Mälardalen university

Project in advanced embedded systems and robotics

UAV

Authors:
Joakim Kareliusson
Jacob Danielsson
Tobias Andersson
Emil Johansson
Ludvig Langborg

January 17, 2016

Abstract

This report will describe the work performed during the fall of 2015 as part of a last year project in intelligent embedded systems and robotics. The goal was to develop a scaled down version of a tilt wing Unmanned aerial vehicle (UAV) with vertical take off and landing capabilities. The platform is intended to be a transportation aircraft for autonomous submarines that are also being developed at the University. The report will cover the design of all individual parts and systems created for this project. The resulting prototype is an aircraft built in carbon fiber with a wingspan of approximately two meters, a length of approximately one meter and a weight of 7.5 kg. It is controlled over wi-fi using keyboard inputs and is capable of flight in vertical mode.

Contents

1 Introduction

This work will describe the development and functionality of an unmanned aerial vehicle (UAV) which was carried out as a last year project of the master programs in robotics and intelligent embedded systems at Mälardalen University. This is the first iteration of the project which is expected to run for many years into the future. The overall project is a collaboration between the aeronautical program, the energy program, the robotics program and the intelligent embedded systems program. Some brief information of the overall goals of the project will be provided herein, but the report will be focused on the work performed during the fall of 2015.

1.1 Background

Mälardalen University has for a number of years conducted research in underwater robotics which has resulted in two autonomous underwater vehicles (AUV:s). The purpose of the AUV:s is to investigate different aspects of the Baltic sea, such as pollution and the state of windmilling farms. The Naiad is the latest and still ongoing of these projects. To avoid transporting the Naiad in car or a similar vehicle to its operating area, the university wishes to explore the possibilities of creating an autonomous transportation aircraft for the AUV.

1.2 Project organization

The organization of resources for this project course differs somewhat from traditional project courses. Rather than having a project group for each project the course was organized as follows. Each project has a project manager. The remaining students belong to competence pools: electronics, software, communication and mechanics. Each of these pools has a pool manager. Only the project manager works on a specific project, the remaining students may work on one or more projects. The electronics group for instance are responsible for electronics on all of the projects. Furthermore the projects have different priority levels, out of the four projects that ran during this time the UAV had the lowest priority, meaning that if the other projects required resources, the UAV would be left with less resources.

1.3 Goals and requirements

To avoid the need for runways, the aircraft should be capable of vertical take off and landing(VTOL). Since flying in vertical mode, such as a helicopter or multicopter, requires much more energy for propulsion than a conventional aircraft, the UAV should be able to transition from vertical flight to horizontal flight. There are different methods for achieving this, it was decided that a tilt wing design should be used in this project, meaning that the wings and hence propellers will be in a vertical position when starting, and then tilt to a horizontal position as a conventional aircraft during flight. This is illustrated in figure 1.

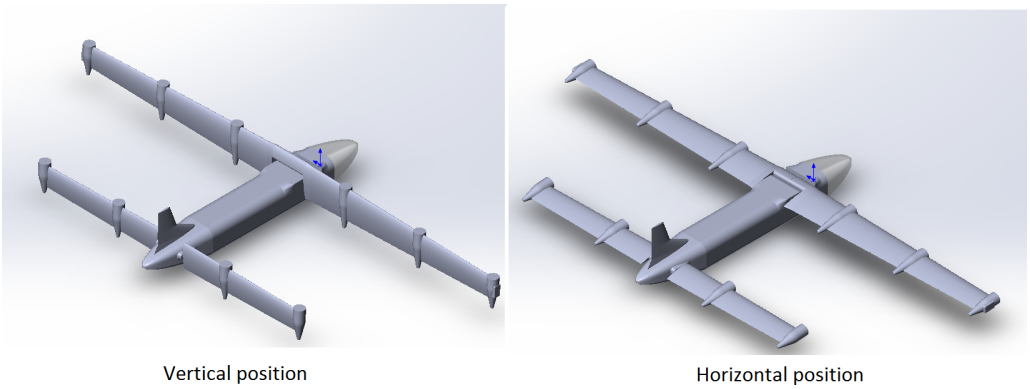

Figure 1: Vertical and horizontal wing position

1.3.1 Overall goals and requirements

The overall project goals, that will reach into the future can be summarized as follows:

- The UAV should be constructed to comply with Swedish transportation agency (transportstyrelsen) class 3.[1]
- It should be able to fly autonomously.
- It should be of a tilt wing type.
- It should be powered by electricity.
- It should have a renewable energy source (e.g solar panels).
- It should be able to emergency charge a Naiad, so it can reach the closest charging station.
- It should be able to start and land vertically (as a helicopter).
- It should also be able to start and land horizontally (as an aircraft).
- It should be able to start and land at land.
- It should be able to start and land at water.
- It should be able to fly at least 150 km.
- It should be able to carry up to three Naiads.
- It should be able to relay communication from the Naiad to the ground station.
- It should have a navigation system based on GPS and vision.

1.3.2 Initial goals and requirements for the first iteration

The initial goals for the first iteration can be summarized as follows:

- A small scale prototype in the size range of ≈1*2 m and weight of ≈7kg.
- It should be able to carry three mini Naiads. [2]
- It should be powered by electricity.
- It should be able to fly indoors in vertical mode only.
- It should be able to fly for ≈10 minutes.
- It should be able to autonomously follow a predetermined path and avoid static obstacles.
- It should use vision for obstacle avoidance.
- It should use vision and inertia for navigation.
- It should have wireless manual control as backup.

1.3.3 Revised goals and requirements for the first iteration

Due to the low priority of this project it was realized during the course that there would not be enough time or resources to create a vision system. Since inertial navigation alone is much to noisy to provide accurate information of positions and speed, and GPS has no signal indoors, all the autonomous parts were dropped and the obstacle avoidance and path planning were limited to simulations only. Instead the project switched its focus to manual control.

[1] https://www.transportstyrelsen.se/sv/luftfart/Luftfartyg-och-luftvardighet/Obemannade-luftfartyg-UAS/Sok-tillstand-for-UAS/Kategori-3/

[2] The mini Naiad is a 3d printed model of a Naiad without electronics with a size of 18 * 10 * 4.5 cm.

1.4 Report organization

The report is organized in the following way. At first a high level description of the entire system will be provided. Following that, the project plan will be described, i.e how the individual subsystems and tasks have distributed among the competence pools. Next, the mechanical design will be covered, including sizing, aerodynamical considerations and construction techniques. After that the subsystems and parts will be described in detail including requirements, methodology, functionality, testing, limitations and suggested improvements for the future. Following this the results of the project in its entirety will be presented together with some overall conclusions and future work. A checklist for flight, rig tests and some considerations when assembling the aircraft will be provided in the appendices, together with some contact information.

2 System description

This section will provide a high level overview of the entire system. Detailed information will be provided in the following sections. The system as a whole can be seen in figure 2.

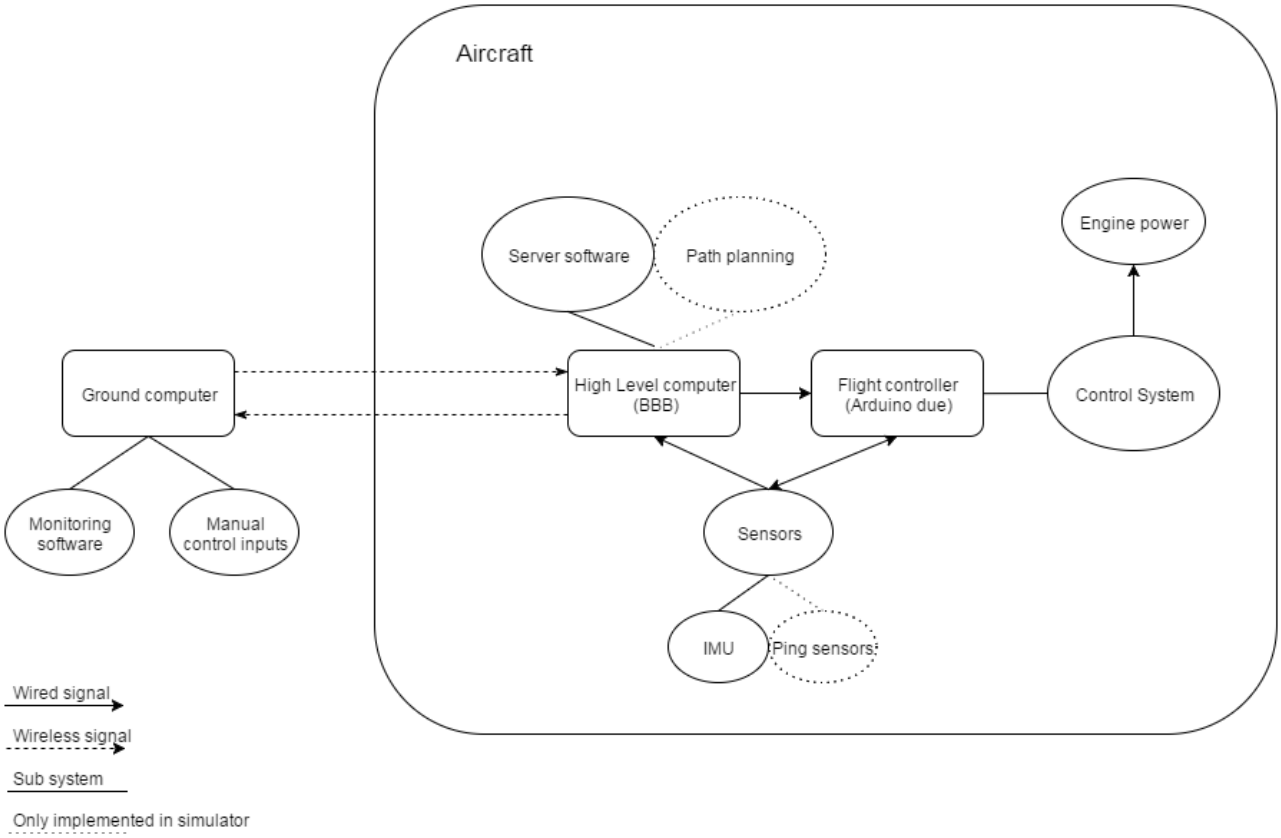

Figure 2: A high level overview of the system.

The ground computer contains a GUI where the parameters from the aircraft, such as pitch roll and yaw, can be monitored. It also plots these parameters over time. Furthermore it contains the manual controls, this means that the aircraft is flown with a keyboard, the inputs are the throttle together with the desired pitch, roll and yaw. It receives the inertial measurement unit (IMU) data from the high level computer and sends the control data to the same computer, this is done wirelessly.

The high level on-board computer is currently only running a server program responsible for relaying the wireless communication between the ground computer and the flight controller. It receives the IMU data and forwards it to the ground computer. It also forwards the control data from the ground computer to the flight controller. This computer was also meant to run the path planner, note that this was never implemented beyond simulations. The beagle bone black [3] was chosen for this. The idea is that this computer will run more programs in the future, such as path planning, image handling and other tasks that require more computational power.

The flight controller computer runs the control system responsible for stabilizing the aircraft in the desired position. It reads sensor data from the IMU and the manual control input from the high level computer and converts this data into appropriate control signals for each of the ten engines. As this is a real-time task with high demands on periodicity, an arduino due was chosen[4].

[3] http://beagleboard.org/BLACK
[4] https://www.arduino.cc/en/Main/ArduinoBoardDue

3 Planning

This section will provide information on how the work was distributed among the competence pools and the project manager.

Software

- Path planner.
  - Create a path planning algorithm that can follow a given path and avoid any obstacles that that may occur by choosing the shortest way around it.
  - Show the functionality of this algorithm in a simulated environment.
  - Implement the algorithm on the high level computer (not done).
- Vision
  - Implement a vision system capable of detecting obstacles (not done).

Communication

- Wireless communication with ground computer.
  - Investigate different communication protocols and choose an appropriate one.
  - Implement a server for this protocol on the high level computer capable of sending and receiving data from the ground computer.
  - Implement a GUI on the ground computer which can receive the IMU data and present them in a intuitive manner, including plots over time.
  - In the same GUI implement a function that can send the control inputs from the keyboard to the high level computer.
- Communication with IMU
  - Implement a function that can communicate with the IMU on the flight controller.
  - Parse the received data into floats so it can be used in calculations.
  - Implement a function on the high level computer that also receives the IMU data and forwards it to the ground computer.
- Internal communication
  - Implement internal communication between the flight controller and the high level computer and forward the ground computer commands to the flight controller.

Electronics

- Power distribution
  - Estimate the total weight of all the electronics.
  - Design and construct a power supply unit according to the specifications.
  - Install the necessary cabling to distribute the power to the components.

Mechanics and project manager

- Aircraft design
  - Design an aircraft that complies with the requirements.
  - Determine how the aircraft should be constructed.

  - Estimate the weight of the finished prototype so work can continue while the aircraft is constructed.
  - Construct the aircraft.

Project manager

- Flight controller
  - Create a simulation environment where the control system can be tested.
  - Investigate what type of algorithms should be used.
  - Create and test the control system in the simulator.
  - Implement the control system on the embedded flight controller computer.
- Get parts
  - Acquire the necessary components, parts, and material for the project.

4 Mechanical design

The mechanical design consisted of designing and constructing everything related to the physical aircraft, the body, wings, fixings for motors, electronics and assembly of the aircraft. It was decided beforehand that the aircraft should be of a tandem configuration. This means that there is no conventional horizontal stabilizer of the aircraft but a front and rear wing. In a traditional configuration the stabilizer provides downwards lift to stabilize the aircraft around its y-axis. In a tandem design both wings provide lift. This is illustrated in figure 3. The reason a tandem configuration was chosen was to provide better control in vertical mode since more engines can be placed on the rear wing. The design has both pros and cons when it comes to conventional flight in horizontal mode, but these are out of the scope of this report.

