Controlling multiple drones autonomously inspired by birds ability to keep formation

University:
University of Southern Denmark

Author:
Mathias Mikkel Neerup

Bachelor Thesis

In cooperation 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

Controlling multiple drones autonomously inspired by drones ability to keep formation

At kontrollere flere droner autonomt inspiret af fugles evne til at holde formation

1. February 206 - 1 June 2016

In cooperation with ViaCopter

Electronic appendix at: https://goo.gl/qrHRcK

Supervised by
Kjeld Jensen

May 31, 2016

Todo list

Abstract

Up until now drones keeps getting bigger and larger to carry bigger batteries with more capacity and to lift heavier payloads. This has lead to drones getting less efficient, less responsive and more dangerous. Lately it has become more popular to make small collaborative drones to solve the task.

Mathias: Abstract: Skal omskrives til sidst

(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 motivation of this project. It starts giving an overview of the topic and ends up focusing on the motivation of the work. It further contains related work, problem statement, defined hypothesis
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. Almost every section starts with an introduction and ends with test results evaluating whether the choices made works or not.

Chapter 3,4 concerns showing and discussing results with respect to the hypothesis. 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 to help the reader.

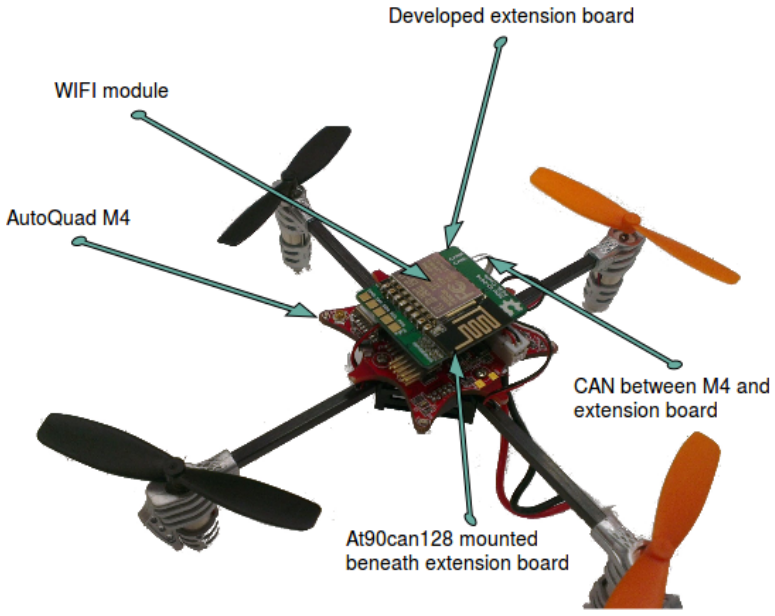


Figure 1 *Ladybird M4 drone with developed extension board on top*

List of Acronyms

ADC Analog to Digital Converter

AQ AutoQuad

BLE Bluetooth Low Energy

CAN Controller Area Network

CPU Central Processing Unit

CRC Cyclic Redundancy Check

DHCP Dynamic Host Configuration Protocol

DIY Do It Yourself

DOP Dilution of precision

ENU East, North, Up

ESC Electronic Speed Control

FMA Flying Machine Arena

FTDI Future Technology Devices International

LL Latitude, Longitude

LLH Latitude, Longitude, Height

GNSS Global Navigation Satellite System

GPIO General-Purpose Input/Output

GPS Global Positioning System

IMU Inertial Measurement Unit

IP Internet Protocol

IR Infrared

ISP In-System Programming

LED Light-Emitting Diode

NED North, East, Down

OTA Over The Air

PCB Printed circuit board

PDB Power Distribution Boards

RGB Read-Green-Blue

ROS Robot Operating System

RPi Raspberry Pi

RTC Real-Time Clock

RTCS Run To Completion Scheduling

RTK Real Time Kinematic

RTK-GNSS Real Time Kinematic-Global Navigation Satellite System

RTOS Real-time Operating System

SDU University of Southern Denmark

SLIP Serial Line Internet Protocol

SOC System On A Chip

SWD Serial Wire Debug

TED Technology, Entertainment, Design

TTY Teleprinter

UART Universal Asynchronous Receiver/Transmitter

UAV Unmanned aerial vehicle

UDP User Datagram Protocol

UKF Unscented Kalman filter

USB Universal Serial Bus

UTM Universal Transverse Mercator

UUID Universally Unique Identifier

WPA2 Wi-Fi Protected Access II

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 proofreading the thesis.
- Leon Bonde Larsen for proofreading 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 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.

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]

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 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

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

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 manually, as is is currently, and save time and 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]

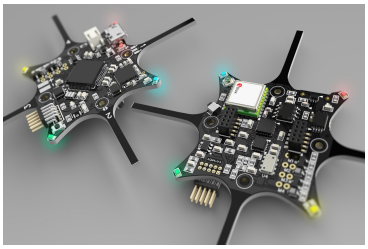
In another project, SDU is working together with Ecobotix ApS, Aarhus Univercity and EWH BioProduction ApS, to avoid the need of pesticides in organic crops. Pesticides are chemical substances used to kill pests from the crops. When using pesticides a health risks exists for those who eat the crops. 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

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

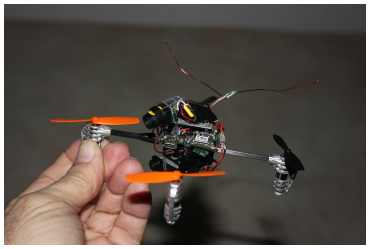
fields. [8]

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*

[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 does not support vision as source of 2D position which is required for the indoor Leader-follower to work. The relative height of the drone is measured using the build-in barometer in each drone providing the third dimension to the drones position. In case the barometer turns out to be too inaccurate due to drift other sensors might be used e.g ultrasound. A PCB for each drone has to be developed for the drones to carry as payload. The PCB will be responsible for receiving messages from the computer and translate them into the CAN-bus of the drone.

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

[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

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 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 vision and spoofed into the drone using CAN, then it is possible for at least 3 drones to follow a leader drone with a preprogrammed flight path and keep a euclidean distance at 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 overall system architecture indoor as well as outdoor. The section goes in details with the components used and how they are connected. Furthermore it describes the information flow through the system. The design choices and implementation details will be covered later in this chapter.

The section has been split into two subsections in order to address indoor and outdoor positioning separately.

2.1.1 Indoor

Figure 2.1 shows the system architecture for indoor positioning. The laptop detects the drones position using vision, sends it through the ROS nodes and transmits the GPS positions to the drones via WIFI. Each drone receives its position and sends it into the M4 flight controller over Controller Area Network (CAN).

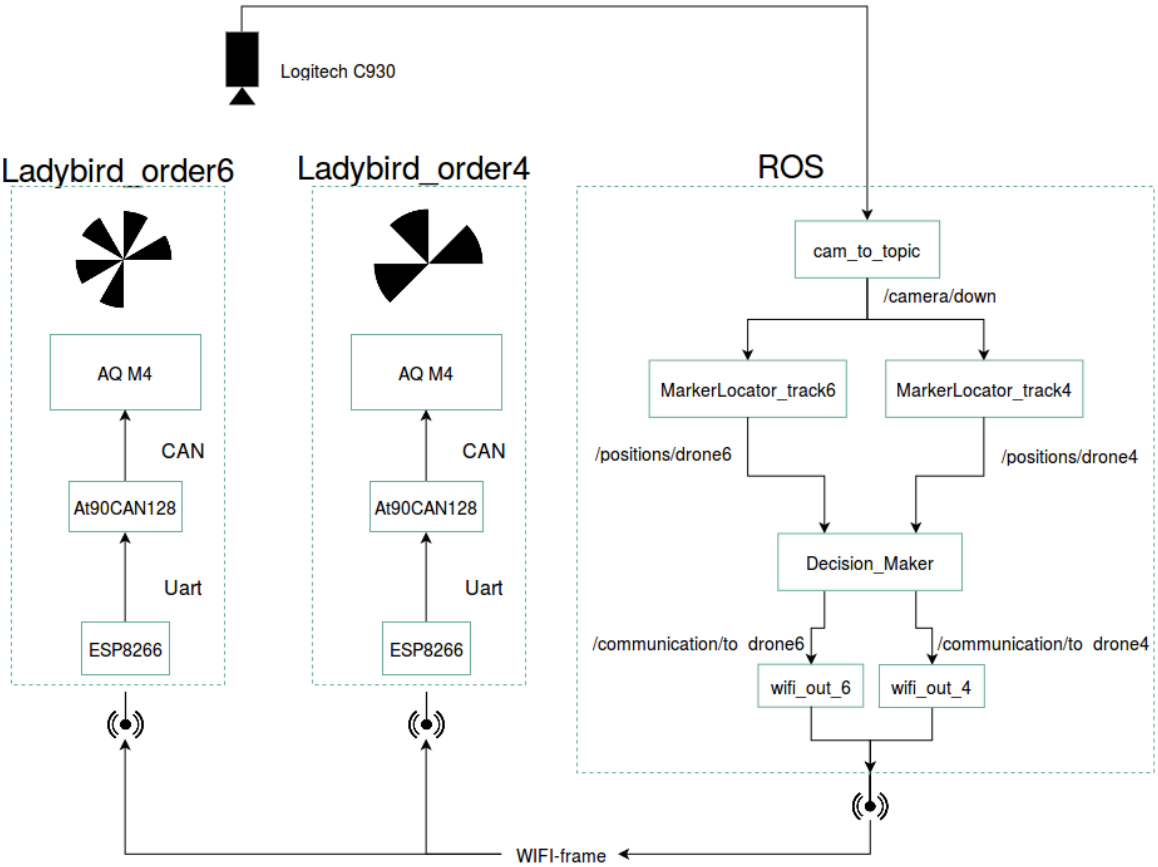


Figure 2.1 *Diagram showing system architecture when used indoor. The markers used to track the drones is shown below the camera.*

A Logiteh C930 camera[1] was mounted below the ceiling to detect the drones when flying. The order of the marker of each drone is unique in order to identify the particular position of the drone.

ROS

Robot Operating System (ROS)[2] is used as middleware on the laptop as inter-process communication. By using ROS it is easier to debug since each node[3] can be isolated and debugged without everything has to be connected. ROS uses subscriber-publisher pattern which means one or more nodes can produce data, and one or more nodes can consume data. Topic is the term used to define a communication channel between nodes. Nodes producing data are referred to as publishers, and nodes consuming data are referred to as subscribers. Furthermore ROS supports running nodes distributed across multiple PCs meaning the MarkerLocators can be distributed if more CPU resources are needed when tracking several drones. The squares shown in the ROS section in figure 2.1 shows the ROS nodes and how they are connected using topics.

- *cam_to_topic* captures frames from the Logitech camera and publish the raw frame to */camera/down* as a RGB picture.

[1]The camera is used in other courses to track mobile robots so the author did not have any influence on which camera to use
[2]http://www.ros.org/
[3]Process in ROS terminology

- *MarkerLocator_trackN* subscribes to *cam_to_topic* and processes the frame in order to localize the marker or order N of the drone in the frame. This node publishes messages to */positions/droneN* containing x,y position, orientation and a boolean telling whether the right marker is found in the frame or not. The MarkerLocator was customized to support receiving frames from a topic instead of the camera directly. By doing this, multiple instances of the MarkerLocator can be run in parallel without their performance impact each other. [4]

- *Decision_Maker* subscribes to the topics containing the position of the drones. This node publishes messages to */communication/to_droneN* containing the Latitude, Longitude, Height (LLH) position of the drone. This node contains the logic for controlling the drones [5]. Due to the modularity of ROS, changing the behavior of the drones is a matter of replacing this node.

- *wifi_outN* subscribes to */communication/to_droneN* and transmits the received messages to the drone using WIFI. The responsibility of this node is to pack the message, calculate Cyclic Redundancy Check (CRC), append it and send it as an User Datagram Protocol (UDP) packet. The content of the frame can be seen in table 2.1.

