doc == CAN_DOC_LON then
      canData.longitude = data

   if doc == CAN_DOC_DOP then
      canData.xDOP = data[x]
      ..
   if doc == CAN_DOC_ACC then
      canData.satellites = data[0]
      canData.fix = data[1]
      canData.xAcc = data[x]
      ..
      canData.heading = data[n-1]

   if doc == CAN_DOC_VEL then
      gpsData.velx = data[x]
      ..
   if doc == CAN_DOC_ALT then
      gpsDatat.height = data
      if gpsData.fix == 1 and gpsData.satellites >= 8 then
         gpsUpdatetime = now()
         setFlag (gpsData.gpsPosFlag)
         setFlag (gpsData.gpsVelFlag)
      else:
         debug_to_QGroundStation
}
```

[48] Described in section **??**

The *data* pointer given as parameter to the callback is of void pointer. Some pointer gymnastics is done in order to get the right elements from the CAN-packet. If the received CAN-packet is of 8*1 bytes, the void pointer is casted to an uint8_t pointer so each byte can be retrieved by using the casted pointer as an array.

When the flags are set, the task updating the UKF will run [49]. Code 2.8 shows a snippet from the task updating the UKF when the position or velocity flag is set.

Listing 2.8 Snippet of run.c as psudeocode which updates the UKF when position flag or velocity flag is set

