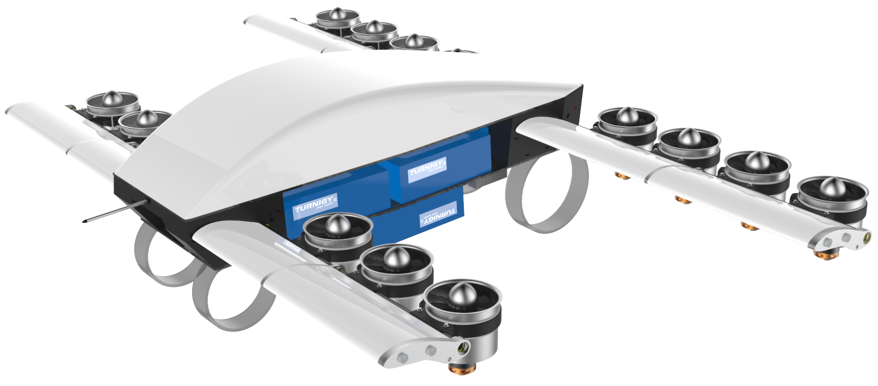
WAZP

Development of a flying personal vehicle



Axel Johansson, David Wallén, Fredrika Kringberg
Gustaf Ingvarsson, Hanna Dotevall, Jiyang Chen
Ludvig Bjärkeback, Magnus Valtersson, Tomas Fredriksson

MF2059, Mechatronics Advanced Course

December 22, 2017

Abstract

This project was part of the Mechatronics Advanced Course, at the Royal Institute of Technology (KTH) in Stockholm. The project group, consisting of nine people, worked with the project over three academic periods, from March 2017 to December 2017. The assignment was given by the sponsor Anton Wass, an entrepreneur with a mind for technology and innovation. The aim of the project was to challenge and improve the means of human transport used today. With already existing technologies, a vehicle that can travel in a superior way compared to todays alternatives should be developed. A design of a flying personal vehicle with the capability of vertical take-off(VTOL) and subsequently transition to a horizontal long-range flight mode, and a reverse process for the landing procedure, was developed. The project was limited to designing and building a scaled model version of this vehicle.

The scaled model was built with a canard wing configuration and a motor tilt system that ensured the VTOL property. Several test cases proved the functionality of the different subsystems that the vehicle consists of, but due to large forces from vibrations and faulty specifications of the purchased components, the model was not capable of stable hovering and forward flight.

This report was conducted during the second and third period of the course and contains a summary of the research done during the spring followed by details of the development done during the latter periods of the course. The report also describes the final product and what could be improved for future work.

Acknowledgements

We would like to thank Anton Wass and Robin Lilja for giving us the opportunity to work with this exciting project. We would like to show our gratitude to Anton for giving us financial support and continuously believing in the team, and to Robin for providing us with good constructive feedback on our technical solutions.
We would also like to thank our supervisor for the first period Björn Möller, our supervisor for the rest of the project Naveen Mohan for helping us with providing a good working environment for us and guiding the team when needed.
We also want to thank the Mechatronics HK teaching team for their continued input throughout the project.
Additionally, we would like to thank Per Wennhage for helping us with the aerodynamics and taking the time to discuss ideas with us.

Contents

1 Introduction **1**
1.1 Pre-study & Background . . . 1
1.2 Scope . . . 1
1.3 Requirements . . . 2
1.3.1 Broad requirements . . . 2
1.3.2 Construction specific requirements . . . 2
1.3.3 Electronics specific requirements . . . 2
1.3.4 Software specific requirements . . . 2

2 Method and Design Process **3**
2.1 Project process . . . 3
2.2 Design and Development Method . . . 3
2.2.1 Mechanical and Electronic Hardware . . . 3
2.2.2 Software . . . 4
2.3 Verification and Validation . . . 4

3 System Design **5**
3.1 Mechanical Hardware . . . 5
3.1.1 Body . . . 5
3.1.2 Wings . . . 6
3.1.3 Motors . . . 6
3.1.4 Motor Tilt System . . . 7
3.2 Electronics Hardware . . . 9
3.2.1 Routing Design . . . 9
3.2.2 Motors, EDFs and ESCs . . . 9
3.2.3 Motor Tilt System . . . 9
3.2.4 Flight Controller . . . 10
3.2.5 Batteries . . . 11
3.3 Simulation . . . 11
3.3.1 Vehicle Dynamics . . . 11
3.3.2 Tilting controller . . . 12
3.4 Software . . . 14
3.4.1 Flight Controller . . . 14
3.4.2 Motor Tilt Controller . . . 15
3.4.3 Communication interface . . . 15

4 Results **17**
4.1 Requirements fulfillment . . . 17
4.2 Mechanical Results . . . 17
4.3 Software Results . . . 18
4.3.1 Simulation . . . 18
4.3.2 Flight Controller . . . 18
4.3.3 Motor Tilt System . . . 19

5 Discussion **21**
5.1 Development Process . . . 21
5.2 Results . . . 21
5.2.1 Requirements fulfillment . . . 21
5.2.2 Mechanical Results . . . 21
5.2.3 Software . . . 22
5.3 Future Work . . . 22
5.4 Conclusion . . . 23

A Pre-study and SOTA Report **1**

B Test Cases **1**
B.1 Test Case 1: Arduino-wing subsystem 2
B.2 Test Case 2: Motor, battery and ESC Benchmark 3
B.3 Test Case 3: Constrained Hover 5
B.4 Test Case 4: Performance of different types of ESCs 7
B.5 Test Case 5: EMI testing 9
B.6 Test Case 6: Functionality of fron right wing 11
B.7 Test Case 6: Functionality of fron right wing 12
B.8 Test Case 7: Startup and shutdown procedure 13
B.9 Test Case 8: Stationary Tilting 15
B.10 Test Case 9: Stress-test of motor setup 17
B.11 Test Case 10: Torque testing on Servo motors 19
B.12 Test Case 11: Wing Strength 20
B.13 Test Case 12: Vertical Take-Off and Landing 21
B.14 Test Case 13: Balancing in Roll Axis 23

C System Design Overview **1**

D Results **1**

E Flight Controller Code **1**

F Motor Tilt System code **1**

G Weight Budget **1**

Nomenclature

Abbreviations

CAD	Computer Aided Design
DC	Direct Current
EDF	Electric Ducted Fan
ESC	Electronic Speed Controller
BEC	Battery Eliminator Circuit
EMI	Electromagnetic Interference
EKF	Extended Kalman Filter
ESC	Electric Speed Controller
GPS	Global Positioning System
IMU	Inertial Measurement Unit
MATLAB	Matrix Laboratory
MEMS	Micro-Electro-Mechanical Systems
MTS	Motor Tilt System
NASA	National Aeronautics and Space Administration
PID	Proportional, Integral, Derivative
PWM	Pulse Width Modulation
RC	Radio Control
SOTA	State Of The Art
VTOL	Vertical Take Off and Landing
MIMO	Multiple Input Multiple Output

Symbols

A Symbol	Symbol [SI-unit]
A	Wing area [m^2]
AR	Aspect ratio [-]
c	Wing chord [m]
c_d	Specific drag coefficient of the aircraft [-]
c_{d0}	Zero-lift drag coefficient of the aircraft [-]
c_{di}	Lift-induced drag coefficient [-]
c_{dl}	Specific lift coefficient of the aircraft [-]
D	Drag force [N]
e	Efficiency factor of the aircraft [-]
g	Gravitational acceleration [m/s^2]
L	Lift force [N]
m	Mass of the aircraft [kg]
S	Wing span [m]
v	Velocity of the aircraft [m/s]
$\dot{x}$	Velocity in horizontal direction [m/s]
α	Angle of attack [°]
π	Circumference-to-diameter ratio of a circle [-]
ρ	Density of air [kg/m^3]

1 Introduction

One century ago, the car replaced the horse carriage as the most common method for personal transportation. However, with the current technological developments in several fields, it is becoming increasingly possibly to replace the car with a personal aircraft. Batteries, motors and software has gone down in scale and increased in efficiency, bringing the idea of a small personal aircraft that replaces the car closer to reality. The benefits of such a revolution would have a large impact on personal commuting, reducing travel-time and infrastructural need. To become the next step in the evolution of personal transportation, two things are required. First the maneuverability of a multicopter, for travel in urban areas, and secondly the efficiency of a small aircraft, for longer and faster commutes.

1.1 Pre-study & Background

To develop this personal transport a pre-study was conducted, for which the following areas of technologies were identified as crucial, listed below:

- Aerodynamics
- Wing-profile types
- Hardware (Motors, material and electronics)
- Control system

The aerodynamics and flight properties of the vehicle heavily depends on the design choices of the construction, but also have the possibility to be extended for commercial usage. Additionally, the design of the wings will affect how the vehicle performs in terms of efficiency and flight-time. The hardware was studied to select optimal configuration for our proposed design, which included looking at different flight controllers, micro-controllers, motors and batteries. The control theory is important to ensure that the vehicle can perform a smooth transition from hovering to forward flight, that could be implemented in a personal flying vehicle. Of course, in addition to this have a stable control system for both flight modes. The entirety of the state-of-the-art research conducted in the spring can be found in Appendix A

1.2 Scope

The aim of this project was to generate a proof of concept for an airborne personal vehicle. It should be able to replace the car as a mean of personal conveyance and therefore, an initial set of requirements were given from the stakeholder:

- The vehicle must be able to carry two people.
- The maximum size is 3x6m.
- The vehicle must have an airtime of 1 h.
- The vehicle must have VTOL
- The vehicle must be electric and battery driven.
- The cruising speed should be 150 - 200 km/h.

This, however, is a broad-ranged project and some further limitations are necessary. Since this is a project in mechatronics, it will focus on the control system that maneuvers the vehicle in hover and level flight, and the transition between the two. The scope of the project only includes as much aerodynamics as necessary and simplifications were allowed in airfoil design and airflow parameters.
To evaluate the concept, a scaled prototype was built, which has the necessary functions to demonstrate the aforementioned control system.
Furthermore, neither the cost of manufacturing the full scaled vehicle nor laws and legislations regarding flying the vehicle will be considered in this project. Safety is not a focus for the project team but may be considered throughout the project.

1.3 Requirements

From the limitations given to the project team from the stakeholder, a list of boiled down requirements were created for the project. The requirements are divided into broad requirements, not specific to our solution. In addition, specific requirements, which were set after the design lock-in. These are divided into *construction*, *electronics* and *software* requirements.

1.3.1 Broad requirements

REQ1.1 The vehicle shall be fully electrical.

REQ1.2 The vehicle shall be able to take off vertically.

REQ1.3 The vehicle shall provide lift in forward flight.

REQ1.4 The vehicle shall be safe for bystanders when performing VTOL.

REQ1.5 The vehicle shall be more efficient in forward flight than a quad copter.

REQ1.6 The vehicle shall reach a flight speed of 200km/h.

REQ1.7 The project shall not extend 50 000 SEK.

REQ1.8 The project shall be completed before 22 December 2017.

1.3.2 Construction specific requirements

REQ2.1 The wings shall support the weight of the vehicle in flight.

REQ2.2 The tilt mechanism shall withstands the torque generated by the motors at full thrust.

REQ2.3 The vehicle footprint shall not exceed 0.6 x 1.2 meters.

REQ2.4 The vehicle shall have a thrust-to-weight ratio of 1.5:1.

1.3.3 Electronics specific requirements

REQ3.1 The motors shall provide thrust 1.5 times higher than the weight.

REQ3.2 There shall be more than 1 motor per wing for redundancy.

REQ3.3 The batteries shall provide enough voltage and current to the motors.

REQ3.4 The batteries shall be rechargeable.

REQ3.5 The power management circuit shall withstand the current requirement.

REQ3.6 The control circuit shall be protected from EMI.

1.3.4 Software specific requirements

REQ4.1 The software shall include a self-stabilizing system.

REQ4.2 The flight controller shall control tilt and flight.

2 Method and Design Process

This section will present how the project team has approached the project and describe the methods and design processes that have been used during the development. Additionally, it describes the verification and validation process for the project.

2.1 Project process

The development has been done iteratively using Scrum with a Scrum Leader and Project Leader with one week sprints. The team initially has been divided into four major work-teams, consisting of *Construction*, *Controlling*, *Motors* and *Power Management*. The project plan has continuously been updated every week and a new set of tasks have been created and prioritized from the previous weeks performance. During the later phases of the project the team however divided into three new work-teams, *Construction*, *Controlling* and *Architecture*:

- Construction, which includes construction of the aircraft, body, wings and assemble of all the hardware, including electronics, wiring, batteries and cabling.
- Controlling, which includes simulation and designing the control system for the aircraft.
- Architecture, which includes the software framework for communication between the different systems, such as the flight controller, motor tilt controller, etc.

Weekly meetings with supervisor has been guaranteed to evaluate the team progress of the prior week. Besides, another focus for the process has also been to make sure everybody in the team would be involved in a broad aspect of the design process, which meant that knowledge transfer is an important part of the project.

2.2 Design and Development Method

The design methodology was differed for the each part of the project. The differences are mainly between hardware and software development, which are described below.

2.2.1 Mechanical and Electronic Hardware

The mechanical components were mainly developed through iterations of requirements creation and breakdown, design, prototyping and testing. If the testing had satisfactory results, the part would be considered complete and that version would be locked in for production. If the test was not accepted, the process would be repeated with modified features and recorded with an increment in version number.

The method described above was suitable for cases when components had a shorter development time and could be tested and verified early in the process. Since the project was rather complex there were also components that could not be manufactured in-house and needed to be ordered (either at a large cost or with a long lead time). For these components the process was different, such components spent a longer time in the design phase going through iterations of computer aided design and simulation. Once the design team agreed that the component fulfilled the requirements it was ordered (or brought up to the entire team for approval for the biggest of parts). This approach bears more resemblance to the Waterfall model [10].

When designing the vehicle the scale was selected first. With the scale and a rough avionics design selected a weight budget was made which can be seen in Appendix G. The weight budget contained the weight of the heaviest parts that were needed in order to satisfy the requirements and with that knowledge motors with sufficient thrust could be chosen. After the motors were chosen the batteries and electric speed controllers (ESC's) could be selected. The speed controllers needed to be able to provide enough current to the motors, and the main requirements on the batteries were that they should have a high discharge rate (called C value) and should allow a hovering time of approximately 3 minutes, which was deemed enough to conduct proper testing.

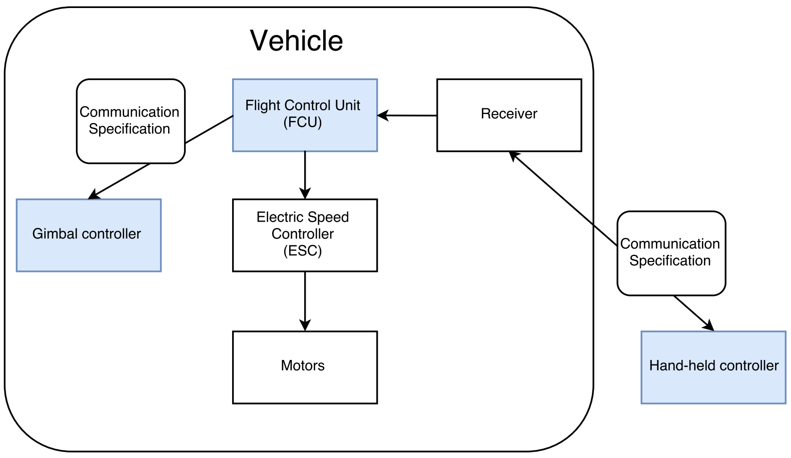

Vehicle
Communication Specification
Flight Control Unit (FCU)
Receiver
Gimbal controller
Electric Speed Controller (ESC)
Communication Specification
Motors
Hand-held controller

Figure 1: Communication between software modules

2.2.2 Software

The software in this project consisted mainly of two different modules. One module was the controller that handles the tilting of the motors, the second module was the main flight controller that handles stabilization of the vehicle and the third and smallest module was the hand-held controller which gives user input to the system. The different modules had to communicate with each other and since each module was developed by a different member in the team there was a need for constant communication in the team as the development progressed. For each connection between different parts of the software where communication was required a *Communication Specification* was created describing the message types that should be sent, how they should be structured, terminated and what the underlying functionality was. The parts where software was written or custom configuration was set up are highlighted in blue in Figure 1.

The purpose of the communication specifications were to create a common view on how the different parts of the software should interact with each and document it properly so that each team-mate could look up and develop the functionality without disturbing each other and to share the knowledge within the team on how the system worked.

For the more complex software development, mainly the Flight Controller (FC) and Motor Tilt Controller, the development method consisted of first defining requirements, breaking them down into features and functions, and then implementing and testing those functions. The chosen method for the software development was Scrum, where the features to develop were written down on a white-board and subsequently assigned and developed. After a large-enough number of functions had been developed and tested on their own, an integration was performed where the system was tested together with the other parts of the system.

The hand-held controller did not contain any software but rather custom configuration that decided what signal ranges to be sent on what channels, this also followed the communication specification documents. All software was developed using git as version control.

2.3 Verification and Validation

To ensure that the requirements for the project were fulfilled a set of test cases were formulated. These test cases can be found in full in Appendix B. A standardized test case document was created by the project team in order to maintain the same standard throughout the project. The methodology for validation and verification was applied from knowledge acquired earlier in the course with some adjustments for our application. The tests were formulated to verify and test smaller subsystems instead of the final product. By using this methodology we could configure and validate the subsystems in a more efficient manner and move forward with the development process. Some aspects of the design process were however not included in the verification process. This was due to time constraints for the project.

3 System Design

This section will present how the different systems within our product is designed, in regard to hardware, electronics, software and control system.

3.1 Mechanical Hardware

This part describes the mechanical hardware used in the project. In order to meet the requirements of the project the design of the vehicle was rather complex and different from any platform existing in the market, this resulted in that many of the parts had to be custom made. The main mechanical parts of interest that were made specifically for this project consisted of the Body, Wings, Motors and Motor Tilt System, as presented in Figure 2. The following chapters aim to detail what the function of the different components are, how they were designed and implemented. test

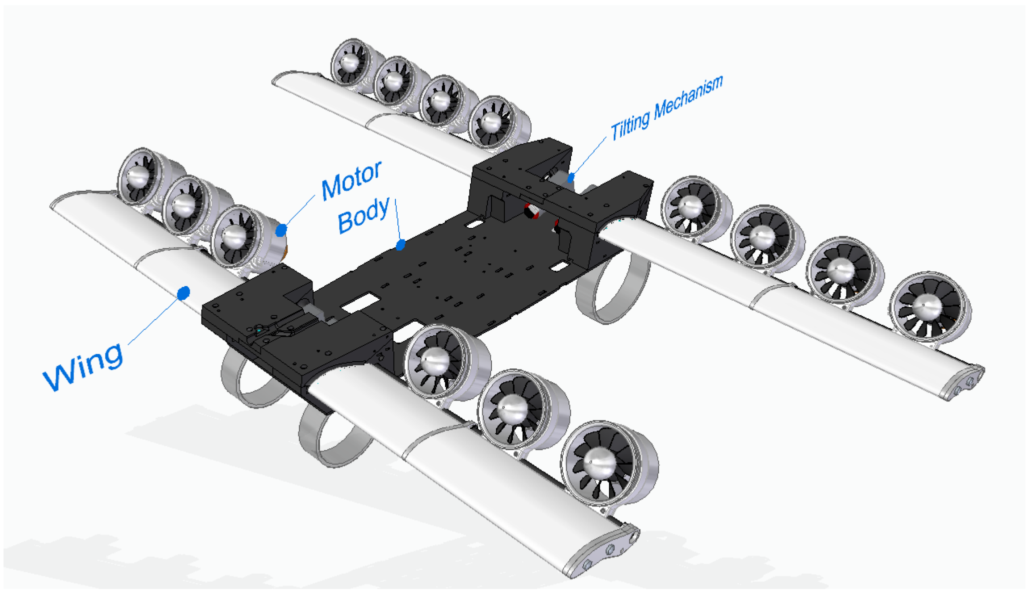


Figure 2: Main mechanical components.

3.1.1 Body

The total amount of motors that the vehicle is equipped with must be able to provide enough thrust to lift the vehicle. A general rule of thumb for multirotors and drones is that the maximum thrust to weight ratio should be two [11] in order to get lift while still obtain controllability of the vehicle. During the design process of the vehicles body were both choice of construction and material considered in order to meet the weight constraint. The vehicle was also designed and assembled in an attempt to make it easy to access the components when necessary and replace them if they broke. The main body of the vehicle is based on a sandwich model and consisted of two horizontal plates held together by two blocks in each end, see Figure 3. The vehicle was 3D-modeled in Solid Edge, from where CAD drawings were used for manufacturing parts. The design components of the vehicle required being custom made due to the complexity of the parts, the majority of the parts were therefore produced in-house. The two plates however required being lightweight and rigid in order to keep the weight down and reduce vibrations and was therefore manufactured by a company that could produce the plates in carbon fiber. Electrical components such as batteries, ESCs, flight controller etc are attached and mounted on both sides of the bottom plate. There are cutouts in the plates where wires can easily be pulled between the plates.