Content	CRC-16	Future	eDOP	nDOP	tDOP	vDOP	Height	Lon	Lat
Datatype	uint16_t	4 byte	byte	byte	byte	byte	double	double	double
Bytes	33:32	31:28	27	26	25	24	23:16	15:8	7:0

Table 2.1 *Table shows the frame sent from the wifi_out_N to the extension-boards. 4 bytes is not utilized but can be used for anything or the frame can be reduced in size*

The frame in table 2.1 is 34 bytes long but can be enlarged or made smaller if needed. However the ESP8266 module has a limitation of 8192 bytes [6]

The initial idea was to also send the velocities of the drone and the accuracies to each extension-board since this information is used by AQ M4, however due to lack of time this was not implemented.

Drone

The drone is a Ladybird using a AutoQuad (AQ) M4 as flight controller with a modified version of the firmware.[7] Is has an extension-board that enables it to receive positions from WIFI and inject them into the AQ M4 over CAN. [8]

- *ESP8266*[9] receives the UDP packet sent from the computer running ROS. When a frame is received, it checks if the received number of bytes it correct. If so, it encapsulates the packet using Serial Line Internet Protocol (SLIP) and transmit it to the At90CAN128.

[4]Indoor localization is described in section 2.5
[5]Decision_maker is described in section 2.6
[6]https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/WiFiUdp.h#L28 last visited 28 Maj
[7]AQ M4 firmware is described in section 2.4
[8]The extension-board is described in section 2.2
[9]The firmware running on the ESP8266 module is described in section 2.3.2

- *At90CAN128* receives the encapsulated packet, decapsulate it, verify the CRC and transmit the packet to the AQ M4 over CAN. Tables tables 2.2 to 2.6 show the CAN-messages send to the AQ M4.

DOC	CAN_DOC_LAT
Value	Latitude
Bits	63:0

Table 2.2 *CAN message to AQ containing 8 byte latitude*

DOC	CAN_DOC_LON
Value	Longitude
Bits	63:0

Table 2.3 *CAN message to AQ containing 8 byte longitude*

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

Table 2.4 *CAN message to AQ containing velocities each of 2 bytes*

DOC	CAN_DOC_ALT
Value	Altitude
Bits	63:0

Table 2.5 *CAN message to AQ 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 2.6 *CAN messages to AQ containing Dilution of precision (DOP)s 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 2.7 *CAN messages to AQ containing accuracies of 1 byte and heading in 2 bytes*

Network configuration

The WIFI network for handling the communication between the PC and the drones had to be configured [10]. The PC was configured as an access point for each extension-board to join. The access point was setup using an Asus USB-N13[11] netcard and hostapd[12] as software. The network was configured as Wi-Fi Protected Access II (WPA2) to make it difficult for other to connect and start messing with the network.

Instead of hardcoding an Internet Protocol (IP) for each extension-board, which would be cumbersome each time a new firmware had to be deployed, a Dynamic Host Configuration Protocol (DHCP) server was configured on the PC. Since the IP might change next time the module connects, Multicast-DNS was configured on the ESP8266 and the PC. Each extension-board was giving a hostname eg. Drone1 so when the PC has to send a frame to Drone1, the underlaying networking will resolve the IP of Drone1. The ESP8266 has a small filesystem build-in which support creating a configuration file that could contain the hostname. When flashing

[10] Wireless communication chosen in section 2.2
[11] Not available on AUSUS' webpage.
[12] https://w1.fi/hostapd/

the ESP8266 module the configuration file[13] would not be overwritten and thereby still have the hostname of the module. Instead of using hostnames the IP of the drone could be entered in the configuration file, however this would make it less portable since the PC would always have to be on the same IP-range. It was chosen to use UDP packets to transfer data from the PC to the ESp8266 module. **TCP!** (**TCP!**) could have been used since it retransmits packets if they are lost and takes care of integrity check. If using **TCP!** and a packet is lost during transmission it will be sent again. However the packet will be outdated when the ESP8266 modules receives the packet. Instead UDP was chosen since the data is sent with 5 hz, nothing happens if a packet is lost.

2.1.2 Outdoor

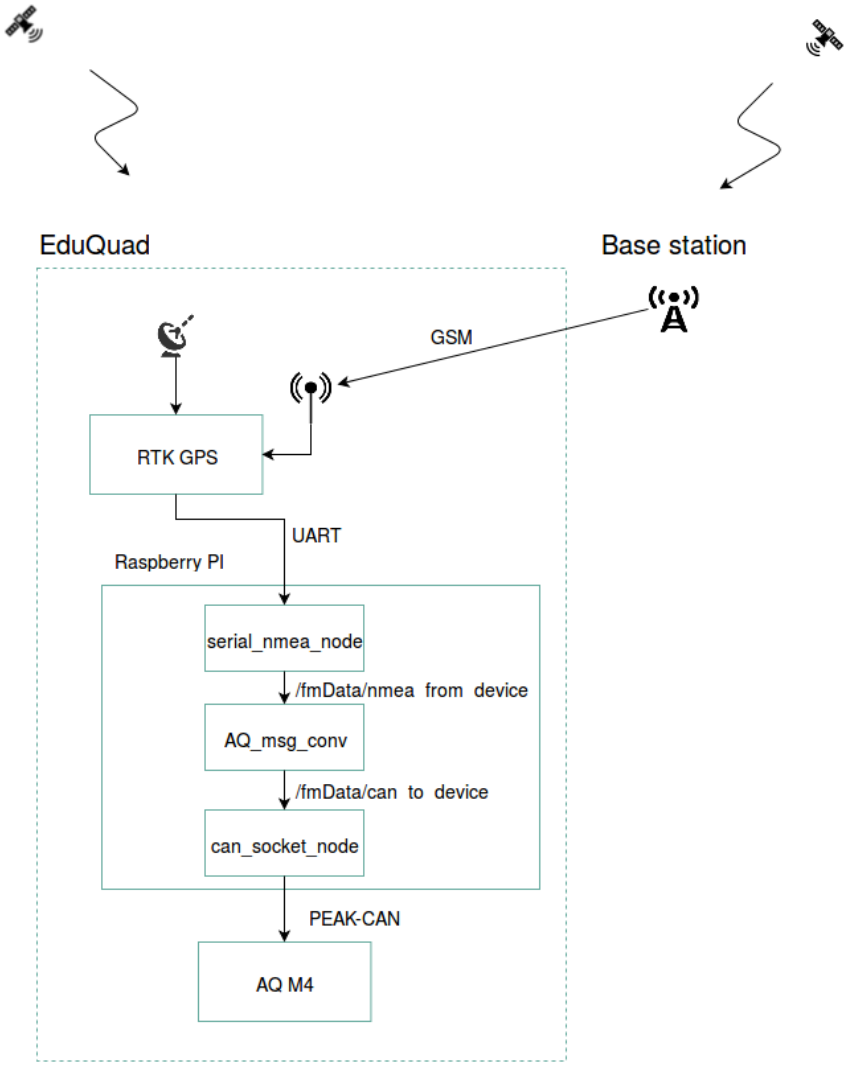


Figure 2.2 *Diagram shows information flow when used outdoor. The RTK-GNSS provides a absolute position which is read and converted to CAN messages by the Raspberry PI*

[13] Due to lack of time the configuration file was not made, and the hostname was hardcoded in the ESP8266 firmware.

The Raspberry Pi (RPi) is also running ROS since it uses two components from Frobomind[14].

- *serial_nmea_node* is used to read and write from the serial port. Further more it decodes the NMEA string sent by the Real Time Kinematic-Global Navigation Satellite System (RTK-GNSS)
- *can_socket_node* is used to communicate with the CAN-adapter[15]

When a GPGGA/GPRMC nmea [16] message is received from the RTK-GNSS its read by the *serial_nmea_node* node. It is then sent to the *AQ_msg_conv* node which simply converts the content of the GPGGA/GPRMC to CAN-messages shown in tables 2.2 to 2.6 .

Even though the positioning is shown using vision and RTK-GNSS, other sources of positioning can be used as well. The same code is running on the AQ M4 whether vision or RTK-GNSS is used, so if other positioning systems should be used one should implement the CAN protocol shown in appendix 4.1. If using CAN is not possible a RPi and CAN-adapter can be used as intermediate, but then the *serial_nmea_node* and *AQ_msg_conv* should be replaced.

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 several requirements it needs to fulfill in order to make the whole system work as expected. A comparison table has been made in table 2.8 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 [17] 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.[18] 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.

[14] http://frobomind.org/ last visited 29 Maj
[15] PEAK-CAN Adapter has been used in this project
[16] http://www.gpsinformation.org/dale/nmea.htm last visited 28 Maj
[17] https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266/open-source-hardware
[18] http://www.esp8266.com/ - 43.000 posts in forum

Their official Arduino GitHub has 2125 commits on their master branch at the time of writing[19]

- **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 [20] 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[21]. Their most active GitHub has 147 commits on their master branch at the time of writing[22].

- **nRF51822**

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[23] {7}	1.5g[24] {8}	7.5$[25] {9}	Great community {9}	?	33
EMW3165	32x16mm[26] {6}	5g[27] {2}	9$[28] {8}	Less available {3}	?	19
nRF51822	18x10mm[29] {8}	3g[30] {4}	7.5$ [31] {9}	Ok documented {2}	30 M[32] {9}	32
XBee	24.38x27.61mm [33] {3}	3g [34] {4}	25$[35] {4}	Lots of DIY {9}	91 M[36] {9}	29

Table 2.8 *Comparisontable used to compare different wireless communication modules*

[19] https://github.com/esp8266/Arduino
[20] https://hackadaycom.files.wordpress.com/2015/07/emw3165.pdf
[21] http://www.emw3165.com/ - 200 posts in forum
[22] https://github.com/SmartArduino/WiFiMCU
[23] https://www.mikrocontroller.net/attachment/243558/fcc_11.pdf last visited 26 Maj
[24] Measured by author on chemistry weight

The products compared in 2.8 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[37]

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[38], 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.

[25] http://www.seeedstudio.com/depot/ESP8266-based-WiFi-module-SPI-supported-p-2486.html last visited 26 Maj
[26] https://hackadaycom.files.wordpress.com/2015/07/emw3165.pdf last visited 26 Maj
[27] http://www.seeedstudio.com/depot/EMW3165-CortexM4-based-WiFi-SoC-Module-p-2488.html last visited 26 Maj
[28] http://www.seeedstudio.com/depot/EMW3165-CortexM4-based-WiFi-SoC-Module-p-2488.html
[29] http://www.fanstel.com/Product/bluenor.htmllast visited 26 Maj
[30] http://www.seeedstudio.com/depot/MDBT40P%C2%A0%C2%A0nRF51822%C2%A0based%C2%A0BLE%C2%A0module-p-2503.html last visited 26 Maj
[31] http://www.seeedstudio.com/item_detail.html?p_id=2503 last visited 26 Maj
[32] https://dl.dropboxusercontent.com/u/54939426/Fanstel_BT600.pdf last visited 26 Maj
[33] http://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4#specifications
[34] http://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4#specifications last visited 26 Maj
[35] https://www.sparkfun.com/products/11215 last visited 26 Maj
[36] https://www.sparkfun.com/pages/xbee_guide
[37] http://www.esp8266.com/ last visited 26 Maj
[38] http://www.frobomind.org/index.php/FMCtrl:FroboMind_Controller last visited 22 Maj

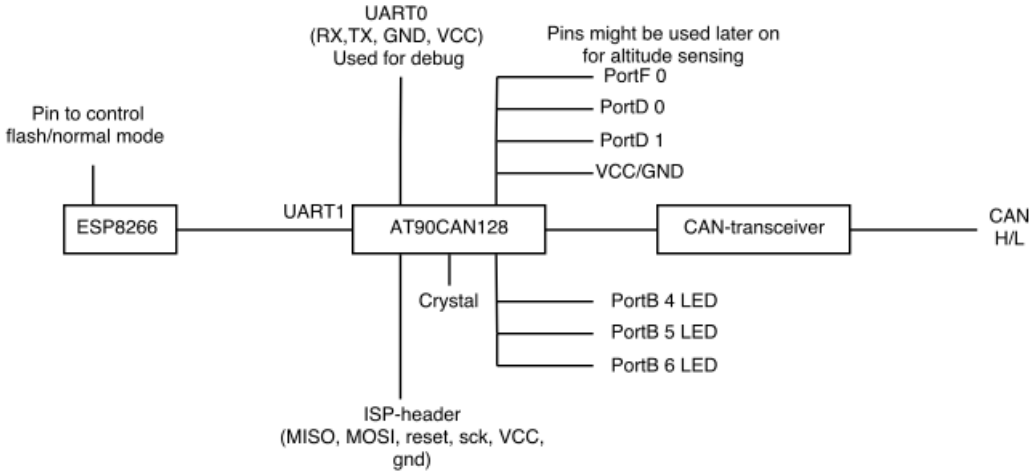


Figure 2.3 *Block schematic of the extension-board developed to AQ M4. It can be seen that the ESP8266 chip is connected to the At90CAN128 using UART1 and that the At90CAN128 is connected to a CAN-transceiver.*