```
1 if IsFlagSet(gpsData.gpsPosFlag) == yes then
2       navUkfGpsPosUpdate(gpsData.lastPosUpdate, gpsData.lat, gpsData.lon,
            gpsData.height, ....);
3       ClearFlag(gpsData.gpsPosFlag);
4
5 else if IsFlagSet(gpsData.gpsVelFlag) == yes then
6       navUkfGpsVelUpdate(gpsData.lastVelUpdate, gpsData.velN, gpsData.velE, ....);
7       ClearFlag(gpsData.gpsVelFlag);
```

[49] https://github.com/bn999/autoquad/blob/master/onboard/run.c#L110 last visited 23 maj

can0 00000000 [0] can0 01C00000 [8] EF BE AD DE 03 09 00 00 can0 0E000041 [6] EF BE AD DE 00 00 can0 14CD0042 [2] 00 08 can0 14CC0043 [2] 0A 00

seq tæller op så rate/value bliver acknowledged can0 00000000 [0] can0 01C00000 [8] EF BE AD DE 03 09 00 00 can0 0E000041 [6] EF BE AD DE 00 00 can0 14CD0042 [2] 00 08 can0 00400002 [8] 00 00 00 00 00 00 00 00 can0 14CC0043 [2] 0A 00 can0 00400003 [8] 00 00 00 00 00 00 00 00

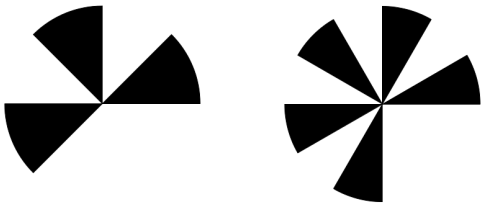
2.5 Indoor localization

In order to do indoor flying there is a need of some sort of localization since GPS is not available indoor. Vision was decided based on what mostly others are using and some further advantages. Instead of making the vision, it is decided to use an existing tracker called MarkerLocator written by Henrik Midtiby. However the MarkerLocator lacks a proper quality measure so that is implemented. A few tests were made and the MarkerLocator with implemented quality measure is working quite well

In order to do accurate flying indoor a localization system is needed. In the Related Work section, it can be seen vision with a camera mounted pointing down is used by others. Compared to use one or more cameras mounted on the drone, an advantage of using a camera mounted on the ceiling is that the camera will not be harmed if the drone crashes and that one camera can be used to detect several drones. However if the position is needed in more than 2D, more than one camera might be needed(depending on the algorithm used) but it still scales better than mounting one or more cameras on each drone.

Based on previous experience, the MarkerLocator written by Henrik Midtiby was chosen as indoor localization. Its using OpenCV[50] and implemented in python. It works by detecting markers as shown in figure 2.14

Mathias: Indoor localization: insert image of markerLocator marker with different orders



(a) *Marker of order 4 used to get the 2D position of a drone*

(b) *Marker of order 6 used to get the 2D position of a drone*

Figure 2.10 *The markers have one arm less than the order of the marker for the MarkerLocator to detect the orientation of the marker*

The MarkerLocator works by making a convolution sum of a complex kernel on each frame obtained from the camera. The location where if fits best is said to be the location of the marker in the frame. How it works in detail is out the scope of this project. The markers shown in figure 2.14 is of different orders. The order is equal to the number of black arms minus one since the missing arm is used to detect the orientation of the marker. Different orders can be used and thereby detect more than one marker in each frame. Unfortunately it can only get the

[50] http://opencv.org/

2D position and 1D orientation of the marker. Since doing a convolution of the entire frame is relatively slow [51], the MarkerLocator has a mode called WindowMode. It works by searching in only a small area around the last known position of the marker and thereby speeding up the progress[52].

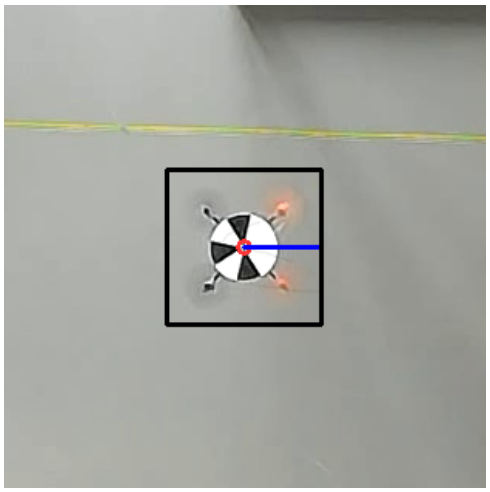


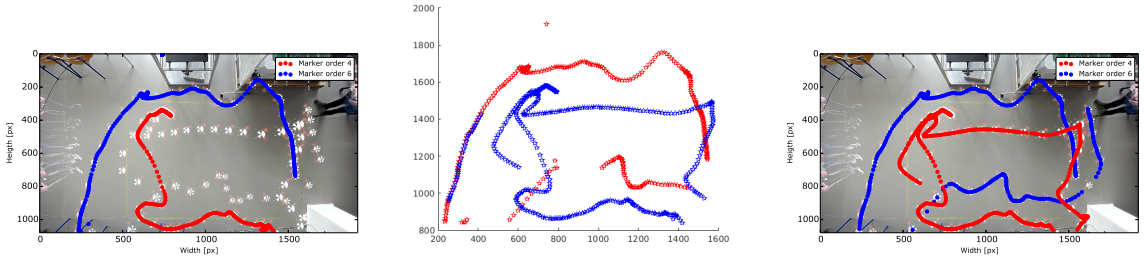
Figure 2.11 *The MarkerLocator searches only for the marker around the last known position of the marker in a frame of 100*100px.*

If the marker moves out of the window, the MarkerLocator is still looking in only that window. Therefore it is important to have a quality measure that can be used to decide whenever a full convolution of the frame has to be done or if the marker has been found within the window.

The MarkerLocator has a quality measure build-in used to tell how well the marker is detected. However it is not used in the MarkerLocator to tell if a full convolution has to be done. The quality measure implemented in the MarkerLocator is a hack [53] and is known from previous applications that it is a bad measure of the right marker is detected.

In cooperation with Henrik Midtiby a few quality measures was developed. The overall idea was to compare the kernel used to do the convolution with the marker found in the frame. Different algorithms where used to give a normalized value telling how similar the found marker is to the kernel. Figure 2.16 shows the different solutions tried.

[51] Processing one fullHD frame takes approximately 0.38 sec. Measured with build in timer in the MarkerLocator
[52] Approximately 0.0041 sec
[53] Said by Henrik Midtiby



(a) *Counting pixel match suffers from being unable to find the markers if lost*

(b) *SSIM suffers from false positives and a few false negative*

(c) *Developed algorithm has no false negative of positive and is able to find the markers if lost*

Figure 2.12 *Different approaches to get a quality measure explaining how well the right marker is found.*

Figure 2.16a is a native approach of comparing all pixels in the kernel with the pixels in the found marker. A counter is incremented upon each match and at the end divided by the number of pixels compared in to normalize. A threshold determines weather a marker is found and the next scan should be done in the window of if a full scan is required. The best result is obtained using a threshold of 0.4. If the threshold were lowered false positives started to occur.

Figure 2.16b uses an algorithm developed to compare similarities in images. The details of this algorithm is out of scope the details of this project. The implementation in scipy-image[54] was used. It can seen it performs better than the native approach but a lot of false positive and a few false negative exists. It also suffers from not detecting the markers compared to the native approach. A threshold of 0.3 were used. If the threshold were increased, it would detect less markers.

Figure 2.16c shows a simple algorithm the author came up with. If the MarkerLocator is looking for a marker of order 4 and it has detected a marker and its orientation, the algorithm calculates where the location of the arms. It then checks a pixel in each arm and increments a counter if the pixel is black. When it has checked all the arms it compares the counter with the order the MarkerLocator is looking for and returns if they match or not. This method does not provide a scalar as output but a binary and thereby has no threshold. However in order to detect if a pixel is black or not, a threshold has been used. The threshold was set to 100 which has shown good results doing tests and flight.

During development of quality measures it was noted that if the MarkerLocator does a full search because it cannot find a marker with order 4, it slows down the detection of the marker of order 6. This is because the MarkerLocator is implemented in one process without threading. However this has been solved by splitting up the MarkerLocator in different processes so they run independently of each other. More about this in section 2.5

Perspective correction

In order to use centimeters as measure of the position of the markers and to correct if the camera is not pointing directly down, the build-in perspective correction in the MarkerLocator were used. It works by using homography[9] to make the transformation from pixels to some chosen unit which in this case is centimeters.

[54] http://scikit-image.org/

By creating four markers with known distance to each other at the expected flight height [55] and find these markers in the frame, it is possible for the perspective corrector to make the transformation. Figure 2.17 shows the setup used. A picture was taken from the ceiling-camera and the four black corners of the whiteboard could be found in the frame. The distance between the black corners in centimeters and the distance in px is given as input to the perspective correcter. The locations of markers is then transformed into the plane made by the whiteboard. Depending on where the whiteboard is placed in the frame origin can be placed. The placement of the coordinate system can be seen in section **??**

Figure 2.13 *Setup used to make 4 points in the flight-height. The four corners of the whiteboard was used and then found in the image.*

A test was conducted to get the accuracy of the MarkerLocator. Two markers with order 4 and 6 was placed at the expected flight-height and the distance between the two markers was calculated using Pythagoras.
This was done with distance 60 centimeters measured using a ruler. The real distance measured by a ruler was 60 cm and and the MarkerLocator gave

Another test was conducted in order to see how stable the MarkerLocator is. 547 position detections was done while the marker was placed steady at the flight height. It shows a mean position in x of -8.1487 with variance of 0, y position with mean -39.5046 and variance 0.

Some of this inaccuracy is caused by measuring error and placing of the whiteboard. If higher accuracy is needed this should be done again more carefully.

It can be concluded that using the MarkerLocator with the improved order detection it is possible to detect two drones without having any false positive/negative. By making a test of the distance between two markers there is an error of 0.81 cm. The MarkerLocator seems stable with a variance of zero.

2.6 Control and coordinate conversion

In order to do indoor flying there is a need of some sort of localization since GPS is not available indoor. Vision was decided based on what mostly others are using and some further advantages. Instead of making the vision, it is decided to use an existing tracker called MarkerLocator written by Henrik Midtiby. However the MarkerLocator lacks a proper quality measure so that is implemented. A few tests were made and the MarkerLocator with implemented quality measure is working quite well

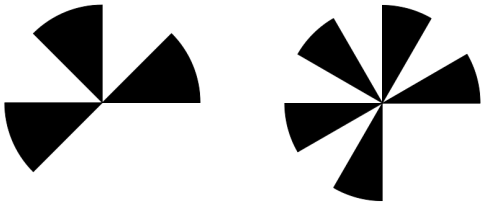
In order to do accurate flying indoor a localization system is needed. In the Related Work section, it can be seen vision with a camera mounted pointing down is used by others. Compared

[55] Since only 2D positioning is available, a plane where the drone is expected to fly is decided

to use one or more cameras mounted on the drone, an advantage of using a camera mounted on the ceiling is that the camera will not be harmed if the drone crashes and that one camera can be used to detect several drones. However if the position is needed in more than 2D, more than one camera might be needed(depending on the algorithm used) but it still scales better than mounting one or more cameras on each drone.

Based on previous experience, the MarkerLocator written by Henrik Midtiby was chosen as indoor localization. Its using OpenCV[56] and implemented in python. It works by detecting markers as shown in figure 2.14

Mathias: Control and coordinate conversion: insert image of markerLocator marker with different orders



(a) *Marker of order 4 used to get the 2D position of a drone*

(b) *Marker of order 6 used to get the 2D position of a drone*

Figure 2.14 *The markers have one arm less than the order of the marker for the MarkerLocator to detect the orientation of the marker*