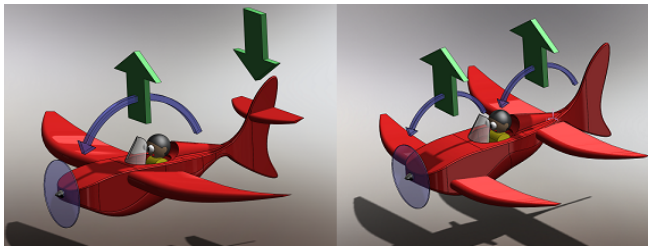

Figure 3: The difference between a conventional aircraft (left) and a tandem aircraft (right).[5]

4.1 Fuselage

The fuselage, or the body of the aircraft, was designed from an inside out principle based on the biggest components it needed to house. The requirement was that three mini Naiads should fit inside. The dimensions of the mini Naiad can be seen in figure 4.

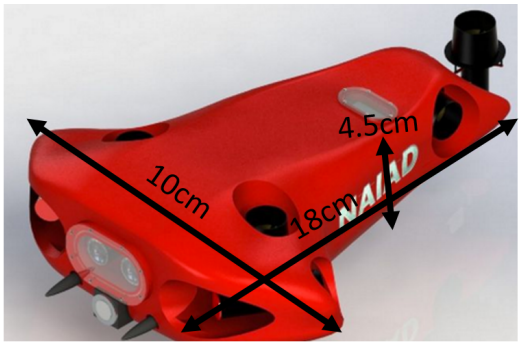

Figure 4: Dimensions of the mini naiad.

Furthermore a GIMMIE2 stereo camera system was decided to be used for vision as they were already available at the university. The dimensions of the camera card is 13 * 8 cm and naturally the lenses need to point forward. The camera system can be seen in figure 5. These two served as a starting point to the sizing of the fuselage.

Figure 5: GIMMIE2.

[5] http://www.nestofdragons.net/weird-airplanes/tandemwings/

Since both components are fairly square shaped, it was decided that a circular conventional fuselage would contain a lot of unused space. The main reason for circular fuselages are that they provide the best structure when pressurizing the cabin [3]. Since this is not a concern here, a square shaped fuselage with rounded corners was considered the best option. The idea was to divide the fuselage into two halves where the bottom half would contain only the Naiads and the top half the electronics. The camera would require both halves in the nose of the aircraft. To allow for some extra space in between and above the Naiads, the dimensions were chosen to be 16 * 12 * 66 cm. A cross section of the fuselage can be seen in figure 6.

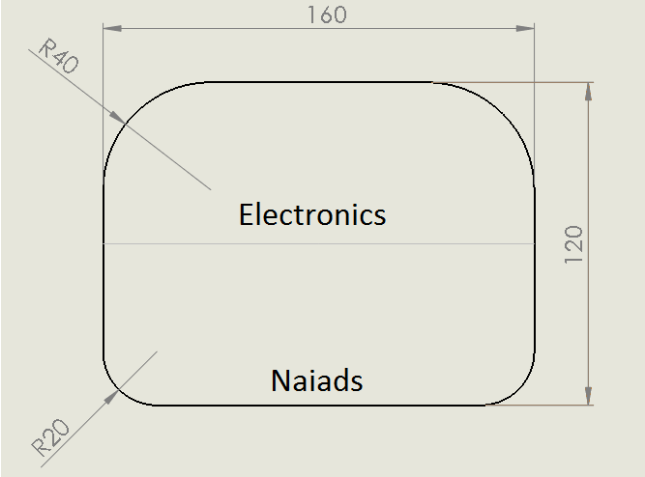

Figure 6: A cross section of the fuselage, dimensions in mm

As a base for choosing the length of the nose and tail, one can consider the ratio ln/d and lt/d, where ln is the length of the nose, lt is the length of the tail and d is the diameter of the fuselage [2]. Typical values for these are:

$$ln/d : 1.2 - 2.5$$

$$lt/d : 2 - 5$$

This landed on a nose of 20cm and a tail of 30cm which gives:

$$ln/d = 1.43$$

$$lt/d = 2.14$$

Since the propellers would be quite large a high wing configuration was chosen to maximize the ground clearance. The full fuselage with dimensions can be seen in figure 7.

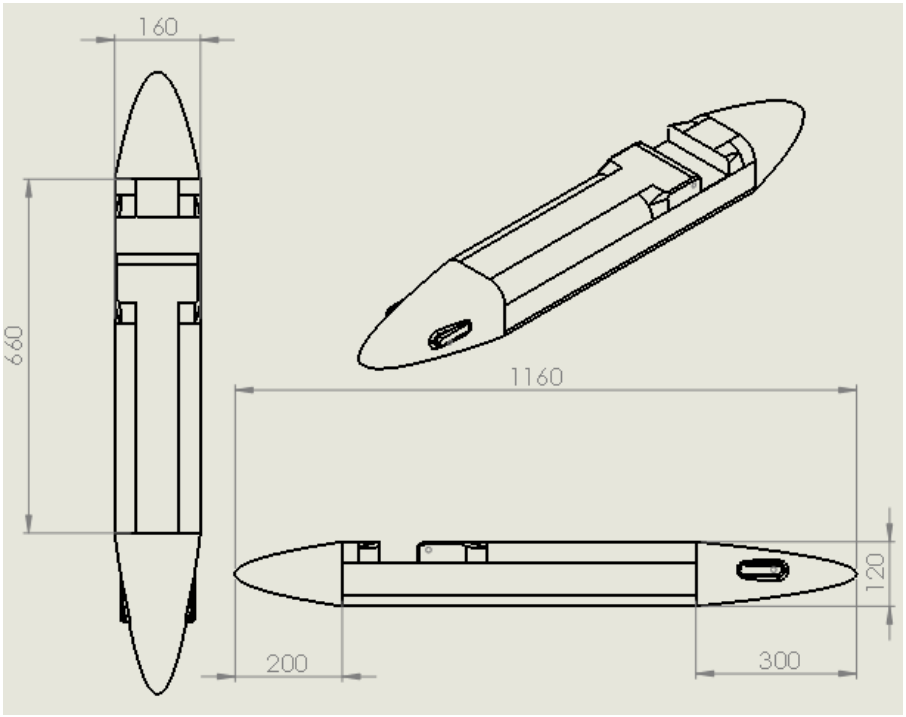

Figure 7: Fuselage with dimensions in mm

4.2 Wings

The wings are what provides lift to a conventional aircraft. In this project the aircraft is only meant to fly in vertical mode and hence all the lift will come from the propellers. The main job of the wings is then simply to hold the propellers in place. To make proper aerodynamical calculations on the wings is out of the scope of this course, nevertheless some basic calculations have been performed to make the aircraft somewhat realistic and to give a good ground for future students to build upon.

4.2.1 Aerodynamical considerations

The basic lift formula looks as follows [4]:

$$L = \frac{1}{2} * v^2 * \rho * A * C_l$$

where,
L = the force,
v = the velocity,
ρ =the air density,
A = the wing area,
C_l = the lift coefficient at the desired angle of attack and Reynolds number [4].

All of these parameters are easily found with the exception of the lift coefficient C_l. This can only be determined by testing the wing. Fortunately standard airfoils with this parameter available exist from the National Advisory Committee for Aeronautics (NACA) [4].

The NACA 2415 was chosen as it is a simple airfoil with decent C_l at low altitudes and speed. Not too much consideration went in to choosing this airfoil as it was not the main goal of this project. This would need to be revised in the future if the aircraft is to fly in horizontal mode. The shape of the airfoil can be seen in figure 8.

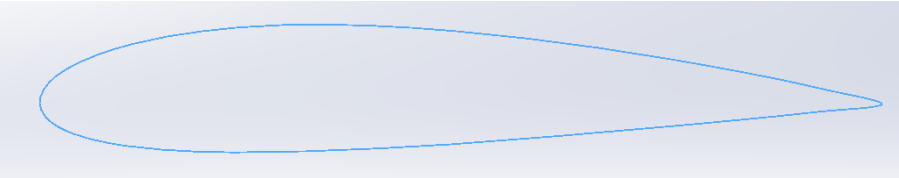

Figure 8: The shape of the NACA2415

As mentioned above C_l is dependent on the Reynolds number and the angle of attack of the wing. For more information on the Reynolds number refer to [4]. A plot of C_l over the angle of attack for the NACA 2415 at the Reynolds number where flight is most likely can be seen in figure 9.

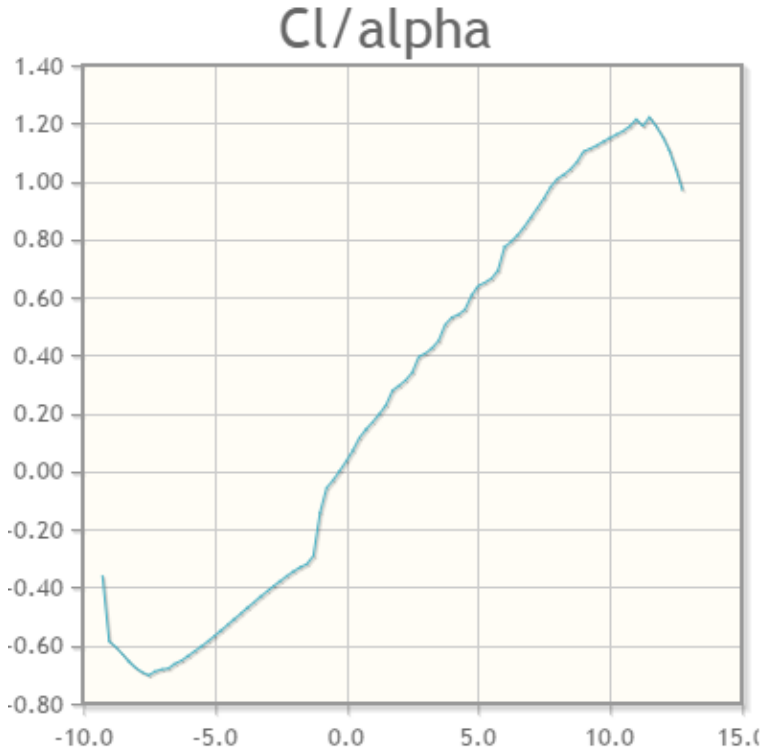

Figure 9: C_l (y-axis) over angle of attack (x-axis) for the NACA 2415[6]

.

From this, it can be seen that the airfoil has a maximum C_l of approximately 1.2 at an angle of attack of approximately 11 degrees.

4.2.2 Sizing

The length of the wings were mainly chosen based on the size of the propellers, at this stage the total weight of the aircraft was just an estimation, they were therefore sized to fit ten 15" propellers to assure that sufficient lift would be available. A tandem aircraft requires one wing to be the main wing, meaning that it provides more lift than the other wing. This can be done in different ways, in this case as ten propellers were chosen, the front wing was simply made larger to fit six propellers and the rear wing was made smaller to fit four propellers.

To save weight, a tapered wing design was chosen, this is structurally more efficient as the wing would get lighter and lighter from the base of the wing towards the tip.

The width of the wings were then chosen based on the total area to provide sufficient lift in horizontal flying mode, combined with what looked aesthetically correct. This landed at a front wing with

[6] http://airfoiltools.com/airfoil/details?airfoil=naca2415-il#polars

a base width of 16 cm, a tip width of 10 cm and a total length of 101 cm. It was also decided that the front wings should be connected to each other to provide a better structure. The rear wing has a base of 12 cm, a tip of 10 cm and a total length of 62 cm. The motor nacelles were given a diameter of 5 cm to fit a variety of motors. Furthermore fixings for ping sensors were placed on each of the outer motor nacelles and in the middle of the front wing. The front wings with dimensions can be seen in figure 10. The rear wings can be seen in figure 11.

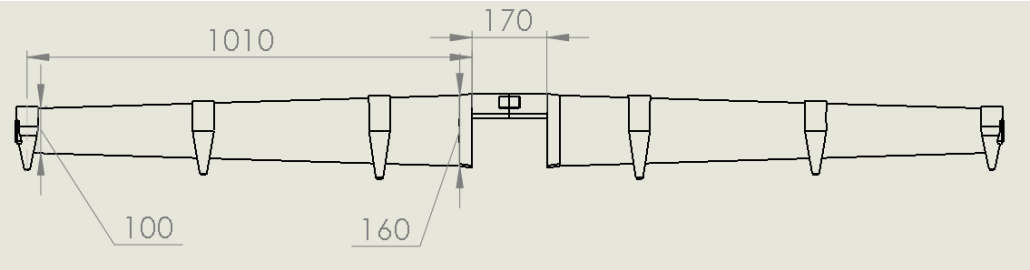

Figure 10: The front wings with dimensions (mm)

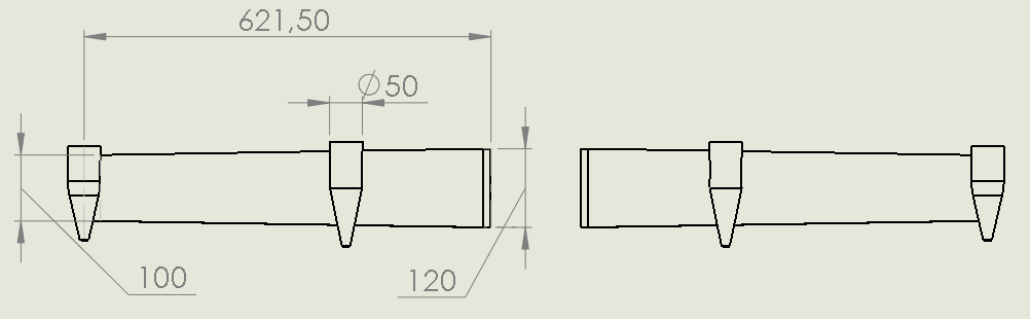

Figure 11: The rear wings with dimensions (mm)

The total weight was very roughly estimated to 8 kg, based on the information provided in section 4.2.1, it can be calculated that the aircraft would need to fly at approximately 60 km/h in horizontal mode to get sufficient lift from the wings which was deemed acceptable. This does not account for the air speed generated by the propellers over the wings which should lower the required speed. If flaps are installed this speed can probably be reduced further at take off, landing and transition between vertical and horizontal flight.

4.3 Vertical stabilizer

The vertical stabilizer is responsible for maintaining the yaw of the aircraft in horizontal flight. It has no effect in vertical flight mode and therefore no calculations were done. It was simply designed to look fairly realistic. It can be seen in figure 12.

The complete design can be seen in figure 1.