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

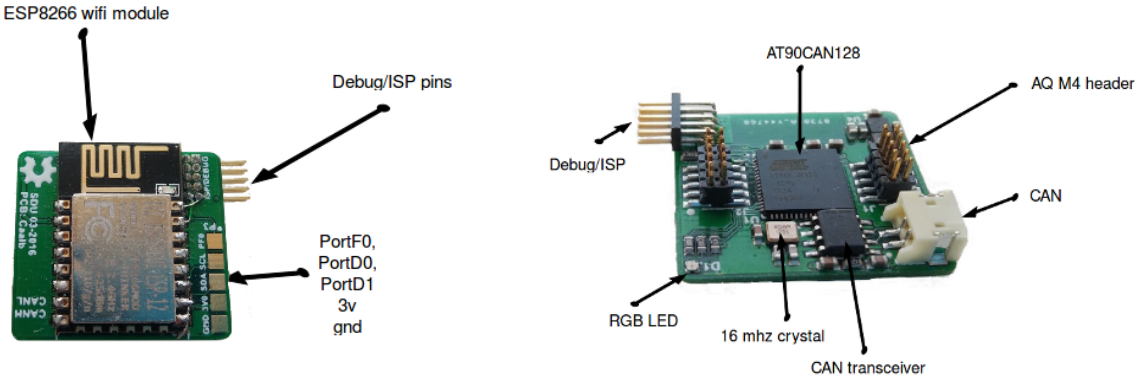
In case the onboard barometer is not accurate enough to keep the drone at a constant height, an alternative distance could be used to measure the drones altitude with respect to the ground. PortF0 has been made available since some distance sensors give output as an analogue value. An example of such sensor is an Infrared proximity sensor.[39]
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.[40]. To detect the high-time, one of PortD1/0 would be useful. Sensors using I2C could be used as well.
Which type of sensor suits best as a distance sensor to provide altitude information to the drone is ouf of scope of this report. The PCB has just been made ready to support different types of sensors.

Debug/ISP

In the final schematic Universal Asynchronous Receiver/Transmitter (UART)0 and ISP pins where combined in one pinheader for easy access through one cable. The connector can be seen in figure 2.4.

[39] http://www.sharpsma.com/webfm_send/1208
[40] http://www.micropik.com/PDF/HCSR04.pdf

To program the At90CAN128 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 At90CAN128 as UART passthrough from UART0 to UART1. Due to a mistake [41] in the final schematic[42], both UART0 and UART1 where made accessible trough the ISP/debug header. This ended up making it easier to program the ESP8266-board without using the At90CAN128 as UART pass-through.



(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 the extension-board was done by making small sketches testing the individual parts of the circuit. The sketches used to blink the leds and proof that the basic functionality works is left out. CAN is tested in section 4.2.

Test of WIFI range

A test of the WIFI was conducted in order to test the range of the ESP8266 module.

To make the test, the firmware of the ESP8266 module had to be flashed to specify which access point [43] it should connect to. The tests were conducted on an open grass field in order to avoid as much disturbance as possible.

The first test was conducted by pinging the module from the authors laptop 200 times, 10 hz with a packet size equal to the size of the frame used.[44]. The purpose of the test was to see how the latency behaves when the distance is increased and at what distance the WIFI-connection is dropped.

Figure 2.5 shows the results of the ping test.

[41] 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

[42] Can be found at *Appendix/ESP8266_CAN_Autoquad29032016_3.PDF*

[43] The network configuration is described in section 2.1.1

[44] The WIFI frame is described in section 2.1.1

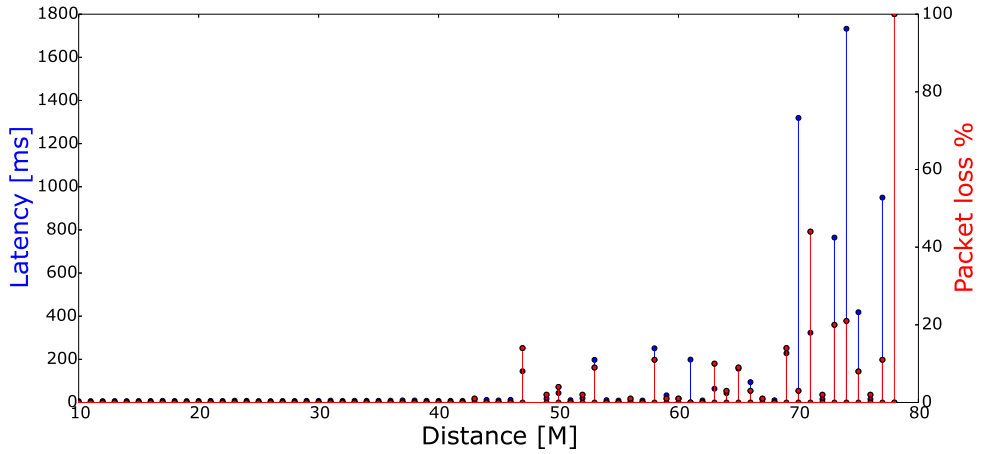


Figure 2.5 *Plot shows the latency, packet loss vs. distance. Up until 46 meters no packets are dropped and the latency is quite low.*

It can be seen, that until 46 Meters no packets is dropped and the latency is low. From 46 to 69 the latency starts to increase and packets begin to get dropped. At 69 meters and up to 80 meters the latency is high and packet loss percentage is increased. Since ping works by sending a request and waiting for the answer this tests communication both ways. However only one way communication from the laptop to the ESP8266 is required.

Another test was done where the authors laptop sends valid frames[45] with CRC[46] to the ESP8266 module. The tests shows for how long a distance the module can be from the laptop and still receive valid packets.

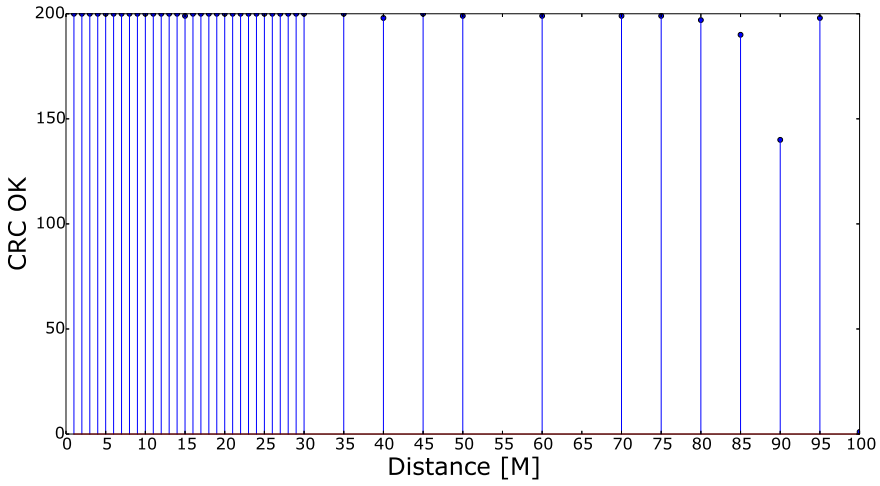


Figure 2.6 *Measurements was initially done at every meter, however to save time the distance was increase to 5 meter and later to 10. When packets began to drop, measurements was done at 5 meter interval again. At 100 meters the WIFI connection was dropped.*

At figure 2.5 and 2.6 it can be seen, that at a distance higher than 46 meters the latency increases but almost every CRC packet is received and valid. At 90 meters the number of valid

[45] Frame described in section 2.1.1
[46] CRC described in section 2.3

frames drops and a ping response is not received. Even though frames it still received and valid they will be delayed.

A final test was made to see how the modules behave when two modules are receiving the packets. At 10 hz 200 frames were sent to two modules at a distance of 10 meters. Unfortunately only one connector to communicate with the extension-board were made which made it difficult to check if both extension-boards received the data without error at the same time. The test was done 4 times while alternating between the modules to verify both modules were receiving all the packets.[47]

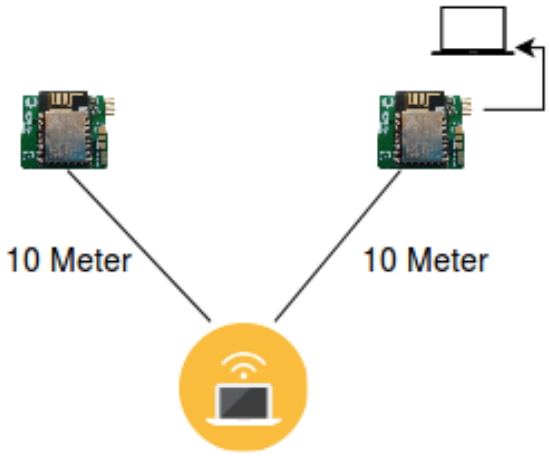


Figure 2.7 *Test setup where two modules receives 200 frames at 10 hz at the same time. It can be seen that the laptop only checks the received number of frames one extension-board at a time.*

Both modules received 200 packets without CRC error

Conclusion

The ESP8266 wireless module was chosen based on requirements in table **??**. The ESP8266 got the highest score and was chosen since it is a widely used module. The extension-board was created based on the block schematic shown in figure 2.3. Three tests was conducted to test the performance of ESP8266 module. If the distance is higher than 46 meters the latency begins to increase and the CRC packets will be delayed.

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. Furthermore the firmware running on the ESP8266 is described. Several tests are conducted to test if it works

2.3.1 Scheduler

This captor concerns only the At90CAN128. The ESP8266 module is described in section 2.3.2
In order to have a timing on the At90CAN128 a scheduler was chosen and implemented. The

[47] If more time were available, a timing-test of the setup would have been done. By measuring the time it takes one drones to receive 200 packets, do the same test but when two drones each receiving 200 packets.

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.

The firmware was written in C++ to use the same AQ CAN-message generation code running on the RPi, on the At90CAN128. The scheduler has been implemented with a minimum functionality.

It supports a software timer which is useful if a Light-Emitting Diode (LED) needs to be toggled every x ticks. To use the timer *wait(ticks)* has to be called from within a task with the number of ticks to wait. The task will not be executed until the wait period has passed. Each task has its own state-variable which is parsed as a parameter to the task. By calling *set_state(state)* from within a task, the state-variable can be updated. This was implemented to avoid the need of static or global variables in order for each task to have its own state-machine.

Listing 2.1 Pseudo code of the RTCS scheduler implemented

```
1  def scheduler():
2     for(ever):
3        while(ticks != 1);
4        ticks = 0
5        for task in tasks:
6           if task.task_state == ST_RUN or task.wait_counter== 0:
7              task.task_state = ST_RUN
8              current_state = task.state
9              task.task_ptr(current_state)
10          else: // task_state == ST_WAIT
11             task.wait_counter--
```

2.1 shows pseudo code of the scheduler. When tick is different from zero it loops through an array of task. If the current state of the task is ST_RUN or the wait_counter is zero, then invoke the callback which is defined when creating a task. If the task is in ST_WAIT state simply decrement the counter.

Queues was implemented as circular-buffer to handle communication between the tasks. The queues was designed to be generic in order to contains elements of different size. When a queue is created, the size of each element is specified as a parameter.

Listing 2.2 Implementation of queues. Notice the queues are generic in size since the size of the element is given as parameter. When a element is put into the queue it is done by multiplying the elements size by the index of the next element in the queue and add that to the beginning of the memory allocated for the queue

```
1 // Create queue of 10 elements of 1 byte (main.cpp)
2 Queue_Uart0_Rx = QueueCreate(10, sizeof(uint8_t));
3
4 // From QueueSend(&Queue_Uart0_Rx, &ch), put element into the allocated memory.
5 memcpy ( queue->mem + ( (queue->typesize)*(queue->wr) ), dataIn, queue->typesize );
```

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 if needed. The task diagram shown in figure 2.8 shows the tasks and how they do inter-task communication using queues.