The blocks are vital for the construction and are of high level of detail. They are acting as a center piece for the wings and are mounted in the front and in the back of the plates. The blocks are also used for mounting the landing gear. Two of the three rods that each wing contains of goes through the front respectively back block in order to obtain wing stability. The third rod in each wing are attached to the motor tilt mechanism that is mounted inside each block. Each side of

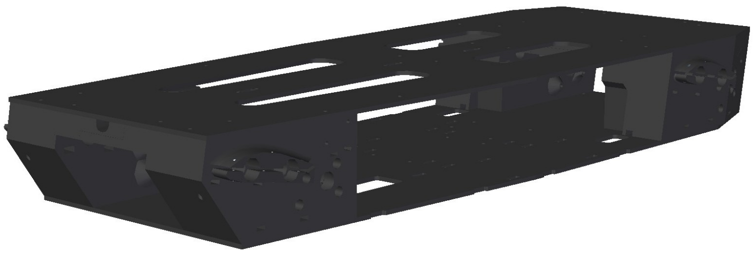


Figure 3: Sandwich model.

the block that are facing the wings have a slot for the custom made wings to fit into in order to achieve added stability.

3.1.2 Wings

Due to the limited size set by REQ2.3 a conventional wing structure was not an option because of the short wingspan. Conventional wing would need to be very thick, having a long chamber, resulting in a lot of drag. Instead a canard wing configuration was chosen and with its four wings can still use the same wingspan but a shorter chamber, resulting in a plane with better efficiency at high speed. A model of the wing design can be seen in Figure 4 below.

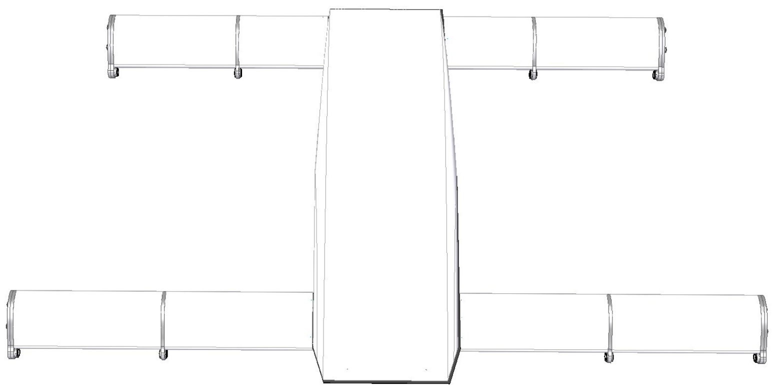


Figure 4: View of wing design.

The four wings also provided a natural structure to mount the motors on, positioning them at a distant from the center of mass resulting in a more effective pitch and roll control.
A wing profile was selected with low speed performance and a low stall speed in consideration. The selected Clark Y profile was modified to fit the carbon rods inside. The structure of the wings consisted of two parallel carbon rods that carry the weight, mounted on these was plastic segments with the wing profile. On top of these plastic segments a thick plastic wing skin was fastened to direct the airflow and help prevent torsion in the wing.

3.1.3 Motors

In order to achieve the VTOL functionality the motors need to be able to provide thrust in both vertical and horizontal directions and this was solved by tilting the motors. The movement limited the size of the motors and this resulted in using electric ducted fans (EDFs) to keep the dimensions of the motors low. This also allowed having multiple motors per wing, creating redundancy thereby achieving REQ3.2. Motorplacement can be seen in Figure 5 below. To create the high thrust with a smaller impeller, the EDFs runs at a high RPM, in this case 46 000 rpm, resulting in the motors being energy inefficient.

The motors were attached to the third carbon fiber rod of each wing. They were fastened by a plastic clamp, fitted to the rod, which in turn was clamped with two M3-screws to an aluminum holder. The two screws were tightened with sufficient force to keep the clamp from

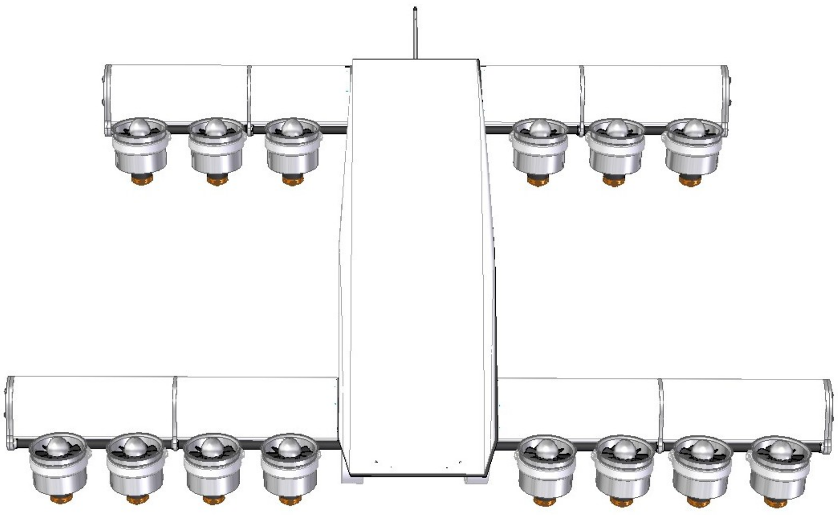


Figure 5: View of motor placement.

moving independently from carbon fiber rod. An exploded view of the motor holder can be seen in Figure 6 below.

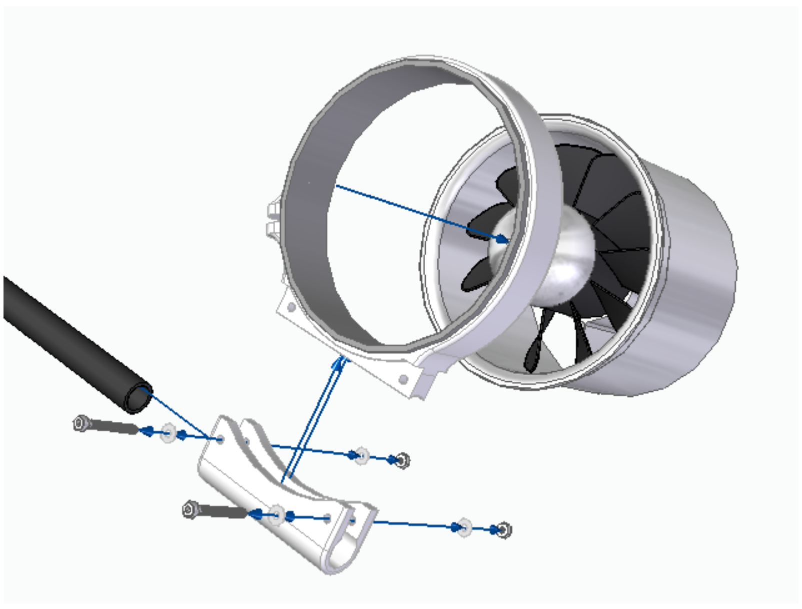


Figure 6: Exploded view of the motorholder.

3.1.4 Motor Tilt System

The tilting mechanism is designed to turn the motors around an axis, thereby allowing for adjustments of the angle of thrust. The vehicle will thereby be able to perform the sought after VTOL. The mechanism consists of a DC-motor, worm gear and shaft connector. In Figure 7 a model of the mechanism can be seen and where the parts are placed in the vehicle.

The servo-motor used was a regular 12V DC-motor, equipped with two hall encoders to sense rotation of the motor shaft. A worm gear with a ratio of 972 was used to increase the torque at the cost of rotational speed on the output shaft. The loss of speed was deemed acceptable as the needed transition time was 10 seconds at full speed, with this gear the transition would take 4 seconds. This type of gear was also beneficial to use as it was self locking, negating the need for constant torque from the servo-motor. Safety was increased as the self locking keeps the motors in place

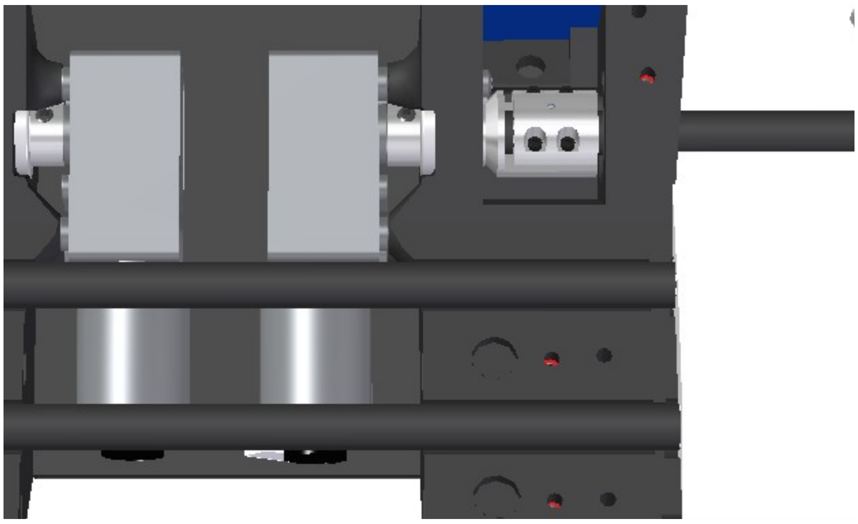


Figure 7: View of mechanism placement in the front.

should the servo-motors fail. After performing a torque test between two gear boxes with different ratios, both capable of delivering the needed torque, the gearbox with ratio 972 was chosen. The test results can be seen in Appendix B.10.

Gearboxes have the problem that they all have some amount of play, allowing for the output shaft to move very small amounts. This is caused by not enough teeth being engaged all the time. Manufacturing also affects this, as inaccuracies in the manufacturing cause faulty geometries. For the vehicle this meant that unwanted vibration was introduced to the system. The vibration in turn caused unwanted wear on the motor fasteners and disturbances for the tilting controller.

The connector was fastened to the motor output shaft using a set screw and clamps the motor rod using four screws, as can be seen in Figure 8, exploded view of the rod connector. The rod connector was made from aluminum after exploratory testing revealed that a 3D printed version in plastic deforms from the torque of the gear output shaft. This system was able to provide the ability to accurately set the motors at a desired angle, fully leveled when flying at high speeds and fully down during low speeds or takeoff and landing, thus giving it VTOL function.

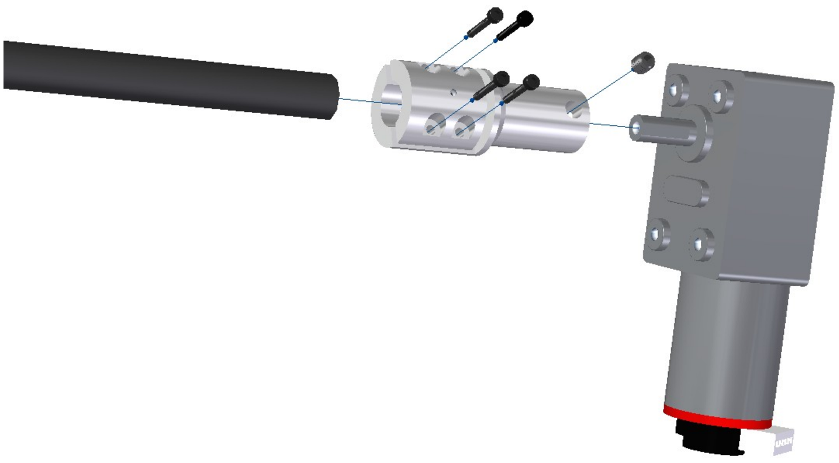


Figure 8: Exploded view of the tilting mechanism.

3.2 Electronics Hardware

This section aims to describe the electronics side of the hardware. It consists of the routing of all cables in the vehicle, the powering of motor systems, as well as descriptions of the micro-controllers. It will also discuss the requirements that were set on the electronics.

3.2.1 Routing Design

The system's electrical routing can be found in its entirety in Appendix C. How the subsystems are connected and physically related can be subtracted from the schematic. The subsystems included are the high power system (Motors, EDFs, ESCs and batteries) as well as the low power part including the flight control unit and motor tilt system. For a more in-depth description and overview, see the respective chapters below.

3.2.2 Motors, EDFs and ESCs

Relevant requirements that was considered was mainly requirements REQ3.1, REQ3.2 and REQ1.4 but also REQ2.3 to some extent. The electrical part of the tilt mechanism consists of a 12V DC-motor and a hall-encoder. These are controlled by an Arduino MKRZero as well as an H-bridge Arduino shield. For more information about these control systems see the respective chapters below.

The Electric Ducted Fans (EDF) [3] uses a 22V outrunner motor and can be seen in Figure 5. It can draw up to 74A during maximum thrust which means that the Electronic Speed Controllers (ESCs) needs to be able to withstand this current. The ESCs that were chosen for these motors can be seen in [5]. These ESCs have a 5V Battery Eliminator Circuit (BEC), typically used to provide a receiver with power. These wires normally connect parallel to the signal wire into the flight control computer but since several ESCs are connected together the BEC needs to be removed in order to prevent current flowing backwards into the ESCs.

The BEC is used to ground the ESC to the receiver resulting in a stable signal. However since GND was cut they had to be grounded another way. It was accomplished using a ground plane connecting GND from all the ESCs to all the batteries. One wire also connected the ground plane to the flight control computer and hence the ESCs are all able to safely communicate with the controller. A full schematic of the wiring can be seen in Figure 9.

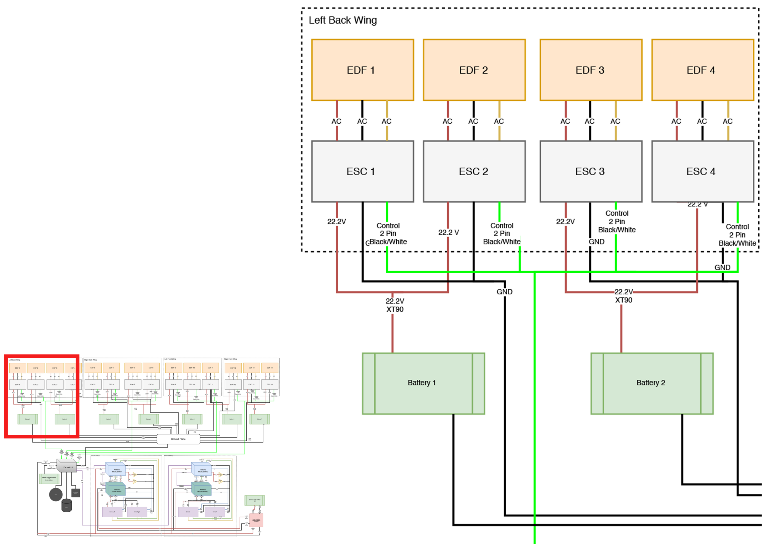


Figure 9: Routing subsystem: High Power Side, including motors, speed controllers and batteries.

3.2.3 Motor Tilt System

The Motor Tilt System (see Figure 10) annotated "Servos" in the figure, is controlled through an Arduino Motor Shield [8]. The Motor Shield is a circuit board with an H-bridge and acts as an interface between the low-voltage digital Arduino controller and the high-voltage DC-motor side. The main function of the Motor Tilt System is to receive reference signals from the Flight Controller and using a PI-controller to calculate and subsequently send control signals to the Motor Shield. The control algorithm implementation is explained further in the software chapter.

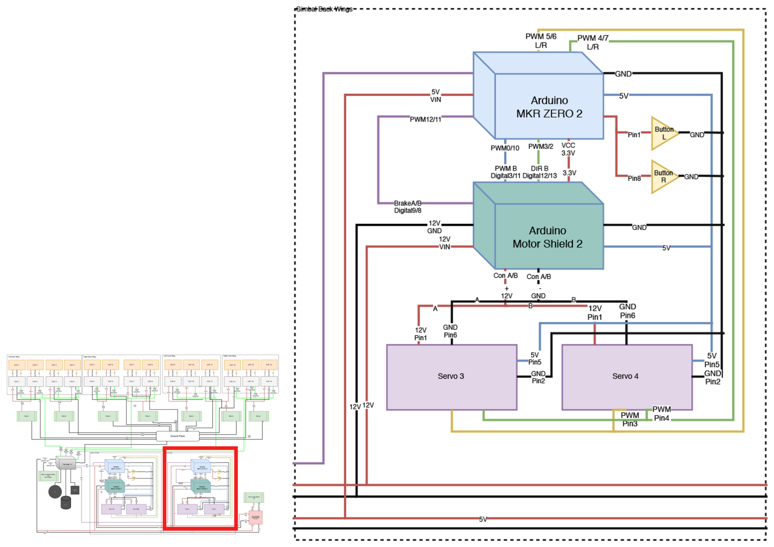


Figure 10: Routing subsystem: Motor Tilt System.

In order to connect the Arduino MKRZERO with the Motor Shield a custom circuit board board had to be developed to allow for a robust connection as well as easy debugging and development. The design of the circuit board can be seen in Figure 11. Connections for debugging buttons, serial communication, tilting motors and encoders were included in the design.

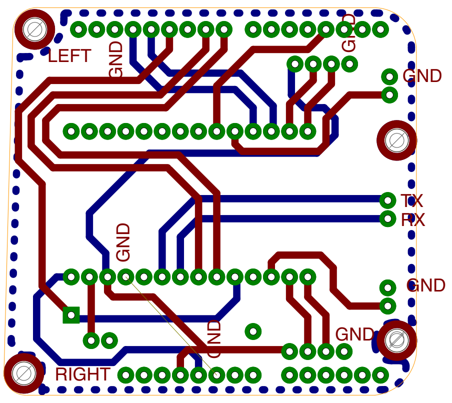


Figure 11: Circuit board design for interfacing motor tilting system.

3.2.4 Flight Controller

The Flight Controller used in this project was an open-hardware platform called Pixhawk, which contains a large open-source codebase especially suited for RC multirotors, tilted rotors and fixed wing planes. The main processor is an STM32F427 running at 180 [MHz] with a 2 [Mb] flash memory. The Inertial Measurement Units (IMUs) included are; accelerometer, gyroscope, magnetometer and a barometer which are all mounted on a separate vibration dampened PCB and used to calculate the yaw, pitch, roll and altitude of the vehicle. The Flight Controller was also fitted with an external GPS for position holding and a Pitot tube to measure the speed through the air. For a overview of connections see the routing scheme in Figure 12.

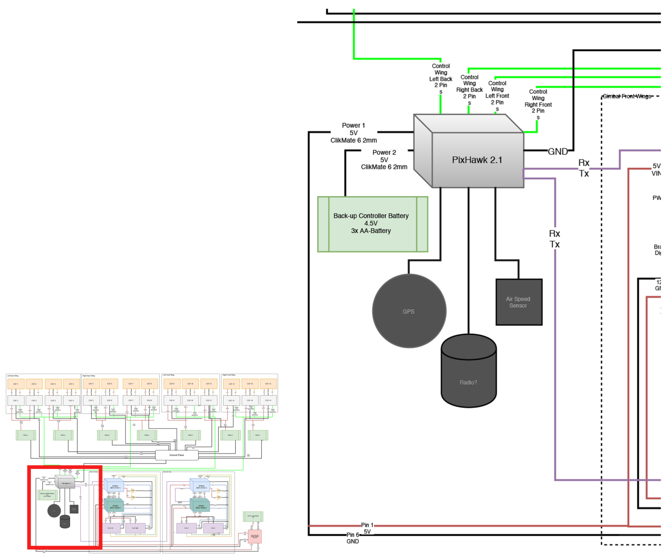

Control Wing Left Back 2 Pins
Control Wing Right Back 2 Pins
Control Wing Left Front 2 Pins
Control Wing Right Front 2 Pins
Gimbal Front Wings
Power 1 5V ClikMate 6 2mm
Power 2 5V ClikMate 6 2mm
PixHawk 2.1
GND
Rx Tx
Rx Tx
5V VIN
Back-up Controller Battery 4.5V 3x AA-Battery
GPS
Radio?
Air Speed Sensor
Pin 1 5V
Pin 6 GND

Figure 12: Routing subsystem: Flight Control Unit.

The Flight Controller had five UART serial ports which could send and receive Micro Air Vehicle Link (MAVLink), a protocol developed for small unmanned vehicles. It could also receive and send serial messages with external devices, which was the selected method for this project.

The Flight Controller was powered by an external battery and a switched voltage-regulator circuit reducing the voltage down to 5V from the 11.1V battery. A backup power supply was also used to provide redundancy if the main battery was depleted or failed. Power to the radio received, GPS module and the air speed sensor was provided by the Flight Controller.

3.2.5 Batteries

For an overview of the batteries in the routing scheme, see Figure 13. There are a few requirements that are set on the batteries, such as REQ1.8, REQ3.3 and REQ3.4. The batteries needs to be rechargeable, lightweight and within the budget of the project but most importantly able to provide the motors with sufficient power. For this reason 5000 mAh 6S LiPo, Lithium Polymer, batteries were chosen [6]. In order for the vehicle to fulfill requirement REQ2.4, "the vehicle shall have a thrust-weight ratio of 1.5:1", it was decided to let one battery provide power to two EDFs. For a 5000 mAh battery to be able to provide a total of 150A it needs to have a discharge rate of at least 40C for it to also have some safety margin.

There also needs to be power provided to the flight controller and the motor tilt controllers. These controllers needs 5V and the tilt motors needs 12V This can be done by letting one of the batteries pass by a voltage regulator that would supply these components. That however would mean that 75A would have to flow past the voltage regulator and that would potentially break it. Instead a separate 3S LiPo battery was acquired for this purpose [7]. The 3S LiPo battery provides around 12V by itself which means that it can be connected directly into the motor shield for the tilt motors. The tilt motors only draws a few ampere which means that a voltage regulator can be safely connected to supply 5V to the flight controller and motor controller.