4.4 Manufacturing

With the aircraft design completed a manufacturing method needed to be decided, a few options were considered such as 3d printing and aluminum. These were abandoned as 3d printing would take up too much time in the 3d printer as other people need to use it, and an aluminum construction could not be made in house and would therefore be too expensive. Instead carbon fiber was chosen as it is strong light weight and could be constructed ”in house”.

Since access would be needed to both the wings and the body to install components, cables and other parts, it was decided to split all wings and the body in half so that one would get one top half

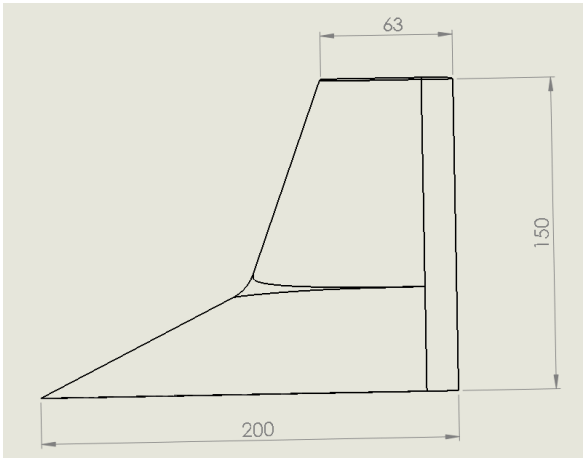

Figure 12: The vertical stabilizer with dimensions (mm)

and one bottom half that would later be put together. This lead to a total of 12 parts that needed to be manufactured:

- Wings
    1. Front left wing top.
    2. Front left wing bottom.
    3. Front right wing top.
    4. Front right wing bottom.
    5. Rear left wing top.
    6. Rear left wing bottom.
    7. Rear right wing top.
    8. Rear right wing bottom.
- Fuselage
    1. Fuselage top.
    2. Fuselage bottom.
- Nose
- Vertical stabilizer.

Since the nose was originally intended to house a camera, it was separated from the body as it needed to be transparent. It was however also built in carbon fiber once it was realized that the vision system would not be created.

The first step was to create molds to lay the fiber on. It was decided to use a wrapping technique meaning that the fiber is lain on top of the mold. The molds were milled out of styrofoam using a CNC milling machine available at the university. A picture of the styrofoam beeing milled and a few finished pieces can be seen in figure 13.

With the molds finished, the fiber could be applied. The fiber cloths used were a 200 g/m^2 0-90 mat, meaning that the fibers are oriented in a square pattern, and a 200 g/m^2 unidirectional mat, meaning that all fibers are oriented in the same direction making it stronger in that direction. The idea was to use the unidirectional cloth on the inside of the wings as most forces on the wing would be in that direction.

Figure 13: Creating the molds.

The molds were covered in packing tape to make the carbon fiber release once it had hardened. Carbon fiber is basically lain by first applying epoxy with a paint brush, then putting down a layer of fiber, then applying new epoxy and so on until the desired number of layers is reached. The more layer the stronger, but also heavier, the structure will be. It was not known exactly beforehand how many layers would be required.

Three layers of 0-90 cloth were lain in this fashion on all the parts except the nose and vertical stabilizer and then the molds were removed. The body and rear wings felt very solid after this and it was decided that no more layers were needed. One extra layer of unidirectional cloth was lain inside the front wings as they are longer than the rear wings. The styrofoam mold was reattached to the wings using epoxy as it provided some structural advantages at nearly no extra cost in weight. The nose was wrapped in only one layer of fiber as it would not take any structural loads. The vertical stabilizer was tricky to wrap due to the shape and size, it was therefor decided that it would be 3D printed instead. One large flat piece of two layers of unidirectional cloth put in a 0-90 pattern was also created. This was to be used as a plate for the electronics inside the fuselage, and for connecting the two pieces of the fuselage. A few pieces in the manufacturing process can be seen in figure 14.

Once the pieces were completed the excess fiber along the edges could be cut off using a dremel tool, and the last few mm could be sanded down. During this the pieces were somewhat abused which slightly ruined the clear finish. They were therefor lightly sanded down all over and sprayed with a clear coat spray.

For future reference, a better result both structurally and aesthetically may be achieved by vaccum bagging the the fiber while the epoxy is curing. This will squeeze the layers together preventing any de-laminations in the the corners and make sure that the fiber gets shaped better after the mold. It should also be noted that this work was very time consuming, reaching over hundreds of man hours, if it is the be redone in a similar manner in the future, one should be prepared to put in some time.

Figure 14: Laying the carbon fiber cloths.

4.5 Assembly

The manufacturing procedure resulted in 12 separate parts with no fixings for neither the motors nor assembly of the aircraft itself.

Fuselage
The body was assembled by cutting two 60 * 4 cm parts of the flat carbon fiber piece. Holes were drilled and the two parts were attached with bolts and nuts to the lower half of the body. The top half of the body could then be put on as a lid on the lower half. Holes were drilled through the top half of the body and the two assembly pieces and the nuts were glued on the assembly pieces so the top half can easily be removed and attached whenever this is necessary. This is shown in figure 15. The lower line of bolts also provides fixings for the internal electronics plate, these should not normally be removed. For the future, new fixings should probably be mounted so the internal electronics plate can be raised to create more space for the Naiads.

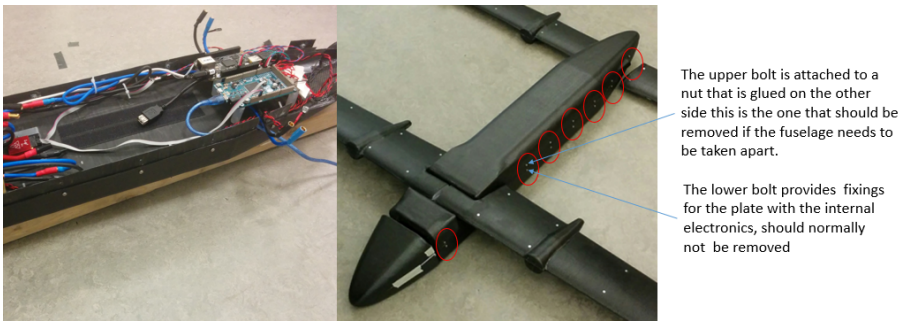

Figure 15: Bolts holding the top and bottom part of the body together.

Wings
The two halves of the wings are simply assembled with bolts from one side and nuts from the other side, seen in figure 16. The front wings are connected at the middle with a 3D printed piece.

Connecting the wings and fuselage
The wings are attached to the body with two 1 m prefabricated carbon fiber rods, one for the front wings and one for the rear wings. Holes were drilled through the body at the correct position and the rod was pushed through. Tracks of the correct size were cut out of the styrofoam inside the wings,

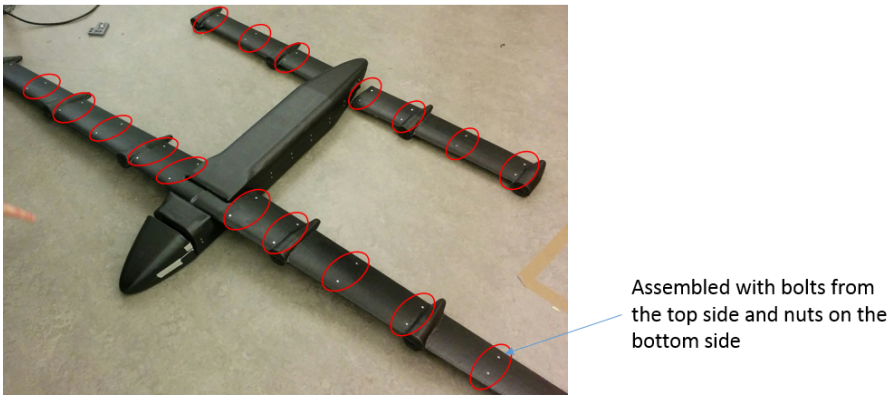

Figure 16: Bolts holding the top and bottom part of the wings together.

by doing so the wings simply slide on to the rod. The rod and wings were then locked together by drilling holes through the wings and the rod itself and attaching bolts and nuts, two holes for each wing. This makes it easy to remove the wings for transportation etc. The wings can now tilt from vertical to horizontal position. Since no servos were to be installed during this iteration of the project, the wings were simply locked in the vertical position using pipe clamps.

Nose and horizontal stabilizer

The two halves of the nose were simply glued together and attached with screws from inside the lower half of the body. The horizontal stabilizer is glued on the body and can be removed with some force.

Motor fixings

The motors are attached with bolts from behind the motor as shown in figure 17. The carbon fiber itself at the front of the motor nacelles is not strong enough to simply bolt the engine to. To overcome this some circular plastic plates were 3D printed and glued on just inside motor nacelle, and on the front of the motor nacelle on the top half on the wings. The styrofoam was also cut out from the nacelles. The motors were then bolted on to these circular plates on the top half of the wing. When the two halves of the wings are connected, the plastic plate and the carbon fiber provide a combined structure to hold the motors in place. This can be seen in figure 18.

Figure 17: Holes for attaching the motor. [7]

.

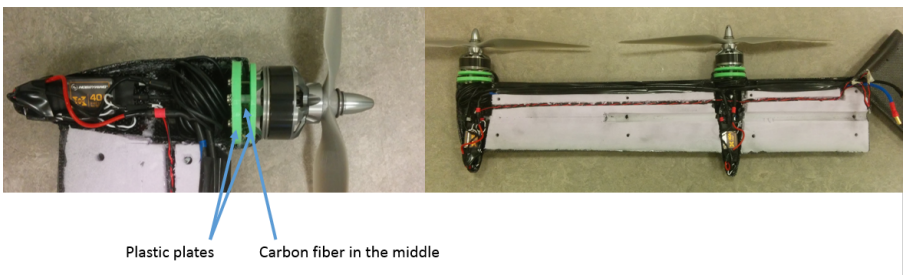

Figure 18: Plastic plates are part of the motor assembly

[7] http://www.rctigermotor.com/html/2013/Navigator_0910/38.html/

5 Control system

The control system design started with a moderate literature review to determine a suiting control strategy. Though many papers have been published on new control system approaches for UAV:s, many seem to never get beyond experimental results, and show little improvement over traditional control systems such as a PID controller. In a survey of advances in UAV technologies [5]. It is mentioned that many researchers have switched the focus from control related research to other areas due to this reason.

It was therefor not deemed necessary to add extra complexity for little gain in performance in this first iteration of the project, and a traditional PID controller approach was chosen. The PID controller is the most widely used control method in most industries and is very easy to implement, but may be tricky to tune. For more information on this controller, refer to [1]. There were no specific requirements for the control system, except for the obvious fact that it should be able to control the orientation of the aircraft.

5.1 Aircraft motion

An aircraft can rotate around three axes: rotation around the y-axis is called pitch, rotation around the x-axis is called roll and rotation around the z-axis is called yaw. This can be seen in figure 19. Conventional aircraft flying in horizontally control these motions with control surfaces, in the most basic setup, the pitch is controlled with the elevator attached to the horizontal stabilizer, the roll is controlled with the ailerons attached to the wings and the yaw is controlled with the rudder attached to the vertical stabilizer. The forward speed is controlled with the engine.

Multi-rotor vehicles such as quadcopters have the same motions but they are controlled in a different manner. All the motions are normally controlled by adjusting the speed of the motors. If one considers a quadcopter as it is the simplest setup, the pitch is controlled by increasing the speed of the rear motor while decreasing the speed of the front motor or vice versa. The roll is controlled in the same way but using the left and right motor. Since the motors provide torque, two motors need to spin in one direction and the other two need to spin in the opposite direction to equalize the torque and keep the vehicle from spinning around. This is used for controlling the yaw. The speed of the two motors spinning in the same direction is increased while the speed of the other two is decreased, thus creating a yaw motion. In all cases the speed is increased and decreased equally on the various motors, thus maintaining the total lift. Pitching a quadcopter will make it go forward or backwards, rolling will make it go left or right, yawing will make the front end point left or right and changing the motor speed will make it go up or down.

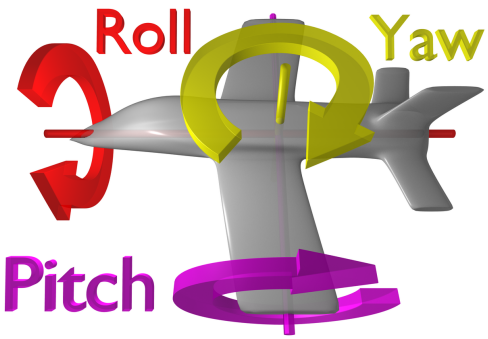

Figure 19: The motion of an aircraft around its axes (x: red, y: purple, z: yellow).[8]

[8] https://en.wikipedia.org/wiki/Flight_control_surfaces#/media/File:Flight_dynamics_with_text.png

5.2 Design

It was decided that during this first iteration, no control surfaces would be installed and hence all motions would be controlled with the speed of the engines in a multi-rotor fashion. The first thing to consider was which motors should control which motions. The roll and pitch are fairly straight forward. The pitch would have to be controlled with the front and rear motors, and the roll with the left and right motors. The yaw however can be controlled in different ways by varying the direction the motors spin. Five motors would always have to spin clock-wise and five motors would have to spin counter clock-wise to equalize the torque. Some sort of symmetry about the center of gravity is also required to not affect the pitch and roll when inputting yaw commands. The rotation of the motors, and the naming convention can be seen in figure 20.