A hardware timer was configured to set *ticks = 1* at every 1 ms. This means the scheduler is running through all tasks with a frequency of 1khz.

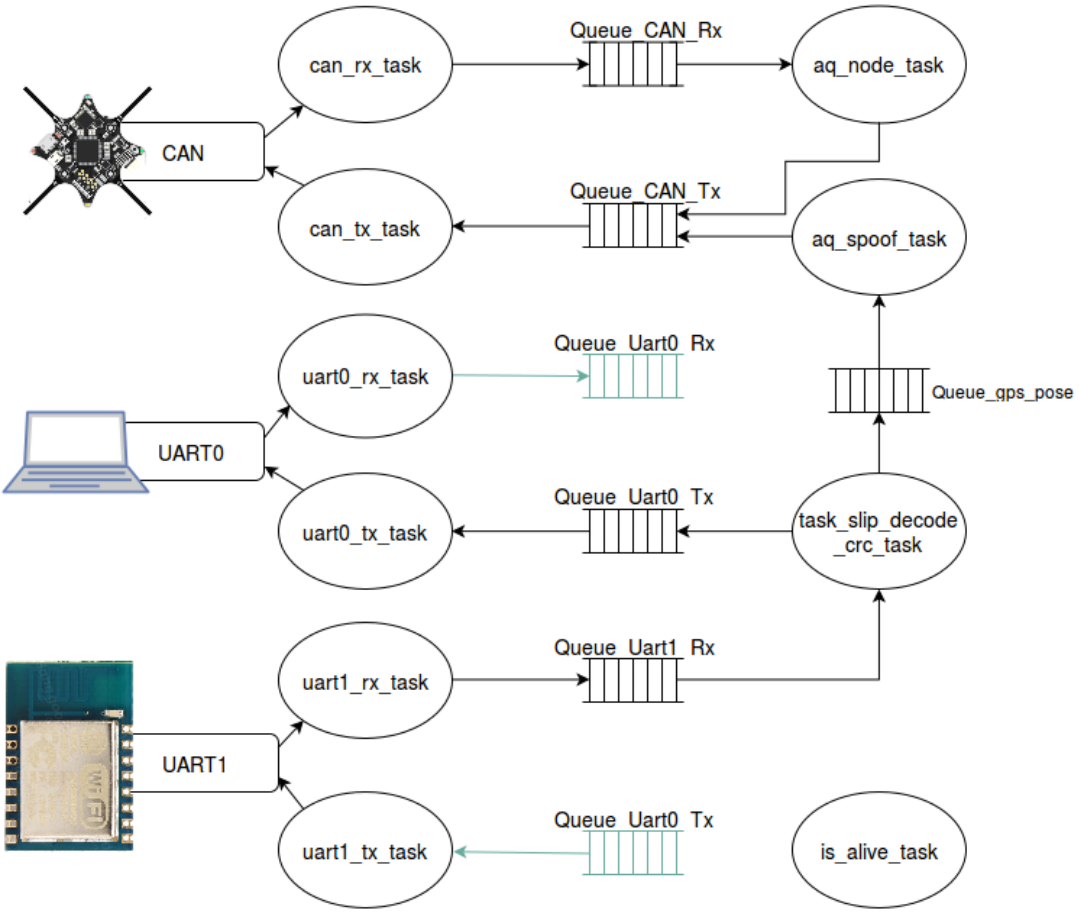


Figure 2.8 *Tasks diagram showing overview of the running tasks on the At90CAN128*

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.
- **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 running the SLIP [48] decapsulation of the received frames [49]. The At90CAN128 is not capable of handling the LLH as doubles due to a limitation of the compiler,[50] so instead the At90CAN128 represents doubles as byte arrays. When a frame has been received it calculates the CRC of the payload and verifies the CRC bits are the same. The code used for CRC on both PC and At90CAN128 was generated by PyCRC[51]. If CRC is valid, the message is put into *Queue_gps_pose* and sent to *aq_spoof _task*
- **aq_spoof_task** Responsible of converting received frames from *Queue_gps_pose* to CAN messages [52].

Test of CAN and Queues

A test of the CAN and queues was conducted by sending a known ID and data to the extension-board over CAN. The At90CAN128 then sent, using UART, to the PC what it received. Then test can be seen in appendix 4.2. As expected, the PC received the same ID and data as sent over CAN. This shows CAN and UART is configured correctly and that the queues transfer data between the tasks.

Test of scheduler

In order to test the timing of the scheduler, a led_task was written. The task can be seen in code 2.3

Listing 2.3 RTCS task used in timing test. It sends the state of the task as ASCII character to UART0 and waits 1000 ticks.

```
1 void is_alive_task(uint8_t my_state){
2    UDR0 = my_state+'0';  // Write state as ASCII to UART0
3    switch(my_state){
4    case 0:
5       INT_LED_ON_GREEN; // Other off
6        set_state( 1 );  // Set next state
7       break;
```

[48] SLIP described in section 2.3.2
[49] SLIP described in section 2.3.2
[50] The compiler used is AVR-GC++ to compile C++ to the At90CAN128, however it defaults handles a double as 4 bytes and not 8.
[51] https://pycrc.org/last visited 29 Maj
[52] Messages shown in section 2.1.1

```
8     case 1:
9        INT_LED_ON_RED;  // Other off
10        set_state( 2 );
11       break;
12    case 2:
13       INT_LED_ON_BLUE; // Other off
14        set_state( 0 );
15       break;
16    }
17    wait( 1000 );  // Wait one second (1000 ticks = 1000 * 1ms )
18 }
```

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 2.4.

Listing 2.4 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()
```

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. This was not done due to lack of time.

2.3.2 ESP8266 firmware

The ESP8266 module was initially flashed with the Arduino [53] bootloader. The modules supports Over The Air (OTA) [54] which means the module can be flashed over WIFI instead of using a Future Technology Devices International (FTDI) cable.

At first, the module had to be programmed using an FTDI cable. The ESP8266 has a programm-pin that needs to be held high or low depending on whether the module should boot from its flash or if its about to be programmed. This pin was connected to the At90CAN128 in order to set the level of the pin without the need of jumpers on the Printed circuit board (PCB). When the At90CAN128 boots, it pulls the programming-pin high or low depending on a define. It can thus easily be chosen whether the ESP8266 module should boot or be programmed. After the Arduino bootloader was flashed, there was no need to further use the FTDI cable.

[53] https://github.com/esp8266/Arduino last visited 29 Maj
[54] https://github.com/esp8266/Arduino/blob/master/doc/ota_updates/readme.md last visited 29 Maj

The ESP8266 is run in a super-loop meaning it treats UDP packages as fast as possible. The code in 2.5 shows the main part of the code running on the ESP8266 module.

Listing 2.5 Snippet from loop() shows how it processes frames. When a UDP packet is available its size is compared with the expected size of a fame. It then loops through each byte received and runs the SLIP encapsulation

```
// loop
int packetSize = UDP.parsePacket();
if(packetSize == PACKET_SIZE){
      Serial.write(SLIP_END);
    for(int i = 0; i < PACKET_SIZE; i++){
         switch(packetBuffer[i]){
            case SLIP_END:
              Serial.write(SLIP_ESC);
              Serial.write(SLIP_ESC_END);
             break;
             case SLIP_ESC:
              Serial.write(SLIP_ESC);
              Serial.write(SLIP_ESC_ESC);
             break;
             default:
              Serial.write(packetBuffer[i]);
       }
   }
   Serial.write(SLIP_END);
}
// End loop
```

The SLIP [55] protocol has been used to encapsulate frames sent by the ESP8266 and received by the At90CAN128 in order to know when a frame is beginning and ending. The naive implementation of separating frames would be to use a special character as separator. However this fails if the separator occurs as part of string. Instead SLIP[56] is used to handle escaping the separator if it occurs in the data.

[55] Inspired by SLIP implementation provided by Kjeld Jensen
[56] The slip protocol is described in details in https://tools.ietf.org/html/rfc1055 last visited 29 Maj

2.4 AutoQuad M4 firmware

AutoQuad *is the flight controller used in this project. The author was among the first students using AQ on University of Southern Denmark (SDU), but the first student to make changes to the firmware and to communicate with AQ over CAN-bus. The AQ firmware was not very documented, 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)*[57] *in order to compile their firmware, waiting for Quatos* [58] *license and getting familiar with the flashing process of the M4 board. Much of the gained knowledge about AQ and how to do debugging has been written down to share with other students.* [59]

Since AQ only supports receiving absolute positioning through a serial port[60] another way had to be found. The M4 board is born with an onboard ublox M8Q [61] which uses ublox's ubx protocol. If the M8Q GNSS was repleace everything sent by eg. a RPI has to be encoded as ubx. The CAN-bus was chosen since it is already implemented and used in AQ. AQ supports sensor inputs through CAN, unfortunately GNSS was not supported and thereby had to be implemented.

AQs CAN protocol is described, implemented and tested in appendix 4.1. When the AQ M4 board boots, it tries to register the CAN nodes on the bus. The registering of CAN nodes has been described in appendix 4.1

When all CAN-nodes have successfully registered, different node-initializing functions is invoked. First a summary run which sends the number and types of the nodes to QGround-Station. [62]. It can thus be validated by the pilot that all the ESC's and CAN-sensors are detected. After the summary the CAN-sensor initialization is invoked which creates a callback that gets attached to the CAN-type which in this case is CAN-sensor[63]. When AQ receives a sensor reading the callback will then be invoked nd the sensor value will be added to the array.

The callback default fills in the received sensor value in an array so the sensor value can be used in another task. The canID [64] is used as index so the task that needs the sensor value knows on which element in the array to look. Code 2.6 shows the default callback [65]

Listing 2.6 Callback invoked when a CAN-sensorvalue is received. It stores the value in an array indexed by canId. Note that the sensor value is casted to a float

```
1 void canSensorsReceiveTelem(uint8_t canId, uint8_t doc, void *data) {
2     canSensorsData.values[canId] = *(float *)data;
3    /* Reception time of the message stored as well */
4 }
```

[57] http://www.rowley.co.uk/arm/ last visited 29 Maj
[58] The controller, Quatos, used in AQ is not Open Source so a license had to be acquired by SDU
[59] It can be found in *Appendix/SDU-UASAutoQuaddocumentation.pdf* and *SDU-UASAutoQuadsourcecodedocumentation.pdf* Text marked with green is written by the author
[60] Seen by inspection of the schematic to the M4-board done by the supervisor
[61] http://autoquad.org/wiki/wiki/m4-microcontroller/m4-gps-antenna-options/ last visited 29 Maj
[62] Graphical userinterface which provides general information about the drone such as battery level, attitude but also waypoint functionality
[63] CAN-types are described in section 4.1
[64] The canID is described in appendix 4.1
[65] https://github.com/bn999/autoquad/blob/master/onboard/canSensors.c last visited 29 Maj

The callback is fairly simple and does not save the doc[66] which is needed in order to tell which message the CAN-GNSS sensor is sending. The messages received over CAN on the AQ M4 board is shown in section 2.1.1

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

Code 2.7 shows a pseudo code of how the GNSS-messages is handled.

Listing 2.7 Modified callback invoked when a sensor-value is received. Shows how DOC is used to tell which GNSS-message is received and when height is received the flags are set to update the Unscented Kalman filter (UKF)

```
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.pDOP = data[0]/10
      ..
   if doc == CAN_DOC_ACC then
      canData.satellites = data[0]
      canData.fix = data[1]
      canData.xAcc = data[x]/10
      ..
      canData.heading = data[n-1]/100

   if doc == CAN_DOC_VEL then
      gpsData.velx = data[x]/100
      ..
   if doc == CAN_DOC_ALT then
      gpsDatat.height = data
      if gpsData.satellites >= 5 then
         gpsUpdatetime = now()
         setFlag (gpsData.gpsPosFlag)
         setFlag (gpsData.gpsVelFlag)
      else:
         debug_to_QGroundStation
}
```

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

[66] DOC is described in appendix 4.1

When the flags are set, the task updating the UKF will run [67]. 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);
```

Testing of the GNSS spoofing was done in cooperation with Kjeld Jensen since results was needed for the paper. Before mounting an expensive RTK-GNSS on the drone, it was tested using a NEO-6P GPS. The tests can be seen in appendix 4.4 When testing was successful using the NEO-6P, the RTK-GNSS was mounted on the EduQuad. Figure 2.9 shows the waypoints generated to test the performance of the RTK-GNSS and the tracked path.