3.3 Simulation

In order to safely explore the control system, a model for simulating the transition was developed. Firstly, a dynamic model of the vehicle was created and secondly, a model for the tilt controller was added. The simulations were done using Matlab and Simulink.

3.3.1 Vehicle Dynamics

At first a model for the dynamics of the vehicle was developed. The base of the model was Newtons second law,

$$F = ma, \tag{1}$$

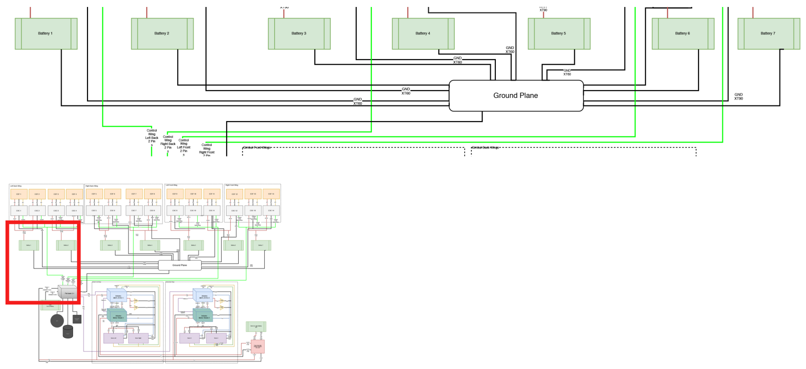


Figure 13: Routing subsystem: Batteries and groundplane.

where force F cause accelerations a inversely proportional to the mass m. Assumptions were made, and one was that the body is completely rigid in order to simply distances and ignore vibrations. Furthermore, it was assumed that there was only movement in two dimensions. This meant that the vehicle can only move upwards, downwards, forwards and backwards, hence not being able to turn left or right. This assumption allowed placement of all motors on the front or back wing at one point, respectively, and only perform calculations with a total of two thrust forces. This simplification significantly reduced the complexity and size of the model. Furthermore, the complete movements in three dimensions are not necessary since the tilting transition will be performed in only two dimensions and no turning will be done. When determining the angle of the vehicle, Newtons second law for moments was used.

The angular inertia was extracted from the CAD model. The angular acceleration was integrated to find the position at every time step. The main force to affect the orientation was the thrust, primarily with the motors in a vertical position. The difference in lift from rear and front wings was also considered, and was compensated for by the control systems.

The dynamic model takes into account the forces shown in Table 1.

Force	Note	Assumption
Air-resistance	Separate for body and wings	Is applied to the center of mass
Lift	One point of lift for each wing pair	Is applied at one point in the center of wing
Thrust	One point of thrust for each motor line. The moment lever is dependent on the motor angle due to the motor movement up and down when tilting	Is applied at one point in the center of the motors
Gravity	Center of mass from CAD model	Constant with respect to height

Table 1: Forces and detailed considered in vehicle dynamic simulation.

3.3.2 Tilting controller

A controller for the tilting was developed to run on an Arduino. The code was written in C and was then ran as a S-function in Simulink when simulated. The entire simulation ran at a fixed step of 2 kHz in order to account for the loop speed of the Arduino. The controller block receive the vehicle angle that should be fed from the flight control computer and the tilt angles that should be calculated internally by the encoder loop run parallel on the Arduino. The output of the controller is the desired angles for the front and rear motors. The vehicle built has two separate Arduinos control the front and rear wing respectively, this would require some modification to the original plan, however in a scenario of real implementation the computer architecture will probably be different.

The output angles should later be sent to the internal servo motor controller that will actuate the desired angle. In the model these servo motors are modeled and have a PID controller to steer them. Some issues with the speed of the servos were encountered and it is concluded that the speed of the controller is essential, it must not be too slow. The implementation used is based on a motor model, but behavior issues caused the control parameters to be set arbitrarily high to essentially allow it to accelerate quickly with only a speed saturation.

The controller developed has the the main goal to keep the vehicle angle at zero by effecting the torque on the body by changing the angle of the motors. When a transition is triggered all motors back and front will slowly keep increasing their angle independently in order to maintain the focus on keeping the vehicle level. The controller is implemented in such a way that the motors back and front will always do a mirrored movement around the desired reference. This is to achieve maximal agility but still only having one control variable. Since the motors are assumed to have maximal thrust, and no stabilization algorithm is run in parallel to control the roll. These drawbacks needs to be considered when evaluating the simulation.
The controller is not proven to be stable in all conditions, but it was tested in simulation and it copes with disturbances such as someone affecting the angle of the vehicle by touching, as can be see in Figure 20 . It can also deal with a sudden decrease in thrust such as a motor suddenly stops working.

3.4 Software

The controller software was separated into two main modules; the flight controller and the motor tilt controller. The flight controller handles the stabilization of the entire vehicle, while sending reference signals to the motor tilt system. The reference signals determine how to move the servos. The motor tilt system runs a position controller that puts the servos at the desired angle.

3.4.1 Flight Controller

When programming the flight controller, the Ardupilot code base was used. Ardupilot is an open source software project that is commonly used in drones [2]. The code base can be used for a variety of vehicles, e.g. multirotors, planes and rovers. The architecture of the code base for drones can be seen in Figure 14.

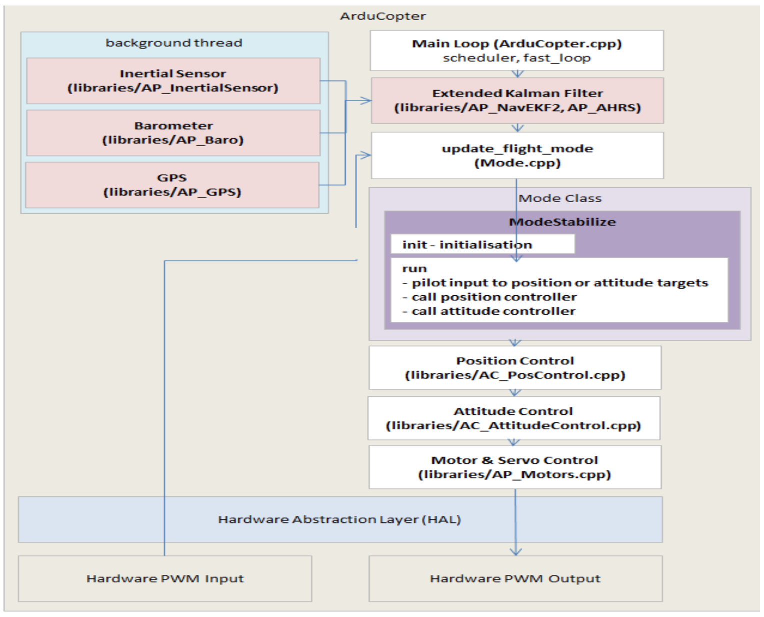

ArduCopter
background thread
Inertial Sensor (libraries/AP_InertialSensor)
Barometer (libraries/AP_Baro)
GPS (libraries/AP_GPS)
Main Loop (ArduCopter.cpp) scheduler, fast_loop
Extended Kalman Filter (libraries/AP_NavEKF2, AP_AHRS)
update_flight_mode (Mode.cpp)
Mode Class
ModeStabilize
init - initialisation
run
- pilot input to position or attitude targets
- call position controller
- call attitude controller
Position Control (libraries/AC_PosControl.cpp)
Attitude Control (libraries/AC_AttitudeControl.cpp)
Motor & Servo Control (libraries/AP_Motors.cpp)
Hardware Abstraction Layer (HAL)
Hardware PWM Input
Hardware PWM Output

Figure 14: Architecture of the Ardocopter software.

The main task of the flight controller is to operate the attitude control, i.e. the controller that sets the roll, pitch and yaw angle using inputs from sensor values and reference values from the radio controller(RC). The inputs were filtered using and Extended Kalman Filter(EKF), to estimate the position of the vehicle.

The attitude controller consists of a proportional controller that converts the error between the desired angles and the actual angles to rotation rate of the motors, followed by a PID controller that converts the motor rate to output signal in terms of pulse width. These consecutive controllers make sure that the vehicle is stable in the air as well as determines its heading. The functionality is shown in Figure 15.

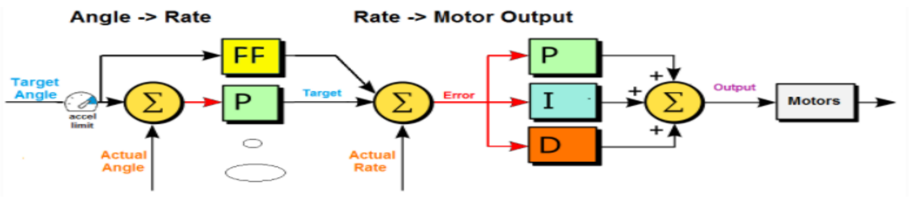


Figure 15: Flow chart of the attitude control.

The Ardupilot code was modified to function with the wing- and motor configuration of the vehicle. Since the number of motors are larger than the available output channels on the Pixhawk, all motors on one wing was considered as one and was given the same control signal. The airframe was modified to account for the mean position of the motors. Since there are four motors on the back wings and three on the front wings, they were considered as further away in the code to give a larger impact on the Euler angles. After that, the PID controller could be tuned to improve the stability of the drone.

A custom made function for controlling the servo motors was added to the main loop, scheduled to run in parallel with the stabilizing control. The function reads the inputs from the hand controller and sends a reference signal to the Arduinos. Two different modes were developed, one regular mode and one calibration mode. The regular mode was used to tilt the motors during runtime, where all the motors were given the same reference signal to move equally. In calibration mode, the motors could be tilted separately to manually level the wings.

3.4.2 Motor Tilt Controller

The Motor Tilt Controller consisted of a PI position controller that used feedback from the encoder reading function and a reference position from the Flight Controller and calculated a control output for the DC-motors that controlled the tilt angle of the EDFs as indicated in Figure 16

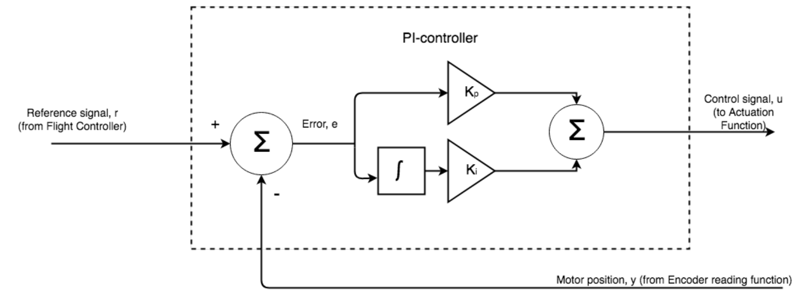

PI-controller
Reference signal, r (from Flight Controller)
+
Σ
Error, e
-
K_p
∫
K_i
Σ
Control signal, u (to Actuation Function)
Motor position, y (from Encoder reading function)

Figure 16: Software architecture: PI controller.

In order to keep a high resolution the reference, r and feedback y signal was based on encoder ticks instead of degrees. There were 31 encoder ticks per degree of rotation on the output shaft of the gearbox resulting in 11 160 ticks per full revolution.

The Tilt controller was developed by simulating the tilt mechanism and optimizing control variables in the simulation. After the simulation was completed and checked, the software was developed.

The frequency of the controller was set to 100[Hz], since the controlled system-plant was very slow. The low value was motivated by having to accommodate for message reading, parsing, actuation and interrupts connected to encoder reading.

The controller was developed with all variables as constants to allow easy manipulation. A debugging interface was developed that outputted the reference values, control signals and plant output to an external computer for debugging and verification purposes.

After the controller model was developed and implemented it was tested and the control parameters were fine-tuned, the final step response of the controller is plotted in the Results section.

3.4.3 Communication interface

For the communication interface several methods were implemented and tested with varying complexity and stability. First an interrupt-based solution that measured the pulse width of a PWM signal was developed. This way the MTS could be controlled both by an analog output from the Flight Controller, but also directly from the radio receiver in order to bypass the Flight Controller. Using this method the industry standard servo control [12] of 1ms, resembling zero degrees actuation, to 2 ms pulses (resembling full actuation) was used in order to verify the subsystems without directly involving the Flight Controller. This communication method was simple and robust but placed a limit on the data that could be sent between the Flight Controller and the MTS, by only allowing a reference setpoint and nothing more, which was not good enough for the final version.

Instead a custom messaging protocol using UART was developed, where the Flight Controller could send various messages resembling different functionalities to the MTS. The messages were constructed as described by Figure 18 and the available commands are listed in Table 2 below.

This communication method allowed for greater flexibility in terms of functionality at the cost of being more dependent on signal integrity. Hence a known limitation of this system was that if

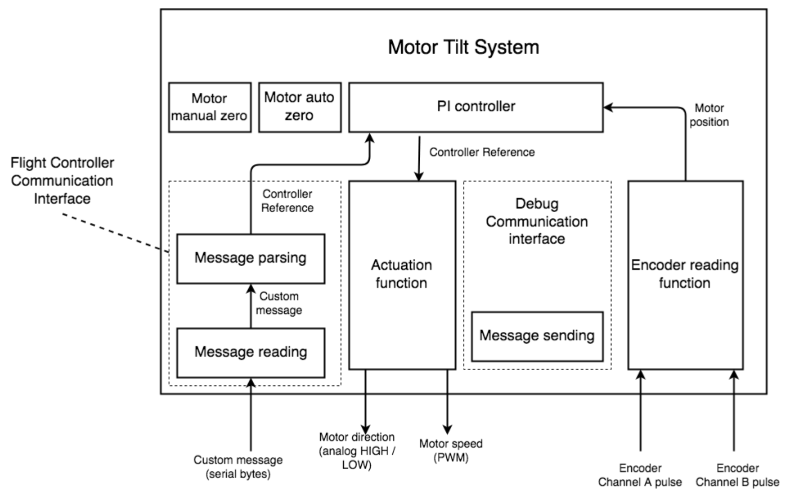


Figure 17: Software architecture: Motor Tilt System.

command	1st char of msg	2nd char of msg	3rd char of msg	4th char of msg	\n

Figure 18: Construction of the message.

Table 2: Custom message commands.

Command	Function
b	Wing reference setpoint
o	Activate motor tilt controller
n	Deactivate motor tilt controller
c	Auto calibrate motors (decrement both motors until zero position is detected, and set zero position)
e	Increment left motor 1 degree and set zero position
f	Decrement left motor 1 degree and set zero position
g	Increment right motor 1 degree and set zero position
h	Decrement right motor 1 degree and set zero position

a message got corrupted on the wire it could potentially lead to an unwanted reference position to be set causing the motors to tilt sporadically, or the zero position of the system could be reset unintentionally causing the controller model to become invalid. To prevent message corruption, causing invalid reference positions to be read, a function that disregarded reference messages was added. The function disregards messages when the resulting derivative of the reference signal was unreasonably large.

Another safety measure was implemented in the Flight Controller, that prevented any decrement or increment messages from being sent, and thereby reset the zero position. This measure was active during flight and whenever the motors were moved by setting new references.

4 Results

This section will present how the prototype performed in regards to the requirements set for the project.

4.1 Requirements fulfillment

In Table 4.1, a full list of the requirements and their method of verification is shown.

In summary 45% of the requirements were fulfilled by demonstrator testing and 70% of the requirements were fulfilled if the simulation results are taken into account.

Table 3: Table 4.1:

Requirement no.	Fulfilled [Yes/No]	Verification method
REQ1.1	Yes	N/A
REQ1.2	Yes	Simulation
REQ1.3	Yes	Simulation
REQ1.4	No	Test case 3
REQ1.5	Yes	Simulation
REQ1.6	No	Not attempted
REQ1.7	No	Stakeholder approved raised budget
REQ1.8	Yes	N/A
REQ2.1	Yes	Test case 11
REQ2.2	Yes	Test case 10
REQ2.3	Yes	N/A
REQ2.4	No	Test case 2
REQ3.1	No	Test case 2
REQ3.2	Yes	N/A
REQ3.3	Yes	Test case 2
REQ3.4	Yes	N/A
REQ3.5	Yes	Test case 2
REQ3.6	No	Test case 5
REQ4.1	Yes	Test case 3
REQ4.2	Yes	Test case 1

4.2 Mechanical Results

Flight was not achieved, and there are a few mechanical issues related to this. The method of fastening the motors proved to be a problem, since it was not firm enough. Movement in the gearbox was amplified by the lever to the motors causing more movement than expected. The carbon fiber axle twisted under the torque and some of the mounts slided due to issues with fastening with friction on a smooth rod. The aluminum connector between the motor and rod did not break, but some instances of the stop screw loosening or the rod sliding were encountered.

The motors had some upsides and downsides. When tested separate from the vehicle they provided good thrust for a small propeller diameter. However problems with quality were encountered as two of them broke down. The motors caused extreme vibrations and combined with the flexible motor mounts this probably led to the axles breaking off.

The overall design and aerodynamics of the vehicle is difficult to assess, since it never flew unconstrained. The sandwich construction of the body proved to be stiff and strong, though it was difficult to work with the on-board electronics as the plate had to be unscrewed to grant access to the upper side of the bottom plate. The 3D printed blocks was the central component of the construction proved to be strong and since they were printed could be customized with precision. It took several iteration until the blocks were fully functional, but the final version provided the stability and functionality required. The blocks biggest issue was that the method of fastening the plates wasn't reliable as the washers could easily start turning in their slots, making it very difficult to unscrew the screws.

4.3 Software Results

The results from the simulations and software implementations. The simulation was done using Matlab Simulink while the software implementations where tested on their corresponding micro-controller.

4.3.1 Simulation

The tilt controller worked well, and it is able to tilt the motors slowly while maintaining control of the vehicle. The balance is kept by only varying the angle of the motors and not changing the thrust at all. This is a limitation as no stabilization for roll is implemented in the simulation. The angles of the motors together with the body angle during a transition can be seen in Figure 19. It can be see that there is an imbalance in thrust between the front and back wing which results in the vehicle pitching forward, this is then compensated by the motors changing angles.

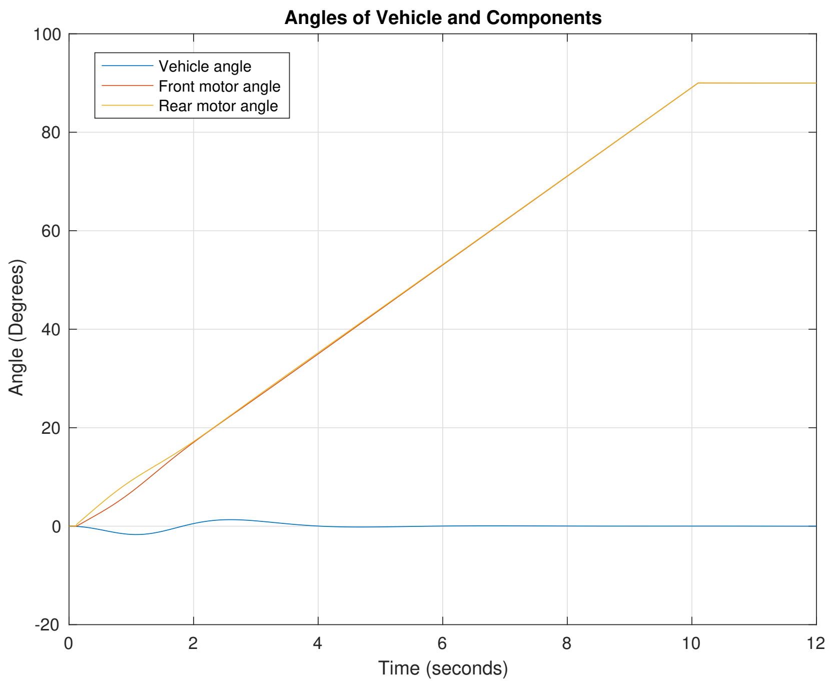


Figure 19: Simulation Graph: Transition.

The controller has also shown some resistance to external influences, such as turbulence represented by the impulse in vehicle angle. In Figure 20 is an example where an impulse to the vehicle angle is added, it can be seen that the motor angles compensate and levels the vehicle while still performing the transition.

4.3.2 Flight Controller

The stability of the flight controller code was tested in test cases 3, 12 and 13, which can be seen in Appendix B.

The result from the Balancing in Roll Axis test can be seen in Figure 21. The controller could compensate for the applied disturbance in 0.2 s.

The result of the unconstrained hover test can be seen in Figure 22. From the graph, one can conclude that stable flight was not achieved, but the vehicle was airborne for approximately 2 s, with a maximum altitude of 0.13 m.