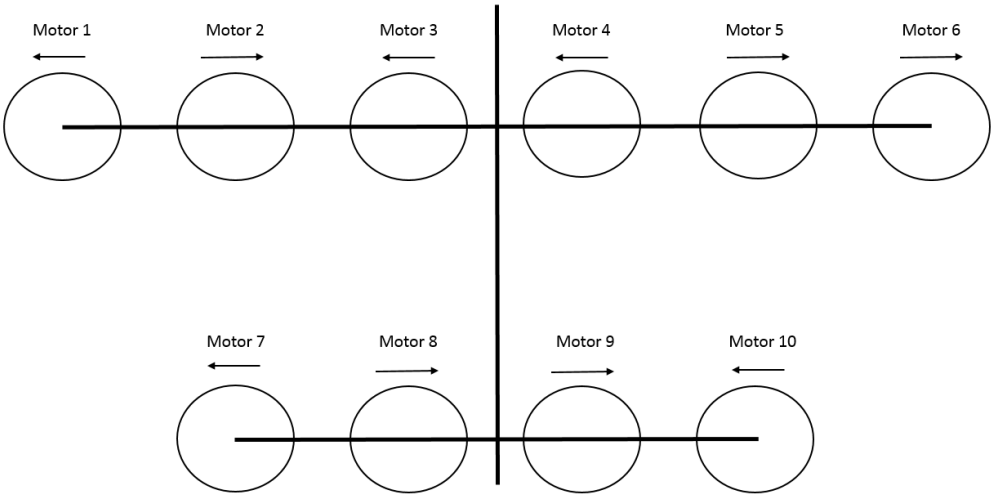

Figure 20: The rotation of the motors and the naming convention.

An overview of the full control system can be seen in figure 21. All in all it is a question of subtracting and adding power to the individual engines to achieve the desired orientation. More details on the individual controllers will follow below.

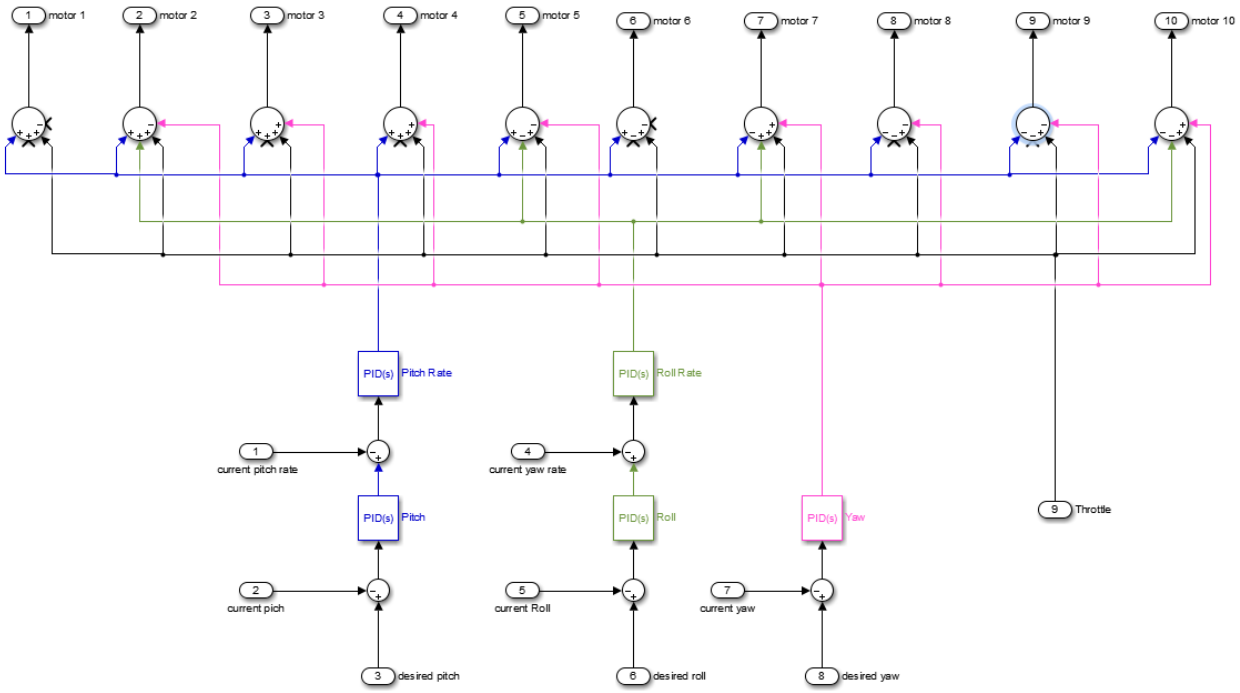

Figure 21: An overview of the control system.

5.2.1 Pitch control

The pitch controller consists of two PID controllers in an inner and outer control loop. At first the pitch and roll were tested with only the outer loop. This worked well in simulations but as pitch and roll motions are sensitive, the response was too slow in real testing. More on simulations and testing will follow later.

The inputs to the outer control loop is the current pitch from the IMU and the desired pitch from the manual commands. The output will be a desired rate of change for the pitch. The input to the inner loop is the desired rate of change from the outer loop and the current pitch rate of change from the IMU. The output is a desired throttle value between 1 and 100% which is sent to all the motors. This value will be added to motors 1 through 6, and deducted from motors 7 through 10 or vice versa, hence pitching the aircraft either forward or backward. See the blue line in figure 21.

5.2.2 Roll control

The roll controller works in the exact same way as the pitch controller but only motors 2, 5, 7 and 10 are affected. The output value will be added to motor 2 and 7 and deducted from motor 5 and 10 or vice versa, hence causing the aircraft to roll either left or right, See the green line in figure 21.

5.2.3 Yaw control

The yaw controller is less sensitive than the pitch and roll, therefore only one PID controller is used to simplify the tuning procedure. The yaw controller reads the current yaw from the IMU and the desired yaw from the manual control and directly outputs a desired throttle value. The value is added to motors 2, 5, 8 and 9, and deducted from motors 3, 4, 7 and 10. Which changes the balance of the

torque and causes the aircraft to yaw either left or right. These motors should not affect the pitch and roll balance of the aircraft. See the pink line in figure 21.

5.2.4 Throttle

The throttle is not a controller, it simply adds the throttle from the manual input to all the motors making the aircraft go either up or down. This could be replaced with an altitude controller that would always output throttle to stay at a desired altitude, which would make the aircraft easier to fly, and it would be a must for autonomous flight. Unfortunately time did not allow for this to be implemented beyond simulations.

5.3 Simulations

For simulations of the control system, SimMechanics which is part of Simulink was used. In SimMechanics, a multibody system can be created using blocks representing bodies, joints, constraints, forces etc. and SimMechanics will automatically formulate and solve the equations of motion. Furthermore, a full CAD model can be imported and once again, SimMechanics will automatically formulate and solve the equations of motion related to mass, inertia and constraints. It also generates a 3D animation to visualize the system [9]. Since the aircraft is only controlled with upwards force and torque, and a CAD model had been created, SimMechanics provided a good and easy to use simulation environment.

An overview of the simulation blocks can be seen in figure 22

[9] http://se.mathworks.com/help/physmod/sm/gs/product-description.html

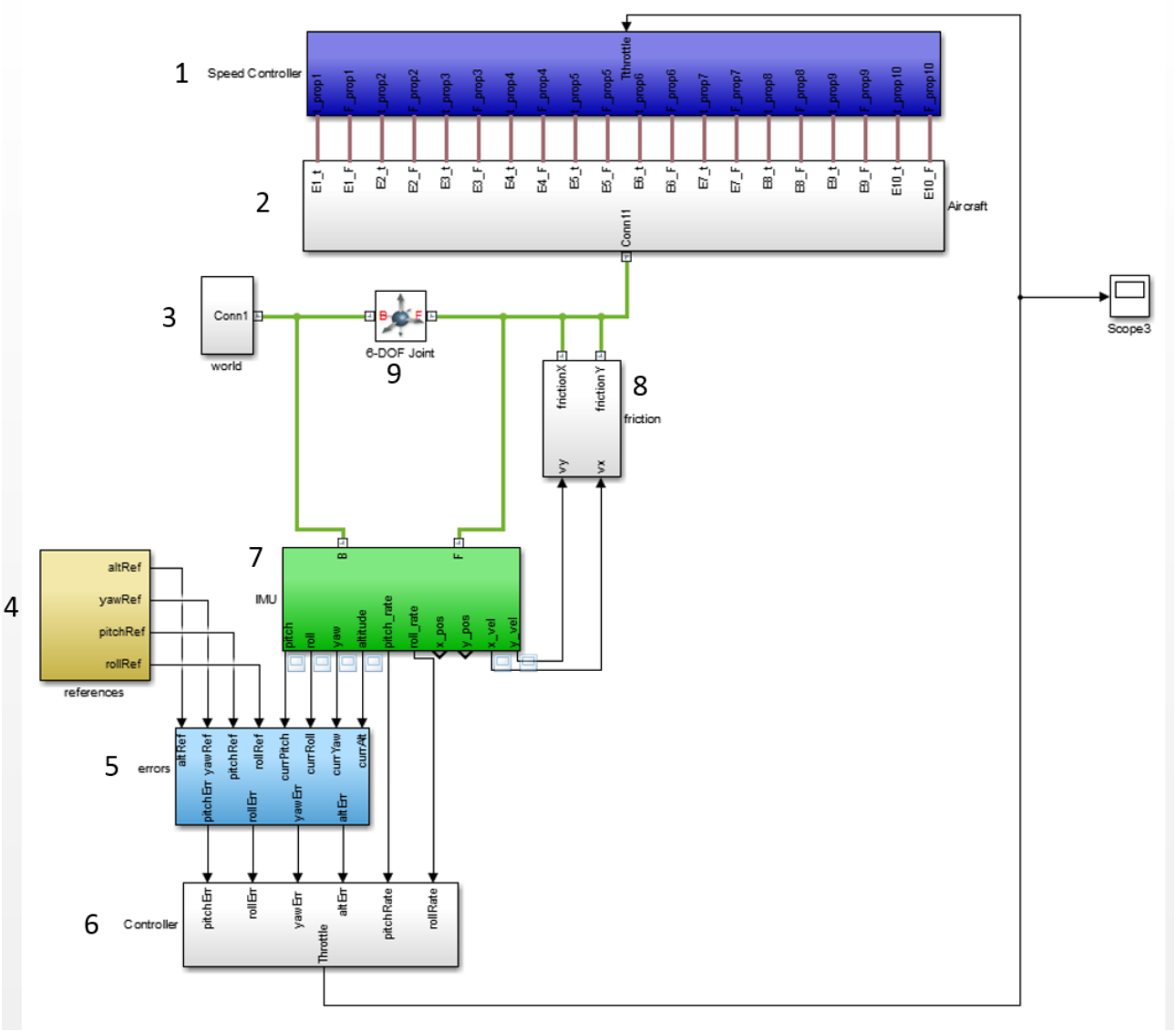

Figure 22: An overview of the simulation environment blocks.

Block description

1. This block converts the throttle from the controller to thrust for SimMechanics. This is done through a table lookup using a table provided by the motor manufacturer that shows the thrust from the motors at different throttle levels.

2. This is the automatically generated SimMechanics blocks from the CAD file. i.e the aircraft itself.

3. This block contains the parameters of the world, also an automatically generated SimMechanics block.

4. These are the pitch, roll and yaw references. This corresponds to manual commands in the real aircraft. Note that the simulator has an altitude reference rather that a throttle input.

5. These are the errors of the other control loops, simply desired position - current position.

6. This block contains the control system.

7. This is the simulated IMU, keeps track of orientation, rate of change, position, velocities etc.

8. A friction block was added since SimMechanics is basically a vacuum.

9. This is the relation between the world and the aircraft, a six degrees of freedom block, meaning that the aircraft can rotate around and move along all the axes.

With the simulator working the control loops could be tested and tuned. This is a very iterative process. Some heuristic tuning techniques were tried, but in the end it was basically a trial and error procedure. The goal was never to achieve perfect control as it was suspected that the real aircraft would not behave in the exact same way(This later turned out to be true). It was rather to provide a decent control system and verify that the aircraft could in fact be controlled with the proposed technique.

An example response from the simulator where the pitch goes from 0 to -10 to 10, and roll goes from 0 to -5 to 5, can be seen in figure 23

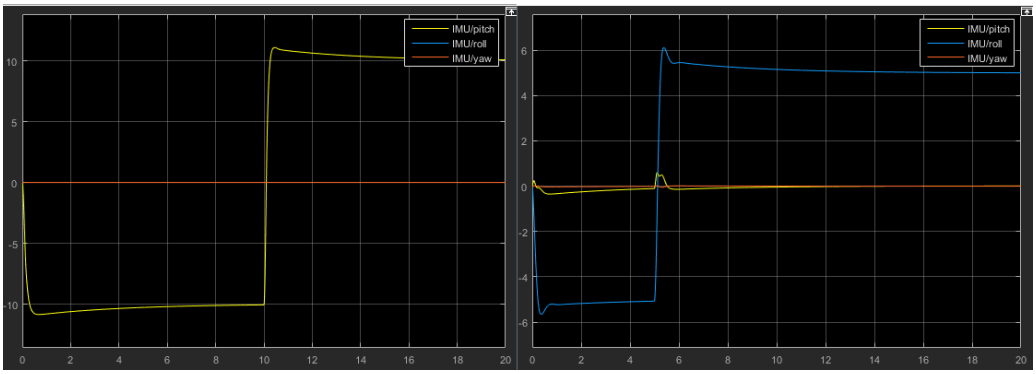

Figure 23: Example response of the pitch (left) and roll (right).

5.4 Hardware

An aircraft control system is a real time system that needs to run at a constant periodicity. Generally speaking, the shorter the period, the better the vehicle can be controlled. An Arduino due was chosen as an embedded computer to run the control system as it provides real time response, and allows for relatively rapid prototyping. The period was set after some experimenting with the response time to run all the required functions. The most time consuming task was reading the data from the IMU, the quickest response that was achieved while assuring that no data was missed was approximately 10 ms. The remaining tasks take only a few ms. The period was set to 15 ms. This is a fairly long period but this is a large aircraft that is not intended for any sort of aerobatic flying.

Simulink supports automatic code generation for the Arduino, but as the algorithms are not too complicated and it is much easier to debug and add functionality later, the Arduino was coded manually. The code was then verified by comparing some example input and outputs values with the Simulink model.

5.5 Initial testing

While the aircraft was still being constructed, a temporary aircraft made out of wood, with the same dimensions as the real one, was created to test the control system in reality. This wooden aircraft was placed one a home made three degrees of freedom joint that was connected to a test rig, meaning that motions around the x, y and z axis(pitch, roll, yaw) were allowed. The joint can be seen in figure 24. Only eight motors were used as it was unclear how much current the temporary cables could handle. The full setup can be seen in figure 25. The temporary aircraft was only tested on the rig, it never flew freely. There is no data available from this test but after some PID tuning the control system performed well and kept the aircraft stable. This gave a good indication that the proposed system should work in reality.