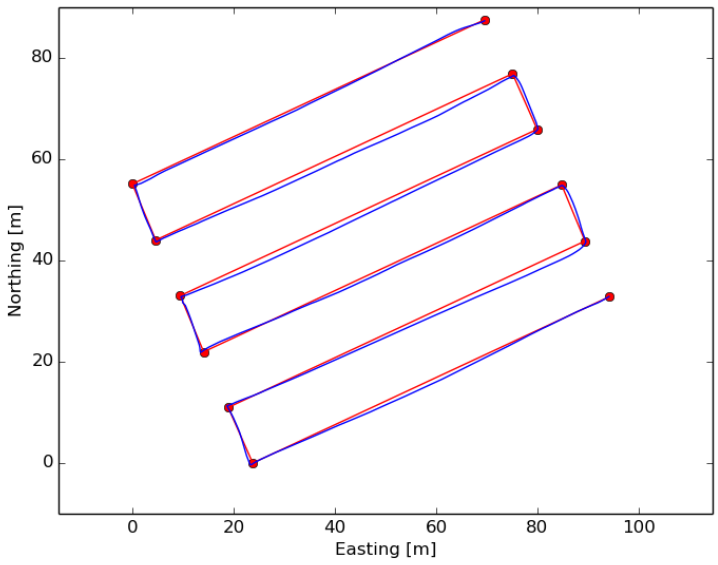


Figure 2.9 *Map of the planned route (red) with waypoints (red dots) and tracked flight (blue)*

In figure 2.9 the dots represents waypoints, straight lines represents legs and the line deviating from the legs are the track of the drone.

Figure 2.9 shows the result of the test flight.

The horizontal lateral distance is calculated as the distance to the closest point of any of the route legs. This result in some inaccuracy at the end of each leg, and the hovering at the end of each leg skews the result slightly as well. The measured average horizontal lateral distance from the route was 0.45 m. The 95'th percentile was 1.11 m and the maximum distance was 1.35 m.

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

Figure 2.10 shows a histogram of the horizontal lateral distances from the planned route. The average vertical distance from the route was 0.26 m, the 95'th percentile was 0.72 m and the maximum distance was 1.52 m. [68]

The reason for the relatively low accuracy of flying with the RTK-GNSS is out of the scope of this project.

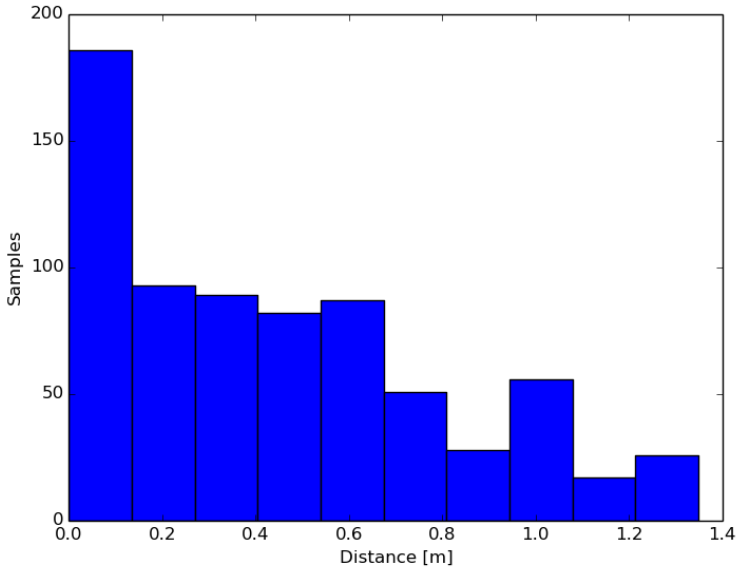


Figure 2.10 *Histogram of the horizontal lateral distance from the drone to the planned route during the experiment. The total number of GPS track points (samples) is 715.*

It can be concluded that it is possible to inject absolute GNSS positions into the AQ M4 board over CAN. The tests made in appendix 4.4 shows it is possible to decide how much the UKF should trust the GNSS positions based on the DOP values.

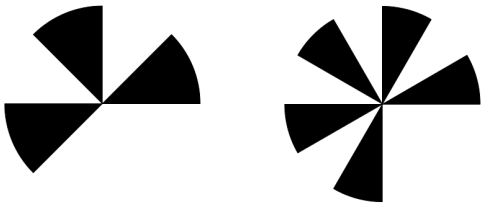
2.5 Indoor localization

In order to do indoor flying there is a need of some sort of localization since GNSS 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 Marker-Locator 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.

[68] 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

Based on previous experience, the MarkerLocator written by Henrik Midtiby was chosen as indoor localization. Its using OpenCV[69] and implemented in python. It works by detecting markers as shown in figure 2.11



(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 2.11 *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 2.11 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 [70], 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[71].

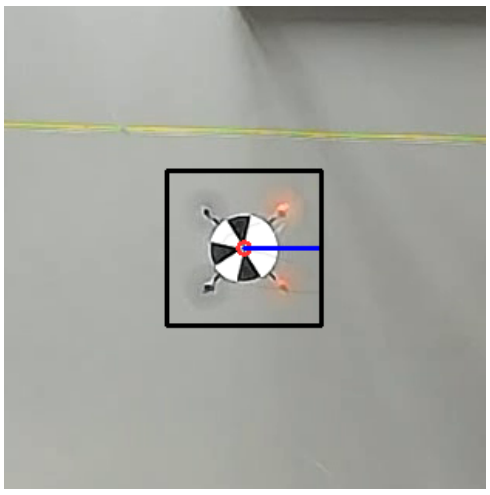


Figure 2.12 *The MarkerLocator searches only for the marker around the last known position of the marker in a frame of 100*100px.*

If the marker moves out of the window, the MarkerLocator is still looking in only that

[69] http://opencv.org/ last visited 29 Maj

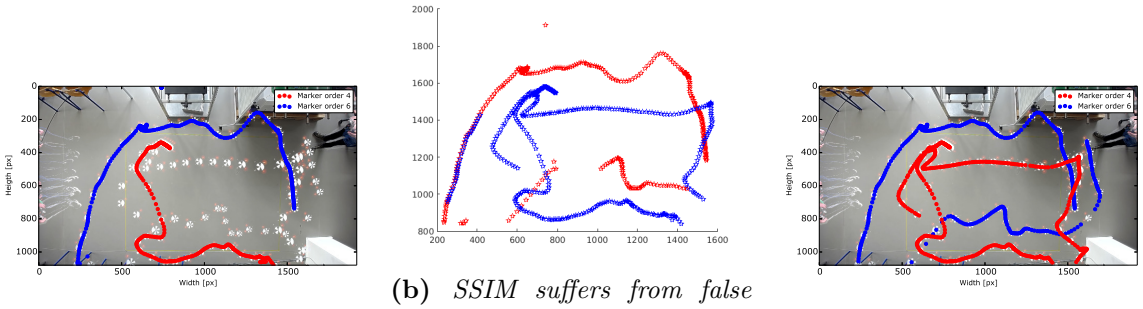
[70] Processing one fullHD frame takes approximately 0.38 sec. Measured with build in timer in the MarkerLocator

[71] Approximately 0.0041 sec

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 right 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 [72] 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 2.13 shows the different solutions tried.



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

(b) *SSIM suffers from false positives and a few false negative. Notice the red and blue color has swapped*

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

Figure 2.13 *Different approaches to get a quality measure explaining how well the right marker is found.*

Figure 2.13a 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 2.13b 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[73] 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 2.13c 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 are. 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

[72] Said by Henrik Midtiby
[73] http://scikit-image.org/

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 2.1.1

2.5.1 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 [74] and find these markers in the frame, it is possible for the perspective corrector to make the transformation. Figure 2.14 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 2.6

Figure 2.14 *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 test the homography. 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. The real distance measured by a ruler was 60 cm and and the MarkerLocator gave 59.19. Section 2.6 tests the localization more in depth as part of the coordinate conversion.

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 the inaccuracy of the MarkerLocator is caused by error when measuring the distance between the corners of the whiteboard and finding the corners in the frame. If higher accuracy is needed this should be done again more carefully.

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

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

2.6 Control and coordinate conversion

This section concerns controlling the drones and converting between coordinates. To send Latitude, Longitude (LL) to the AQ M4 when obtaining the position from the MarkerLocator, there has to be made a transformation and coordinate conversion in order for AQ M4 to know where it is in the world. This section will first describe the coordinate conversion and then how to control the drones

2.6.1 Coordinate conversion

The *Decision_Maker* shown in figure 2.1.1 is at the moment of writing responsible for converting between centimeters from the MarkerLocators to LL in decimal degrees which is sent to the AQ M4. Figure 2.15 shows how the coordinates are converted.

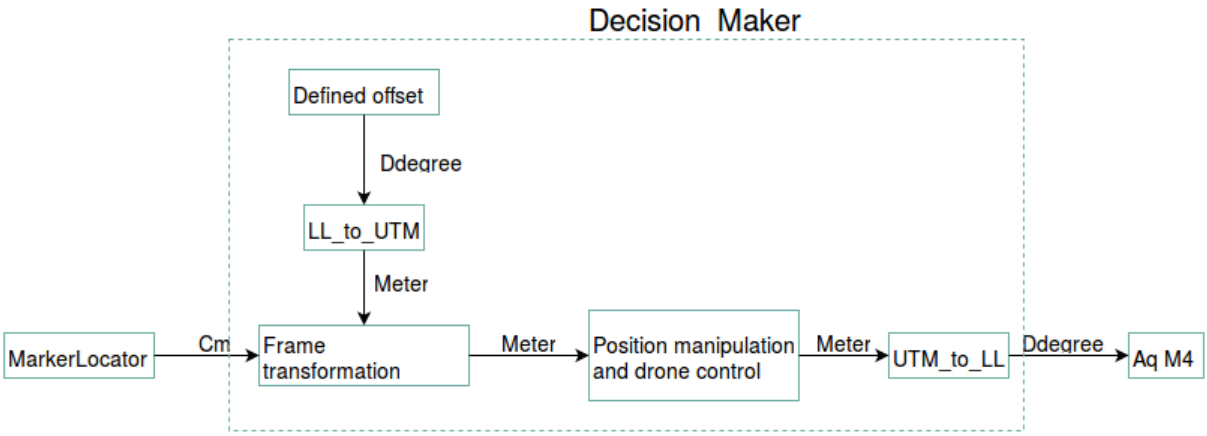


Figure 2.15 *Shows how the coordinates are converted from centimeters, added to a lat/lon offset, converted to meters, manipulated and converted to lat/lon*

The dotted square in figure 2.15 shows the coordinate conversation flow within the *Decision_Maker*-node. The MarkerLocator provides a position in the unit of centimeters[75] which gets added to a defined offset. If no offset is defined , the drones would be flying around lat = 0, lon = 0, which is in the middle of the ocean. The location of SDU's RoboLab was chosen since plotting LL will then be shown in QGroundcontrol as if flying is happening in RoboLab, however this is not required. Another advantage of using RoboLab as offset is, that Universal Transverse Mercator (UTM) zone(32) is already specified and tested working in the UTM-converter[76] used.

When the offset is converted into meters it can be added to the MarkerLocators output which also has to be converted into meters. Since UTM works in northing and easting the x,y axes of the MarkerLocator have to be aligned with north and east axes. Whether y is north or easting depends on which axes convention is used. The East, North, Up (ENU) axes convention

[75] This should be changed to meters, it was not changed due to lack of time

[76] Frobomind's UTM converted was used, https://github.com/FroboLab/frobomind/tree/master/fmLib/math/geographics/transverse_mercator/src last visited 29 Maj

has been used, however North, East, Down (NED) could have been used as well. How the camera-frames quadrant is laying in the frame is a matter of how the homography was made insection 2.5.1.

The camera mounted below the ceiling used to track the drones was align with north by coincidence so that x,y could simply be added to the northing and easting respectively of the UTM converted.

Figure 2.16 shows the coordinate-frame of the camera with respect to north and East in RoboLab.