The MarkerLocator works by making a convolution sum of a complex kernel on each frame obtained from the camera. The location where if fits best is said to be the location of the marker in the frame. How it works in detail is out the scope of this project. The markers shown in figure 2.14 is of different orders. The order is equal to the number of black arms minus one since the missing arm is used to detect the orientation of the marker. Different orders can be used and thereby detect more than one marker in each frame. Unfortunately it can only get the 2D position and 1D orientation of the marker. Since doing a convolution of the entire frame is relatively slow [57], the MarkerLocator has a mode called WindowMode. It works by searching in only a small area around the last known position of the marker and thereby speeding up the progress[58].

[56] http://opencv.org/

[57] Processing one fullHD frame takes approximately 0.38 sec. Measured with build in timer in the MarkerLocator

[58] Approximately 0.0041 sec

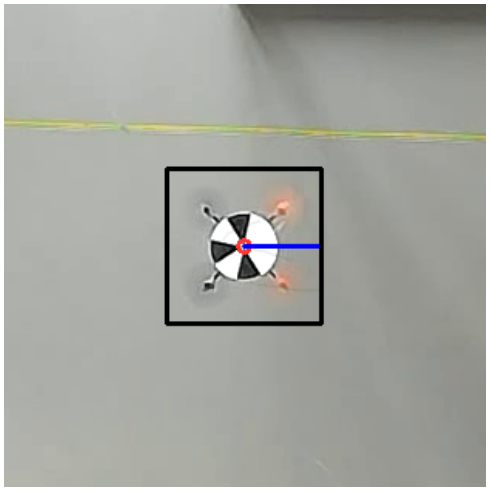
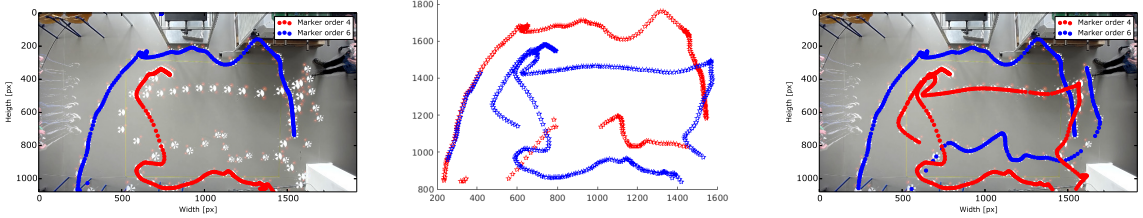


Figure 2.15 *The MarkerLocator searches only for the marker around the last known position of the marker in a frame of 100*100px.*

If the marker moves out of the window, the MarkerLocator is still looking in only that window. Therefore it is important to have a quality measure that can be used to decide whenever a full convolution of the frame has to be done or if the marker has been found within the window.

The MarkerLocator has a quality measure build-in used to tell how well the marker is detected. However it is not used in the MarkerLocator to tell if a full convolution has to be done. The quality measure implemented in the MarkerLocator is a hack [59] and is known from previous applications that it is a bad measure of the right marker is detected.

In cooperation with Henrik Midtiby a few quality measures was developed. The overall idea was to compare the kernel used to do the convolution with the marker found in the frame. Different algorithms where used to give a normalized value telling how similar the found marker is to the kernel. Figure 2.16 shows the different solutions tried.



(a) *Counting pixel match suffers from being unable to find the markers if lost*

(b) *SSIM suffers from false positives and a few false negative*

(c) *Developed algorithm has no false negative of positive and is able to find the markers if lost*

Figure 2.16 *Different approaches to get a quality measure explaining how well the right marker is found.*

Figure 2.16a is a native approach of comparing all pixels in the kernel with the pixels in the found marker. A counter is incremented upon each match and at the end divided by the number of pixels compared in to normalize. A threshold determines weather a marker is found and the next scan should be done in the window of if a full scan is required. The best result

[59] Said by Henrik Midtiby

is obtained using a threshold of 0.4. If the threshold were lowered false positives started to occur.

Figure 2.16b uses an algorithm developed to compare similarities in images. The details of this algorithm is out of scope the details of this project. The implementation in scipy-image[60] was used. It can seen it performs better than the native approach but a lot of false positive and a few false negative exists. It also suffers from not detecting the markers compared to the native approach. A threshold of 0.3 were used. If the threshold were increased, it would detect less markers.

Figure 2.16c shows a simple algorithm the author came up with. If the MarkerLocator is looking for a marker of order 4 and it has detected a marker and its orientation, the algorithm calculates where the location of the arms. It then checks a pixel in each arm and increments a counter if the pixel is black. When it has checked all the arms it compares the counter with the order the MarkerLocator is looking for and returns if they match or not. This method does not provide a scalar as output but a binary and thereby has no threshold. However in order to detect if a pixel is black or not, a threshold has been used. The threshold was set to 100 which has shown good results doing tests and flight.

During development of quality measures it was noted that if the MarkerLocator does a full search because it cannot find a marker with order 4, it slows down the detection of the marker of order 6. This is because the MarkerLocator is implemented in one process without threading. However this has been solved by splitting up the MarkerLocator in different processes so they run independently of each other. More about this in section 2.5

Perspective correction

In order to use centimeters as measure of the position of the markers and to correct if the camera is not pointing directly down, the build-in perspective correction in the MarkerLocator were used. It works by using homography[9] to make the transformation from pixels to some chosen unit which in this case is centimeters.

By creating four markers with known distance to each other at the expected flight height [61] and find these markers in the frame, it is possible for the perspective corrector to make the transformation. Figure 2.17 shows the setup used. A picture was taken from the ceiling-camera and the four black corners of the whiteboard could be found in the frame. The distance between the black corners in centimeters and the distance in px is given as input to the perspective correcter. The locations of markers is then transformed into the plane made by the whiteboard. Depending on where the whiteboard is placed in the frame origin can be placed. The placement of the

Figure 2.17 *Setup used to make 4 points in the flight-height. The four corners of the whiteboard was used and then found in the image.*

[60] http://scikit-image.org/

[61] Since only 2D positioning is available, a plane where the drone is expected to fly is decided

coordinate system can be seen in section
??

A test was conducted to get the accuracy of the MarkerLocator. Two markers with order 4 and 6 was placed at the expected flight-height and the distance between the two markers was calculated using Pythagoras.
This was done with distance 60 centimeters measured using a ruler. The real distance measured by a ruler was 60 cm and and the MarkerLocator gave

Another test was conducted in order to see how stable the MarkerLocator is. 547 position detections was done while the marker was placed steady at the flight height. It shows a mean position in x of -8.1487 with variance of 0, y position with mean -39.5046 and variance 0.

Some of this inaccuracy is caused by measuring error and placing of the whiteboard. If higher accuracy is needed this should be done again more carefully.

It can be concluded that using the MarkerLocator with the improved order detection it is possible to detect two drones without having any false positive/negative. By making a test of the distance between two markers there is an error of 0.81 cm. The MarkerLocator seems stable with a variance of zero.

Chapter 3

Results

Chapter 4

Discussion

Chapter 5

Conclusion

List of Figures

List of Tables

Bibliography

[1] Suraj G Gupta, Mangesh M Ghonge, and PM Jawandhiya, "Review of unmanned aircraft system (uas)," *technology*, vol. 2, no. 4, 2013.

[2] Andrew Gibiansky, "Quadcopter dynamics, simulation, and control," 2010.

[3] Vijay Kumar, "Robots that fly ... and cooperate," 2016.

[4] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

[5] Hyun-gyu Kang, Byoung-kyun Kim, and Wangheon Lee, "Indoor localization of dron using vision based sensor fusion," in *Control, Automation and Systems (ICCAS), 2015 15th International Conference on*. IEEE, 2015, pp. 932–934.

[6] Wikipedia, "Background subtraction — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Background%20subtraction&oldid=697931050, 2016, [Online; accessed 17-May-2016].

[7] Jose Luis Sanchez-Lopez, Jesus Pestana, Paloma de la Puente, Ramon Suarez-Fernandez, and Pascual Campoy, "A system for the design and development of vision-based multi-robot quadrotor swarms," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*. IEEE, 2014, pp. 640–648.

[8] Timothy Stirling, James Roberts, Jean-Christophe Zufferey, and Dario Floreano, "Indoor navigation with a swarm of flying robots," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4641–4647.

[9] Jan Erik Solem, *Computer Vision with Python*, O'REILLY, first edition, 2012.

[10] Anna B.O. Jensen Keld Dueholm, Mikkel Laurentzius, *GPS*, Nyt Teknisk Forlag, third edition, 2015, Page 45-47.

Appendices

5.1 AutoQuad CAN protocol

In order to spoof the onboard GPS, the CAN-bus was chosen as input to AQ. The CAN-bus is already implemented and widely used on a ViaCopter drone to communicate between AQ and hardware like ESCs, PDB etc. Much time was spend reading through the source code and trying to figure out how the protocol works. The developers behind AQ says, that the code is the documentation and thereby have not written any real documentation.
Figuring out how it works was done using debugging utilities such as breakpoints and looking at the content of the different memory locations in CrossWorks.
More debugging tricks can be seen in appendix **??**. To further investigate the protocol a PEAK-CAN adapter were connected to a Ladybird drone and several python scripts were developed to send and receive messages from AQ.
The author did not go into details about messages used between AQ ↔ ESC's and AQ ↔ OSD.

CAN Protocol description

A Peak-CAN adapter were used throughout the project. It supports High-speed ISO 11898-2[1] which is a standard that states the properties of physical layer. Its max speed is 1 mbit which is the speed AQ uses to communicate on the CAN-bus. AQ uses CAN 2.0B which has 29 identifier bits. AQ is not using an existing protocol on the application layer. The developers created their own to suit their needs.

Identifier bits

A Generic look at a AQ CAN messages can be seen in table 5.1. The normal function of the identifier bits is the priority of the messages. A CAN controller can then be setup to only allow certain messages to be processed depending on their identifier bits. The identification bits has been split up in AQ to contain information about the sender, receiver etc. Which is not normally saved information in a CAN message.

CAN Message							
Identfier bits [29 bits]							Data bits [max 64 bits]
LCC [28:27]	TT [26]	FID [25:22]	DOC [21:16]	SOID [15:11]	TID [10:16]	SEID [5:0]	

Table 5.1 *Table shows the identifier bits used in AutoQuad CAN messages*

In figure 5.2 the abbreviations can be seen.

LCC	Logical Communications Channe
TT	Target Type
FID	Function ID
DOC	Data Object Code
SOID	Source ID
TID	Target ID
SEID	Sequence ID

Table 5.2 *Table shows abbreviations used in table 5.1*

[1] http://www.peak-system.com/PCAN-USB.199.0.html?&L=1

Each of the elements in an AQ messages will be explained how they are used in AQ.

Logic Communication Channel

LCC is the priority of the message. To understand how LCC works, one needs to look into CAN arbitration. As stated in ISO-11898-2, a 0 is the dominant bit and a 1 is the recessive bit. The example in figure 5.1 shows two nodes each transmitting a packet. The arbitration only happens during the transmission of the identifier bits. In the example on bit 8, Node 16 losses the arbitration and stops transmitting. Node 15 keeps transmitting its packet because it has lower identifier bits and thereby higher priority.

	Start Bit	ID Bits										The Rest of the Frame
		10	9	8	7	6	5	4	3	2	1	0
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN Data	0	0	0	0	0	0	0	0	1	1	1	1	

Figure 5.1 *Example of CAN arbitration where node 15 has the lowest ID and thereby highest priority*

LLC in a message can take one of the defines in code 5.1

Listing 5.1 LLC defines