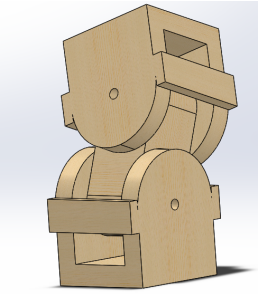

Figure 24: 3DOF joint.

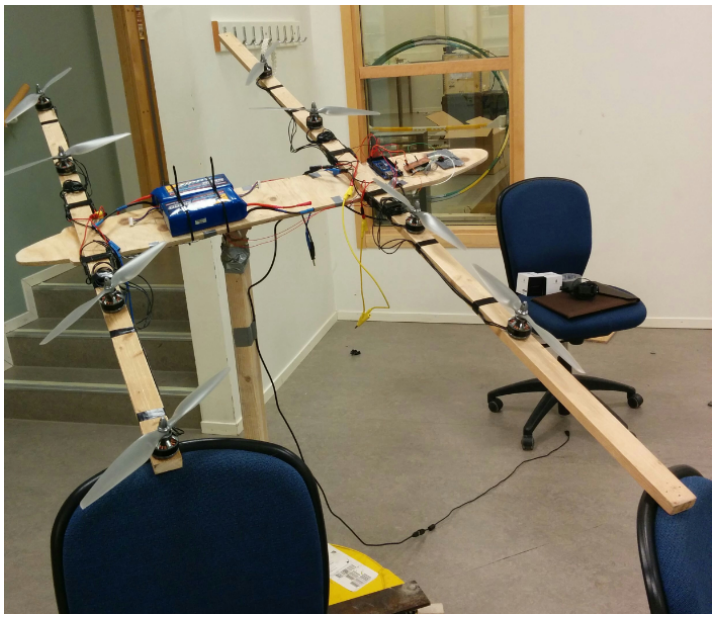

Figure 25: A temporary wooden aircraft used for initial testing.

5.6 Future work

All in all the control system works quite well. The altitude controller was only implemented in the simulator, The next step would be to implement this on the aircraft itself. Low altitudes could be sensed with for example a ping sensor, code for this was written but never implemented on the aircraft. Higher altitudes can be sensed either with the barometer available on the IMU or the GPS. It may also be a good idea to add a real time operating system for the control system. It may provide better timing for the individual tasks. A version of freeRTOS was successfully tested on the arduino, but there was not enough time to implement the whole control system in this manner. FreeRTOS together with some other real time operating systems ported to arduino can at the time of writing be found at *https://github.com/greiman?tab=repositories*.

The rotational direction of the motors can be configured in different ways to control the yaw. The current configuration is not necessarily the best. A few different ones were tested in the simulator with little to no difference. There was however no time to test them in reality. If control surfaces are installed the yaw could also be controlled via the ailerons by deflecting the air in different directions on the left and right wing, which would probably provide a more responsive and better yaw control.

A more advanced control system could also be tested, for example some sort of adapting system.

6 Motors and propellers

The motors were chosen based on table data from the manufacturer containing the total thrust with different propellers at different throttle levels. The aircraft weight was estimated to 7 - 8 kg. The goal was to make the aircraft hover at approximately 50% throttle, meaning that each motor needed a thrust of approximately 0.75 kg at 50% throttle ($0.75 * 10 = 7.5kg$). Many motors are available at the market, the chosen motor was recommended by the vendor as it is light and provides sufficient power. The data for the chosen motor can be seen in figure 26.

Item No.	Volts (V)	Prop	Throttle	Amps (A)	Watts (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating temperature(℃)
MN4010 KV370	22.2	T-MOTOR 12*4CF	50%	3	66.60	600	5000	9.01	40
			65%	4	88.80	730	5600	8.22	
			75%	5	111.00	860	6200	7.75	
			85%	6.6	146.52	1090	6800	7.44	
			100%	7.7	170.94	1200	7300	7.02	
		T-MOTOR 13*4.4CF	50%	3.1	68.82	650	4800	9.44	47
			65%	4.3	95.46	810	5300	8.49	
			75%	5.5	122.10	970	6000	7.94	
			85%	7.1	157.62	1160	6700	7.36	
			100%	8.5	188.70	1340	7100	7.10	
		T-MOTOR 14*4.8CF	50%	3.5	77.70	780	4400	10.04	51
			65%	5.7	126.54	1060	5100	8.38	
			75%	7.6	168.72	1310	5700	7.76	
			85%	9.8	217.56	1590	6300	7.31	
			100%	11.7	259.74	1830	6600	7.05	
		T-MOTOR 15*5CF	50%	4.2	93.24	940	3900	10.08	52
			65%	7	155.40	1300	4800	8.37	
			75%	9.3	206.46	1620	5350	7.85	
			85%	12.2	270.84	1950	5700	7.20	
			100%	14.6	324.12	2240	6100	6.91	
Notes:The test condition of temperature is motor surface temperature in 100% throttle while the motor run 10 min.									

Figure 26: Motor data for the T-motor MN4010, with the chosen propeller marked.[10]

14" propellers were chosen as they provided sufficient thrust. The aircraft can fit 15" propellers and should more power be required in the future, they can easily be replaced. As can be seen in figure 26, the thrust at 50% power from all 10 motors should be around 7.8kg, and at maximum power the thrust should be around 18 kg.

This table also served as requirements for the power supply unit and cable dimensions.

[10] http://www.rctigermotor.com/html/2013/Navigator_0910/38.html

7 Power Supply Unit

7.1 Description

DO NOT TOUCH THE HEATSINKS WHEN THE BATTERY IS CONNECTED, THEY ARE NOT ISOLATED!

The power supply is responsible for distributing the power and to some degree protecting all the different systems inside the UAV. The input power comes from a 6S LiPo battery which powers the motors and the 22,2V is converted to 12V and 5V to power the different electronics.

7.2 Requirements and Limitations

Design characteristics		
Inputs:	22,2V @ Battery current	
Outputs:	22,2V @ 100A 12V @ 2A 5V @ 2A	
Battery:	6S LiPo	
Protections:		
Short circuit:	NO	
Reverse polarity:	NO	
Over-voltage:	NO	
Reverse current:	YES	
Measurements:		
Current:	NO	
Voltage:	NO	
Software operated:	NO	
Input connectors:		
LiPo battery	x2	
Output connectors:		
Motors	22,2V	x 3
22,2V	22,2V	x 3
12V	12V	x 4
5V	5V	x 4

Table 1: Design characteristics

7.3 Design and Interface

The PSU was designed in Multisim and the PCB was designed in Ultiboard. In Multisim it was designed to be modular and easy to follow. This was achieved by using subcircuits, which will be described below with the same name as in the Multisim file. The subcircuits makes it possible to hide the components and give a good overview, without being overwhelming. The Ultiboard design was the big issue, when the motors are all on and at max throttle they will consume about 100A, which is not easy to route on copper traces. Because of this the PSU is a PCB, cable hybrid.

The missing connections in the Multisim file is where wires have been used and in the Ultiboard file, the wires are represented by silkscreen. The blue silkscreen is 22,2V and the red one is GND.

7.3.1 Safety motors

Description:
It works as an ON/OFF switch and protects from reverse currents that the motors can generate.

Requirements:
A circuit that can switch a load of 100A and protect from reverse currents.

Design and Interface:
3 BTS555s were chosen for the job, and they are rated for 45A. They were chosen because they had already been tested in the Butler and ROARy PSUs and the high current rating. Each BTS555 provides current to a front wing or the tail. They are turned on by a transistor that allows a small current to pass from pin 2 to GND.

Notes:
The transistor is a BJT, which means the system uses a small amount of current that wouldn't be used with a MOSFET.

7.3.2 Buck 5V

Description:
Convert the battery voltage to 5V.

Requirements:
Provide a little more than 2A at 5V.

Design and Interface:
A LM2678-5.0 was used, since they were used in the Naiad and had proven themselves to be quite good. The components used were taken straight out of the LM2678 datasheet where they have tables corresponding to input voltage and what output current is required. It can be turned off by allowing a small current to pass out of it, which is done with a transistor.

Notes:
The LM2678-5.0 should be able to output 5A, but from the tests in the Butler it collapsed at 2A, this design is not the same so hopefully it works at higher currents but this has not been tested.

The coil should probably be moved away from the other components to reduce the noise in sensitive parts of the converter, like the feedback.

The transistor is a BJT, which means the system uses a small amount of current that wouldn't be used with a MOSFET. This is quite bad because the PSU is supposed to be in standby when the BUCKs are turned off, and then you want to consume a minimal amount of current.

7.3.3 Buck 12V

Description:
Convert the battery voltage to 12V.

Requirements:
Give enough current for an Arduino at 12V was the minimum, but it is good to have some extra for future upgrades.

Design and Interface:
A LM2678-12.0 was used, since they were used in the Naiad and had proven themselves to be quite good. The components used were taken straight out of the LM2678 datasheet where they have tables corresponding to input voltage and what output current is required. It can be turned off by allowing a small current to pass out of it, which is done with a transistor.

Notes:
The LM2678-12.0 should be able to output 5A, but from the tests in the Butler it collapsed at 4A, this design is not the same so hopefully it works at higher currents but this has not been tested.

The coil should probably be moved away from the other components to reduce the noise in sensitive parts of the converter, like the feedback.

The transistor is a BJT, which means the system uses a small amount of current that wouldn't be used with a MOSFET. This is quite bad because the PSU is supposed to be in standby when the BUCKs are turned off, and then you want to consume a minimal amount of current.

7.3.4 Safety 12V

Description:
There are two identical versions of this circuit on the PSU, one for the 5V channel and one for 12V. It works as a switch for these channels and protects from reverse currents that may occur.

Requirements:
Be able to switch 5A (current rating of the BUCKs) and protect from reverse currents.

Design and Interface:
The main part of the circuit is the BTS50085-1TMA which is the same kind of component as the BTS555, but they don't have the exact same features or current rating. It is switched on by allowing a current to pass out of pin 3 and there is a big capacitor that is supposed to protect from voltage drops.

Notes:
The transistor is a BJT, which means the system uses a small amount of current that wouldn't be used with a MOSFET.

7.4 Using the system

The battery inputs are the biggest cables, the positive inputs are on top and they are red and the GNDs are on the bottom and they are black. There are three outputs for the different wings and they have the same arrangement with top being positive and bottom GND, the positive outputs use blue cables and the GND uses black cables. The button is supposed to connect with the U4 connector but since it broke, the cables have been soldered directly onto the PCB and there is a connector between two cables instead. When the two cables are closed, all the outputs are supposed to turn on.

7.5 Test and simulation results

There weren't any tests conducted apart from connecting a small DC-motor and making sure the on and off button works. Then it was pretty much immediately used in the UAV because of the time constraints, but it has been working fine.

7.6 General notes

The flip button is used to switch the output channels ON in one direction and OFF, as well as putting the PSU in standby, in the other direction. The button has a connector and can be pulled off, causing the outputs to switch OFF which can be useful for emergencies.

7.6.1 Issues

On the back of the PCB, one of the traces to the transistors is not properly connected in the Ultiboard file and this was solved by soldering a small wire over the hole. On the PSU that was used at the time of writing this, the connector U4 is not used because the copper pads broke off when trying to find a short circuit that seems to have been a production fault.

7.7 Troubleshooting

If none of the outputs give power, check the connection between pin 1 and 2 on U4, it needs to be a closed connection to enable the outputs. If one of the outputs doesn't work, check the transistor and make sure current can pass through it.

If one of the BUCKs doesn't work, check the input voltage to the corresponding BTS50085-1TMA, if it is the correct voltage go to the subsection above and make sure the BTSs are closed, if there isn't a voltage check the soldering of the components.

7.8 Future work

A new revision should look into the possibility of having all the current routed on the PCB. There are options like Würth Wirelaid where they have wires embedded underneath the visible copper trace, and PCB bus bars which is a bar that you solder to the PCB that conducts electricity.

The BUCK converters should be tested and if they don't perform as desired, the design should be revamped. There are calculators and example layouts available for the LM2678 and using this would most likely be a good next step.

8 Cabling

This section will provide a brief overview and schematics of the cabling running onboard the aircraft, including the power distribution for the motors and the internal electronics.

8.1 Power distribution

The power is distributed through 3 cables, or 6 if you include the GND cables. The cable has a branch for each motor to avoid having one cable to each motor. There are connectors by each wing so the wings can be taken off without too much trouble. As of now the cables coming out of the motors are very long and rolled up inside the motor nacelle. If the current configuration is kept these cables should be shortened to save weight and space.

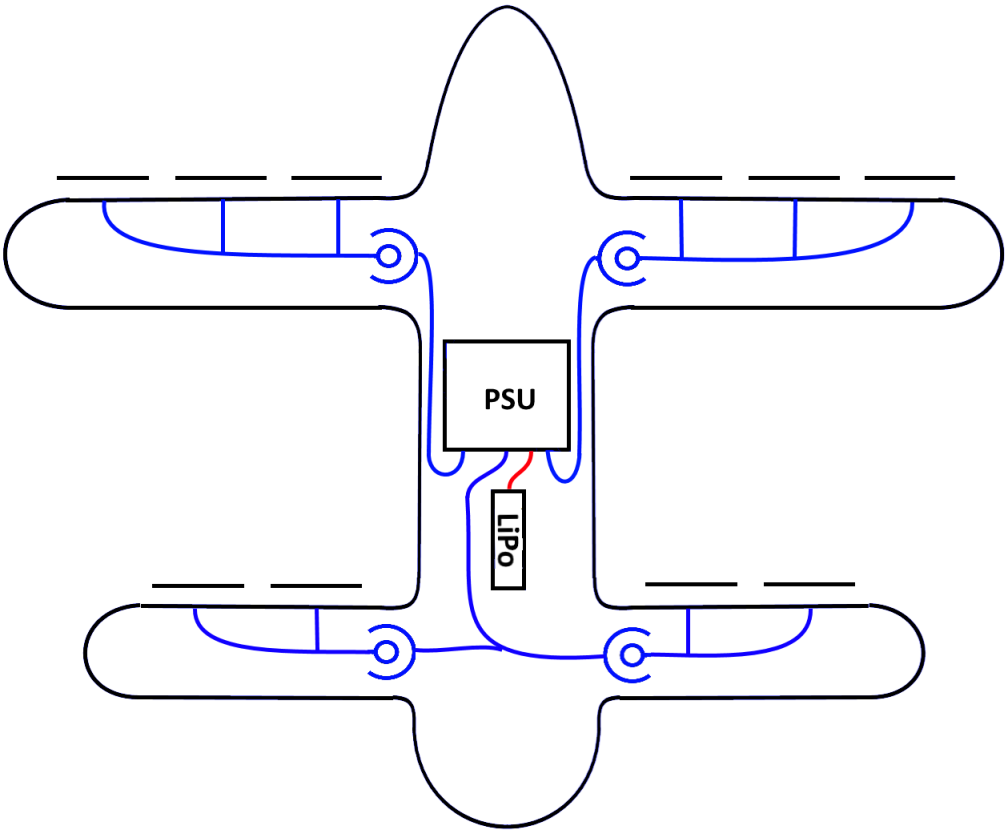

Figure 27: Power distribution sketch

8.2 Internal electronics

The internal electronics and how it should be connected can be seen in figure 28. The signal cables for the motors also have connectors at the base of each wing making it easy to remove the wings should this be required.