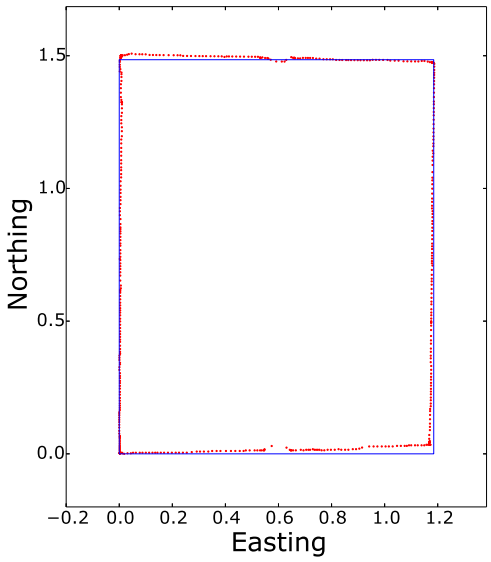
Figure 2.16 *Picture generated from Google Maps. Rob The frame in which the MarkerLocator provides output in is shown with blue arrows. The x-axis is aligned with east and y is aligned with north. The offset used is the origo of the famera-frames coordinate system*

If alignment of the camera-frame with north was not possible, the coordinate frame can be rotated using the rotation matrix[10].

In figure 2.15 the manipulation and control happens in meters but converted back to LL in decimal degrees and sent to AQ M4 board.

2.6.2 Control and manipulation of drones position

The initial idea was to send waypoints to the AQ M4-boards, however this requires an extension-board used for telemetry mounted on the drones. Since only one extension-board can be mounted on the AQ M4 board, no telemetry was available and is thereby not available to send waypoints when it is airborne.
Instead it was chosen to control the drone by manipulating its belief in where it is. If the drone is supposed to fly to the right, its being manipulated to belief it is more to the left than it is in order to make it fly to the right. Eg. the drone is in (x = 0,y = 0) and it has to go to (x = 1, y = 0). Instead of giving it a waypoint telling it to go to 1,0, it is told that its position is (-1,0) and thereby gets tricked into flying to (1,0). In order for this to work, the AutoQuad M4 should be in *Position-Hold*-mode which is set by a switch on the transmitter.



(a) *Blue square is the whiteboard seen in figure 2.17b. The red dots is output in LL converted to UTM to be plotted*

(b) *A 4 order marker held along the edge of the whiteboard all the way around. The size of the whiteboard is 118.5cm*148.5 cm*

Figure 2.17a shows a blue square representing the whiteboard shown in figure 2.17b. The anomalies shown in the top and bottom of the square is caused by the mountings holding the whiteboard. The software setup used to track the marker is shown in figure 2.5. In this test, the *Decision_Maker* was programmed to the output position is the input position. By doing this, it makes it possible to simply plot the position of the drone. Instead of transmitting the LL frames to the drone they were instead saved and converted back to UTM in order to be drawn together with the blue square. The lines were analyzed independently in order to see how well the fit a line. Since the whiteboards sides is straight, it was expected to see the marker moved in straight lines as well.

Table 2.9 *Results from positions obtaining by moving marker along the edge of whiteboard*

Line	Mean [M]	Std. Deviation [M]
Left side	0.004	0.13
Bottom side	0.014	0.01
Right side	0.009	0.14
Top side	0.017	0.25

A more generic design of the *Decision_Maker* would have been to split the current *Decision_Maker* into three nodes in order to avoid doing coordinate conversation within each *Decision_Maker* node. A node before the *Decision_Maker* to handle camera-frame transformation and a node after to convert the coordinates to LL.

Chapter 3

Discussion

Chapter 4

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] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, Cambridge, MA, June 2005.

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

Appendices

4.1 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 4.1. 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 4.1 *Table shows the identifier bits used in AutoQuad CAN messages*

In figure 4.2 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 4.2 *Table shows abbreviations used in table 4.1*

[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 4.1 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 4.1 *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 4.1

Listing 4.1 LLC defines

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

Table 4.3 *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 4.2 Target type defined in AQ

```
1 #define CAN_TT_GROUP      ((uint32_t)0x0<<29)
2 #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 4.3

Listing 4.3 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 4.4 describes the FIDs shown in listing 4.3.

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 4.3 that the FID's is moved 25 positions to the left but FID is shown in table 4.1 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 4.4 *Descriptions of the FIDs mentioned in code 4.3*

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 4.4 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 4.4 shows a function using DOC when message FID is CAN_FID_CMD.

Listing 4.4 Snippet of AQ's can.c:365

```
void canProcessCmd(... ,uint8_t doc, ...) {
    switch (doc) {
        case CAN_CMD_STREAM:
         ...
```

[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

```
5             break;
6         case CAN_CMD_TELEM_VALUE:
7          ...
8             break;
9         case CAN_CMD_TELEM_RATE:
10         ...
11            break;
12    }
13 }
```

In section 4.1 DOC is used to tell which part of the GNSS data is getting sent but where FID is CAN_FID_TELEM.

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.

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 4.5

Table 4.5 *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 4.5 shows how it is implemented in AQ.

Listing 4.5 Snippet showing UUID generated in ESC32v2

```
1 can.h:20
2 #define CAN_UUID 0x1FFFF7E8
```

[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

```
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 out a message to that specific node. The sequence diagram in figure 4.2 shows the described sequence.

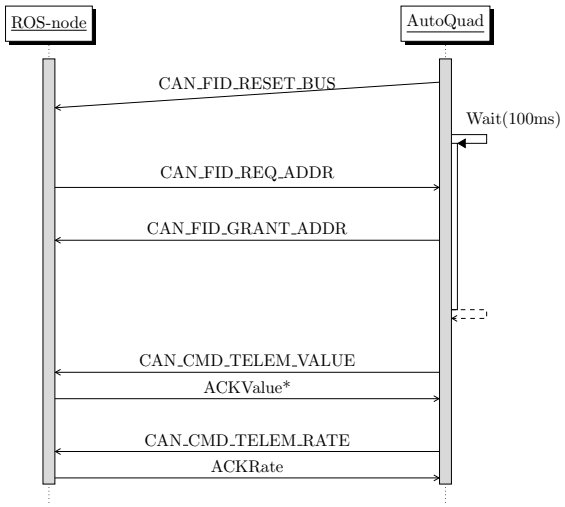


Figure 4.2 *Registering a new node in AQ*

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 4.2 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

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

information is not used but might be later on.

Code in 4.6 shows a snippet[7] from AQ where AQ waits for nodes to request an address.

Listing 4.6 Snippet showing AQ waits for devices

```
canResetBus();

// wait 100ms for nodes to report in
micros = timerMicros() + 100000;
while (timerMicros() < micros)
    // extend wait period if more nodes found
    if (canCheckMessage(0))
        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.

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 4.7 shows a snippet of main.py.

Listing 4.7 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()

```

[7] https://github.com/bn999/autoquad/blob/master/onboard/can.c:547
[8] http://python-can.readthedocs.io/en/latest/socketcan_native.html

```
25 # Parse message
26 msg = CanMessage(msg)
27
28 # ACK TelemRate
29 autoquadNode.AnswerRequestTelemRate(msg)
```

When a node has successfully registered, it is shown in QgroundStation as shown in figure 4.3. It can be seen 4 motors were connected to the CAN bus, but also that the spoofed sensor was registered.

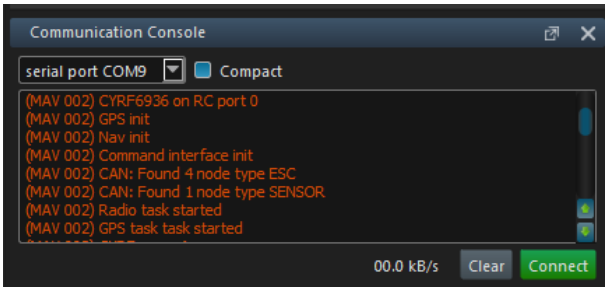


Figure 4.3 *Successful SENSOR registration*

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 4.7 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 4.8 shows the implementation where a sinus is generated to simulate a current and how it is transmitted to AQ.

Listing 4.8 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 4.8 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 4.4

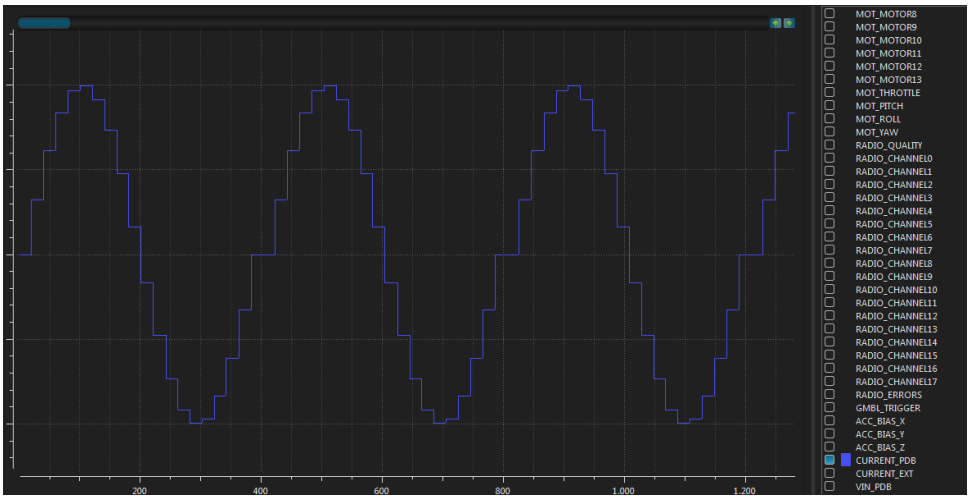


Figure 4.4 *Test of python CAN test*

Figure 4.4 shows the sinus generated from the node as expected.

4.2 Scheduler Queue and CAN test

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 4.9.

Listing 4.9 Snippet of task used to test CAN

```
1  CAN_frame frame;
2  if(QueueReceive(&Queue_CAN_Rx, &frame)) {
3     char ch;
4     /* ID out uart0 */
5     for(int i = 3; i >=0; i--){
6        ch = ((frame.id >> i*8) & 0x000000FF);
7        QueueSend(&Queue_Uart0_Tx,&ch);
8     }
9     /* ID end*/
10
11    /* MSG out uart0 */
12    for(int i = (frame.dlc-1); i >=0; i--){
13       ch = ((frame.msg >> i*8) & 0x00000000000000FF);
14       QueueSend(&Queue_Uart0_Tx,&ch);
15    }
16    /* MSG end */
17
```

```
18      /* End of line */
19      ch = '\r';
20      QueueSend(&Queue_Uart0_Tx,&ch);
21      ch = '\n';
22      QueueSend(&Queue_Uart0_Tx,&ch);
23  }
```

The command used to send CAN-messages can be seen in code 4.10.

Listing 4.10 Task used to test CAN

```
1  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 4.11

Listing 4.11 Command used to get UART messages

```
1  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 **??**.

```
00020de: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:37 CEST 2016
00020ec: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:38 CEST 2016
00020fa: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:39 CEST 2016
0002108: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:40 CEST 2016
0002116: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:41 CEST 2016
0002124: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:42 CEST 2016
0002132: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:43 CEST 2016
0002140: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:44 CEST 2016
000214e: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:45 CEST 2016
000215c: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:46 CEST 2016
000216a: 1dea dbef fedc ba98 7654 3210 0d0a  ........vT2...  Wed May 11 11:09:47 CEST 2016
```

Figure 4.5 *Left shows output from command 4.11 and right shows command 4.10*

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

4.3 Datatype for latitude and longitude

In order to know the required bytes to send latitude and longitude with a precision of 1 cm, the author used the datum WGS84[9] to estimate the worst-case error at equator.

[9]WGS-84 used in AQ

WGS-84 [10] 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 4.3

$$b = a \cdot (1 - \frac{1}{298.257223563}) \tag{4.1}$$

Given the formula of the circumference of an ellipsoid:

$$circumference = 2 \cdot \pi \sqrt{\frac{1}{2}(a^2 + b^2)} \tag{4.2}$$

Equation 4.3 gives a circumference at equator of 39873752.0 meters.

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

$$\frac{39873752.0}{360} = 110760 \frac{m}{deg} \tag{4.3}$$

Table 4.6 was made to see how many decimals is required to obtain different precisions.

Table 4.6 *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[11].

Listing 4.12 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 4.12 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.