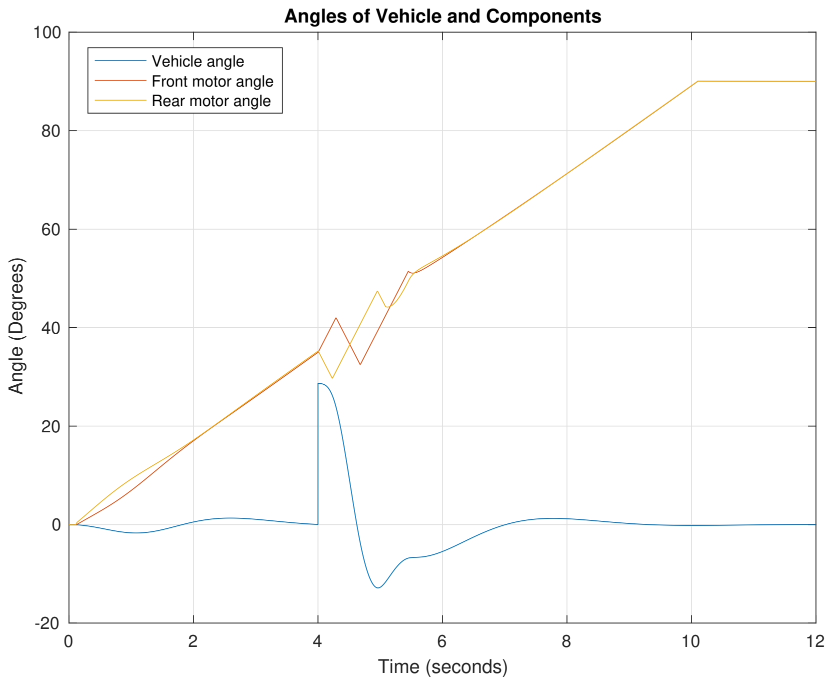


Figure 20: Simulation Graph: Disturbed Transition.

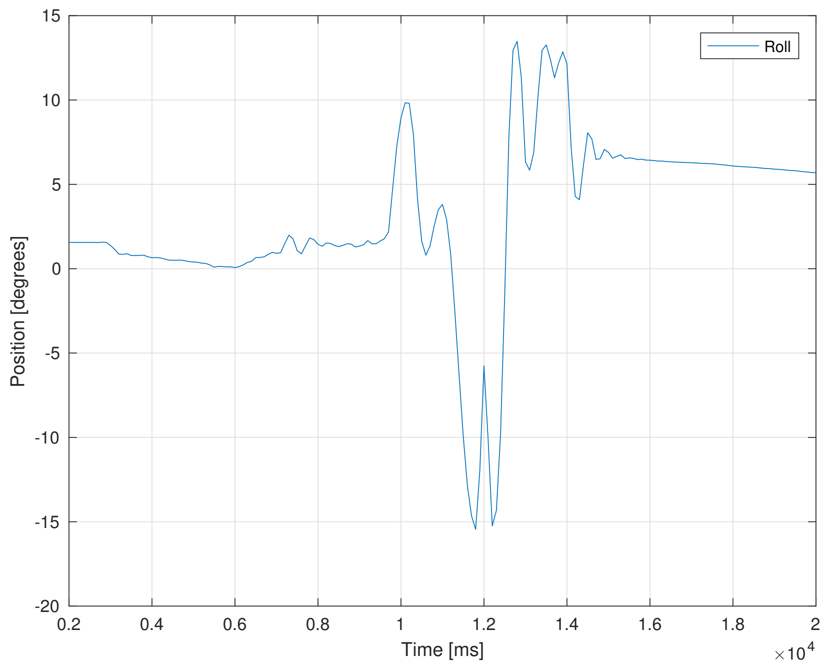


Figure 21: Stabilizing controller.

4.3.3 Motor Tilt System

The result of the tuned PI-controller for the Motor Tilt System can be seen in Figure 23 with an input step of 60 [deg]. The data has been extracted from the controller in real time while the step input was provided with a computer connected in stead of the hand-held controller and the rise time was calculated to 2.0 [s].

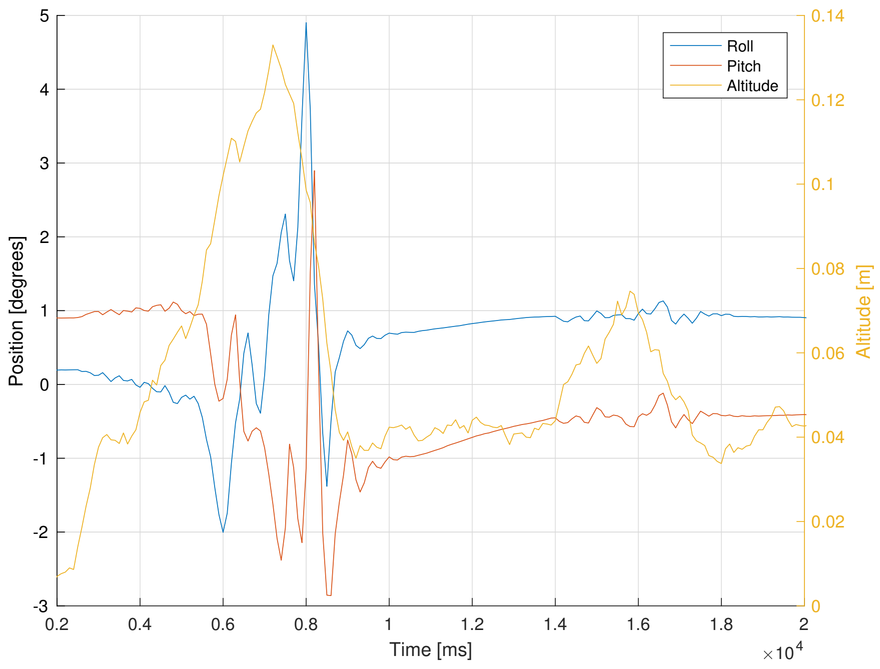


Figure 22: Unconstrained flight.

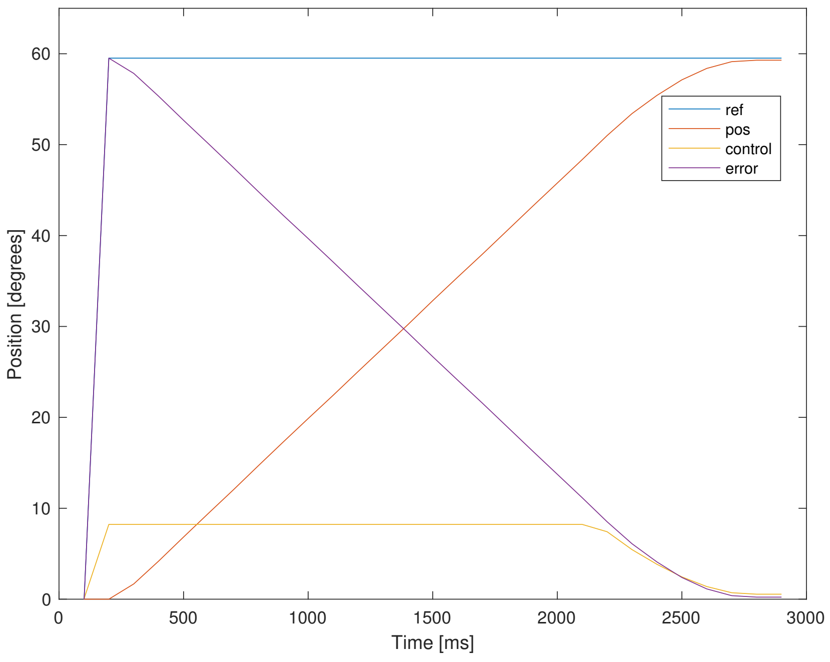


Figure 23: Controller plot: Motor Tilt System.

5 Discussion

This section will discuss the development process, the results conducted during the project and also touch on what future work could include for the development of a personal flying vehicle.

5.1 Development Process

This project has been performed by a team consisting of nine people. When performing a project with a team of that size it is important to make sure that everyone is focused on the things that brings the most value to the project. This is where a Scrum development process is useful since it puts a lot of focus into creating value for the project. However during the later stages of the project a lot of bottlenecks were encountered when the whole team needed to work on the prototype simultaneously.

Another issue that had to be tackled was information sharing. With nine people working somewhat individually on problem solving it takes energy to go back and share with the rest of the team how the problem was solved. It was much easier to just keep going and solving the next problem. Some information sharing was done during the weekly meetings or on requests, but not enough to cover the more complex solutions. Unfortunately this lead to in the later stages of the project people started to be irreplaceable since they might be the only person who knows how something, like the controller for example, works. That means that it is hard to work alone and if someone is absent the whole team might stop. This is something that could have been solved better during the project. When working with Scrum you are however always focused on creating more value so for the team to put energy into sharing knowledge it must be decided from the start that there is value in it.

5.2 Results

This section discusses the results that were produced. Elaborating on the results from the mechanical and software side and how the requirements were fulfilled.

5.2.1 Requirements fulfillment

As can be seen in 4.1 not all requirements have been fulfilled. The requirements that have been fulfilled but are without verification were fulfilled during the design/lock in phase of the project. They were accomplished by limiting wingspan and vehicle length for REQ2.3. Requirements 3.2 and 3.4 were met by other requirements, while REQ3.2 was fulfilled through REQ2.4 and REQ3.4 was accomplished by REQ3.3.

REQ1.2, 1.3 and 1.5 have only been confirmed in a simulation and are therefore still in need of a real-life test to fully verify that these have been fulfilled. Mainly due to delays and unexpected set backs during the project resulting in a loss of time. Fulfilling these requirements are in turn a prerequisite to being able to verify REQ1.6. The rest have been fulfilled or failed as can be seen in their respective test-cases in Appendix B.

REQ1.4 was deemed unfulfilled as in several test-cases various parts suffered catastrophic failure, which in some cases resulted in parts catapulting away. REQ3.6 has the same issue that unforeseen EMI caused total failure of the Arduino. Further information can be found in the Appendix B.5. REQ2.4 and REQ3.1 were not fulfilled as the thrust to weight ratio was lower than set due to added weight not included in the weight budget. For REQ4.1 only roll stability has been verified during the project.

Overall the requirements could have been more narrowed down in order to provide more steering. Narrowing down would provided extra information to simplify the design process. Testing and verification would also have benefited by providing a bigger basis for tests to be performed and what they verify. Further some requirements could have been merged as is the case for REQ2.4 and REQ3.1 which both require a certain thrust to weight ratio.

5.2.2 Mechanical Results

The demonstrator showed that most of the mechanical subsystems worked as intended. The wings and body withstood the load during vertical suspension without too much flex. The construction also withstood full thrust from the motors when fixed to the ground and also while suspended.

The most unforeseen mechanical issue was the twisting of the carbon fiber rod that the EDF motors were mounted on, which was solved by mounting each motor at an angular offset. Once thrust was applied and the rod started twisting they would line up better, but not perfectly since their individual incline was a function of the thrust applied.

Vibrations from the EDF motors was another phenomenon that proved to be an issue as the subsystems were assembled and tested together. It was known from the beginning that the high-rpm EDF motors would cause vibrations and measures to counter it were taken early in the design process. The precautions of using rubber suspensions for both the wings and the individual EDF motors proved to not be enough and during some of the earlier test cases the motors vibrated so much that a motor axis sheared. This caused the motor to be irreparable and it had to be replaced. Additional measures that were taken during the later stages of the project was replacing the plastic EDF motor holders with aluminum ones to increase resistance against vibrations. The replacement was done because the vibrations caused the rod clamps to shear.

One of the reasons that changes were made to the mechanical design at such a late stage in the process was that the final assembly process started later than initially planned and some issues did not arise until integration. This led to delays in the planned testing and subsequent planned tuning. This resulted in that the only unconstrained tests that were performed were unsuccessful lift-off tests, with little time for analyzing and rewriting the software afterwards, which compromised its stability. This resulted in the vehicle failing to fulfill the last requirements.

5.2.3 Software

The Flight Controller was verified to have stability in one degree of freedom at a time during the constrained flight tests, which partially satisfied the requirement "REQ4.1 The software shall include a self-stabilizing system". The unforeseen mechanical breakdowns put limitations on the amount of testing that was possible in the given time and thus stable hover and forward flight was not achieved.

When the whole system was tested in unconstrained hover it was proven that the controller was not capable of stabilizing the vehicle with sufficient speed. A higher thrust to weight ratio and finer tuning of the PID controller would improve the control performance.

The Motor Tilt System was functioning with input from the hand controller. The step response in Figure D shows that it is able to stabilize the motors in an adequate manner. The control signal is saturated by the software in order to not output higher voltage than the motors can handle, which gives the system its primary speed limitation. Running the Motor Tilt System as a stand-alone system verified was functioning as intended and performed closely to the simulation, however it did experience some stability issues possibly related to EMI from the high-current part of the system, as can be seen in Appendix D.

Simulations showed that the transition with the selected components would be possible, however the simulation contained a few simplifications and in order to fully verify the concept the real tests should be performed.

Another important part that was verified in the transition simulation was the fact that the Motor Tilt System (with its controller and speed limitations) was quick enough to compensate for a sudden 30 [deg] disturbance of the vehicle angle, which could be caused by turbulence or sudden movements of passengers as displayed in Figure 20.

If the requirements for the software would have been more detailed it would have been easier to evaluate the software that was developed. Ideally the requirements should be more quantifiable to for example a span of degrees that the vehicle should stay within to be defined as stable.

5.3 Future Work

A lot of work has been done with this project, but there is still a lot more to be done before an electric car can be put in the air. For this product to be safe for human passengers, a lot needs to be done in terms of safety. For example, the communication within the vehicle is done using a serial bus with a protocol that has no message confirmation, and it is thus not sure that a message send it received or even correct. A much more complex protocol with an error detection system such as a checksum should be implemented. It is also recommended to implement acknowledgements on the message transmission to prevent packet loss.

Furthermore, a lot of work needs to be done to the flight controller. Currently the flight controller is an open source software run on a Pixhawk without the full understanding of all algorithms involved. For this vehicle to transport human passengers, experts within fluid dynamics and airplane control needs to be consulted and a custom made software should be developed.

While some conclusions can be draw from the design used, a lot more needs improvement. The method of mounting motors needs significant rework. Due to the numerous issues with mounting the motors on a rod, its strongly advised to develop a new method of mounting and tilting the motors which can additionally allow individual tilting as well. The structural design of the wings needs some rework. The wings had issues with torsion throughout the project, although this was reduced by the addition of a hard shell to the wing. With proper solid mechanics analysis the rigidity and weight of the wings can certainly be improved further.

The focus in this project has been to get the complex vehicle to hover. Due to this focus on getting all of the mechanics and electronics together, the task of creating and implementing a tilt controller has not been completed. A software was written and simulations was done, but since the vehicle has not yet flown, the test of this has not yet been possible. The limitations from the open source software used has also slowed this integration down, further motivating a need for a more custom software designed by experts in the field. Flight in a horizontal mode has not yet been investigated and since the wing configuration is quite rare,it will need more consultation from aerospace experts.

The scalability of the project has not been investigated heavily and needs more attention. While some guidelines can been seen, such as the efficiency of the propeller will increase, this is a non-linear curve and needs investigation. The battery efficiency can roughly increase when custom made buttery pack is used, but this, again, is a rule of thumb. The scalability of the design and materials have not been investigated and the effect of vibrations is unknown. The control systems and algorithms used are developed particularly for small hobby projects and the implications when scaling this up needs to be considered. The investigation of scalability is very large and complex, but extremely necessary.

5.4 Conclusion

As the project has progressed, the development process has improved and the team has learned to use work breakdown structure and Scrum in order to improve the efficiency of the project work.

The prototype built in this project partially proves the concept of personal flying vehicle. Especially, the constructed tilt-rotor mechanism, essential for the VTOL property, was fully functional. The prototype could also prove that the subsystem controlling the rotor angle could be integrated in the flight controller software, without interfering with the stabilizing controller . However, the forces of vibration and torsion in the tilting mechanism has been a major issue that was not solved in this project. Furthermore, a higher thrust to lift ratio is desired to improve the stabilizing controller. The instability in the aircraft design has lead to that forward flight could not be tested.

There is still a lot of work to be done before flying vehicles replaces cars as the most common mean of human transport. Further prototyping at this scale is recommended to The focus in this project has been the mechatronics parts; integration of electronics, hardware, software and control theory, and for future work, experts in aerodynamics and solid mechanics should be consulted. The work done in this project has been a good starting point, and has taken flying cars one step closer to reality.

References

[1] Department of machine design, *Maskinelement Handbok*, Royal Institute of Technology, Stockholm, 2008.

[2] Ardupilot, *ardupilot.org*, 2017.

[3] EDF info, 2017-12-15, `https://hobbyking.com/en_us/dr-mad-thrust-70mm-11-blade-alloy-edf-1900kv-mot` `html?___store=en_us`

[4] Tilt Motor info, 2017-12-15, `https://www.aliexpress.com/item/Wholesale-JGY-370B-12v-DC-Motor-with-Encoder-Disk-High-Torque-Low-Speed-Gear-box-Worm/32361264555.html?spm=a2g0s.9042311.0.0.23GzC3`

[5] ESC info, 2017-12-15, `https://hobbyking.com/en_us/turnigy-ae-80a-brushless-esc.html?___store=en_us`

[6] 6S Battery info, 2017-12-15, `https://hobbyking.com/en_us/turnigy-5000mah-6s-40c-lipo-pack-xt90.html`

[7] 3S Battery info, 2017-12-22, `https://hobbyking.com/en_us/turnigy-3600mah-3s-30c-lipo-pack-xt-60.html`

[8] Arduino Motor Shield Tech Specs, 2017-12-18, `https://store.arduino.cc/arduino-motor-shield-rev3`

[9] Arduino MKRZERO Tech Specs, 2017-12-18, `https://store.arduino.cc/arduino-mkrzero`

[10] Waterfall software development model, 2017-12-21 `https://www.oxagile.com/company/blog/the-waterfall-model/`

[11] Efficiency vs. performance - How to build a drone with long flight time, 2014-11-27 `https://www.flyingtech.co.uk/blog/efficiency-vs-performance-how-build-drone-long-flight-time`

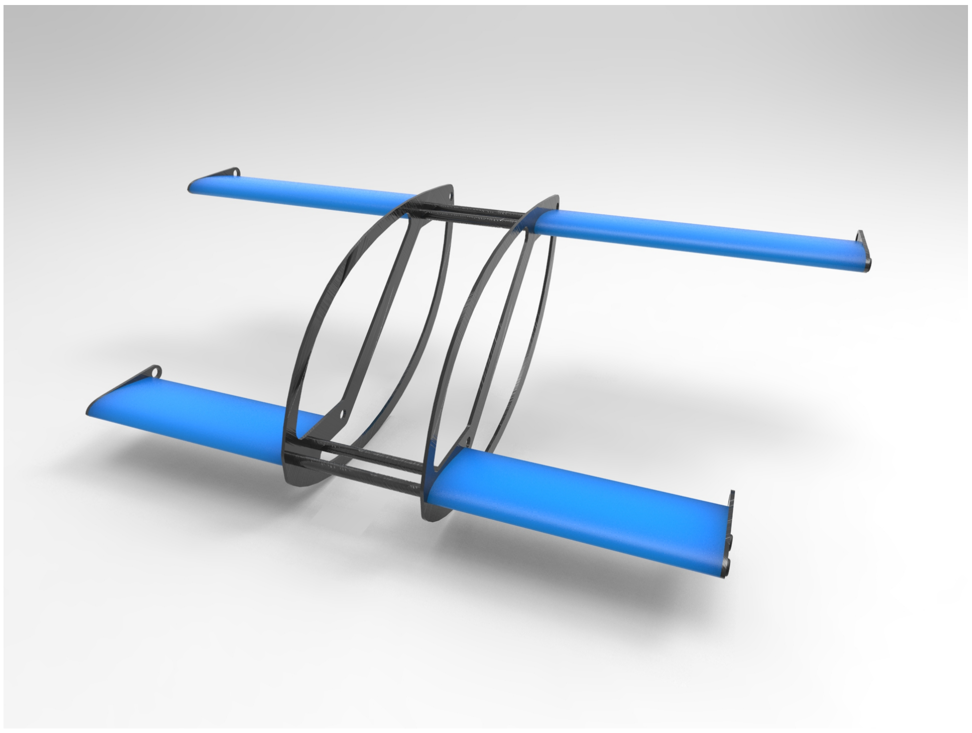
[12] Hobby Servo Fundamentals, Darren Sawicz, 2017-12-22 `https://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf`

A Pre-study and SOTA Report

This page intentionally left blank

WAZP

Pre-study and State Of The Art Report



Axel Johansson, David Wallén, Fredrika Kringberg
Gustaf Ingvarsson, Hanna Dotevall, Jiyang Chen
Magnus Valtersson, Tomas Fredriksson

June 1, 2017

Abstract

This project was part of the Mechatronics Advanced Course, at KTH (Royal Institute of Technology). The group working with the project consists of eight students and will work with the project over three periods, from March 2017 to December 2017. The assignment was given by Anton Wass, an entrepreneur with a mind for technology and production. The project was founded to challenge and improve how human transport is made today. With already existing technologies, a vehicle that can travel in a superior way to todays alternatives should be developed. The project is limited to design and build a scaled model version of a vehicle that can go from being parked on the ground to taking of vertically and subsequently transition into a horizontal long-range flight mode, and reverse the process for the landing procedure.

This report was conducted during the first period of the course and contains general details of a pre-study and a state of the art research (SOTA) as well as future work and a project plan for next semester.

Acknowledgements

We would like to thank Anton Wass and Robin Lilja for giving us the opportunity to work with this exciting project. We would like to show our gratitude to Anton for giving us financial support and continuously believing in the team, and to Robin for providing us with good constructive feedback on our technical solutions.
We would also like to thank our supervisor Björn Möller for helping us with providing a good working environment for us and guiding the team when it is needed.
Additionally, we would like to thank Per Wennhage for helping us with the aerodynamics and taking the time to discuss ideas with us.