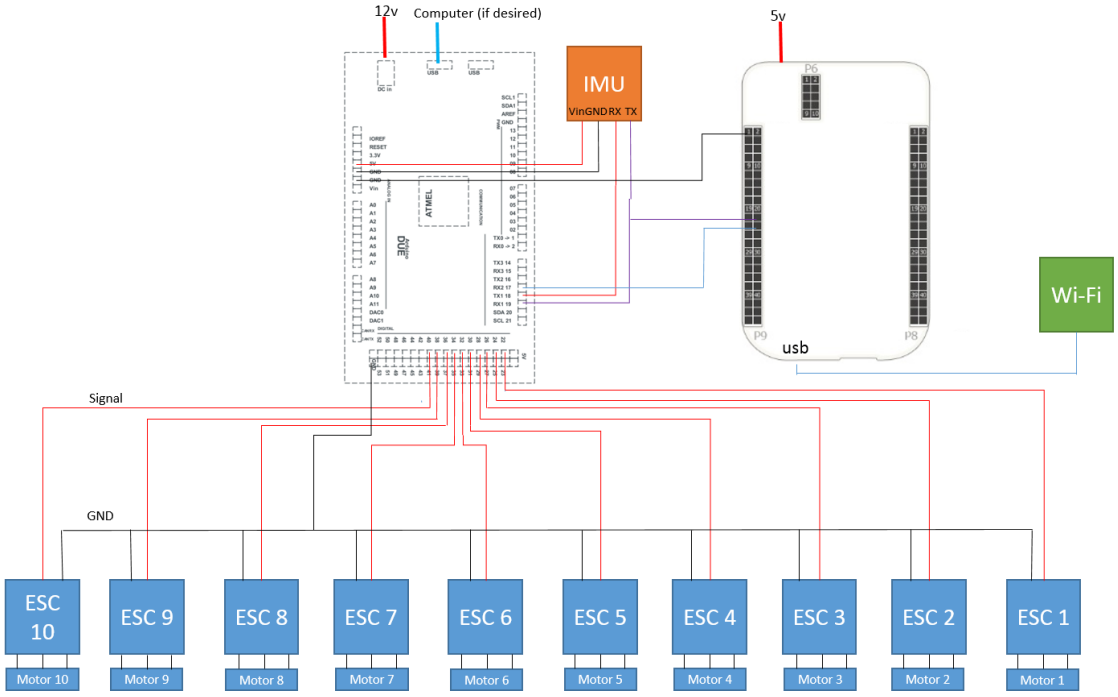

Figure 28: Schematics of the internal electronics.

9 Communication

9.1 Description

The UAV has a simple communication structure which should be easy to understand and modify. The processing units used in this project is an Arduino, a beaglebone black and a ground windows computer. The picture below shows the current external communication architecture of the UAV:

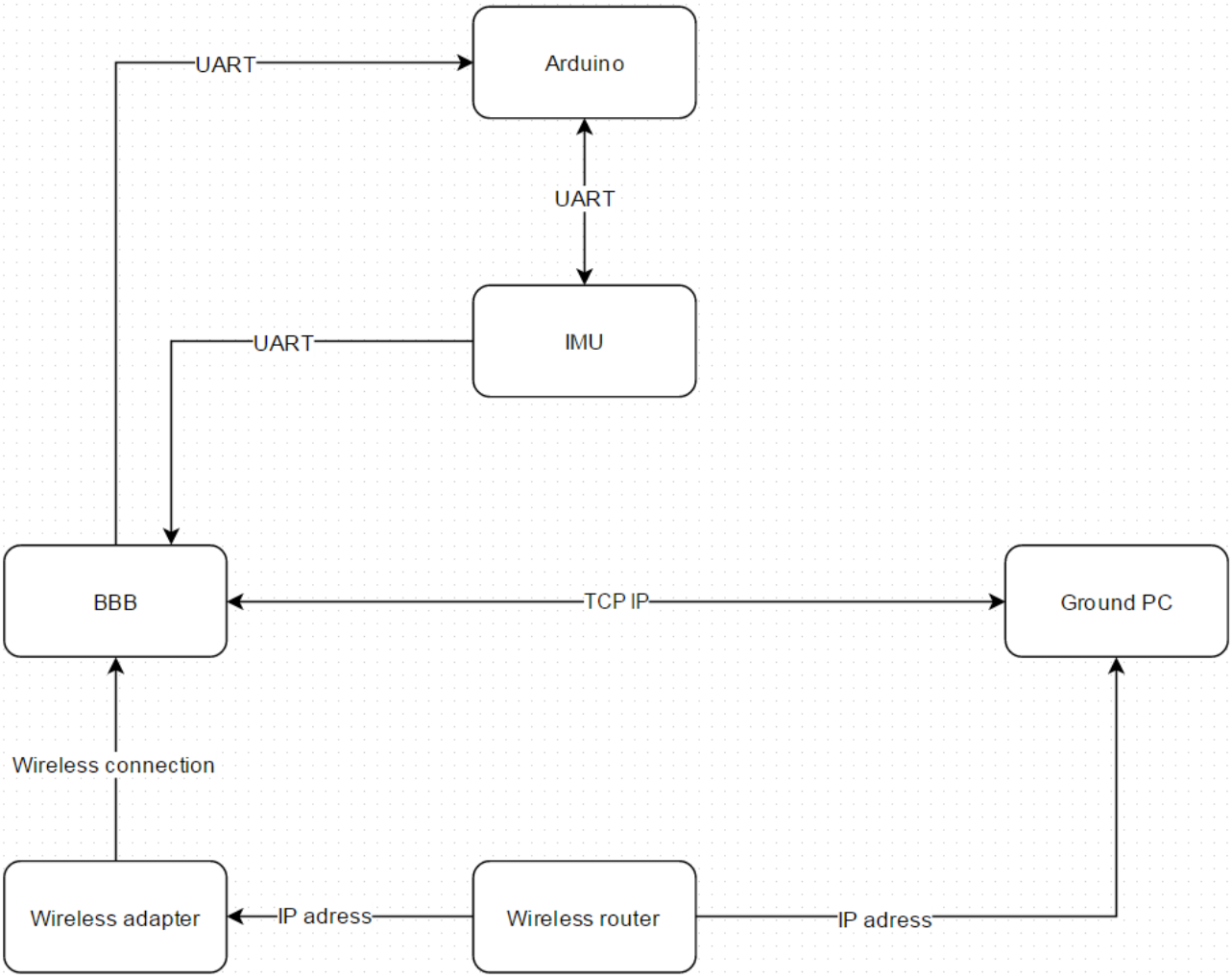

Figure 29: Current UAV communication architecture

The current architecture shows a communication between the Arduino and the BBB. This communication is done according to the UART interface in order to send steering control messages to the PID controller located in the arduino. The UART line is also used to fetch data from the IMU continuously. Furthermore, another UART line is connected to the arduino. The reason for the double IMU connection to both the BBB and the arduino is to reduce the processing power needed from the arduino. Using this UART connection, the arduino can recieve data from the IMU and send configuration commands to the IMU. The TCP/IP connection between the BBB and ground PC is used for sending manual steering commands from the PC to the BBB and receiving IMU data sent from the BBB.

The ground PC program which is being run to fetch IMU data and send manual commands is called UAVdata and the BBB program which is run as a server on the BBB is simply called BBBprogram.

9.2 Requirements and limitations

Some requirements exist for this project to be run, this section describes these requirements as well as limitations to the current product.

9.2.1 Requirements

UAVdata is required for the user to clearly see what is happening with the UAV, and for this UAVdata will display the IMU data. The most recent data must be displayed in text, so UAVdata is required to continuously gather data from the IMU as the program runs. The program must also be able to show charts of the data gathered from the IMU to clearly display how the UAV has been moving. For these functionalities to work UAVdata needs to be able to receive messages in some kind of way, to be able to access the data from the IMU. UAVdata also has the requirement to be able to control the UAV manually with the keyboard, so for this it must be able to send messages to the UAV.
A few things are needed to be configured for the BBB to work:

- Ubuntu 14.04 arm installed on the BBB
- Wifi dongle installed on the ubuntu
- UART TTY-PORT 0 and UART TTY-PORT 1 opened on the BBB

When these bullets are configured, the IMU TX (the white marked cable pin) should be connected to UART2_RX(1) on the beaglebone (BBB pinout can be found at [11])UART1_TX should also be connected to the arduino UART2 RX. Lastly, the wifi dongle should be connected to the BBB USB output.

9.2.2 Limitations

The server program faces some limitations regarding the connection to the UAVdata program. Firstly, there can only be one connection opened at the time, meaning the server only accepts one client connection and cannot be used on multiple ground PC's. This can however be fixed if using an array of connections instead of just one connection. Another limitation is that the UAVdata program should be closed first. When this procedure is done correctly, the server program will close it's connection to the port and the program can be restarted again. If this is not done, the UAVdata has to wait for the router to reset the port in order to connect to the BBBprogram, this may take several minutes.

9.3 Design and interface

The programs are designed so that all of the work is done on the actual UAV in the Beaglebone Black and Arduino. They then send the necessary information over TCP/IP to the ground computer, allowing users to view what is happening. Figure 30 shows how the graphical user interface looks like on the ground computer.

Once BBBprogram receives the data from the IMU in a text string it will send the information over TCP/IP to the ground computer. The ground computer then looks for a 'V', one of the first characters found in the string sent from BBBprogram, and once it finds the character it will read another 159 characters to get the full message. This is done to allow for different kinds of messages, and the 159 extra characters are there because the string is simply that long.

To make sure that the messages sent from the BBBprogram are not corrupt in some kind of way, there is a CRC check. It takes the string from the IMU and calculates its CRC, and through that it compares the value with the CRC sent from the IMU by the end of the string.

[11] http://pix.cs.olemiss.edu/csci581/BBBlackGPIOMap.png

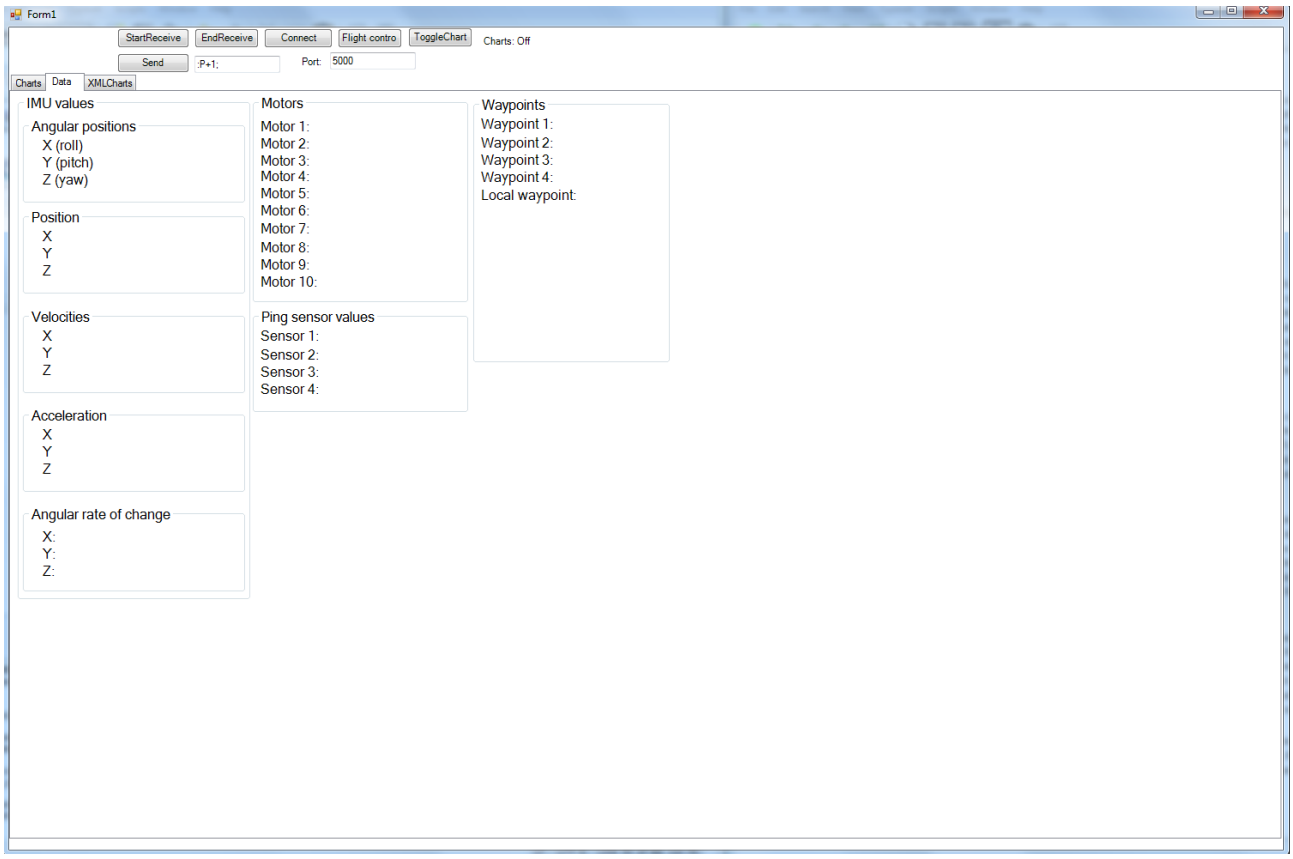

Figure 30: Showing how the GUI of the UAVdata program looks like.