```
#define CAN_LCC_EXCEPTION ((uint32_t)0x0<<30)
#define CAN_LCC_HIGH      ((uint32_t)0x1<<30)
#define CAN_LCC_NORMAL    ((uint32_t)0x2<<30)
#define CAN_LCC_INFO      ((uint32_t)0x3<<30)
```

Table 5.3 *Table showing the 4 types of priority in AQ*

Name	Value	Priority (lowest first)
EXCEPTION	0000	0
HIGH	0001	1
NORMAL	0010	2
INFO	0011	3

Target Type

Target type can either be "node" or "group" depending on if the receiver(s) is one or more nodes.

Listing 5.2 Target type defined in AQ

```
#define CAN_TT_GROUP      ((uint32_t)0x0<<29)
#define CAN_TT_NODE       ((uint32_t)0x1<<29)
```

The GROUP is used when AQ sends its reset-msg upon startup.
When the authors node sends a sensor-measure to AQ it will be of type CAN_TT_NODE.

Function ID

Function ID describes the function of the CAN message. If it is a PING, ACK, NACK, etc. It simply states the function of packet so the receiver node knows what to do with the message. Different types of functions can be seen in code 5.3

Listing 5.3 Excerpts from AQ's list of function defines

```
#define CAN_FID_RESET_BUS  ((uint32_t)0x0<<25)
#define CAN_FID_ACK        ((uint32_t)0x1<<25)
#define CAN_FID_REQ_ADDR   ((uint32_t)0x7<<25)
#define CAN_FID_GRANT_ADDR ((uint32_t)0x8<<25)
#define CAN_FID_CMD        ((uint32_t)0x3<<25)
```

Table 5.4 describes the FIDs shown in listing 5.3.

Mathias: : Dette er forkert, eftersom alle defines er taget fra AQ's can.h så der passer det..

If a message is if FID CAN_FID_CMD, the Data Object code will be used as the command. It should be noted from code 5.3 that the FID's is moved 25 positions to the left but FID is shown in table 5.1 is at bits 22:25. The offset of three bits is due to the way ARM's CAN

registers is implemented. The three initial bits in CAN_TIxR and CAN_RIxR (registers used to transmit and receive respectively) [2] registers are general information about the CAN messages like whether it is extended (Part B.0) or not.

Table 5.4 *Descriptions of the FIDs mentioned in code 5.3*

FID	Description
CAN_FID_RESET_BUS	Message sent by AQ when powered up
CAN_FID_ACK	Message sent to AQ when acknowledging a received message
CAN_FID_REQ_ADDR	Messaged used when a node on the bus tries to register itself as a node on the bus
CAN_FID_GRANT_ADDR	Messaged sent by AQ when it registers a node as a node on the bus.

Table 5.4 shows a description of the 4 FIDs used when a node is registered.

Data Object Code

Data Object code is used as a parameter to the FID. Data Object Code can have different values depending on the fid. Code 5.4 shows a function using DOC when message FID is CAN_FID_CMD.

Listing 5.4 Snippet of AQ's can.c:365

```
1  void canProcessCmd(... ,uint8_t doc, ...) {
2      switch (doc) {
3          case CAN_CMD_STREAM:
4           ...
5              break;
6          case CAN_CMD_TELEM_VALUE:
7           ...
8              break;
9          case CAN_CMD_TELEM_RATE:
10          ...
11             break;
12     }
13 }
```

In section 5.1 DOC is used by the author to tell which part of the GPS data is getting sent but where FID is CAN_FID_TELEM.

Mathias:: Fixe ref til section om GPS spoof

Source/Target/Network ID

When referred to source/target ID but without respect to either sender or receiver but just a CAN-node, then it is referred to as NetworkId. When a node registers itself in AQ, it gets assigned a NetworkId.

[2] http://www2.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf:688

Sequence ID

The sequence id is incremented on transmission of each message. It is used when eg. A CAN-node sends a reply to a get-message, it is then including the sequence id of the get-message so AQ knows what it receives a reply for.

Data bits

The data bytes does not have a general definition, it depends on the function id. With some FIDs the data field may contain additional parameters.

Eg. When FID is CAN_FID_REQ_ADDR, data has the fields shown in table 5.5

Table 5.5 *Packet sent from node when registering in AQ*

64 bits							
0	0	CanId	CanType	UUID3	UUID2	UUID1	UUID0
7	6	5	4	3	2	1	0

UUID is a unique address generated by each node on the bus. In ESC32v2[3] the address is calculated using XXH-hashing algorithm. The algorithm generates a 32 bit hash from a given input value and a salt. The input to XXH in ESC32v2 is the unique ID that every ARM in the STMF32[4] family has. Code 5.5 shows how it is implemented in AQ.

Listing 5.5 Snippet showing UUID generated in ESC32v2

```
1 can.h:20
2 #define CAN_UUID 0x1FFFF7E8
3
4 can.c:753
5 canData.uuid = XXH32((void *)CAN_UUID, 3*4, 0);
```

CanType is the type of the node trying to register. Eq. SENSOR, ESC and SERVO.

CanID is the number of the CanType node trying to register. When an ESC32v2 is trying to register it sends CanId as the ESC number ranging from 1 to the number of ESC's mounted on the drone. The CanID is assign to each ESC manually as part of the configuring.

Registering a node in AQ

When the AQ is starting up, it is transmitting a CAN_FID_RESET_BUS to the bus. When each CAN-node receives the messages, they send out a CAN_FID_REQ_ADDR message in order to get registered as a node in AQ. If a node on the bus does not get registered it will not be able to communicate with AQ.
When AQ receives an CAN_FID_REQ_ADDR, it sends back a CAN_FID_GRANT_ADDR to tell the node that the registration is successful and to assign the node its networkID. This id is used later to identify the node when it is transmitting a message or when AQ wants to send

[3] https://github.com/bn999/esc32/blob/master/onboard/can.c
[4] http://www2.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf

out a message to that specific node. The sequence diagram in figure 5.2 shows the described sequence.

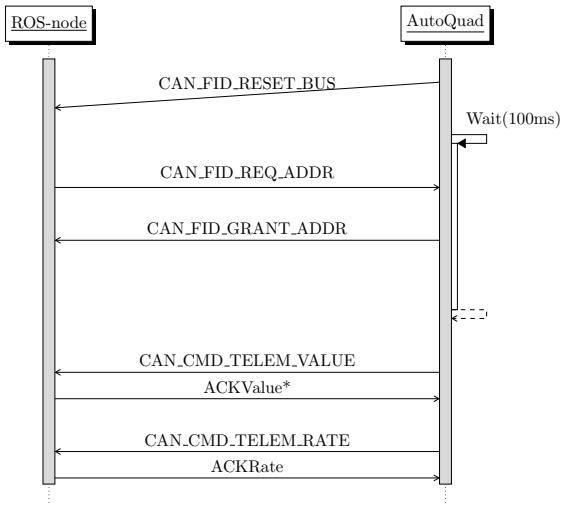


Figure 5.2 *Registering a new node in AQ*

Depending on the canType set in the data-field when a node is transmitting a CAN_FID_REQ_ADDR, AQ might send zero or more messages back to the node.
In figure 5.2 it can be seen that AQ sends back CAN_CMD_TELEM_VALUE and CAN_CMD_TELEM_RATE because the CanType is set to SENSOR. Each of the two messages has two fields in the data field, that contains information about the stream. CAN_CMD_TELEM_VALUE is used to request telemetry from a node that registered itself as a SENSOR. CAN_CMD_TELEM_RATE is the requested telemtry rate from AQ. Default is 10 hz [5].
A timeout timer is started when each message has been sent. If an ACK is not received after each message within 250 ms [6] a timeout counter is incremented. At the time of writing this information is not used but might be later on.

Code in 5.6 shows a snippet[7] from AQ where AQ waits for nodes to request an address.

Listing 5.6 Snippet showing AQ waits for devices