Contents

1 Introduction **6**
1.1 Scope . . . 6

2 State of the Art **7**
2.1 Aerodynamics . . . 7
2.2 Wing types . . . 8
2.2.1 Conventional wing . . . 9
2.2.2 Duo wing . . . 10
2.2.3 Mono-wing . . . 10
2.3 Vertical Take-Off and Landing . . . 12
2.3.1 Tilt wing . . . 12
2.3.2 Rotor-tilt . . . 12
2.3.3 Pitching . . . 13
2.4 Existing Designs . . . 13
2.4.1 The Lilium Jet . . . 13
2.4.2 Greased Lightning 10 . . . 14
2.4.3 WingtraOne . . . 14
2.4.4 Ehang 184 . . . 15
2.5 Hardware . . . 15
2.5.1 Motor . . . 15
2.5.2 Energy . . . 16
2.5.3 Material . . . 17
2.5.4 Electronics . . . 18
2.6 Control . . . 18
2.6.1 Hybrid Control . . . 19
2.6.2 Vertical control . . . 19
2.6.3 Horizontal control . . . 21
2.6.4 Transition . . . 21

3 Future Work **22**
3.1 Project Plan . . . 22
3.2 Scalability . . . 22
3.3 Component Selection . . . 22
3.4 Testing . . . 22
3.5 Construction . . . 22

4 Discussion **24**
4.1 Conclusion . . . 24

A Project GANTT Schedule **27**

Nomenclature

Abbreviations

CAD	Computer Aided Design
DC	Direct Current
EDF	Electric Ducted Fan
ESC	Electric Speed Controller
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LQR	Linear-Quadratic Regulator
MATLAB	Matrix Laboratory
MEMS	Micro-Electro-Mechanical Systems
NASA	National Aeronautics and Space Administration
PWM	Pulse Width Modulation
SOTA	State Of The Art
VTOL	Vertical Take Off and Landing
MIMO	Multiple Input Multiple Output
LQR	Linear Quadratic Regulator

Symbols

A	Wing area [m^2]
AR	Aspect ratio [-]
c	Wing chord [m]
c_d	Specific drag coefficient of the aircraft [-]
c_{d0}	Zero-lift drag coefficient of the aircraft [-]
c_{di}	Lift-induced drag coefficient [-]
c_{dl}	Specific lift coefficient of the aircraft [-]
D	Drag force [N]
e	Efficiency factor of the aircraft [-]
g	Gravitational acceleration [m/s^2]
L	Lift force [N]
m	Mass of the aircraft [kg]
S	Wing span [m]
v	Velocity of the aircraft [m/s]
$\dot{x}$	Velocity in horizontal direction [m/s]
α	Angle of attack [°]
π	Circumference-to-diameter ratio of a circle [-]
ρ	Density of air [kg/m^3]

1 Introduction

One century ago, the car replaced the horse carriage as the most common method for personal transportation. However, with the current technological developments in several fields, it is becoming increasingly possible to replace the car with a personal aircraft. Batteries, motors and software has gone down in scale and up in efficiency, taking the idea of a small personal aircraft that replaces the car close to reality. The benefits of such a transition would have a big impact on personal commuting, reducing travel-time and infrastructural need. To become the next step in the evolution of personal transportation, two things are required. First the maneuverability of a multicopter, for urban areas, and secondly the efficiency of a small aircraft, for longer and faster commutes.

1.1 Scope

The aim of this project was to generate a proof of concept for an airborne personal vehicle. The vehicle should be able to replace the car as a mean of conveyance and therefore, an initial set of requirements were given:

- The vehicle must be able to carry two people.
- The maximum size is $3x6$ m.
- The vehicle must have an airtime of 1 h.
- The vehicle must have VTOL.
- The vehicle must be electric and battery driven.
- The cruising speed should be $150 - 200$ km/h.

This, however, is a broad-ranged project and some limitations are necessary. Since this is a project in Mechatronics, it will focus on the control system that maneuvers the vehicle in hover and level flight, and in the transition between the two. The scope of the project only includes as much aerodynamics as necessary and simplifications were allowed in airfoil design and airflow parameters.

To evaluate the concept a scaled prototype was built, which has the necessary functions to demonstrate the aforementioned control system.

Furthermore, neither the cost of manufacturing the full scaled vehicle nor laws and legislations regarding flying this will be considered in this project. Safety is not a focus for the project team but may be considered throughout the project.

2 State of the Art

This section will present the current state of the art concerning different areas that could be interesting and useful for the development project, including aerodynamics, wing types, motors, energy, control theory among others.

2.1 Aerodynamics

In this chapter the basics of how to generate lift is discussed together with the latest wing configurations and their respective advantages and disadvantages.

Ever since the Wright brothers performed the first powered flight in 1903 [2] the field of aerodynamics and flight have been heavily researched, tested and optimized. The reason that all commercial airplanes today look very similar (long wings, high aspect ratio, sleek fuselage) is that it is the optimal way of transporting multiple passengers in terms of cost. Even though the focus of this project is not on aerodynamics it is still important to research and understand aerodynamics and the latest wing designs in order to create an efficient vehicle that fulfills the requirements.

Every aircraft, whether it is a fixed-wing airplane, quad-copter or rocket (although rockets are not generally classified as a subclass of aircraft) needs a way of generating lift. Simply put, lift is the upward acting force counteracting the force of gravity and allowing the aircraft to take off from the ground and subsequently stay in the air. The lift can be generated either by thrust from engines directed downwards or by pressure differences between the upper and lower side of an airfoil. Conventional airplanes use lift generated by their wings and they have engines providing thrust in the direction of travel and subsequently they get drag caused by friction in air flowing around the fuselage and wings, resulting in a force opposed to the direction of travel. All of these forces are presented in Figure 1.

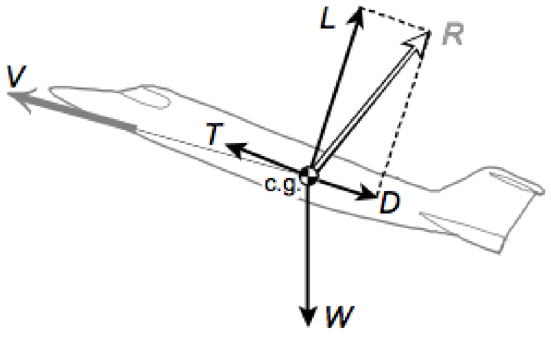


Figure 1: Forces acting on an aircraft.

To calculate the lift given by a specific airfoil it is required to look more closely on how the airflow, or freestream, is hitting the airfoil. This is illustrated in Figure 2.

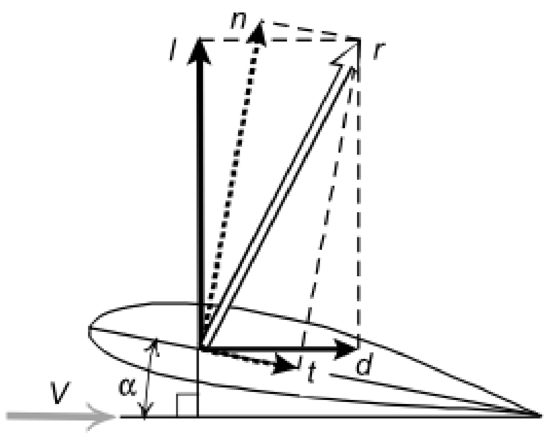


Figure 2: Forces acting on an aircraft.

The lift generated is dependent on the density of the air ρ, the speed of the freestream v, the specific lift coefficient c_l and the surface area of the wing S and is calculated by

$$L = \frac{1}{2}\rho v^2 c_l S. \tag{1}$$

The drag is calculated with

$$D = \frac{1}{2}\rho v^2 c_d S \tag{2}$$

where c_d is the specific drag coefficient which are calculated using

$$c_l = 2\pi\alpha \tag{3}$$

where α is the angle of attack as indicated Figure 2. C_d can also be described as

$$c_d = c_{d0} + c_{di} \tag{4}$$

where

$$c_{di} = \frac{c_l^2}{\pi ARe} \tag{5}$$

and c_{d0} is taken from the Table 1 and AR is the aspect ratio which is calculated using

$$AR = \frac{S^2}{A} \tag{6}$$

where S is the wing span and A is the total wing area.

	Aircraft type value	c_{d0}	e
1	Twin engine piston prop	0.022 – 0.028	0.75 – 0.8
2	Large turboprop	0.018 – 0.024	0.8 – 0.85
3	Small GA with retractable landing gear	0.02 – 0.03	0.75 – 0.8
4	Small GA with fixed landing gear	0.025 – 0.04	0.65 – 0.7
5	Agricultural aircraft with crop duster	0.07 – 0.08	0.65 – 0.7
6	Agricultural aircraft without crop duster	0.06 – 0.065	0.65 – 0.75
7	Subsonic jet	0.014 – 0.02	0.75 – 0.85
8	Supersonic jet	0.02 – 0.04	0.6 – 0.8
9	Glider	0.012 – 0.015	0.8 – 0.9
10	Remote controlled model aircraft	0.025 – 0.045	0.75 – 0.85

Table 1: Typical values for c_{d0} and e for several aircraft [3].

Observing Equation 1 and Equation 2 one can conclude that both the lift and the drag increase linearly with increased wing area.

In order to generate sufficient lift all that is needed is to have a high angle of attack. Although increasing the angle of attack and subsequently the lift also the lift induced drag is increased. This can be understood by incorporation Equation 5. It then becomes apparent that it is also necessary to have a high aspect ratio, AR reduces the drag and thus increases the efficiency.
Furthermore, by fixing parameters such as air density ρ and varying the design parameters such as angle of attack α optimization curves can be drawn as in Figure 3.

For a given airfoil the measurement of efficiency is the fraction lift over drag $\frac{L}{D}$ or $\frac{c_l}{c_d}$ which represents how much lift you get at the cost of drag, as represented in Figure 5.

While the latest commercial aircraft use many advanced mechanical flaps, slats and slots (as pictures in figure 6) in order to change the angle of attack and move the operating points on the curves in figure 5 depending on the conditions the scope of this project will include selecting an optimal fixed angle of attack.

2.2 Wing types

One of the most important design parameters of the vehicle that is to be built is the selected wing configuration. Therefore many possible solutions were evaluated early on. Everything from conventional wings, mono wings and duo wings to complicated canard configurations were evaluated. The most reasonable ones are presented in the sub sections below.

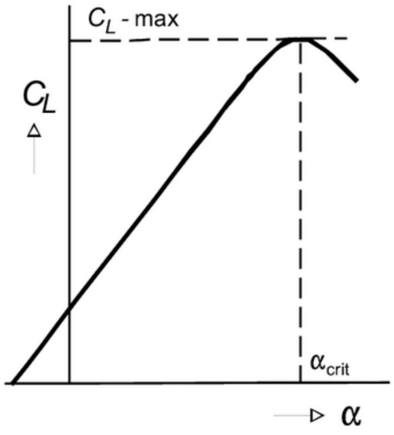


Figure 3: C_L alpha curve [2].

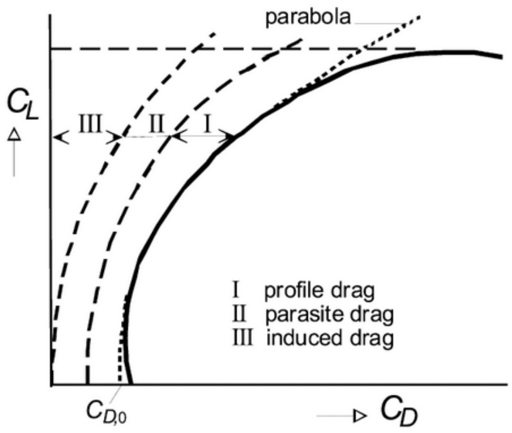


Figure 4: $\frac{C_L}{C_D}$ curve [2]

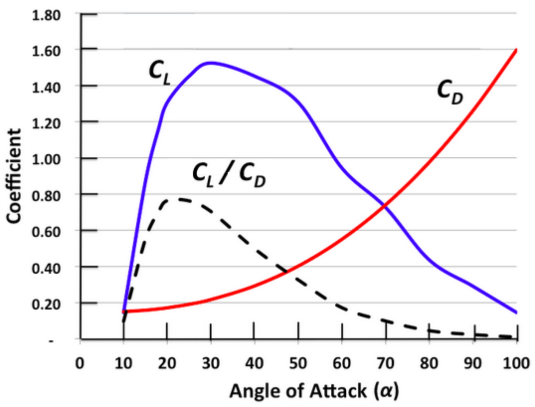


Figure 5: $\frac{\frac{C_L}{C_D}}{\alpha}$ curve [2].

2.2.1 Conventional wing

The conventional wing configuration used on all modern commercial aircraft, consists of two wings, one bigger for generating lift upwards and one smaller in the rear generating a force downwards for control and stability, as indicated in Figure 7. This configuration has the advantage of being statically stable, which means that any change in pitch will cause a torque in the opposite direction

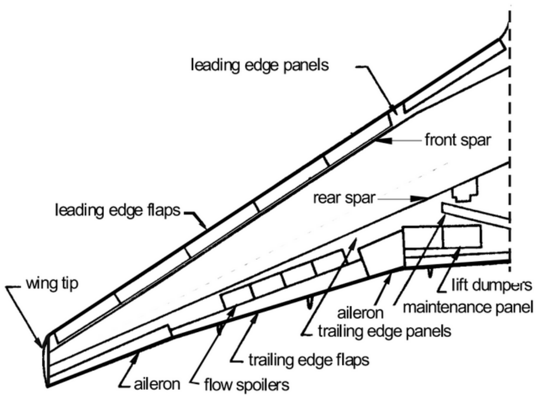


Figure 6: Common wing devices on a modern commercial aircraft [2].

and the aircraft will converge back to stable forward flight without the need for a control system. This is achieved by having the center of mass positioned in front of the center of pressure (the point where all of the lift force of the main wing) whilst the tail wing is providing a downward force.

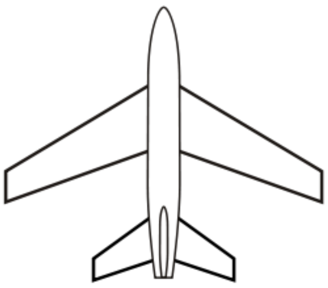


Figure 7: Conventional wing configuration.

While the conventional wing type has many advantages in terms of aerodynamic stability it also has some disadvantages of requiring a large wing span which causes problems when adapting the vehicle to the footprint of a car. With this configuration the aspect ratio can be very high in order to have an efficient aircraft but it also puts high demands on the structural stability of the wing construction.

2.2.2 Duo wing

In a duo wing configuration the main wing can be smaller since there are two wings generating lift and the aircraft can be designed with a smaller wingspan compared to conventional wing design. However the two wings will create interference and a down-wash effect (air moving down caused by the first pair of wings affecting the second pair of wings) can occur resulting in less lift and more induced drag from the wing furthest aft. Although the lack of a tail wing generating negative downward forces reduces the drag in this configuration.

Potential problems with canard wing configurations (figure 8) are if the center of gravity is misplaced, a dangerous stall situation can occur if the main wing (the back wing) stalls before the smaller wing in the front resulting in the aircraft falling without reducing the angle of attack resulting in a situation without the possibility to recover. This is not a problem for the tandem wing (figure 9) where the front wing will take 50 % or more of the load and therefore stalling before the wing furthest aft resulting in a dive if the front wing stalls, reducing the angle of attack and increasing the speed.

2.2.3 Mono-wing

A mono wing design is where the fuselage is fused together with the wings, creating a single aerodynamic profile combined with the characteristics of the non existing tail elevator, as seen in

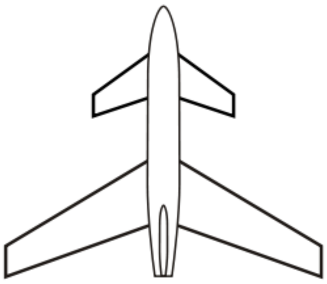


Figure 8: Canard wing configuration.

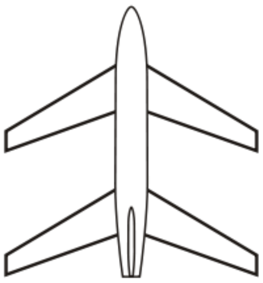


Figure 9: Tandem wing configuration.

figure 10.

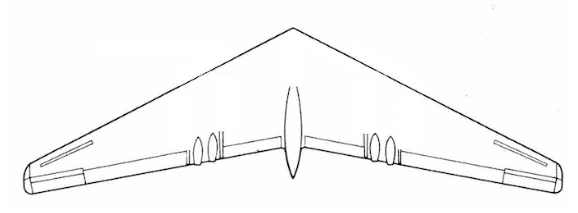


Figure 10: Mono wing configuration.

To generate the counterforce needed for stability the flying wing is often swept and negative lift is generated at the tips of the wing creating a bit of torque. The mono-wing is theoretically the most aerodynamically efficient design configuration for a fixed wing aircraft due to its naturally high aspect ratio, low drag and it also offers high structural efficiency. The main downside of the flying wing is that if it should also possess VTOL capabilities it needs to be in what is called a tail-sitter configuration, as seen in figure 11.

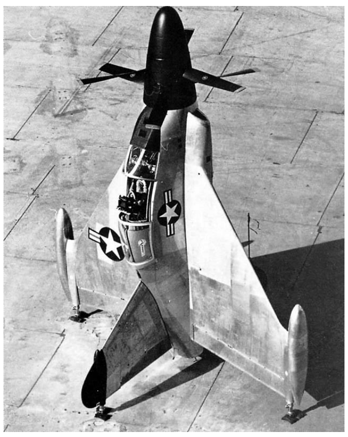


Figure 11: Tail-sitter configuration.

The tailsitter setup makes it vulnerable to wind gusts during take-off and landing and it has an added complexity for pilots entering and exiting the cockpit, where some sort of gimbaling cockpit mechanism would have to be adopted. Another problem with the flying wing design is the need for bigger ailerons since the distance from the force to the center of mass is less compared to most conventional wing designs, resulting in an increase in weight. Without a rudder the controlling of the yaw is often controlled by the speed or direction of the motors, or by creating drag with spit ailerons or spoilers (resulting in reduced lift) which requires some extra control features.

2.3 Vertical Take-Off and Landing

A vertical take-off and landing (VTOL) aircraft has the ability to take off, hover and land vertically removing the need of takeoff and landing strips. There are other similar classifications depending on the take-off lengths, CTOL (conventional take-off and landing), STOL (short take-off and landing) and STOVL (short take-off and vertical landing).

The most famous fixed-wing VTOL aircraft is the Harrier Jump Jet, using a single engine directing the propelled air downwards to generate lift vertically.

To achieve VTOL with todays aircrafts, it in most cases, requires parts of the craft to be able to tilt or rotate around a horizontal axis. This allows for the thrust to gradually be redirected from downwards to forwards in a safe manner as the needed lift from the rotors are reduced by the increased airspeed. The aircraft is thereby able to profit from the benefits of VTOL and level flight.

2.3.1 Tilt wing

This VTOL configuration has motors fixed on the wing, generating thrust in line with the wings as a conventional aircraft. The wings are then tilted so the thrust generated by the motors creates a force lifting the plane vertically. To convert into conventional flight the wings tilts back while the plane is in the air. An operation that is being carefully executed by a computer to ensure it is secure [29]. The transition from VTOL to horizontal flight can be seen in figure 13. To land the aircraft again it needs to decelerate and tilt the wings back to compensate the loss of lift from the wings. In this configuration the losses are substantial as the tilting reduces wing efficiency. Once it has decelerated low enough to be hovering, it can land as a helicopter.

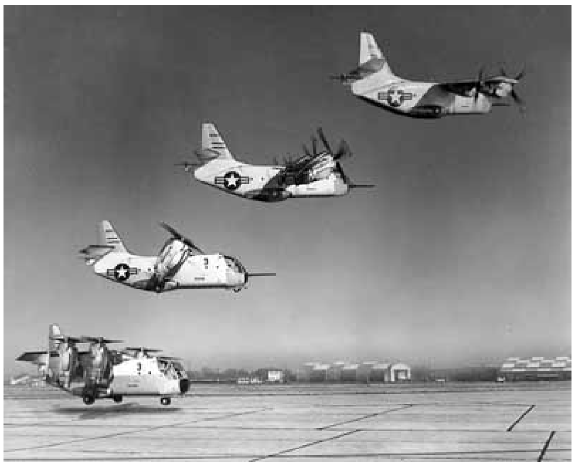


Figure 12: The transition from VTOL to horizontal flight by the LTV-Ryan-Hiller [30].

2.3.2 Rotor-tilt

The rotor-tilt is one of the more common VTOL configurations, where only the motors are being tilted. This allows it to, in the same manner as the tilt-wing, tilt the thrust to the needed angle. The operation is handled by a computer that can adjust for disturbances as needed, keeping it secure. Typical for this configuration is the use of nacelles at the wingtips that hold the motors. The transition is made in a similar manner to figure 13, with the difference being, as before mentioned, only the nacelles tilt instead of the entire wing. To land the aircraft again it decelerates and tilts the rotors back again to compensate for the lost lift. The rotor-tilt biggest benefit in comparison to the very similar tilt-wing configuration is the immediate generation of vertical lift from the wings even at low speeds.