9.4 Method

A review was made of ZigBee, Bluetooth and Wi-Fi in order to find a suitable way to communicate between the GUI, in this case UAVdata, and the server on the UAV. As the main upside of ZigBee and Bluetooth over Wi-Fi is their lower battery consumption, while their downside is that it is not quite as widely available on all devices, Wi-Fi was decided since it is usually always available on any laptop.

UAVdata was created with Windows Forms and uses a lot of functionalities from this, such as buttons and graphs, to give the user a graphical user interface to work with. The data gathered from the IMU by the other devices is sent from the Beaglebone Black to UAVdata, where it is saved in a list of strings and displayed in labels. The list of strings can be saved to an XML document. These XML documents can be loaded into UAVdata to display all the data in charts, as seen in figure 31.

There are two different ways to get data from the IMU. It either automatically writes the data over and over and a user simply has to read it all, or it can be requested. In this case the Arduino requests the data by sending a message to the IMU, then reads it and sends it on. The data is requested because it should always be the latest available data. Otherwise the Arduino could possibly get an overflow in its buffer, or end up sending old data that is no longer useful. The IMU and Arduino should both be able to handle a baudrate of around 1000000, however the highest rate that has been successful without corrupt data was 230400. A manual for the IMU can be found at the Vectornav website [12].

The BBBprogram located on the Beaglebone Black handles all the messages that are sent between the devices, as it has access to both the Arduino, through serial communication, and the UAVdata, through TCP/IP. Messages sent over the TCP/IP from UAVdata are forwarded from the Beaglebone Black to the Arduino, allowing the UAVdata program to control the UAV. Before any message from

[12] http://www.vectornav.com/products/vn200-rugged

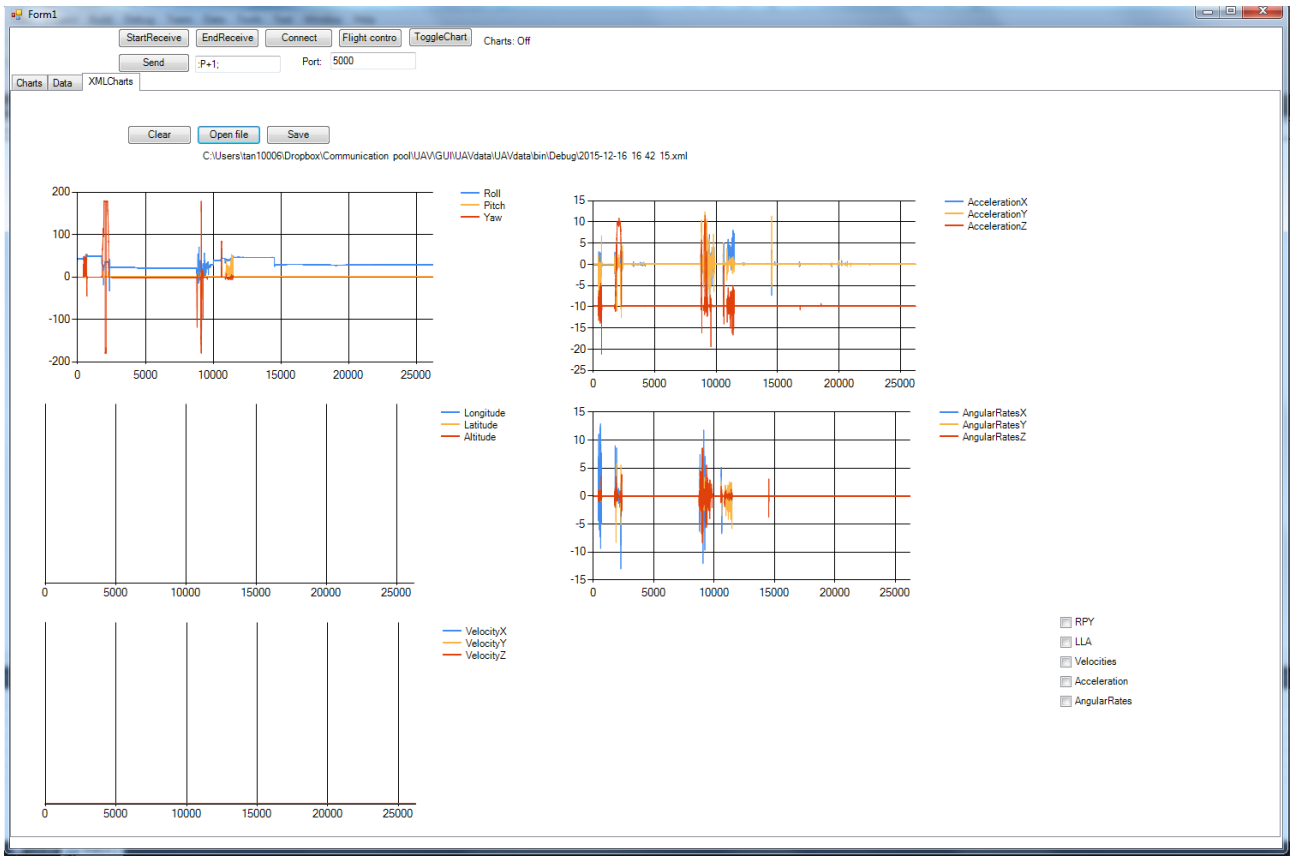

Figure 31: Charts in UAVdata showing some data gathered from the IMU. Velocities and longtitude, latitude and altitude were not connected at the recording.

the BBBprogram is sent on over the TCP/IP it will do a CRC check to make sure the data has not been corrupted.

9.4.1 Program setup

The program setup currently used configures the Beaglebone black as a server connection and the UAVdata as a client connection. This mean, the BBBprogram uses sockets configured as a server while UAVdata creates client sockets. Since the original thought was to have the router on-board the UAV plane, this was the most logical setup, since this now has changed, the current setup can be debated. The BBBprogram does not terminate connections, instead a connection should be terminated from the UAVdata program. This allows the BBBprogram to close all currently opened sockets so that they can be re-used again as soon as possible, exceptions to this can happen if the user decides to close the BBBprogram before the UAVdata program. Then the user has to wait for the ports to be reset by the router before it is possible to connect to the BBBprogram again.

9.5 Test and simulation results

The IMU was tested a lot through the Arduino in order to find out how it worked. These tests consisted of simply sending messages to it and then reading the returned message received from the IMU. These returned messages state if an error occurred with the command sent to the IMU or if the command was run. Through these tests it was possible to find out that the two characters '\r' and '\n' must be added at the end of the string for the IMU to process it, and the Arduino's inbuilt function Serial.println() provided these automatically.

In order to make sure that UAVdata received the correct data from BBBprogram, the programs were simply run and BBBprogram sent the real IMU data to UAVdata. These tests sometimes resulted in the data sent from the BBBprogram being corrupt, with some characters being switched out

at random. This problem was solved by removing the TX cable connected in the Beaglebone Black.

The graphs in UAVdata were tested in order to see how stable they are in a long run. This showed that re-drawing the graph repeatedly is too slow in Windows Forms as the data shown in the graphs was quick to fall behind the actual data. If the program is only run for a short time then this is only a problem in the sense that the data is behind, however if the program runs for a longer time then eventually the program will crash. This crash happens because the buffers overflow with the data waiting to be inserted into the charts. It is recommended to simply save the data and then load it all in at once after a full reading if the data must be shown in graphs.

9.6 Using the system

This section will cover information about the dependencies, installation, configuration and execution of the programs. After reading this section, the reader should be able to run the programs without problems.

9.6.1 Dependencies

To be able to run UAVdata, .NET 4.0 Client Profile is required.

The BBB program needs GCC installed and all essential linux headers, as well as ubuntu arm.

9.6.2 Installation

The latest version of UAVdata has been compiled in both Microsoft Visual Studio 2010 and Microsoft Visual Studio 2013. As long as .NET 4.0 Client Profile is available it may work on other versions too, however these are the ones that have been tested.

9.6.3 Configuration

For the UAVdata program to properly connect to the Beaglebone Black it must first consider the IP address of the Beaglebone Black. If the IP address does not work with the current values, the ipAdress variable found in networks.cs can be changed to the correct address. The current IP address of the device is 192.168.0.104, but could change if another router is used, this can be noted if it is no longer possible to ssh into 192.168.0.104. The beaglebone is currently configured to automatically connect to the network ROARy with the password storstark. This needs to be changed if another router is used and can be changed in /etc/network, but has to be done by USB. The BBB also uses the TP-link dongle to connect to the router, changing dongle can be done, but will be very problematic, as it took some extensive work to get the TP-link drivers to work for the BBB.

9.6.4 Execution

UAVdata can be run at any time, but the BBBprogram must be on and the PC and the Beaglebone Black must both be connected to the same router for UAVdata to be able to do anything useful. To run the UAVdata program, all that is needed is to simply run the UAVdata.exe file found in the bin/release folder. To run the BBBprogram the user must first navigate from the default folder to software/Server then run the command ./BBBprogram in the terminal.

9.7 Future work

If the UAV is to be flown a longer distance it would definitely be beneficial to put a router inside of it, so that only the ground PC has to be moved along behind it instead of also a router. It could be helpful to add support for sending messages from the Arduino to the ground PC so that it is possible to for example send messages about where the current waypoint is, how far away the ground is according to the ping sensor and similar things.

Currently, all checksum and error handling functions are disabled. This was due to a short amount of time to test them in a real environment, these should be enabled again and tested. More connections could also be allowed to the BBB, so that it is possible to run the UAV from more computers than one and the server should also be able to handle if one client disconnects, right now it kills the program. Lastly, the arduino to BBB communication has to be refined, the UART connection to arduino generates many checksum errors, this should be investigated why and fixed.

Finally, the flight control could be altered in to a more safe way. By connecting digital pins from the beaglebone to interrupt driven pins on the arduino, it would be possible to send manual commands in real-time, this would also prevent any risk of package losses.

9.8 Running the programs

To run the program first of all a user must power up the Arduino and Beaglebone Black. The code running on the Arduino will then run automatically, but the code on the Beaglebone Black must be started manually. This can be done by first using SSH to connect to the Beaglebone Black through the Linux terminal, using the following command:

```
ssh ubuntu@192.168.0.104
```

One must however remember that the IP address for the Beaglebone Black may change if another router is used, so it is not guaranteed to be the same. Simply change the IP address in the above command to the correct one. Next step is to navigate to the correct folder. This can be done from the default folder with the following command:

```
cd software/Server
```

From this folder it is possible to run the program. However, if some changes have been made it must first compile. This is done with the following two commands:

```
make clean
```

```
make
```

Then the newest version of the program will be run with the command:

```
./BBBprogram
```

The UAVdata program can be started before or after BBBprogram, this is done by running the file UAVdata.exe found in the /bin/Release folder, navigated from the project. The timing importance comes now where the 'Connect' button must be pressed after the BBBprogram is already running. Once the ground PC has been connected to the BBBprogram it is possible to press 'StartReceive'. StartReceive will make the program start looking for data sent from the BBBprogram, this data can be seen under the 'Data' tab, or in the Charts tab if 'ToggleCharts' is on. The program can be slowed down if ToggleCharts is on while it is reading IMU data, so if this is a problem then it is possible to get the data into charts by saving it in the 'XMLCharts' tab and then loading it. It should be noted that closing UAVdata with the cross in the top right corner will automatically save the data that has been gathered so far. These saves are found in the default folder where 'Open file' starts looking once it is pressed, all named after what date and time they were saved and they are all .xml files.

To manually fly the UAV the UAVdata program must connect to the Beaglebone Black as mentioned above and the Beaglebone Black must be connected to the Arduino. Then press the 'Flight control' button, this will open up a new window. With the new window in focus, pressing the prompted keys will send messages to the Beaglebone Black that are forwarded to the Arduino, and these will control the UAV manually.

10 Path planner

The Path planner is intended to guide the UAV around objects to a given target position. In its current state the planner can assign waypoints and crudely simulate following them to the goal in Matlab. The Matlab simulation also contains basic obstacle detection and dynamic mapping of these obstacles. The planner never got further than the Matlab simulator.

10.1 Requirements and limitations

The planner should be capable of guiding the UAV from point A to point B in a 2D plane without colliding with any obstacles. It must work indoors, but only in a small area. It was decided to only use a 2D plane for this first iteration, meaning that if the shortest path is above the object it would still go around it.

10.2 Algorithm

Find all corners around objects in the map. Current and target positions are also considered corners. If the target position is outside the map, a waypoint is located, on the edge of the map as close as possible to a line running from the center of the map (our position) and the target.

Check which corners can be connect with straight lines without crossing an object. Do a shortest path node search among the corners and their available edges between the start and end positions. Dijkstra's is used in this version.

As the robot moves around, the map is updated to keep the robot within the central pixel of the map in relation to the world. So everything in the map, and the list of corners, is moved in the opposite direction to the direction of travel. The map does not rotate even if the robot does.

Every time an object is detected by the robot the map is checked to see if we've seen it before. If we have, we do nothing. But if we have not, we stop and start the process over.

Corner
A Point is considered a corner if (this Point is always spelled with a capital P to avoid confusion):

- It is itself unblocked.
- A different point two pixels away diagonally from the Point is blocked (now referred to as the blocked point).
- The point between the Point and the blocked point (now referred to as middle point) is unblocked.
- The two points adjacent to both the middle and the blocked points are both unblocked.