```
1  canResetBus();
2
3  // wait 100ms for nodes to report in
4  micros = timerMicros() + 100000;
5  while (timerMicros() < micros)
6      // extend wait period if more nodes found
7      if (canCheckMessage(0))
8          micros = timerMicros() + 100000;
```

AQ waits default 100 ms for nodes to request an address. If canProcessMessage() returns true a node tried to register and the registration period is extended 100 ms.

[5] https://github.com/bn999/autoquad/blob/master/onboard/canSensors.h:24
[6] https://github.com/bn999/autoquad/blob/master/onboard/can.h:71
[7] https://github.com/bn999/autoquad/blob/master/onboard/can.c:547

Registering node test

Implementation of a CAN-node running on a PC was initially done in python. The python module *SocketCan* [8] supports PeakCAN adapter out of the box.

Code 5.7 shows a snippet of main.py.

Listing 5.7 Snippet showing AQ registration from python

```
# Create autoquad handle
autoquadNode = AutoQuadNode('can1', 'socketcan')

# Wait for reset msg transmitted by AQ.
autoquadNode.WaitForReset(timeout=1000);
print "Recevied readymsg"

# Register node, CanType as sensor and CanID as PDB ampere
msg = autoquadNode.RegisterNode(CAN_TYPE_SENSOR, CAN_SENSORS_PDB_BATA)

# Wait for telemetryTryValue
msg = autoquadNode.recv()
# Parse message
msg = CanMessage(msg)

# Print all values received, _str__(self) overwrited
print msg

# ACK TelemValue
autoquadNode.AnswerRequestTelemValue(msg)

# Wait for telemetryRate
msg = autoquadNode.recv()

# Parse message
msg = CanMessage(msg)

# ACK TelemRate
autoquadNode.AnswerRequestTelemRate(msg)
```

When a node has successfully registered, it is shown in QgroundStation as shown in figure 5.3. It can be seen 4 motors were connected to the CAN bus, but also that the spoofed sensor was registered.

[8] http://python-can.readthedocs.io/en/latest/socketcan_native.html

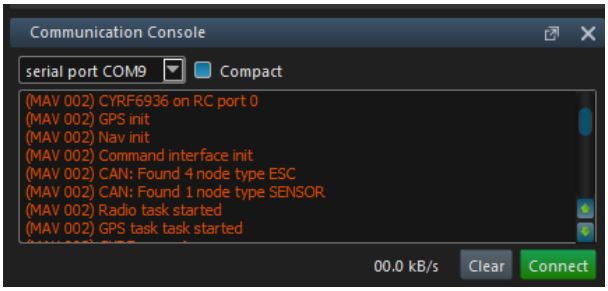


Figure 5.3 *Successful SENSOR registration*

Spoofed current test

When an AQ PDB is mounted on a drone, the PDB measures voltage, current and temperature and sends the measurements to AQ where they get logged.
The code in section 5.1 was further developed into a sensor simulator that spoofs current measurements into AQ to test the registering works probably.
A logger extension-board was mounted on the M4-board in order to save the measurements to a MicroSD-card.

Code 5.8 shows the implementation where a sinus is generated to simulate a current and how it is transmitted to AQ. [**?**]

Listing 5.8 Snippet spoofing current measurements into AQ