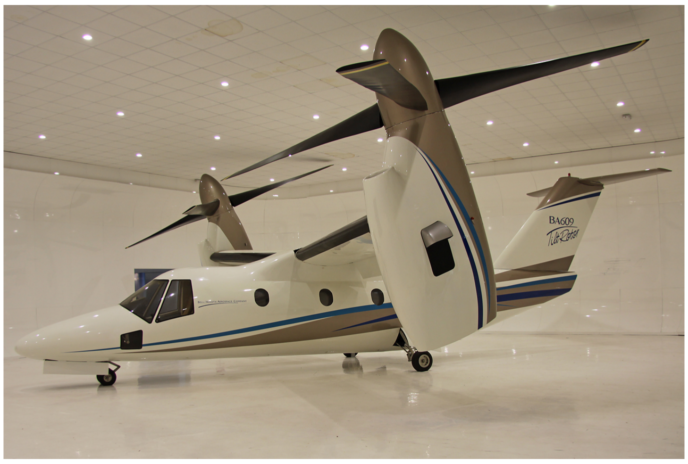


Figure 13: Tilted rotor type airplane.

2.3.3 Pitching

The pitching VTOL configuration is the mechanically simplest solution for VTOL. The aircraft accelerates and pitches hard in either direction. This will force the plane to rise while at the same time leveling out until it is parallel to the ground, which can be seen in figure 14. This configuration needs no extra tilting mechanisms, instead it has extra requirements on the motors and flaps.

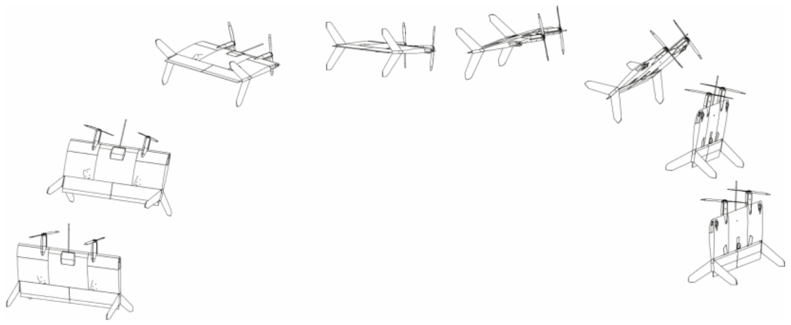


Figure 14: Hover, level, and transition of Wingtra

Landing is achieved by reducing the pitch and then decelerate, causing the aircraft to transition into hover mode [22]. The largest benefit from this configuration is the simplicity, that reduces moving parts and control complexity.

2.4 Existing Designs

This section will describe the designs of existing aircrafts that can achieve VTOL and forward flight in some form. The aircrafts described are of varying size and capability, though the main thing looked at is their concept for VTOL and transition to level flight.

2.4.1 The Lilium Jet

This aircraft was developed by Lilium Aviation in southern Germany, Figure 15. The goal of the company is to create an aircraft with a range of 300 km and a cruise speed of 300 km/h. The aircraft is addition to that emission free, by using batteries and electrical motors instead of combustion engines. In addition to that the electrical motors and ducted fans used keep noise levels low [19].

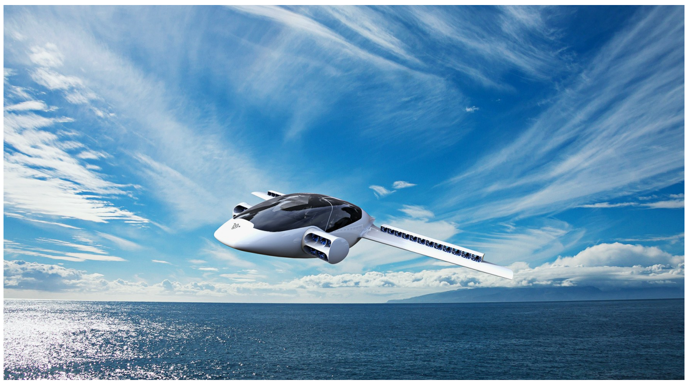


Figure 15: The Lilium Jet

It is an aircraft capable of VTOL by using a multitude of individual engines spread out on the wings. These motors power ducted fans that generate the needed thrust. These packages of motor and fan can be tilted on an axis, along the wing. The setup enables the transition from vertical to level flight, making it a tilt-rotor aircraft. The prototype in its current configuration has wings at the aft of the aircraft, while at the front it is equipped with extra motors for added thrust. It has performed successful unmanned test-flights during the year 2017, proving the concept of VTOL and level flight capability with this configuration.

2.4.2 Greased Lightning 10

The NASA GL-10 is still in its design and testing phase, the goal is a VTOL aircraft used for small package delivery, farmland surveillance and other mapping tasks. A scaled up version is also being considered, capable of carrying up to four persons, figure 16.

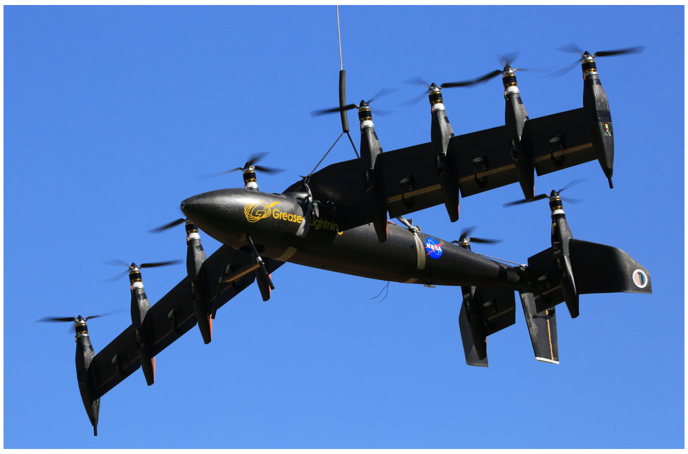


Figure 16: Greased Lightning 10

The Greased Lightning is a til-twing aircraft, that is capable of VTOL and level flight. Tilt-wing aircrafts , as before mentioned, are able to tilt its wings to adjust the angle of the thrust. The GL-10 has wingspan of 6.1 m on with a total of 10 motors. These motors are powered hybrid engines meaning it is not emission free, but has lower emissions than many other aircrafts [20].

2.4.3 WingtraOne

This tail-sitter aircraft is manufactured by Wingtra. It is most commonly used for land survey, data gathering or hobby purposes, depending on need, see figure 17 [21].

The WingtraOne is only a RC-model sized aircraft with a wingspan of 1.25 m, it has however achieved a flight time of about 55 minutes. This at a cruising speed of 55 km/h in horizontal flight [21]. The VTOL transition of this concept is one of the simpler ones, as it uses pitching. This VTOL configuration, as mentioned earlier, requires no extra mechanics, but add requirements on

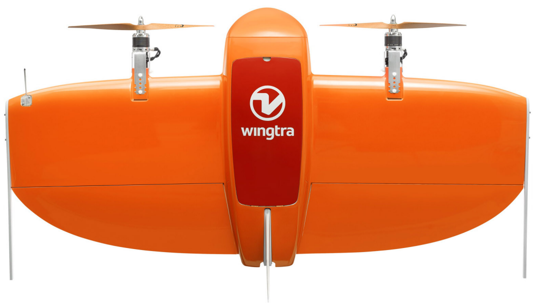


Figure 17: WingtraOne

motor and flap performance[22]. Scaling up this concept to a manned aircraft will however result in added complexity as the cockpit must be able to gimbal.

2.4.4 Ehang 184

The Ehang 184 is being developed by the Chinese company named Ehang. The company's goal is to develop a low altitude autonomous aerial vehicle capable of short to medium distance travel. The aircraft is currently capable of 25 minutes of flight, with a maximum cruising speed of 60 km/h and a maximum output of 152 kW. It is designed to have a payload capacity of 100 kg, which combined with the net weight of 240 kg the Ehang lands at a total weight of 340 kg [23].

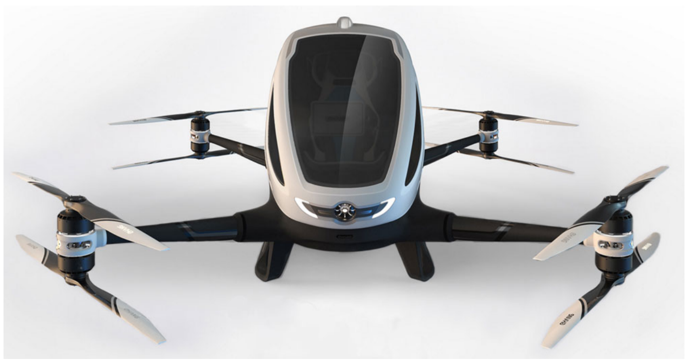


Figure 18: Ehang 184

The transition from vertical to horizontal flight is achieved by tilting the entire aircraft. This method is a derivative of pitching, as the entire aircraft pitched using the rotors. This pitching is also the most inefficient one. The motors need to provide lift and forward thrust as no wings are there to add lift, adding even higher requirements on the motors.

2.5 Hardware

2.5.1 Motor

When choosing a design for a VTOL plane one has to consider the amount of motors that are optimal. When looking at the existing designs it can be seen that almost all of them are using a design with more than just the minimal amount of motors required. Here, WingtraOne is the exception which most likely is because it's not intended to be a personal vehicle which the other three are. The extra motors gives redundancy which means that even if one or a few motors break down the vehicle can still land in a secure fashion. But if we start by ignoring the safety factor there are other parameters that should be considered.

The efficiency of the motors are quite relevant since the batteries that exists today has a very limiting amount of energy storage. Internal combustion engines have an efficiency that ranges between 25-50% but the efficiency of a standard brush-less DC-motor is around 90%. Since the energy we can bring is quite limited it has to be considered if there is a general difference in efficiency for a smaller or a bigger electric motor. There are several different losses in a brush-less DC-motor. Some are connected to the copper wiring, some to the iron core and some to other factors [31]. Considering these types of losses there is no evidence that the size of the motor should have a large impact on the efficiency. To prove this a study was conducted where several electric ducted fans of different sizes were looked at. To be able to compare them their thrust was divided by their input energy to see how much thrust they could provide per watt. The results of this can be seen in table 2

Motor name	Diameter [mm]	Net-thrust [kg]	Power [W]	$\frac{Net-thrust}{Power}$ $[\frac{g}{W}]$
Dr. Mad Thrust 40mm 8-Blade Alloy EDF	40	0.285	330	0.86
Dr. Mad Thrust 50mm	50	0.557	400	1.39
HK EDF64	64	0.961	520	1.85
Hobbyking 64mm Alloy EDF	64	0.815	850	0.96
Hobbyking 64mm Alloy EDF 4300kv - 750w (4s)	64	0.905	750	1.21
Dr. Mad Thrust 68mm	68	2.35	1950	1.21
EDF64 with ADH300	75	0.343	252	1.36
Mercury Aluminum Alloy 104mm 11 Blade EDF	104	4.108	2550	1.61
Dr. Mad Thrust 120mm 12-Blade Alloy EDF	120	5.419	6000	0.90

Table 2: Commercially available EDF.

From this it can be seen that there are differences between the motors and brands but there is no apparent difference in efficiency between a bigger or smaller motors.

Another important design concept to consider is air separating from the wing during flight. Separation occurs when the angle of attack becomes to large [32] and is usually countered by using Spoilers on commercial airliners [33]. This is something that could also be countered with a line of smaller motors along the wing, like it has been done on the Lilium Jet. What the motors do is that when the air wants to separate from the wing they will suck the air through over the wing and keep the airflow laminar for higher angles of attack.

In that aspect the Greased Lightning 10 and the Lilium Jet are quite similar that they have several propellers mounted along the wing. But the difference is that Lilium uses EDF´s while Greased Lightning uses more conventional propellers. The purpose of the ducted fan is that they will provide a more centered airflow which will make the thrust higher for their size compared to a conventional propeller and motor. EDF's are also quieter than propellers and safer to start close to other people which is a necessity if the vehicle is to be used in a city.

2.5.2 Energy

The technology for providing electrical power in our application would lie in utilizing batteries and their promising future. Batteries on the market today have specifications which can be found in Table 3, batteries there are general use lithium-ion battery, [4].

It is concluded that batteries are in a good state at the moment but not good enough for our application, but it can be identified that the energy density of batteries are currently improving drastically, see figure 19, for reference [5].

Batteries are currently improving at a fast rate, we see trends that many companies are investing in new research areas as well as improving the current technology even further, there are several examples of this in for example Tesla's Gigafactory [8] and the new Swedish investment in Northvolt [7], proposed to be the largest battery factory in Northern Europe. This trend is backed up by previous articles but also in how the cost for batteries are decreasing [6], see Figure 20. It can be seen that the combination of a higher energy density together with cheaper batteries is setting the baseline for a bright future.

Description	Value
Specific Energy	100-265 $\frac{W \cdot h}{kg}$
	(0.36-0.875 $\frac{MJ}{L}$)
Energy Density	250-676 $\frac{W \cdot h}{L}$
	(0.90-2.43 $\frac{MJ}{L}$)
Specific Power	250-340 $\frac{W}{kg}$
Charge/Discharge Efficiency	80-90 %
Energy/Consumer Price	2.5 $\frac{W \cdot h}{US\$}$
Self-discharge rate (per month)	8% at 21 °C
	15% at 40 °C
	31% at 60 °C
Cycle durability	400-1200 cycles

Table 3: State of the Art Lithium-Ion Battery

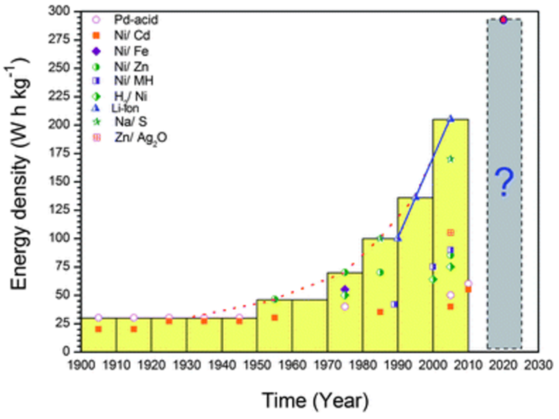


Figure 19: Energy density development since 1900.

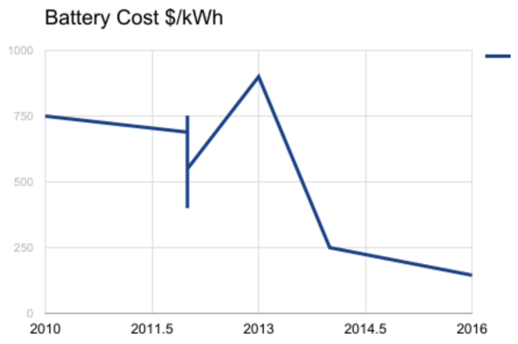


Figure 20: Battery cost development since 2010.

2.5.3 Material

The primary factors to be considered to choose the material is the stress on the parts of aircraft, including tension, compression, torsion, shear and bending, shown in figure 21. Depending on the stress on the components, the material should be selected to have different mechanical properties such as stiffness, strength, hardness and plasticity etc. Some components should also have strong resistance to corrosion. The material of an aircraft should be light and strong. It can be classified as metallic material and nonmetallic material [18].

Metallic material includes aluminum alloy, magnesium alloy, titanium and steel alloy. The aluminum is the most commonly used material in aircraft due to its benefits of high strength-to-weight ratio and corrosion resistance. Magnesium alloy is the world's lightest structural metal

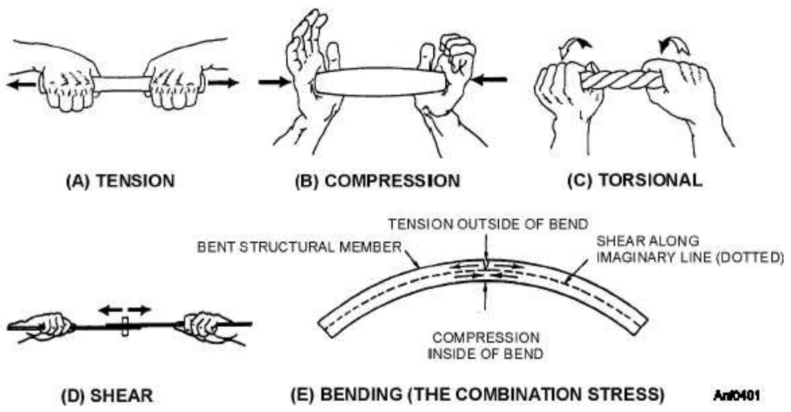


Figure 21: Specific action of stresses

which is commonly used in helicopters. But its low corrosion resistance limits its use. Titanium has excellent mechanical properties but is expensive compared to the other materials. It is usually applied in parts where steel is too heavy and aluminum too weak. The steel alloy is stronger but heavier than aluminum alloy.

Non-metallic material can be divided into transparent plastic, reinforced composite and carbon fiber materials. The transparent plastic is used in canopy, windshield and window of the aircraft. Reinforced plastic is usually made of several different materials in a sandwich-type which contains two surfaces of several layers and a center layer made in a honeycomb structure to enhance the strength. Composite and carbon fiber is used in high-performance aircraft now. They fulfill the requirement of the high strength-to-weight ratio. The process to manufacture carbon fiber for aircraft requires high technology.

Wing construction usually consists of a framework made of spars and ribs which are the main structure of the wing [18]. And the spars and the ribs are covered with skin. Most of the stresses in the spars are bending. So, in order to tolerate the bending stress, the main structure of spars and ribs should be made of aluminum alloy. And the surface cover can be made of carbon fiber to reduce the weight.

Propellers can be made of wood, carbon fiber or metal. Metal propellers are the heaviest among the three. If the propellers are lighter, the vibration of the motor can be dampened and more rotation speed could be obtained, since less load would affect the engine. Wood propellers are the cheapest and also the easiest to break. Regardless of the cost, the carbon fiber is the best choice for the material to make propellers.

The Lilium aircraft design claims to use the customized carbon fiber material for the whole jet structure in order to reduce weight. Meanwhile, Ehang uses reinforced composite with carbon fiber and epoxy for the main frame structure and aerial aluminum alloy for other stiff structures.

2.5.4 Electronics

The Pixhawk 2 is a high-performance open source autopilot system with support for different kinds of airframes such as VTOL, multi-copters, planes, boats and general robotics [24]. It is an ideal system that is used by hobbyists, academic and industrial communities. The Pixhawk 2 includes a triple redundant vibration damped IMU, which ensures that the readings are accurate. The IMU consists of 29 MEMS sensors, having 3 accelerometers, 3 gyroscopes, 3 magnetometers and 2 barometers. The flight controller also comes with 14 PWM outputs which can be used for servos and ESC control to drive the motors. It is equipped with two power supply inputs, which allows redundancy in case one of the power systems is breaking down [25]. The controller receives data from the IMU, GPS and compass that is used for balance and stabilization during flight.

2.6 Control

This section will describe the methods used to control an airborne vehicle while hovering, levering and transitioning between the two.

Figure 22: Pixhawk 2 flight controller

The most common way to define the system equations is to use either the Newton-Euler formulation or the Lagrangian formulation, where the former describes the system behavior in terms of forces and momentum and the latter uses generalized coordinates and describes the system using equations of energy and forces [13].
The system is a highly coupled MIMO system, and for this type of control problem a LQR has proven to be an efficient method [34]. Another approach to the control problem is to use the PID controller we all know and love.

2.6.1 Hybrid Control

The system as a whole can be described as a hybrid automata, as seen in figure 23 [9].

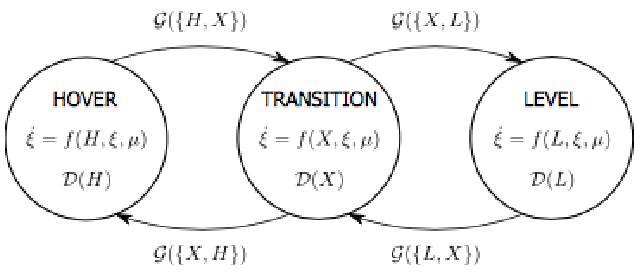


Figure 23:

The three states $Q = (H, X, L)$, hover, lever and transition, each have their own constitutive equations that describes the dynamics of the system in that state. The guards $G(Q_i, Q_j)$ describes the conditions that need to be fulfilled in order to switch from one state to the other.

When designing a controller for a hybrid system, one must keep in mind that for the system to be stable, it is not sufficient that each of the states are stable. To ensure stability of the system as a whole there are several approaches to use, such as the Common Lyapunov approach or the Multiple Lyapunov approach [17].

2.6.2 Vertical control

The propellers can be divided into clockwise propellers and counterclockwise propellers. The clockwise propeller will generate upward thrust when it rotates clockwise while the counterclockwise propeller generates upward thrust in an opposite way, shown in the figure 24. In a quad-rotor design, it would generate both thrust and corresponding reaction torque to the body when propellers rotate. The reaction torque might make the body spin if it is not correctly compensated. The way to eliminate the reaction torque is to pair a clockwise propeller with a counterclockwise propeller. The clockwise and counterclockwise propeller would generate opposite torques to the body so the reaction torques will be compensated and they will not have impacts on the aircraft body. The typical quad-rotor design is shown in figure 25 below.

When the thrust generated by the propellers equals the weight of the whole aircraft, the aircraft remains in hover mode. It can be stable at the same level of altitude. When the thrust is increased and larger than the weight, the aircraft would lift vertically. On the other hand, when the thrust

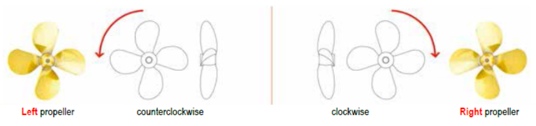


Figure 24: Typical propeller direction for both left and right type.