4.4 Test of spoofed GNSS

Testing of the spoofed GNSS was split intro three subtests.

- Test first test was done indoor where the drone were laying on a table. See test in 4.4.1.

[10] http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf last visited 22 maj
[11] http://se.mathworks.com/help/matlab/ref/eps.html

- The second test was conducted as the first test, but when the drone. was airborn See test in 4.4.2
- The third test was done by walking with the drone and comparing the onboard GNSS on the AQ M4 board and an external GPS. See test in 4.4.3

4.4.1 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.

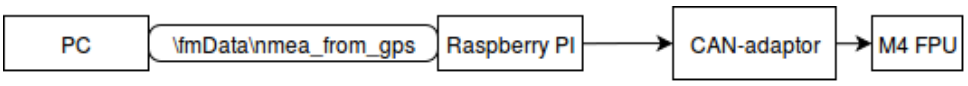


Figure 4.6 *Block schematic of the connected components*

Figure4.6 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.

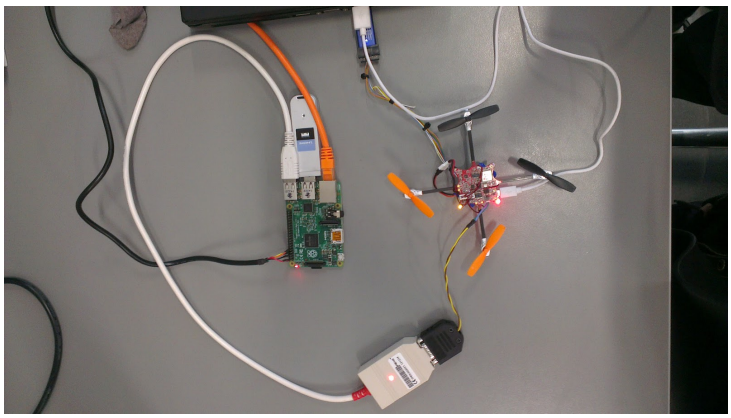


Figure 4.7 *Test-setup shown*

Figure **??** shows 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 Teleprinter (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 4.8

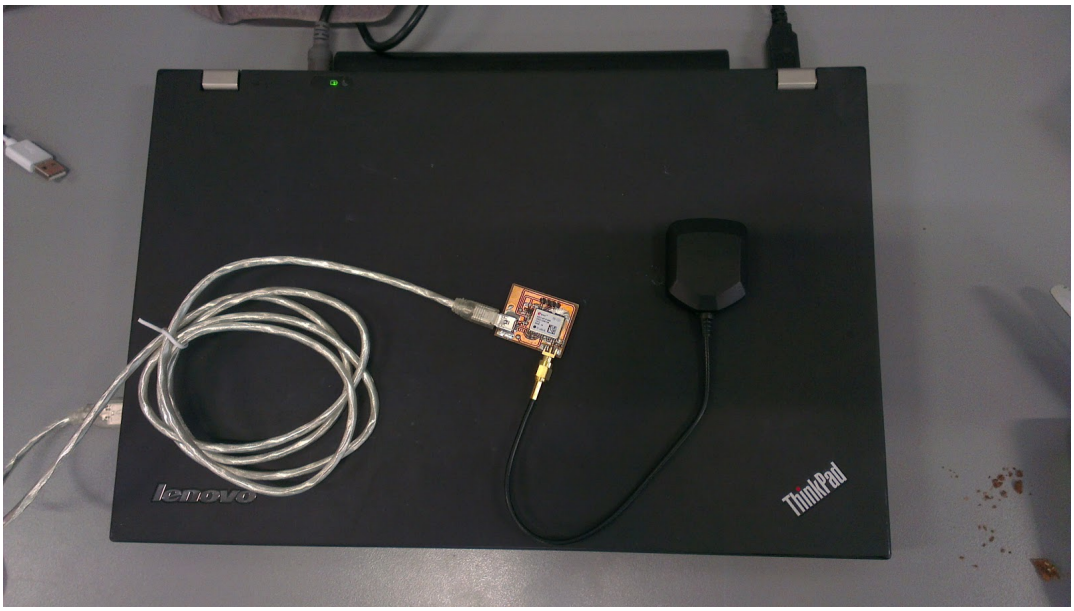


Figure 4.8 *The Neo6-P GPS (PCB developed by SDU), u-blox active GPS antenna* [12] *and authers laptop used in this test*

The coordinates recorded by the rosbag is shown in figure 4.9. FreeNMEA[13] was used to plot the GPGGA messages obtained from the GPS.

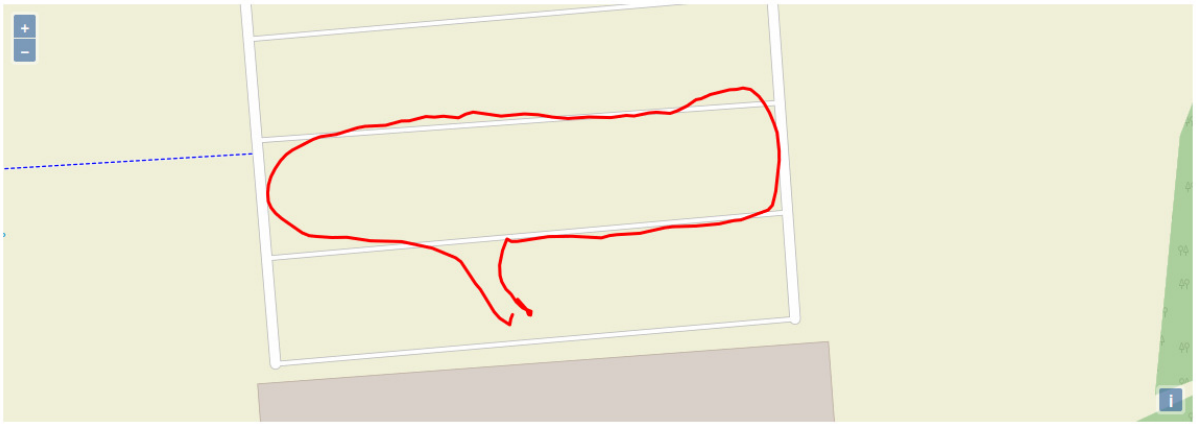


Figure 4.9 *Test coordinates plotted*

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 4.10

[13] http://freenmea.net/

Figure 4.10 *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.

4.4.2 Initial test flight with spoofed GNSS

The purpose of this test was to see if the GNSS spoofing worked when the drone was airborn. The same hardware and software were used as in section 4.4.1, though with a few changes. The GNSS used in test 1 to collect coordinates were mounted on the EduQuad. The idea was to pass-through the GNSS 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.

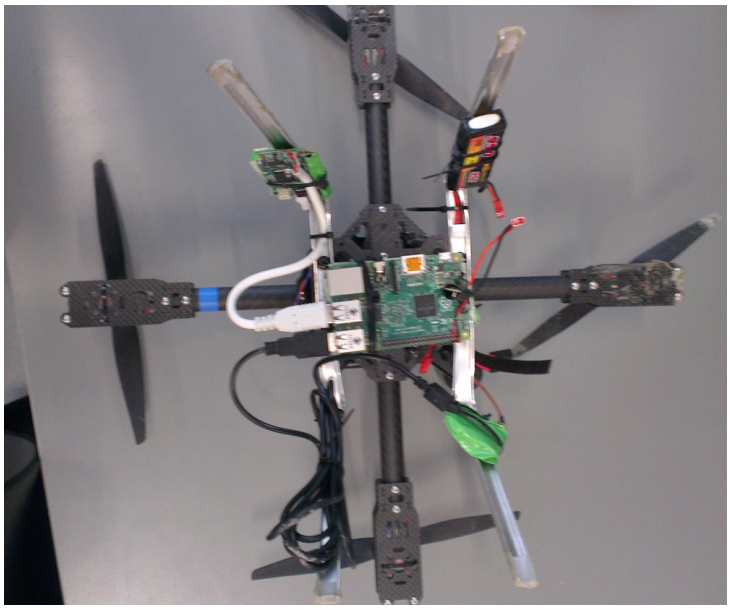


Figure 4.11 *The EduQuad drone with the hardware used in the test. The figure shows the Raspberry PI, PEAK-CAN adaptor, green Neo-6p GPS and extra battery for the Raspberry Pi*

Figure 4.11 shows how the hardware were mounted on the EduQuad. It was temporary mounted with strips and tape to avoid short circuiting and getting the wires into the robots. 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.

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[14] uses the altitude from the GNSS to estimate the position of the drone. Furthermore the checks shown in code 4.13 where inserted to make sure only valid GNSS coordinates were fed into AQ.

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

Listing 4.13 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);
}
```

When the ROS-node was 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 [15]

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 GNSS antenna was not mounted in the middle of the frame as the onboard antenna.

Figure 4.13 shows the DOP value is lowering witch indicates that AQ was using its onboard GNSS.

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

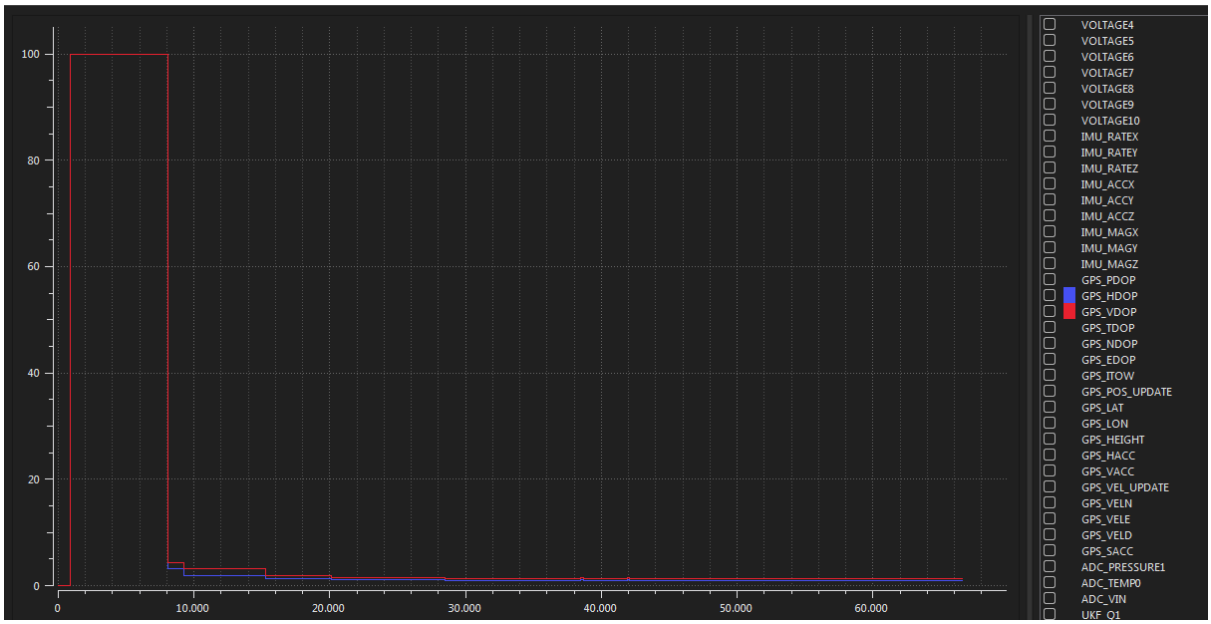


Figure 4.12 *Plot of h/vDOP from logs when the drone was airborn*

4.4.3 Comparison of onboard and spoofed GNSS

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 anteanna 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-mesasages. The test was done by walking 20 meters with the EduQuad two times to compare the output of the UKF with the different GNSS's. The values used to spoofed the GNSS was found by reading about the ratio between the different DOP-values. Values available from the Neo6-p GPS, was used directly to spoof the onboard GPS and values not available was either calculated or hardcoded. 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

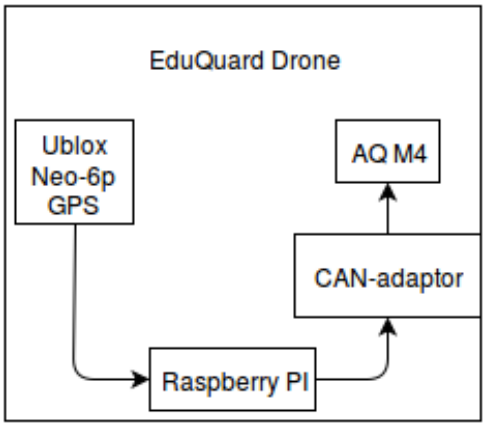


Figure 4.13 *Nep-6p connected to a AQ M4 through a RPI using a PEAK-CAN adapter*

In order to quickly switch between the onboard GNSS and the Neo-6P GPS during the test,

a switch on the Spectrum DX9[16] 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 4.14 is filled in.

Listing 4.14 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;
```

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 Real-Time Clock (RTC) was set from the onboard GPS.