```
1  def ReqistrerTelem(self):
2     # self.sinus is a list of sinus values
3     sensor_value = self.sinus[self.phase]
4
5     # Get next value next time and do overrun
6     self.phase+=1
7     self.phase = self.phase % 20
8
9     # Pack as IEEE754 float
10    sensor_value_float = pack('>f', sensor_value)
11
12    # Create list of float-bytes as integers
13    data_float = [int(ord(elm)) for elm in sensor_value_float]
14
15    # Reverse list and append empty data
16    data_float.reverse()
17
18    # Send to AQ. id = CAN_FID_TELEM (02c00000) | Sourceid 1 (00000800) |
          CAN_SENSORS_PDB_BATA (00004000)
19    self.send(0x02c04800, data_float)
```

The method defined in code 5.8 was run every 100ms.
After the code ran for a few seconds the SD card was plugged into a PC and QgroundStation configured to make a plot of the PDB_CURRENT-field. The plot can be seen in figure 5.4

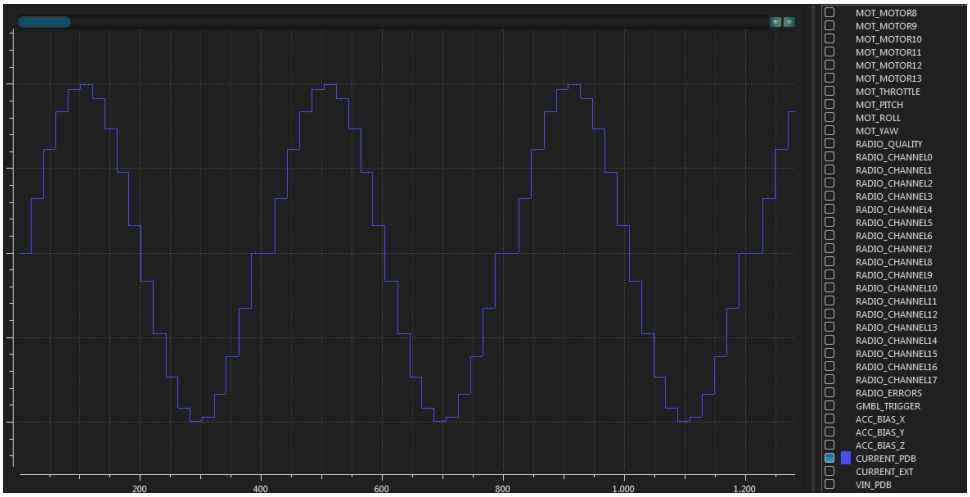


Figure 5.4 *Test of python CAN test*

Figure 5.4 shows the sinus generated from the node as expected.

5.2 Test of spoofed GNSS

Testing of the spoofed GNS was split intro two subtests.
The first test was conducted in order to validate the GPS spoofing using the CAN-bus works when the drone is laying on a table.
Test 1 can be seen in 5.2.1.
The second test was conducted as the first test, but when the drone is airborn.
Test 2 can be seen in 5.2.3

Figure Y shows the waypoint list uploaded to the drone. It is expected that the drone will behave as if it was using its onboard GPS. The GPS positions will be gathered in a rosbag to be used later on in test2. A flight with the onboard GPS will be done in order to have a reference flight.
Testx Test two will be conducted much the same way as test1. However the GPS will be replaced with a lightweight RTK GPS. The RTK GPS positions will also be saved in a rosbag for later analyse. Is is expected that when using the RTK GPS the drone is closer to its waypoints shown in figure Y, than when using a normal GPS.

5.2.1 Test of spoofed positions indoore

Introduction The purpose of this test was to see if the GPS coordinates was spoofed correctly and read probably by AQ and to validate the ROS node responsible for creating CAN messages was working as long as the drone is laying on a table. If this works, it shows that the UKF works with the spoofed coordinates as expected, and that the author of the report can spoof the GPS position from a vision based localization system.

A rosbag was used to collect data from the GPS since it publishes data to a topic when played as a node did when the recording took place. This results in no code has to be written and when the code works with a rosbag it also works when it is replaced with a node providing real-world data.

Test

Figure 5.5 *Block schematic of the connected components*

Figure5.5 shows the connected componenets.

To get the NMEA strings from the GPS, an already existing Frobomind component will be used. It reads from a serial port, and publishes the parsed NMEA string to */fmData/nmea_from_gps*. Rosbag was then used to save the messages. The code written by the author will then subscribe to */fmData/nmea_from_gps* and publish the CAN-messages to another topic. A third node which is also a part of Frobomind will then subscribe and send the CAN-messages.

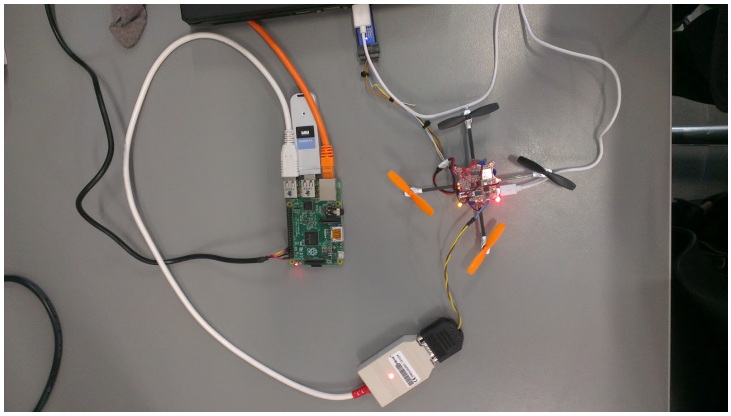


Figure 5.6 *Test-setup shown*

The Rpi is connected directly to the PC using the orange ethernet cable. The rosbag was stored and played on the PC, so the ethernet cable were used to share rostopics between the PI and the PC. The roscore where running on the PC.
The Ladybird drone is connected to the RPI using a PEAK CAN-adapter to transfer CAN messages. The Ladybird is powered by the white USB-cable and to show the position of the drone in QgroundStation. The PC is connected to the drone using ST-Link SWD to easier start, stop and restart the drones firmware during the test. To access a TTY on the PI, an FTDI cable has been connected to the PC.

In order to obtain GPS-testdata to spoof into AQ, a rosbag was used to record data collected by the GPS mounted on the authors laptop. The setup can be seen in figure 5.7

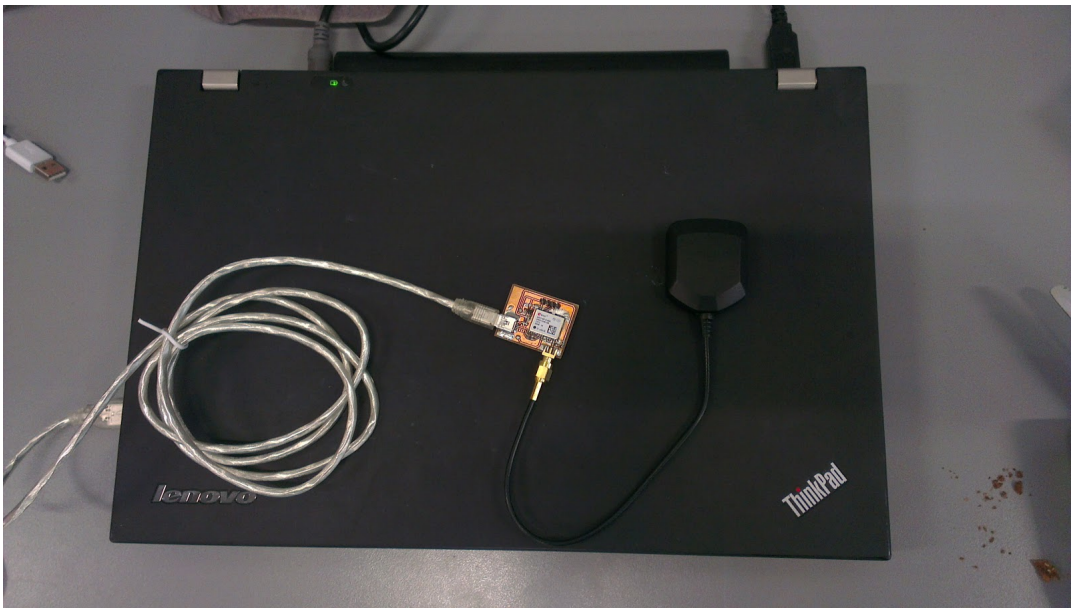


Figure 5.7 *The Neo6-P GPS (PCB developed by SDU), u-blox active GPS antenna* [9] *and authers laptop used in this test*

The coordinates recorded by the rosbag is shown in figure 5.8. FreeNMEA[10] was used to plot the GPGGA messages obtained from the GPS.

[10]http://freenmea.net/

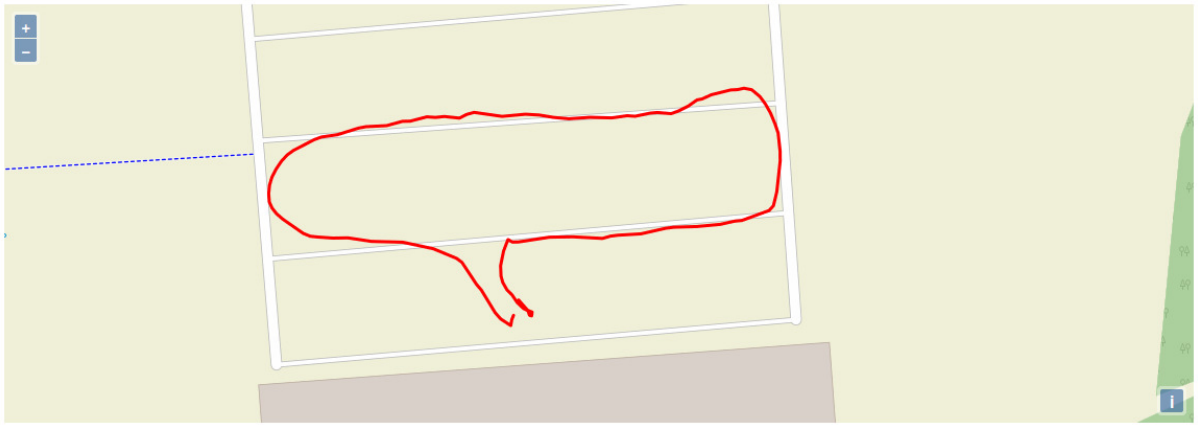


Figure 5.8 *Test coordinates plotted*

The rosbag containing the GPS coordinates where played on the Rpi, to see if the ROS node responsible for converting GPGGA messages into CAN messages were working. QgroundStation plots AQ's belief in where the drone is. The plot can be seen in figure 5.9

Figure 5.9 *QgroundStations plot of AQ's belief in where the drone is*

Conclusion This test visually confirms that the coordinates is spoofed correctly and read probably by AQ as long as the drone is laying on a table. Passing this test means there is reason to believe that it also works when the drone is airborne.

5.2.2 Initial test flight with spoofed GPS

Introduction The purpose of this test was to see if the GPS spoofing worked when the drone was airborn. The same hardware and software were used as in section 5.2.1, though with a few changes.

The GPS used in test 1 to collect coordinates were mounted on the EduQuad.

The idea was to pass-through the GPS coordinates from the Neo 6-P GPS to AQ. If the CAN spoofing works as expected the drone should behave normally when flying in position-hold

mode.

Test

Figure 5.10 shows how the hardware were mounted on the EduQuad. It was temporary mounted with strips and tape to avoid short circuiting.

Mathias: Initial test flight with spoofed GPS: Opdater billede med test-equipment

In test 1 the altitude of the drone was hardcoded since it had no relevance in the test. Though in test 2 if was necessary since the UKF[11] uses the altitude from the GPS to estimate the position of the drone. Furthermore the following check where inserted to make sure only valid GPS coordinates were fed into the AQ:

Listing 5.9 Quality checks added to discard bad positions