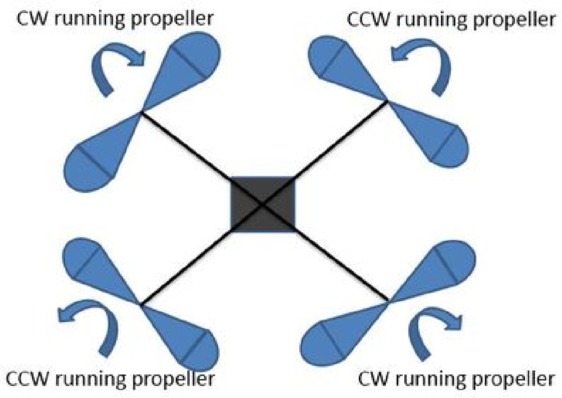


Figure 25: Typical quad-rotor propeller design.

is less than the weight, the aircraft would lose in altitude. The Ehang 184 is based on the quad-rotor model for VTOL. Ehang uses eight motors instead of four as a redundancy in case of motor malfunction, the aircraft can still work sufficiently. The tailsitter WingtraOne, uses one clockwise propeller and one counterclockwise propeller to generate thrust. Besides, two control surfaces are designed to increase the stability when taking off.

The output of the aircraft should be six degrees of freedom when flying. The six degrees of freedom is shown in figure 26 below. However, the input of the quad-rotor and the tail-sitter is only four actuators. The control system is an under-actuated system [15]. Given a fixed size of the aircraft, the maximum propellers size of a quad-rotor would be bigger than the maximum propellers size of hex-copter which means the efficiency of the propellers in quad-rotor is higher than the hex-copter. Though the under-actuated system can increase the efficiency of the motors, the control system is more difficult to design to make the system stable. Because the under-actuated system is a special nonlinear control system and difficult to be linearized.

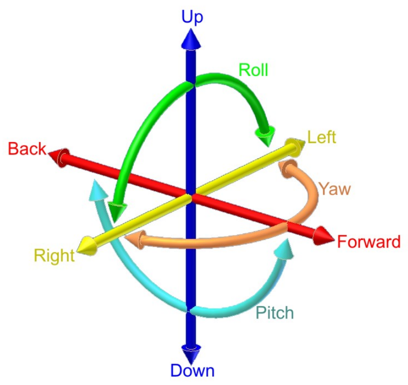


Figure 26: Six degrees of freedom.

The Lilium aircraft and NASA Greased Lightning use multiple motors and propellers for take off. These control-systems are over-actuated systems and the redundancy of the control increases. The control allocation to the models can be robust [16]. If some of the motors are broken, the aircraft can still take off and land vertically as usual with the optimal control allocation strategy and the redundancy of the control.

2.6.3 Horizontal control

Conventional aircrafts with a fixed-wing design are using three primary flight control surfaces (i.e. ailerons, elevators and rudder) to adjust and control the aircrafts flight attitude during horizontal flight [26]. Aircrafts with other type of fixed-wing configurations are using the same basic principle for the control surfaces, but the control surfaces themselves can be placed differently. Movement of any of the three primary control surfaces mentioned above creates a difference in airflow and pressure over and around the airfoil. This generates forces on the aircraft that allows the pilot to control the movement of the aircraft about the three axes of rotation [27].

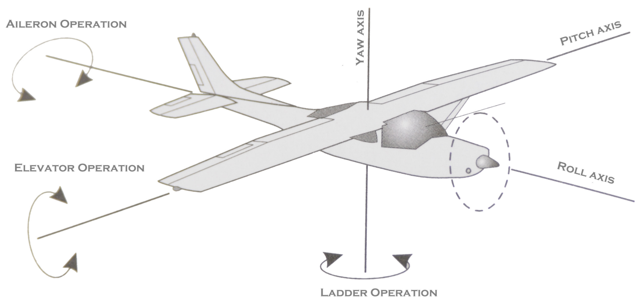


Figure 27: Pitch, roll and yaw in an aircraft

2.6.4 Transition

How the transition between hovering and leveling is controlled is highly dependent on the design of the aircraft. During the transition, the control objectives are to remain stable with a constant altitude while quickly gaining forward speed. The transition is the part of the flight state that requires the most thrust from the motors, substantially

In a tail-sitter configuration, like WingtraOne, the system does not require a switched controller which simplifies the control task. The transition from hovering to leveling is made by increasing the pitch angle to generate lift from the wings. The major drawback with the transition control of the tail-sitter is the transition back to hovering. This can cause unstable behavior due to nonlinear aerodynamic effects caused by the sudden increase of the angle of attack [14]. The transition trajectory of WingtraOne can be seen in figure 14.

For a tilt-rotor design, such as Lilium Jet, the transition is done by adjusting the speed and angle of the rotors to create an airflow over the wings that generates lift. The vehicle leaves the transition state and enters the horizontal control state when the velocity has increased to the point where sufficient lift is generated, which corresponds to when

$$\dot{x}^2 > \frac{mg}{S}, \tag{7}$$

where $\dot{x}$ is the velocity in the horizontal direction, m is the mass of the vehicle, g is the gravitational constant and S is the wing span [12]. When the condition is accomplished, the vehicle will operate like a regular airplane.

3 Future Work

This section will touch on how the project team will continue their work during the next semester and next phase of the project.

3.1 Project Plan

The project plan is summarized in a GANTT-chart that can be seen in appendix A. When the ordered parts have arrived at the start of the semester in September, the testing of the models subsystem will begin. During this phase any needed custom parts will be developed in parallel. As testing and construction is being completed the team will move over towards a first implementation. The end product is likely to require more than one iteration before reaching satisfactory results. A scrum model will be followed to work in an agile and efficient environment to maximize progress. For more details regarding construction and testing, see sections below.

3.2 Scalability

One large focus for the project will be to build a prototype in a scaled down size, which then requires the team to research more on how the scalability of the prototype works. One deliverable will be to produce a concept that could be implemented in a larger scale. This has not been included in the current scope, so this is something for the continuous work in the fall.

3.3 Component Selection

The next steps for the project is the selection of components for all parts of the vehicle. The electronic components that are currently needed are: motors, electronic speed controller, batteries, radio receiver & transmitter and a flight controller. Some optional components that could be utilized but not necessary for the project is a camera, additional speed sensors and a GPS.

The most important parts are the motors, where some motors in different ranges will be purchased and investigated to find the most suitable option.

The current working progress is approximating the weight of the vehicle and calculating what will be the optimal motor size and amount. Efficiency of the motors available is also taken into account when selecting the individual components.

After the motor and corresponding electronic speed controllers have been chosen the rest of the components are easier to select. The criteria for the batteries (except for having the correct voltage) is mainly to be able to provide enough flight time for the testing. There is still a need to decide on the gimbal mechanism before selecting any parts for it, but ideally it will be ran on a servo, which is an easier component to acquire.

3.4 Testing

Before the prototype is build requirements will be set for each specific subsystem. They will be designed to be easily measured and will in return be useful for the validation and verification part. These requirements will be implemented and utilized in the project management tool Polarion, [28]. Test cases will also be managed in this software to maintain traceability and quality.

All subsystems need to be tested, including flight controller, motor control and communication. The verification and validation phase is an iterative process that will follow the agile method of working, with sprints for each specific sub-team. Also, the gimbal mechanisms that will be developed during the project, the vehicle body and the final product will be required to go through this verification and validation process.

3.5 Construction

A scaled prototype will be manufactured, that will mainly consist of a vehicle body and frame along with a gimbal mechanism. The vehicle body and frame will be built in stages starting with a small frame upon which the concepts first can be tested. This frame will then be gradually fitted more body parts as concept progress comes along. The goal is then to build a larger scale version of said body. This version will be an aluminum frame encased in styrofoam and with coated in carbon fiber.

A gimbal mechanism will be custom made for the aircraft. This is also planned in two stages with first a mechanically simpler version that fulfills set requirements. The goal is to then replace it with a more efficient one. During this phase any other needed custom parts will also manufactured if needed.

4 Discussion

Designing a plane is a complex and difficult task, especially if the plane is designed to be optimized and meet harsh requirements. The first obstacle is to choose all the basic design parameters to create a basic platform, such as size, wing configuration, propulsion system and control systems. It is necessary to combine them together while taking aerodynamics and overall structural stability of the plane into account.

This is a product aimed to eventually be sold to civilians and must therefore be somewhat recognizable to the general public. This was one of the main reasons the tail-sitter design was rejected, combined with the fact that the cockpit had to rotate in order to keep the passengers leveled. The canard wing design instead radiates stability and is what people think of when they imagine a flying car, important aspects to the stakeholder. The conventional wing configuration was not suited for the task or VTOL in a restricted area since the configuration requires a quite long body to create the momentum needed for stability. If however a conventional wing design could work as a VTOL aircraft not exceeding the set size it would be easier to control while in horizontal flight due to its relaxed stability property.

The plane's center of gravity and overall balance will affect how the plane handles and will require precision work during the design and build phase. This is not the focus of the course but it's critical for the success of the project and can somewhat be avoided or eased by outsourcing manufacturing of complicated and core parts.

The most complicated part of the aircraft is the mechanism which allows the motors to tilt. The mechanism should be robust enough to tolerate a slight crash however it ought to be as light as possible. In the beginning, a simple version of the mechanism will be implemented to avoid hindering or delaying the rest of the project. The simple mechanism can be developed into a more complex and efficient design parallel to the development of the main build, and will in the end be implemented in the final build. The simple design consists of a servo motor and a planetary gear, two bearings and a carbon fiber rod to mount the motors on.

An advanced control system is required to ensure stable flight in hover, level and transition mode. To increase the safety of the passengers in the aircraft there must be redundancy in both software and hardware. Therefore, it is preferable to use an over-actuated control system.

The aircraft has three different flight modes; hovering, leveling and transitioning, where each of them has separate control objectives and equations describing the dynamics. Stability must be ensured in the system as a whole, for example using the Common Lyapunov approach or the Multiple Lyapunov approach. To somewhat simplify the control task the Pixhawk has been chosen as a flight controller, which has some built in functions that helps to maneuver and stabilize the aircraft.

Because the team consists of students all attending the same master's program, mechatronics at KTH, the lack of variety in the group limits the width of the project. The team also lack experience working with the components vital to this project and this will result in an overall slower development process. This project is part of the Higher Course at the KTH mechatronics master's program, and must therefore be directly linked to mechatronics. That means the team will not be spending too much time on aerodynamics, construction and design aspects. Optimization of wing profile or aerodynamic design will therefore not take place and a simple wing profile will be picked from basic equations and guide rules.

4.1 Conclusion

The canard wing configuration has been selected as the best wing configuration since it can produce a lot of lift without having a big wingspan while still keeping a fairly high aspect ratio increasing the lift-to-drag ratio. The four wings also creates intuitive placements for the motors, which will be placed at the end of each wing in rows. These rows will have the ability to tilt 100° so they can be used to generate thrust in both vertical and horizontal directions. This design was also the most accepted and trusted design by the general public.

Multiple smaller EDFs will be mounted in rows on each wing. EDFs were chosen instead of open blade propellers since it adds a layer of security, both for the passengers and for people near the plane. EDFs are also easier to handle while tilting since they can be smaller and still produce the same thrust. The experienced noise can also be lowered using EDFs over props since the sound is directed by the ducts. Using multiple motors per wing instead of one helps with the space, the potential torque on the tilting mechanism and safety by having redundancy.

References

[1] Department of machine design, *Maskinelement Handbok*, Royal Institute of Technology, Stockholm, 2008.

[2] Flight Physics, *E. Torenbeek, H. Wittenberg*, Springer Science 2009.

[3] Aircraft Performance Analysis, *Sadraey M*, , VDM Verlag Dr. Müller, 2009.

[4] Rechargeable Li-Ion OEM Battery Products, *Panasonic*, "https://web.archive.org/web/20100413182032/http://www.panasonic.com/industrial/batteries-oem/oem/lithium-ion.aspx" , 2010.

[5] Thermodynamic analysis on energy densities of batteries, *C. Zu, H. Li*, "http://pubs.rsc.org/en/content/articlehtml/2011/ee/c0ee00777c", Royal Society of Chemistry, 2011.

[6] Chevy Bolt Production Confirmed for 2016, *J. Cobb*, "http://www.hybridcars.com/chevy-bolt-production-confirmed-for-2016/", Hybrid Cars, Retrieved: April 16 2017, 2015.

[7] Energimyndigheten stöttar satsning på ny svensk batterifabrik, *Energimyndigheten*, http://www.energimyndigheten.se/nyhetsarkiv/2016/energimyndigheten-stottar-satsning-pa-ny-svensk-batterifabrik/, Retrieved: April 16 2017, 2016.

[8] Tesla Gigafactory, *Tesla*, "https://www.tesla.com/gigafactory", Tesla, Retrieved: April 16 2017, 2014.

[9] Autonomous Transition Flight for a Vertical Take-Off and Landing Aircraft, *Pedro Daniel Graça Casau*, "https://fenix.tecnico.ulisboa.pt/downloadFile/395142046008/tese.pdf", 2017.

[10] Introduction to Robotics, *Harry Asada*, "https://ocw.mit.edu/courses/mechanical-engineering/2-12-introduction-to-robotics-fall-2005/lecture-notes/chapter7.pdf", 2017.

[11] Design and control of an unmanned aerial vehicle for autonomous parcel delivery with transition from vertical take-off to forward flight, *Menno Hochstenbach, Cyriel Notteboom*, ”$http : //www.tmleuven.be/thesisprijs/laureates/2014Hochstenbach_Notteboom_samenvatting.pdf$”, Retrieved: April 16 2017.

[12] Transition Flight Control of the Quad-Tilting Rotor Convertible MAV, *Gerardo Florest and R. Lozano*, ”$https : //www.researchgate.net/profile/Gerardo_Flores3/publication/258517455_Transition_Flight_Control_of_the_Quad-Tilting_Rotor_Convertible_MAV/links/0c9605288847834af5000000.pdf$”, Retrieved: May 16 2017.

[13] Modeling and Simulation for Automatic Control, *Olav Egeland and Jan Tommy Gravdahl*, "https://pdfs.semanticscholar.org/9df2/34714fa70751c9ee4fee316e6e56ee02c4e9.pdf", 2002.

[14] Full Attitude Control of a VTOL Tailsitter UAV, *S. Verling, B. Weibel, M. Boosfeld, K. Alexis, M. Burri, and R. Siegwart*, "http://ieeexplore.ieee.org/document/7487466/metrics", 2016.

[15] Introduction to feedback control of underactuated VTOLvehicles: A review of basic control design ideas and principles, *M. D. Hua and T. Hamel and P. Morin and C. Samson*, 2013.

[16] Benefits of over-actuation in motion systems, *M. G. E. Schneiders and M. J. G. van de Molengraft and M. Steinbuch*, 2004.

[17] Stability of Hybrid Systems, *Michael S. Branicky*, "https://www.kth.se/social/files/569bc6f9f2765433866d71d1/ec

[18] AIRCRAFT BASIC CONSTRUCTION, *test*, ”$http : //home.iitk.ac.in/mohite/Basic_construction.pdf$”, 2010.

[19] The Lilium Jet, *Lilium*, "https://lilium.com/", Retrieved: May 16 2017.

[20] NASA GL-10 Tilt-Wing VTOL UAS Flight Validation Experiments, *William J. Fredericks, David D. North, Mark A. Agate and Zachary R. Johns*, AIAA Aviation 2015 Conference, NASA Langley Research Center, Hampton, VA, 23681, 2015.

[21] Wingtra - Aerial Robotics, *Wingtra*, "https://wingtra.com/product", Retrieved: May 16 2017.

[22] Transition between Level Flight and Hovering of a Tail-sitter Vertical Takeoff and Landing Aerial Robot, *K. Kita, A. Konno and M. Uchiyama*, Advanced Robotics, 24:5-6, p.763-781, 2010.

[23] EHANG|Offical Site-EHANG 184 autonomous aerial vehicle, *EHANG*, ”$http : //www.ehang.com/ehang184/$”, Retrieved: May 16 2017.

[24] Pixhawk Autopilot specification, *PX4 autopilot*, ”$https : //pixhawk.org/modules/pixhawk$”, Retrieved: May 15 2017.

[25] Pixhawk Autopilot description, *RC Tech*, ”$http : //www.rctech.se/index.php?route = product/product\&product_id = 453$”, Retrieved: May 15 2017.

[26] Flight control surfaces, *Wikipedia*, ”$https : //en.wikipedia.org/wiki/Flight_control_surfaces$”, Retrieved: April 17 2017.

[27] Pilot’s Handbook of Aeronautical Knowledge, chapter 6, *U.S. Department of Transportation, Federal Aviation Administration*, ”$https : //www.faa.gov/regulations_policies/handbooks_manuals/aviation/phak/media/08_phak_ch6.pdf$”, 2016.

[28] Application Lifecycle Management (ALM) Requirements Management QA Management | Polarion Software, *Siemens*, https://polarion.plm.automation.siemens.com/, Siemens Product Lifecycle Management Software Inc., 2016, Retrieved: May 16 2017.

[29] Wingtip mounted counter-rotatin proprotor for tiltwing aircraft *James K. Wechsler, John W. Rutherford*, United States Patent, pat.Nr:5,381,985

[30] VOUGHT / HILLER / RYAN XC-142A TILTWING VSTOL TRANSPORT, *Norton, Willam*, Air Force Legends. N°213 (First ed.). California, United States: Ginter Books, 2006.

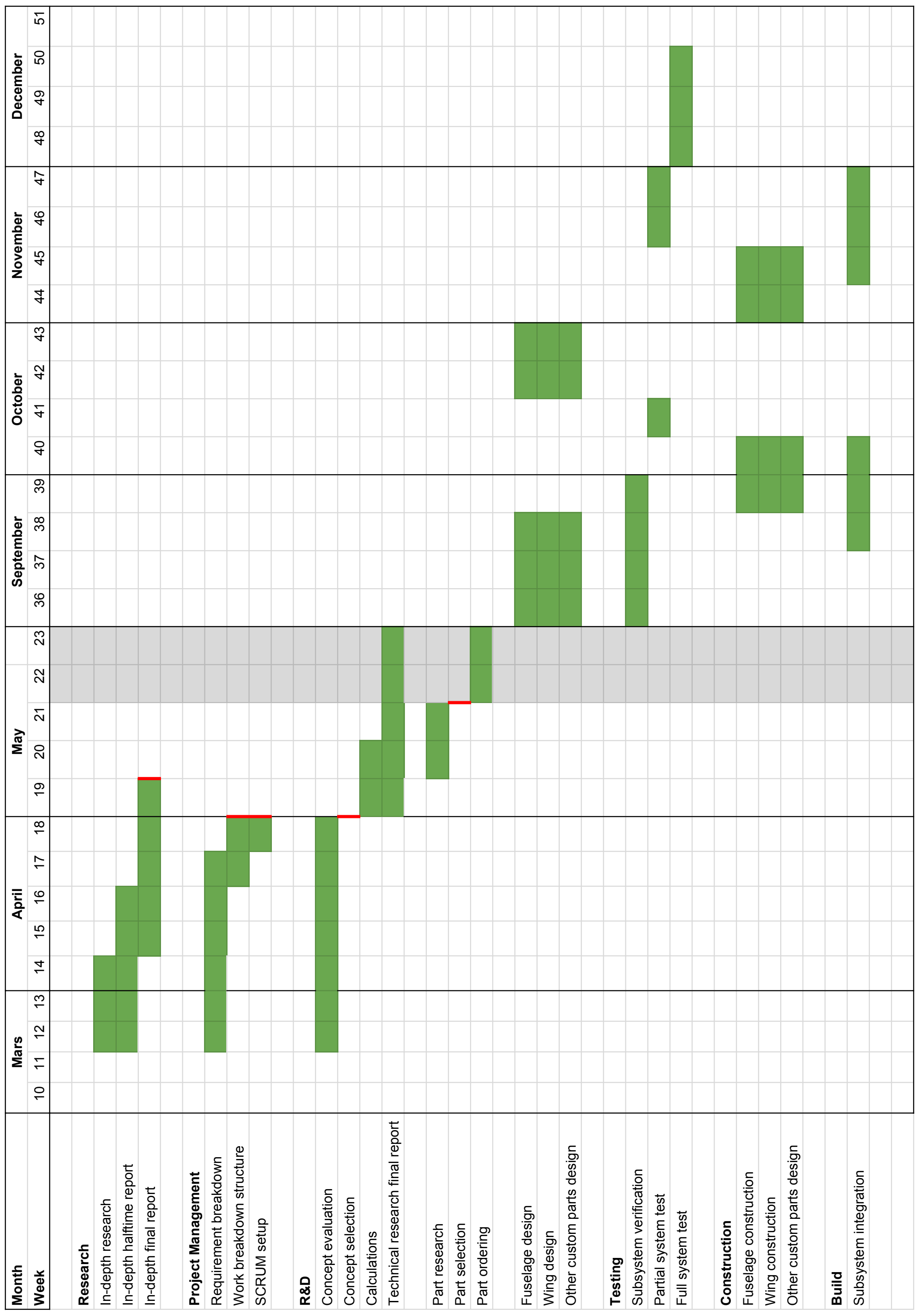
[31] Losses in DC Machine | Electrical4u, *electrical4u*, ”$https : //www.electrical4u.com/losses - in - dc - machine/$”, Retrieved: 17 May 2017.