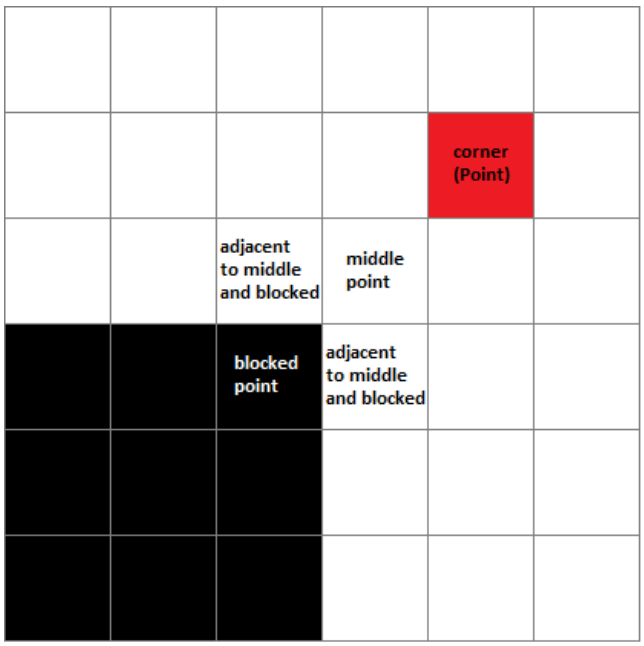

Figure 32: Corners in the path planner.

10.3 Using the system

The simulator is run using the 'setup.m' file. The setup handles which simulated world to use, referred to as 'course', as well as settings for the planner itself such as size of the dynamic obstacle map. The course can be any 128x128 pixel image where black is an obstacle and white is traversable space. If the edge around the course is not black however the the robot may try to go outside the world because the planner will not recognize the edge on its own. The setup also handles the initial position of the robot by asking for it when started, and a new target whenever the position has been reached.

10.4 Future work

This is a work in progress and much can be done here. The path planner can be extended to work in a 3D environment as this is much more suitable for an aircraft. A navigation module should be implemented that converts the next way-point from the path planner into pitch, roll and yaw commands for the control system.

11 Results

The final prototype has a total weight of approximately 7.5 kg and can be seen in figure 33. The total flight time should according to calculations be approximately 13 minutes at 50% throttle with the current battery fully charged assuming it is in good condition, the actual exact flight time has not been tested.

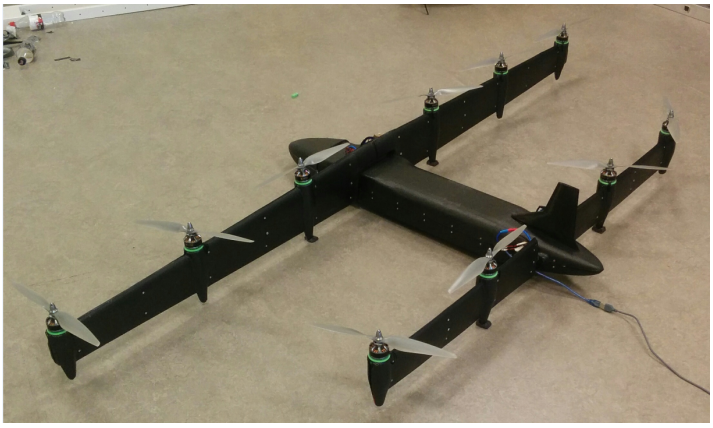

Figure 33: Final prototype.

11.1 Final testing

The aircraft was once again mounted on the test rig to start the testing. The PID controllers were re-tuned until a satisfactory control was achieved. The manual controls were also tested on the rig.

Due to the limited space available on the university, only limited free flight tests were performed. The throttle was gradually increased until the aircraft hovered, but the project manager was always holding the aircraft by its nose. It was deemed to risky to let go of the aircraft as the room was too small and there would be very little time to adjust it in case it started to drift. It was also kept just above the ground. A still image from one of the tests can be seen in figure 34

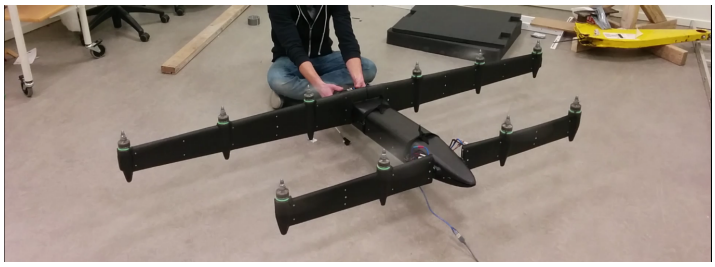

Figure 34: A still image from a test video.

The attached USB cable does not transfer any signals, everything is done over wi-fi. It powers the arduino and is simply one of more precautions to stop the aircraft should something go wrong. It can be powered internally instead.

A graph over the pitch, roll and yaw from one of these tests can be seen in figure 35. The reference values are 0 for pitch, 0 for roll, and 4 for yaw.

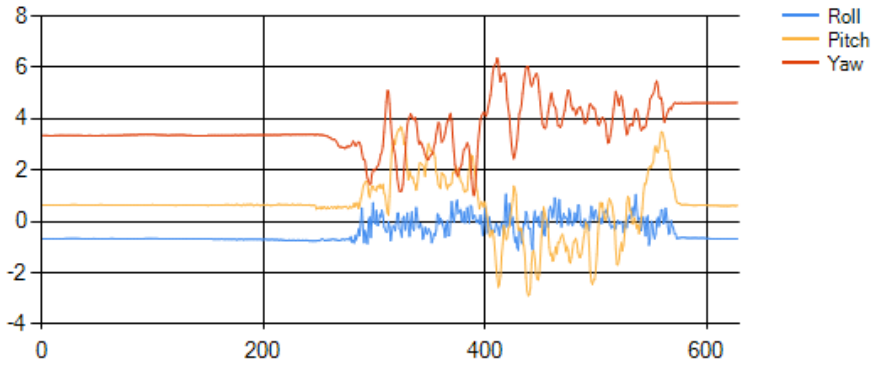

Figure 35: Graph over pitch, roll and yaw in degrees from one of the flight tests. Lift off takes place at ≈ 300 in the time line.

The reason the pitch is somewhat noisy is probably because the project manager is affecting it as he is holding the aircraft in its nose, this was not the case during rig testing where it was stable. The yaw is also noisy, this is due to the fact that the yaw controller is not great, note however that a deviation of approximately 4 degrees is not that much considering a whole spin is 360 degrees. The roll is quite stable.

12 Conclusions

Due to the low priority of this project, few resources were available and all the initial goals were not reached. The aircraft has no autonomy and is basically a radio controlled vehicle. It does however work and we do feel that we have created a solid base for future generations to build upon. The aircraft performs well in the tests but more extensive testing may be required. The main problem is the yaw controller and it may be advisable to adjust it to work as mentioned in 5.6. Due to the size of the vehicle testing is not a simple task and does involve some risks, especially in the small spaces available at the university.

13 Future work

The next step would likely be to make the aircraft autonomous. To implement way point navigation based on GPS would be a relatively simple task. The downside is that the aircraft has to be outside and is affected by weather conditions and the risk of loosing control and flying away. By implementing vision, one could use a vision - inertial based navigation and fly inside. Another area where much work can be done is the pick-up system and communication with the Naiad. The wing tilting procedure and flight in horizontal mode can also be developed. This is however not a simple task and requires a great deal of complicated aerodynamics, and most likely a modification of the current aircraft.

14 Note for future students

During the last days of the project the Beagle bone black broke down for an unknown reason. It also broke all the ports it was connected to on the other components. This means that serial ports RX1 and RX2 on the arduino stopped working. The TTL TX on the IMU stopped working at any baudrate higher than 115200 bps. This was solved temporarily by using a private Rasberry Pi connected to Serial 0 on the arduino and and moving the IMU to serial 3. Since the IMU register containing GPS data takes to long to send at 115200 bps, we switched to a shorter register as only the pitch, roll, yaw and x,y rate of change is used at the moment anyway. This lead to a similar response time. Connecting hardware to serial 0 is however not advisable since that is the port the USB uses to communicate with a computer and the data may be mixed up.

To resume the project the following things should be done:

- A new BBB or a similar embedded computer needs to be purchased. The server code needs to be uploaded according to the instructions in section 9.6, and the wifi needs to be configured.
- A new arduino due should be acquired to allow for serial communication on different ports.
- As long as no position data is required, the IMU can continue running a shorter register at 115200 bps. Should one need to increase the baudrate, the serial messages can either be converted to RS-232 messages since that port is still working fine, or a new IMU can be acquired.

References

[1] Avr221 discrete pid controller. http://www.atmel.com/images/doc2558.pdf. Accessed: 2016-01-10.

[2] Fuselage design, aerospace design and systems engineering 1. http://aerostudents.com/files /aerospaceDesignAndSystemsEngineeringElements1/ Summary/FuselageDesign.pdf. Accessed: 2016-01-09.

[3] D. W. Anderson and S. Eberhardt. *Understanding Flight*. McGraw-Hill, second edition, 2009.

[4] J. Anderson. *Introduction to Flight*. McGraw-Hill, 7th edition, 2011.

[5] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.

A Checklist and trouble-shooting for flight

- If the aircraft has not been used for a while start by connecting the arduino to the computer and run the program to make sure that IMU data is recieved. Open the serial monitor in the arduino IDE and make sure that values are received. Should all values be zero something is wrong, make sure all cables are connected properly.
- feel the motors and propellers and make sure they are properly screwed on.
- Connect the battery and place it close to the IMU to get the center of gravity in the correct place. Attach a battery sensor to warn for low voltage. Be careful with the cables of the battery, if one cable is connected and the other touches a non-isolated part of the PSU the battery may shortcut. The ground cable has a male connector and this should be connected first and disconnected last as it is more likely for this to touch the PSU in case it is dropped.
- Put to top half of the aircraft, put the signal and power cables for the front wing through the holes on the top half of the body and attach the bolts. Straps can also be used for testing as they are much quicker to remove and attach when the battery needs to be replaced. Connect the power and signal cables according the their markings. When the battery needs to be replaced it should be enough to just lift up the back of the top half to get access to it.
- Make sure the wings are in a vertical position, they are locked but can be twisted with some force. One way is to use a try-square tool and use the floor as a reference.
- With the current code the USB from the arduino must be connected to a PC, it needs a command to start running its loop and it can be stopped by pressing any key. This is not necessary and these lines can be commented out in the code. It does however provide some extra safety features. If a USB is not used it should be connected to the 12V output on the PSU.
- If the aircraft has not been used for a while it is a good idea to start the BBB program and run the full system without connecting power for the motors to make sure everything works.
- Turn on the power via the power switch, All the motors should beep once, meaning that they all are connected and are receiving a 0% throttle value. If any motor keeps beeping it is most likely not connected. First make sure that red is connected to red and black is connected to black on the external connectors of the wings. If these are correct a cable may have fallen out from the arduino inside the aircraft.
- Once everything is connected and on, clicking shift once from the manual controls inside the UAVdata program should turn on the motors. Once they are running make sure that the kill key 'k' is working. Pressing this should turn off the motors. Keep pressing shift to increase the throttle, at 40% the control system will turn on which will be noticed as a quick burst in motor power. at approximately 50% the aircraft should start to lift. It has so far never flown completely by itself as the space available for testing was to small. One person has always been holding it by its nose.
- Should the aircraft start to loose control, the first action should be to gradually decrease throttle to make it land. Should it be more of an emergency there are several ways to turn it off. Pressing 'k' from the manual commands should turn off the aircraft. If the USB is connected any key can be pressed in the serial monitor (assuming that code has not been commented out), or the USB cable can be pulled out. The hardware switch can be either switched off or the entire switch can be pulled off its connectors. All of these should be considered last resort as the aircraft will shut off completely and crash. It should be noted that this has so far never been necessary.
- Be cautious, safety goggles are recommended.
- Always remove the battery if the aircraft will be left unattended for a longer period of time. Over night etc.

B Checklist for rig test

- Follow the above instructions for connecting the aircraft.
- Place the aircraft on the rig with straps, try to attach the rig as close to center of gravity as possible. Place chairs under the wings and avoid that the joint is at its maximum position when the aircraft is at rest since it may brake the joint.
- The arduino code can be changed to apply a base throttle automatically, This line is prepared in the code and only needs to be un-commented. It is advisable to set the base throttle to 30 - 45%. If no base throttle is applied the aircraft will not control itself properly as it is only adding and removing values from 0 which is not sufficient to control the aircraft. If the base throttle is less than 41% the if statement to run the control system also has to be changed to a lower value. These tests are better to perform with the USB cable connected. The wi-fi does then not need to be connected unless one wants to "steer" the aircraft. The aircraft can be started and stopped from the serial monitor in the arduino IDE.
- One person should hold the aircraft in its zero position when starting the code. If it starts in any other position the controller will make heavy adjustments to get it to zero. This may work fine but it is not advisable.
- When shutting off the motors make sure that one person is there to catch the aircraft. Otherwise it will fall to the bottom position of the joint and this will likely brake the joint.
- Be cautious, safety goggles are recommended.

C Mechanical assembly instructions

To put the aircraft together is quite straight forward and should be fairly self explanatory by looking at the bolts and nuts. A few things should be considered.

- If the wings have been completely taken apart, the bolts and nuts should be attached loosely. You should then put the wings on the carbon fiber rods and start by attaching the m4 bolts that connect the wings to the rod. Then the remaining bolts and nuts can be tightened. This is because the m3 bolts give the wing some wiggle room and if these are tightened first the hole inside the wing may not align.
- The rod runs through the first motor nacelle. Sometimes the speed controller inside that nacelle moves and end up in the way of the rod. If this happens the bolts from the nacelle and towards to base of the wing can be loosened up more or taken off completely and the wing can be pulled apart a little bit so the rod can slide through and then push the speed controller out of the way. As an alternative the wing can be taken apart and the tape holding the speed controller in place can be replaced.
- When attaching the top half of the body, make sure that the holes are aligned by pulling and squeezing the body a bit. Otherwise the nuts that are glued on the inside may fall off and have to be re-glued.

D Contact info

- **Project manager:**
  - Joakim Kareliusson, jkn11013@student.mdh.se
- **Communication:**
  - Jacob Danielsson, jdn11003@student.mdh.se
  - Tobias Andersson, tan10006@student.mdh.se
- **Electronics:**
  - Emil Johansson, ejn11014@student.mdh.se
- **Path planning:**
  - Ludvig Langborg, llg11005@student.mdh.se