```
// default low accuracy
gpsData.hAcc = 5; // Meters
gpsData.vAcc = 5; // Meters

if(gpsData.fix == 1){ // Single point solution
   if(gpsData.hdop < 3){
      if(gpsData.satellites >= 5){
         AQ_PRINTF("Recv. Valid GPGGA\n",0);
         // High accuracy
         gpsData.hAcc = 2.5; // Meters
         gpsData.vAcc = 2.5; // Meters
         // Set flag to process new message
         CoSetFlag(gpsData.gpsPosFromCanFlag);
        } else {
         AQ_PRINTF("Satellites: %f \n", gpsData.satellites);
        }
    } else {
       AQ_PRINTF("HDOP: %f\n", gpsData.hdop);
    }
} else {
   AQ_PRINTF("Fix: %f\n", gpsData.fix);
}
```

[11] https://github.com/bn999/autoquad/blob/master/onboard/run.c#L111

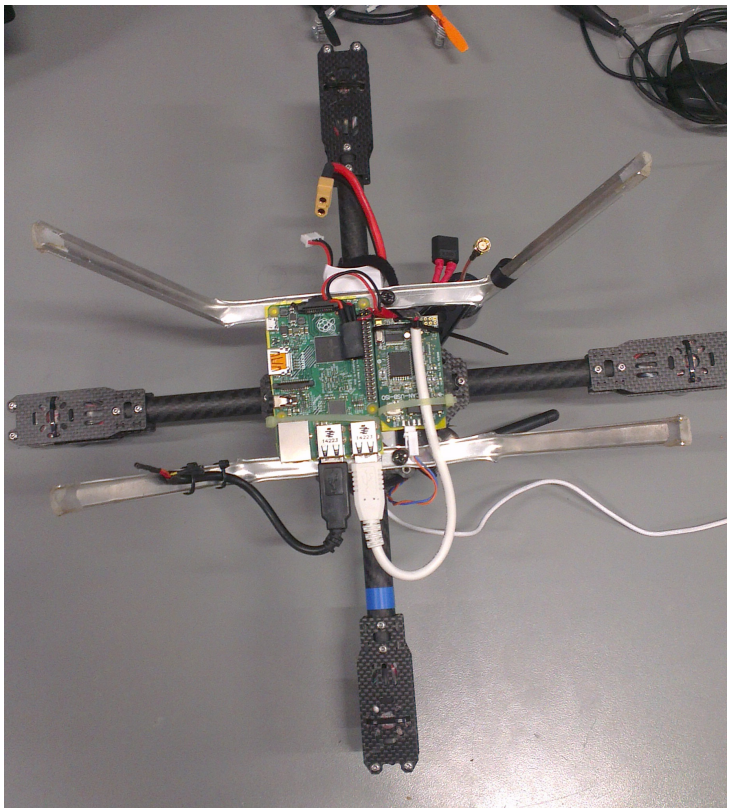


Figure 5.10 *The EduQuad drone with the hardware used in the test*

An extra battery was mounted on the drone to supply the Rpi. The ROS nodes running on the Rpi had to be start up before AQ starts registering notes on the CAN-bus. If the nodes was not started up, they would not reply to AQs reset-msg it sends when it is booting.

When the ROS-node were connected to the CAN bus, it showed that the ROS-node needed extra checking on CAN messages in order to detect difference between messages targeted the ROS-node and messages targeted the ESC32-nodes.

A bug in the *can_socketcan* -node used to communicate with the Peak-Can adapter where found a fixed. It was later on merged back into Frobomind [12] **Conclusion**

The test did not go as expected. The test was carried out under a time pressure since the author's supervisor only had 30 minutes to carry out the test.

The author did though gain experience when having multiple CAN devices connected to the CAN-bus.

Several things went wrong through the test:

- The onboard GPS was not disabled
- The author accidentally forgot to set the DOP values used in AQ
- No reference flight after the firmware were flashed for the first time.
- The GPS antenna was not mounted in the middle of the frame as the onboard antenna.

[12] https://github.com/FroboLab/frobomind/pull/12

Figure 5.14 shows the DOP value is lowering witch indicates that AQ was using its onboard GPS.

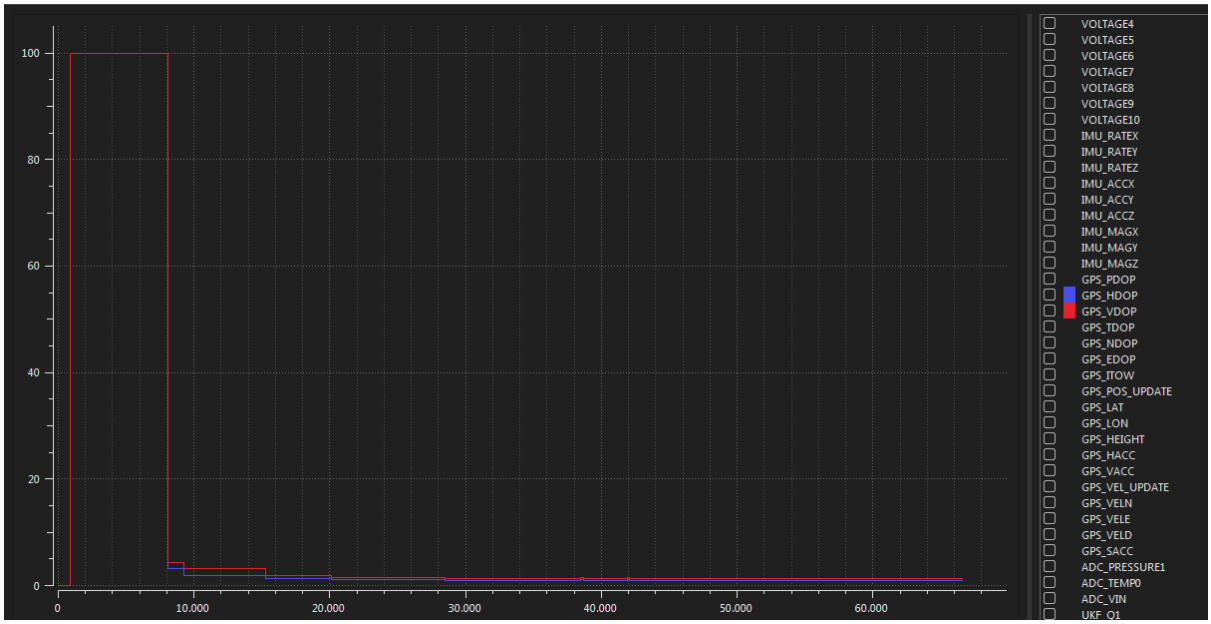


Figure 5.11 *Plot of h/vDOP from logs when the drone was airborn*

The test will be carried out again.

5.2.3 Comparison of onboard and spoofed GPS

This test was made together with Kjeld Jensen in order get one step further to spoofing the drones onboard GPS with an RTK GPS. A ublox Neo6-P GPS and an active ¡ANTENNE¿ was mounted on a EDUQuad drone to obtain the drone's position. A RPI was used on the EDUQuad to convert GPGGA and GPRMC from the Neo6-p GPS to CAN-mesasagess. The test was done by walking 20 meters with the EduQuad two times to compare the output of the UKF with the different GPS's. The values used to spoofed the GPS was found by analyzing a logfile from a regular flight to see the range of the values such as accuracy, DOP etc. when the drone was in the air. Values available from the Neo6-p GPS, was used directly to spoof the onboard GPS and values not available was either calculated or hardcoded. Three tests were made due to a few code-errors but only the first and the last will be described. This test was also relevant for the indoor flying since the indoor application requires the position of the drone can be spoofed and the accuracies and DOPs can be set

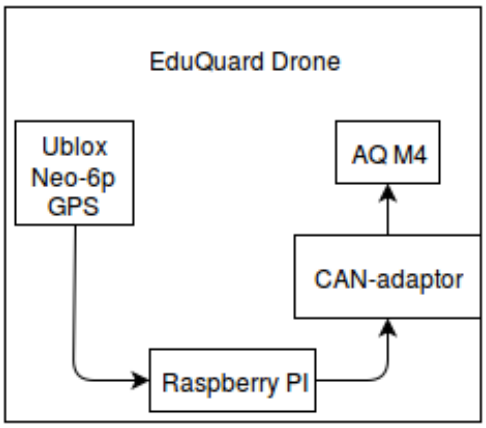


Figure 5.12 *Nep-6p connected to a M4 through a RPI*

In order to easily switch between the onboard GPS and the Neo-6P GPS during the test, a switch on the Spectrum DX9[13] was read in AutoQuad's firmware to decide which of the two GPS's data should be used. When a GPS update(time, position or velocity) is received from the onboard GPS, the struct showed in code 5.10 is filled in.

Listing 5.10 Quality checks added to discard bad positions

```
typedef struct {
   ...
    double lat;
    double lon;
    float height; // above mean sea level (m)
    float hAcc;   // horizontal accuracy est (m)
    float vAcc;   // vertical accuracy est (m)
    float velN;   // north velocity (m/s)
    float velE;   // east velocity (m/s)
    float velD;   // down velocity (m/s)
    float speed;  // ground speed (m/s)
    float heading; // deg
    float sAcc;   // speed accuracy est (m/s)
    float cAcc;   // course accuracy est (deg)
    float pDOP;   // position Dilution of Precision
    float hDOP;
    float vDOP;
    float tDOP;
    float nDOP;
    float eDOP;
    float gDOP;