To tell the UKF in AQ how much to believe in the GNSS coordinates, the onboard GPS feeds the UKF with DOP factors. The DOP factors expresses how well the satellites are geometrically 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. [11]

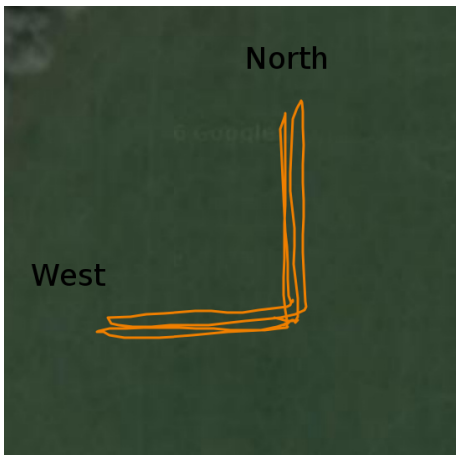
The values listed in table 4.14a was used and multiplied by the hDOP received from the Neo-6P GPS. The pDOP was set to 1.75 times higher than the hDOP since it incorporates error from both vDOP and hDOP [11].
vDOP is usually 1.5-2 tims higher than hDOP. [11].
tDOP was sat to 1.5 higher than hDOP since timeerror causes position error. [17]

[16] http://www.spektrumrc.com/Products/Default.aspx?ProdId=SPMR9900 last visited 21. Maj
[17] http://www.environmental-studies.de/GPS/Dilution-of-precision/dilution-of-precision.html last visited 22

Since hDOP can be split into nDOP and eDOP, they each have 0.7 of hDOP. [11]

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

(a) *Scaling factors used when using Neo-6p GPS. The DOP values grayed out is values not used by AQ directly but gets logged.*



(b) *Path walked. If looking closely it can be seen the path was walked twice, once with onboard GPS then using Neo-6p.*

Figure 4.14

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 4.14b shows the walked path.

Figure 4.15 shows a plot of the position with x-axix as time.

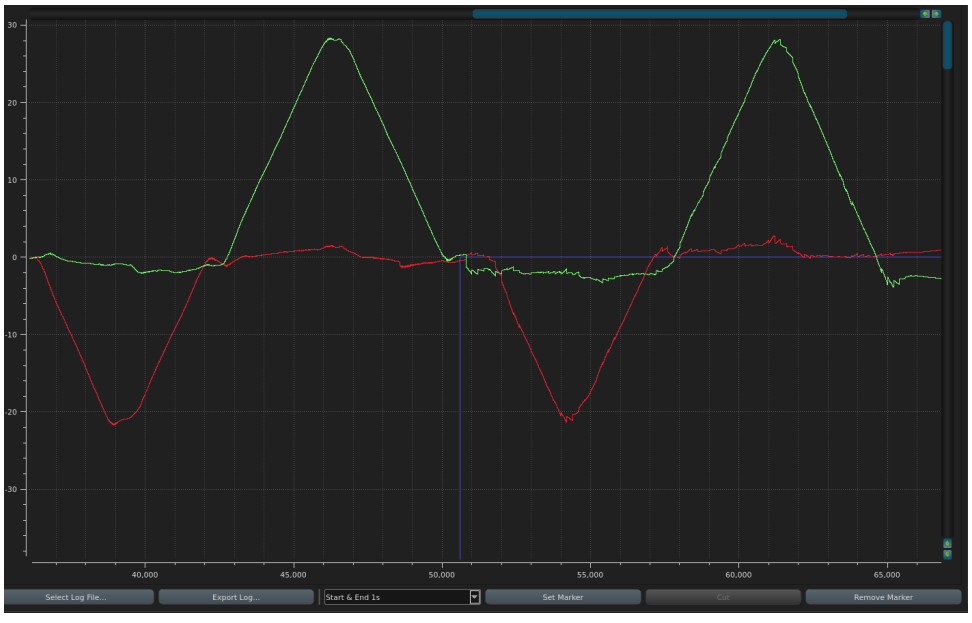


Figure 4.15 *Path walked with x-axis as time and y-axis as nort/west. The blue vertical line in the middle of the image shows the switch changing value. Before the blue line is the onboard GNSS used, after the blue line is the NEo-6P used. If looking closely it can be seen the path was walked twice. The read and green line are output from the UKF*

maj

[18] Measured by counting footsteps

Figure 4.15 matches with the walked path seen in figure 4.14b. The blue line crossing in the middle is the switch on the Spectrum DX9 transmitter getting activated to switch between the GNSS'. It can be seen how the path repeats itself after the switch has been activated. Figure 4.17 shows the same as figure 4.15 however vDOP and hDOP has been added to the plot.

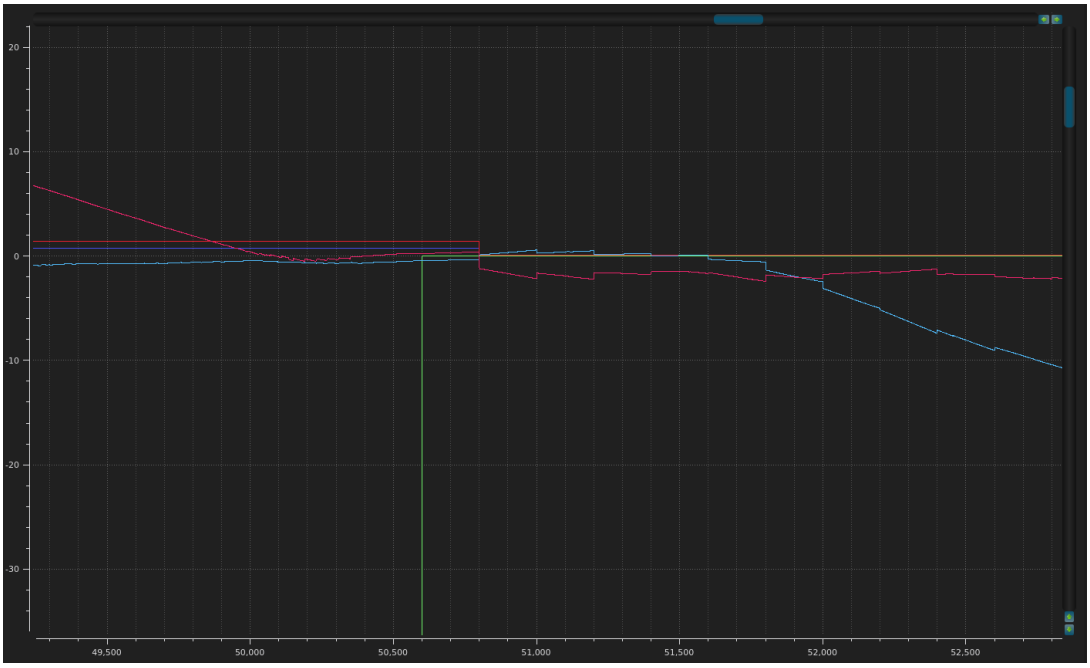


Figure 4.16 *Figure shows a zoom-in on the GNSS switch. Read is pos-easting, light blue is pos-northing, yellow is switch on transmitter and the two resulting colors are hDOP and vDOP. Note how the position gets less smooth after the GNSS switch. Furthermore, notice how the DOP values decreases when the first GNSS position is received after the switch.*

Figure 4.17 shows the same plot but where the vDOP and hDOP has been added. It can be seen if looking carefully, how hDOP and vDOP change from 1 and 1.5 respectively to 0. This was due to a divide error in the code. This shows that the belief of the GNSS can be changed by adjusting the DOP values.

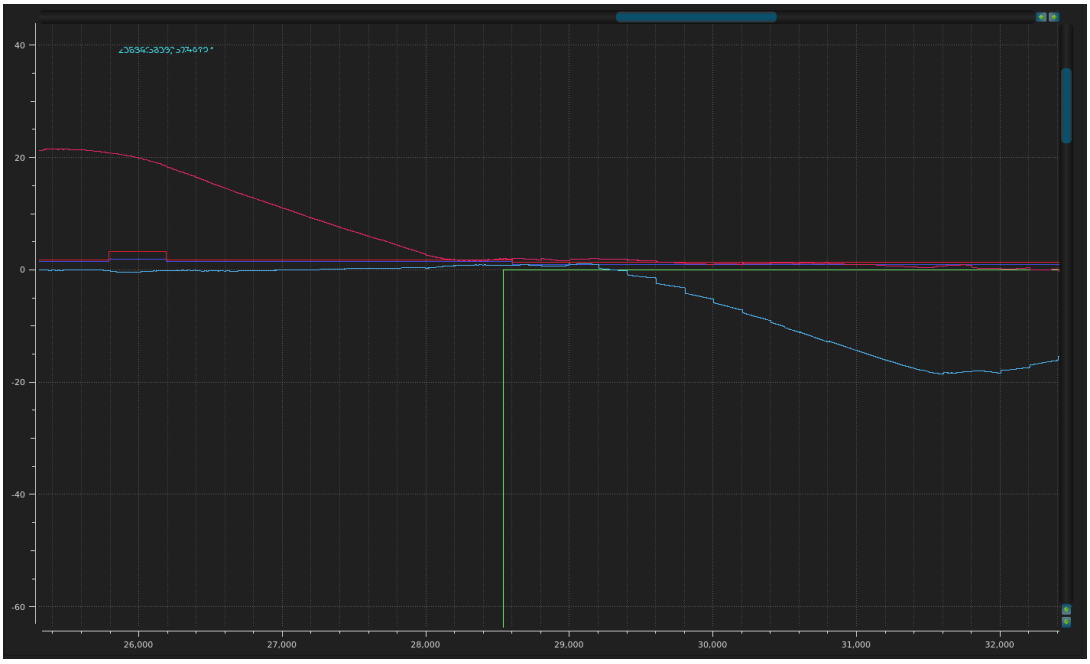


Figure 4.17 *Figure shows a zoom-in on the GNSS switch. Read is pos-easting, light blue is pos-northing, yellow is switch on transmitter and the two resulting colors are hDOP and vDOP. Notice how the position is more smooth than in figure 4.17. By inspection in QGroundControl it can be seen the vDOP and hDOP drops 0.5 from 1.8 and 1.4 respectively which is probably caused by the antenna on the Neo-6P is better*

It can be seen in figure 4.17 that the UKF performs better when the DOPs are provided correctly.

The velocities was also compared as shown in figure 4.18.

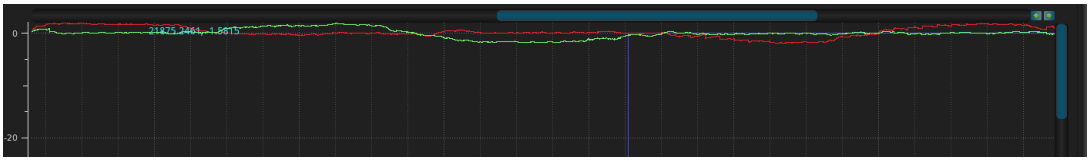


Figure 4.18 *Read line is velocity north, green is velocity east and the vertical blue line shows the GNSS switch. When the switch is zero, the spoofed GPS is used. It can be seen the velocity looks simular, however there exists more ripple on the spoofed GPS*

Figure 4.18 shows the velocities with onboard GNSS and Neo-6P GPS. It can be seen there exists some ripple. However it was later discovered that the onboard GNSS provides positions and velocity at 5 hz but the graphs shown aboive is only when the NEo-6p was sending at 1 hz. Unfortunately no log is available with the increased frequency. [19]

It can be concluded that spoofing the onboard GNSS works over CAN works. The values are compared and it can be seen the two GNSS performs almost equally well.

[19] If more time was avaiakble, the 20 meter west, 20 meter north test should be conducted again with 5 hz instead of 1 hz.