    //Used in GPS spoof
    uint8_t fix;
    uint8_t satellites;
    int out_cnt;

   ...
} gpsStruct_t;
```

[13] http://www.spektrumrc.com/Products/Default.aspx?ProdId=SPMR9900 last visited 21. Maj

In order to spoof the onboard GPS correctly, the author and his supervisor decided to overwrite all of the relevant elements in the struct. However, the RTC was set from the onboard GPS.

Tables tables 5.6 to 5.10 show the CAN-messages created to send data to fill in the struct.

DOC	CAN_DOC_LAT
Value	Latitude
Bits	63:0

Table 5.6 *CAN messages from RPI to AutoQuad containing 8 byte latitude*

DOC	CAN_DOC_LON
Value	Longitude
Bits	63:0

Table 5.7 *CAN messages from RPI to AutoQuad containing 8 byte longitude*

DOC	CAN_DOC_VEL			
Value	VelN	VelE	VelD	speed
Bits	63:48	47:32	31:16	15:0

Table 5.8 *CAN messages from RPI to AutoQuad containing velocities each of 2 bytes*

DOC	CAN_DOC_ALT
Value	Altitude
Bits	63:0

Table 5.9 *CAN messages from RPI to AutoQuad containing 8 byte altitude*

DOC	CAN_DOC_DOP						
Value	gDOP	eDOP	nDOP	tDOP	vDOP	hDOP	pDOP
Bits	55:48	47:40	39:32	31:24	23:16	15:8	7:0

Table 5.10 *CAN messages from RPI to AutoQuad containing DOPs each of 1 byte*

DOC	CAN_DOC_ACC						
Value	Heading	vAcc	hAcc	cAcc	sAcc	Fix	Satellites
Bits	63:48	47:40	39:32	31:24	23:16	15:8	7:0

Table 5.11 *CAN messages from RPI to AutoQuad containing accuracies each of 1 byte and heading in 2 bytes*

In order to know the required bytes to send latitude and longitude with a precision of 1 cm, the author used the datum WGS84[14] to estimate the worst-case error at equator.

WGS-84 [15] states that the semi-major-axis(a) of the earth is 6378137.0 m and that the flattening factor is 1/298.257223563. From these two values the semi-minor-axis(b) can be calculated using equation 5.2.3

$$b = a \cdot (1 - \frac{1}{298.257223563}) \tag{5.1}$$

Given the formula of the circumference of an ellipsoid:

$$circumference = 2 \cdot \pi \sqrt{\frac{1}{2}(a^2 + b^2)} \tag{5.2}$$

[14]WGS-84 used in AQ

[15]http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf last visited 22 maj

Equation 5.2.3 gives a circumference at equator of 39873752.0 meters.

By dividing 39873752.0 by 360, one get how many meters pr. degree.

$$\frac{39873752.0}{360} = 110760\frac{m}{deg} \tag{5.3}$$

Table 5.12 was made to see how many decimals is required to obtain different precisions.

Table 5.12 *Table shows precision required*

Decimal place	Decimal degrees	Distance
0	1	110760 m
1	0.1	11076.0 m
...	..	...
7	0.0000001	7.75323 cm
8	0.00000001	0.775323 cm

It can be seen that 8'th decimal is necessary to get below 1 cm. To see if a float can handle number smaller than $1 \cdot e - 8$, *eps* from MatLab was used[16].

Listing 5.11 Check if float is precise enough of a double has to be used

```
1 eps(single(180.0)) = 1.5259e-05
2 eps(double(180.0)) = 2.8422e-14
```

In code 5.11 eps is given with 180 as input since 180 is the largest integer possible with using latitude and longitude. It can be seen that the smallest number a float can represent is 1.5259e-05 but a precession of $1 \cdot e - 8$ is needed. However a double can easily handle that which makes the latitude and longitude of the CAN-messages 8 bytes each.

2 bytes are allocated for each velocity, speed and heading. In order to send 2 decimals, each value was multiplied by 100 when sending and divided by 100 when received by AutoQuad.

Only one decimal for rest of the values was deemed necessary and was thereby multiplied and divided by 10.

To tell the UKF in AQ how much to believe in the GPS coordinates, the onboard GPS feeds the UKF with DOP factors. The DOP factors expresses how well the satellites are placed. If they are all located around the same spot, the DOP factors will be high since the triangulation becomes less accurate. The DOP factors has no unit but is simply a scaling of the error estimate. [10]

Mathias: Comparison of onboard and spoofed GPS: Insert image from normal flight

[16] http://se.mathworks.com/help/matlab/ref/eps.html

Figure ?? shows a flight using onboard GPS where the DOP values where read. The values listed in table 5.13 was used and multiplied by the hDOP received from the Neo-6P GPS.

The pDOP was set to 1.75 times higher than the hDOP since it incorporates error from both vDOP and hDOP [10]. vDOP is usually 1.5-2 tims higher than hDOP. [10].
tDOP was sat to 1.5 higher than hDOP since timeerror causes position error. [17]
Since hDOP can be split into nDOP and eDOP, they each of some of the error factor.

Figure 5.13 *Scaling factors used when using Neo-6p GPS*

Dop	Value
pDOP	1.75*hdop
hDOP	1*hdop
vDOP	1.5*hdop
tDOP	1.5*hdop
nDOP	0.7*hdop
eDOP	0.7*hdop
gDOP	1*hdop

Mathias: Comparison of onboard and spoofed GPS: Gdop, skal den ikke være højere siden den har både vDOP, hDOP og tDOP

[10]

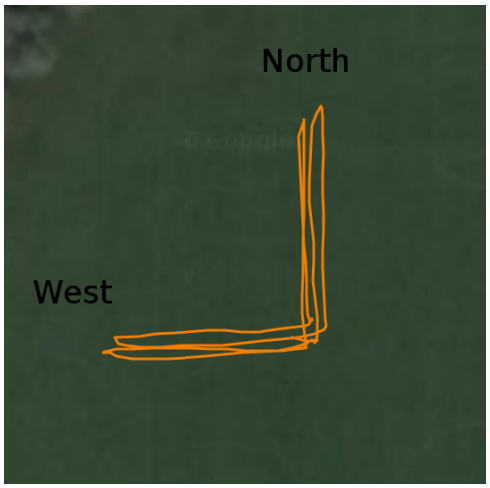


Figure 5.14 *Path walked. If looking close it can be seen the path was walked twice, once with onboard GPS then using Neo-6p*

The test was conducted by walking 20 meters [18] north, 20 meters south, 20 meters east and 20 meters west. After walking 20 meters a small break of five seconds was held. Figure ?? shows the walked path.

Mathias: Comparison of onboard and spoofed GPS: Figure med gpgga plot

The coordinates plotted in figure ?? was from the first test. Figure ?? shows a plot of the position with x-axix as time.

Mathias: Comparison of onboard and spoofed GPS: Plot af northing and easting fra ukf

Figure ?? matches with the walked path seen in figure ??. The line crossing severel times is the switch on the Spectrum DX9 transmitter. It can be seen how the path repeats itself after

[17] http://www.environmental-studies.de/GPS/Dilution-of-precision/dilution-of-precision.html last visited 22 maj
[18] Measured by counting footsteps

the switch has been switched.

Mathias: Comparison of onboard and spoofed GPS: plot af pos og switch samt dop så man kan se de ændrer sig ved skift.

Figure **??** shows the same plot but where the vDOP and hDOP has been added. It can be seen when zoomed in, how the DOP values is also changing since the DOPs are higher when switching to the Neo-6p GPS.

Mathias: Comparison of onboard and spoofed GPS: Billede der viser højden/hastigheden er mere woobly siden DOP er mega lav, den stoler for meget på GPS

Figur der viser højden samt højdehastigheden
figur der viser vele og veln der ikke virker.
Ny test, samme måde som første test, dog med følgende resultater [19]
[10]

5.3 Documentation made throughout this thesis

[19] fejlen skyltes forkert blevev hevet ud fra C++ vector samt en variabel var brugt forkerte sted.