[32] AIR FOIL, *Mealani Nakamura*, MIT, ”$http : //web.mit.edu/2.972/www/reports/airfoil/airfoil.html$”, 1999. Retrieved: 17 May 2017.

[33] Spoilers, *Nancy Hall*, NASA - Glenn Research Center, ”$https : //www.grc.nasa.gov/WWW/K - 12/airplane/spoil.html$”, Last Updated: May 05 2015, Retrieved: 17 May 2017.

[34] Control System Design for a Ducted-Fan Unmanned Aerial Vehicle Using Linear Quadratic Tracker, *Junho Jeong, Seungkeun Kim, and Jinyoung Suk*, "https://www.hindawi.com/journals/ijae/2015/364926/", 2015.

A Project GANTT Schedule



B Test Cases

This page intentionally left blank

B.1 Test Case 1: Arduino-wing subsystem

Test Case: *Arduino-wing subsystem*

Test date:

Performed by: *Gustaf, David*

Purpose

Test subsystem functionality, servo steering motor-rod, servo calibration, pcb setup

Method

Parts tested:

- arduino mkr zero with pcb connector to motorshield
- front wing (mounted)
- single servo fully assembled

Testing equipment:

- laptop with arduino code

Procedure

- *1. send calibration message from laptop to arduino.*
    - *see that calibration button is pressed*
    - *expected that servo stops.*
    - *see that reference is zeroed in the serial output*
- *2. send a refernce degree (90,45,30,10)*
    - *use degree scale to see if rod angle corresponds to reference*

Verdict

1. *calibration works, the servos run "backwords" till it hits the button. As the button is hit the rods stop their movement.*
2. *problemo, the servos dont stop at the requested angle. The error is suspected to lie in the code.*

B.2 Test Case 2: Motor, battery and ESC Benchmark

Test Case: Motor, Battery and ESC benchmark

Test date: 2017-09-28

Performed by: Gustaf, Magnus

Purpose:

To benchmark motor thrust, this validates that the motors will give the thrust needed. The ESC are also benchmarked to evaluate the controller characteristics. The Battery is also included to validate it´s performance.

In order to verify Requirements or parts of Requirements:

- REQ1.1 The vehicle shall be fully electrical.
- REQ3.1 The motors shall provide thrust 1.5 times higher than the weight.
- REQ3.3 The batteries shall provide enough voltage and current to the motors.
- REQ3.5 The power management circuit shall withstand the current requirement.

Method

Parts tested:

- Battery: Turnigy 5.0, 6cells, 22.2V, 20-30C, 5000mAh
- Motor: Dr.MadThrust ..... CounterClockwise
- ESC: Turnigy AE 80A, 2-6S sbec 4A

Testing equipment:

- Radio controller
- Scale
- Plywood mounting for motor
- Receiver

Procedure

Benchmarking Motor:

Connect motor to the ESC, which in turn is connected to the battery and radio Transmitter. The motor is then placed in the mounting and secured with tape if needed. The entire mounting is then placed on scale. The scale is then reset so that the weight of mounting and motor is deducted.

The radio controller is now used to send thrust requests to the motor via the ESC. These request are given in the table below. At every signal step the generated thrust in kg is read from the scale. When all controller steps have been tested the controller behaviour can be extracted from the data. The generated thrusts are also to be evaluated against the data given for the motor.

If motor thrust matches the given data, motor and battery are considered good?

Controller set	Kg on scale round 1	Kg on scale round 2
0	0	
1	102g	200
2	494g	470
3	860g	850
4	1,3kg	1,2
5	1,6kg	Battery drained
6		
0-max (instant)	1,8 burst- 1,5 cont (kg)	

2 motors 1 battery.....

The rig is setup the same as before, this time though the batterycables are forked to accomodate 2 motors.

The radio controller is setup so that each motor is connected to a different joystick.

The mototrs are to be run at full power simultaneously. The one set in the rig will generate a thrust that will be readable on the scale. As they both run on full speed they will generate the same torque.

Motor:	Kg on scale
motor in the rig	1,6kg

verdict

The motors have a lower max thrust than when one motor runs on one battery. The loss is 0,2kg of thrust when running 2 motors on 1 battery. This was deemed acceptable by the group as the combined max thrust will still give a big enough safetymargin.

B.3 Test Case 3: Constrained Hover

Test Case: *Constrained Hover*

Test date: 2017-12-04

Performed by: *(Team WAZP)*

Purpose

Hover the vehicle with it constrained to the ground to make sure that:

- The motor holders are sturdy enough to handle 100% thrust
- The motors generate positive thrust
- The vehicle tries to lift straight up

In order to verify Requirements or parts of Requirements:

- REQ1.2 The vehicle shall be able to take off vertically
- REQ2.2 The tilt mechanism needs to withstands the torque generated by the motors
-

Method

Parts tested:

- Motors
- Pixhawk controller

Testing equipment:

- Radio controller
- Strings
- Cardboard plate
- Styrofoam
- Weights

Procedure

Prepare the vehicle:

- The styrofoam shall be mounted on the cardboard plate
- Strings shall connect the vehicle to the cardboard plate
- Weights shall be mounted on the cardboard plate

Perform startup procedure

Increase thrust until the vehicle lifts from the ground

Try to keep the vehicle stable

Slowly reduce thrust back to zero

Expected Outcome

Vehicle should be stable in the air, only constrained to the ground by the test-rig

Verdict

Test was repeated several times.

For the first test the left and middle motor on top right wing was vibrating at 30% thrust. This test also caused the MKRZero in the front to break (not detected through serial and does not execute code). The motor-clamps for these motors were swapped and the MKRZero was swapped for a new one.

For the second test at 40% thrust the same motors loosened from their clamps. The new MKRZero in the front did now break again and since we do not have another spare it was left in with the motors in "hover mode". The rubber was then glued and the clamps were fastened harder with zip-ties.

For the third test at 50% thrust the stop screw for the top-right wing loosened causing the axis to rotate. The screw was the tightened again with lock-tite.

For the fourth test the back part of the vehicle lifted from the air at about 55-60% thrust. Front part did not lift. After this test something is not letting us tilt the back motors either.

These tests has caused the arduino MKRZero in the front to break and has also done something to the tilting in the back. Nothing apparent seems to be broken on the Arduino nor the pixhawk so there might be something with the connections.

B.4 Test Case 4: Performance of different types of ESCs

Test Case: Performances of different types of ESCs

Test date: 2017-11-08

Performed by: ***Jiyang Chen***

Purpose

The purpose of this test is to evaluate the performances of different types of ESCs and verify if they can be implemented together. There are three models of ESCs needed to be tested which are New Turnigy AE 80A, Repaired Turnigy AE 80A with a swapped capacitor and YEP 80A.

Method

Parts tested:

- ESC: New Turnigy AE 80A, Repaired Turnigy AE 80A and YEP 80A.

Testing equipment:

- Radio Transmitter and receiver
- Weight scale
- Carton mounting platform for motor

Procedure

Setup for the test:

After mounting the motor in the testing platform, connect the ESC to the motor. Then the ESC and the radio receiver is connected. The last step for the setup is to connect the battery to the motor. After all the components are connected and the green light is solid in the receiver, the setup is finished.
Notice: the throttle in the transmitter should be in the bottom during all the setup procedure!

Increase the throttle slowly, when the motor begins to turn and record the input signal in the receiver;
Continuously increase the throttle which makes the throttle increases by 100 us each time and then record the input signal in the receiver and the thrust measured in the scale.

Verdict

The data is recorded below:

Input signal (us)	1280 (start point)	1300	1400	1500	1600
Thrust (g)	85	171	360	680	920

Table1: Tested data for YEP 80A

Input signal (us)	1200 (start point)	1300	1400	1500	1600
Thrust (g)	81	358	650	900	1230

Table2: Tested data for New Turnigy AE 80A

After connecting the Repaired Turnigy AE 80A and the receiver, the ESC started to beep continuously. From the manual of the ESC, the kind of beeping means the throttle is not in the bottom. The new capacitor which was repaired in the ESC was badly influencing the performance of ESC.

These three kinds of ESCs cannot be implemented together. Two of them can work individually and the repaired model is malfunctioned.

B.5 Test Case 5: EMI testing

Test Case: *EMI testing*

Test date: 2017-12-04

Performed by: *(Team WAZP)*

Purpose

After the previous test, *Test Case: Constrained Hover*, there was some breakage of the arduinos. This test is designed to test if the EDFs and the high power part of the vehicle interferes with the arduinos

In order to verify Requirements or parts of Requirements:

- REQ3.6 The control circuit should be protected from EMI.

Method

Parts tested:

- Gimbal architecture
- High power system
- Arduinos

Testing equipment:

- Radio controller
- Strings
- Cardboard plate
- Styrofoam
- Weights

Procedure

Prepare the vehicle:

- The styrofoam shall be mounted on the cardboard plate
- Strings shall connect the vehicle to the cardboard plate
- Weights shall be mounted on the cardboard plate

1. Test gimbal functionality with the hand controller:
    a. Calibrate the controller
    b. Move the controller to the max angle, all wings should move simultaneously to the max position
    c. Move the controller to the min angle, all wings should move simultaneously to the min position
2. Disconnect the arduinos from the rest of the circuit and disconnect the power to the H-bridge

3. Perform startup procedure
4. Increase thrust until the vehicle lifts from the ground
5. Try to keep the vehicle stable
6. Slowly reduce thrust back to zero
7. Connect the Arduinos and the power to the H-bridge
8. Test the gimbal functionality according to step 1 and verify that the arduinos are working
9. Disconnect only the rear arduino and make sure that the front arduino is still working
10. Increase thrust again with the front arduino still connected
11. Let the vehicle run for 10 seconds
12. Lower thrust again
13. Test the gimbal functionality on the front arduino

Expected Outcome

The functionality of the gimbal system should not be changed after running the vehicle for a longer period of time. Neither the pixhawk nor the arduinos will be destroyed by the test

Verdict

After running this test the front arduino was destroyed. Lights are shining but it does not recognize as a device when plugged in nor does it execute any code. From this test it is very likely that the high power electronics send some conducted EMI into the low power electronics.

B.6 Test Case 6: Functionality of fron right wing

Test Case: *Functionality front right wing*

Test date:

Performed by: *Jiyang, Gustaf, David, Ludvig*

Purpose

Test construction of wing against motor torque and vibration. heat check will also be performed by manually checking for hotspots (Batteries, ground plane, ESCs)

In order to verify Requirements or parts of Requirements:

- REQ2.1 The wings needs to be able to carry the weight of the vehicle.
- REQ2.2 The tilt mechanism needs to withstands the torque generated by the motors.
- REQ3.5 The power management circuit shall withstand the current requirement.

Method

Parts tested:

- Fully assembled front section (wing with motors attached)
- batteries and ESCs for motors

Testing equipment:

- Radio controller and receiver
- Laptop connected to arduino with controller code

Procedure

Start the radio controller

Connect the batteries and ESCs that are to be tested to the ground plane

Run low throttle and make sure that the motors run in the correct direction

If any motor runs in the wrong direction stop the test and connect the motor correctly

When all motors are run in the correct direction increase thrust to max

With thrust high look for bending or twisting in the wing

Reduce thrust to zero

Disconnect batteries and ESCs

Check for heating in the motors, ESCs, batteries and ground plane

Verdict

Max throttle for this test was ~70%

Motor fastenings are not breaking from these forces

The wing is holding and not bending

B.7 Test Case 6: Functionality of fron right wing

ESCs are somewhat hot but not hotter than expected

Ground plane and batteries have no notable hotspots

Motors are slightly rotating around the motor axis

Due to the torsion in the carbon fiber rod the motors twist along the axis when running throttle above ~50%. This causes motors further out on the axis to twist more causing an offset.

B.8 Test Case 7: Startup and shutdown procedure

Test Case: *Startup and shutdown procedure*

Test date: 2017-11-27

Performed by: *(Fredrika, David)*

Purpose

Test the startup procedure to make sure that

- The ESCs all get power from the 6S batteries
- The two arduino and pixhawk get power from the 3S battery
- The four servos get power.
- The hand controller (transmitter) is able to connect to the microcontrollers

In order to make sure that

- The startup and shutdown procedure works and can be performed in this way for further testing

Method

Parts tested:

- ESCs
- Motors
- Batteries
- Servos
- Hand controller

Testing equipment:

None

Procedure

Perform startup procedure by:

- Connect all the ESCs to the ground plane.
- Check all battery voltages (batteries should NEVER be below 20V or they will break)
- If the battery voltages are low, charge them before continuing
- Connect all the batteries to the ground plane.
- Make sure that all ESCs get power and give correct "beeps" in the motors.
- Start the hand controller.
- Switch on the 3S battery to the microcontrollers.
- Check that the two arduino gets power by looking at the power indicator light
- Check that the pixhawk gets power by looking at the power indicator light on the GPS

- Check that the servos gets power by looking at the power indicator light
- Perform “zeroing procedure” for the servos on the arduino.

Perform shutdown procedure by:

- Switch off the power switch on the 3S battery
- Disconnect the ESCs
- Disconnect the 6S batteries.

Expected Outcome

All ESCs give 6 beeps (one for each cell on the battery)

The power indicator lights on the ESCs, arduinos, pixhawk GPS and servos are lit

Zeroing procedure works

Verdict

Startup procedure works as expected

Shutdown procedure works as expected

B.9 Test Case 8: Stationary Tilting

Test Case: Stationary Tilting

Test date: 2017-11-27

Performed by: *(David, Fredrika)*

Purpose

Tilt the motors in order to:

- Test the Arduino controller
- Communication from hand controller via pixhawk to Arduino

In order to verify Requirements or parts of Requirements:

- REQ4.2 The flight controller shall control tilt and flight

Method

Parts tested:

- Arduino controller
- Servo motors
- Physical connection between Pixhawk and Arduino
- communication protocol between Pixhawk and Arduino

Testing equipment:

- Hand controller
- Entire vehicle, including Arduinos, Pixhawk and Servos.

Procedure

1. Connect and switch on the small battery used for the control systems.
2. Turn on the hand controller and verify the connection to the receiver by looking for the green light
3. Turn the appropriate knobs on the hand controller to move one of the wings up and down. do this for all four wings separately.
4. Turn the knob that moves all of the wings at once, both up and down.

Expected Outcome

when performing step 3, one of the motor rows should move according to the knob commands, both up and down. the other motors should not move at all.

When performing step 4, all motors should move in unison, at the same speed and equally many degrees.

Verdict

Correct motor tilting is confirmed

B.10 Test Case 9: Stress-test of motor setup

Test Case: *Stress test of motor setup*

Test date:

Performed by: *Ludvig, Jiyang*

Purpose

The purpose is to determine the durability of the setup and safe time that can be run without accidents regarding the battery, ESC and motor. And also determine the battery duration.

One motor, one repaired ESC and one fully charged 22.4V battery are connected. And the motor is run in full throttle until battery is close to recommended minimum voltage (20V).

In order to verify Requirements or parts of Requirements:

- REQ1.1 The vehicle shall be fully electrical
- REQ3.3 The batteries shall provide enough voltage and current to the motors
- REQ3.5 The power management circuit shall withstand the current requirement

Method

Parts tested:

- One Motor
- One repaired Turnigy AE 80A ESC
- One fully charged Turnigy 6-cell battery

Testing equipment:

- Radio Transmitter and receiver;
- Mounting platform for motor;
- Oscilloscope;

Procedure

Describe how the test is setup,

- The ESC is repaired and the beginning of the throttle is lower than normal ESCs, the input signal of the transmitter is set to 800us which is lower than default 988us;
- The battery is connected to the ESC and the ESC is connected to the motor in a correct way.
- Also an Oscilloscope is connected in parallel to the battery to monitor the voltage level;
- Turn on the transmitter and run the motor in full throttle;
- Start to count the time and monitor the voltage level of battery. When it is close to 20V, stop running the motor and stop counting;

Verdict

The test was performed for 6 minutes and 15 seconds. The voltage level dropped faster when it was below 20.5V. It was stopped because it reached the recommended minimum voltage level.

The battery slightly increased the size and temperature was warm when the test was ended. The size of battery returned to default size when the temperature returned to room temperature. We expected that no damage was sustained. No motor or ESC was broken during the test. Though we use the repaired ESC, we assume the new ESC works fine in this case.

We verified that the motor and ESC were able to run in a significant amount of time without incidents. When one battery is connected to two motors, we would expect that the duration would be around 3 and half minutes. Since we are not going to run the motors in full throttle all the time, the duration would be longer than expected.

The ESCs would not be a hazard when the components are connected properly. The Pixhawk was not used in the test, so the wiring would be slightly different from the test case when we used the Pixhawk.

B.11 Test Case 10: Torque testing on Servo motors

Torque testing Servo

In this test the aim is to test the torque delivered by the servos. Specifically the stall torque along with "start again torque".

Setup

Needed parts:

- Servo (4rpm and 7rpm, dcmotor with wormgear)
- Arduino (UNO or MKRZERO)
- Motor driver (MOTORSHIELD REV.3)
- Lever with known length (screwdriver, red handle, 0.09m)
- Dynometer

The setup is for the Arduino to run the driver wich in turn runs the motors. The lever is attached in an appropriate manner to the outputshaft. The dynometer is then attached to the end of the lever.

Procedure of the test

The motors is run while one person the dynometer. The force exerted on the dynometer is now read. First value to read is the stall torque, when the motor can no longer turn the lever. When the stall torque value has been read the person holding the dynometer now slowly reduces its counterforce until the motor turn the rod again. The force read when it starts is the "start again force". These are to be compared against the estimated torque from the motors on their mounting rod.

Motor	**Top stall torque[Nm]**	**"start again torque"[Nm]**
4RPM	0,81	0,63
7RPM	??	0,45

Verdict

From the torquetest it can be concluded that both servos deliver enough torque to overcome the motortorque. The servo with 4rpm has been choosen as the one to use because of its higher safteymargin and slower speed.

B.12 Test Case 11: Wing Strength

Test Case: ***Wing strength***

Test date: 30 nov 2017

Performed by: Magnus Valtersson and Tomas Fredriksson

Purpose

To verify that the wings can:

- Hold the weight of the vehicle

In order to verify Requirements or parts of Requirements:

- REQ2.1 The wings needs to be able to carry the weight of the vehicle

Method

Parts tested:

- Wings
- Body

Testing equipment:

- Vehicle with mounted parts
- Safety Strings

Procedure

Prepare the vehicle:

- Mount all components and top plate
- Remove all temporary parts and cables

Lift the vehicle manually in the wings

Then try to bend the wings by applying force in one of the sides of the wings

Leave one side of the wing and only hold the wings in one of the wing sections.

Expected Outcome

Wings and body should remain stable, and be able to hold the weight without breaking.

Verdict

The wings could take the weight of the vehicle in an excellent manner.

The wings could also receive a large amount of twist without problems.

B.13 Test Case 12: Vertical Take-Off and Landing

Test Case: *Vertical Take-Off and Landing*

Test date: 2017-12-18

Performed by: *(Team WAZP)*

Purpose

Test the controller during take-off and create a safe, repeatable way to tune the controller for a VTOL. This test is supposed to be repeated until the vehicle is able to lift vertically.

In order to verify Requirements or parts of Requirements:

- REQ1.2 The vehicle shall be able to take off vertically
- REQ2.1 The wings needs to be able to carry the weight of the vehicle
- REQ2.2 The tilt mechanism needs to withstands the torque generated by the motors
- REQ3.1 The motors shall provide thrust 1.5 times higher than the weight
- REQ4.1 The software should include a self-stabilizing system

Method

Parts tested:

- Pixhawk PID controller

Testing equipment:

- Entire vehicle
- Radio controller
- Styrofoam test-rig

Procedure

Before starting the test:

- Make sure that the battery voltage is above 22V before starting the test or continued testing may compromise the batteries
- Disconnect the power cable to the arduinos
- Connect the batteries
- Tie down the vehicle to the styrofoam test-rig

Perform the test by:

1.) Power on the pixhawk
2.) Power on the radio controller
3.) Verify connection between controller and pixhawk
4.) Arm the motors
5.) Without manually compensating for movement, increase thrust until the vehicle takes off

6.) If the vehicle is able to do a vertical take-off the test is completed, else perform steps 7-12
7.) Lower the thrust back to zero
8.) Disarm the motors
9.) Power off the radio controller and the pixhawk
10.) If the tuning takes longer than 5 minutes the batteries are to be disconnected before any tuning is made
11.) Connect the pixhawk to the computer to tune the parameters
12.) Unplug the pixhawk from the computer
13.) Restart the test again from step 1

Expected Outcome

The vehicle is not expected to be able to perform a VTOL on the first try so some repetitions are expected. It should however be able to do this after 5-10 tries.

Verdict

The vehicle was not able to take off. The rigidity of the motor axes are not strong enough to withstand the torque that the motors generate which causes the motors to point forward during flight. Compensating for this by individually shifting the motors back before take-off and tilting the entire axis backwards helps slightly but not enough. To much thrust has to be put into stabilizing and that doesn't leave enough for the vehicle to take of and it just slides around.

B.14 Test Case 13: Balancing in Roll Axis

Test Case: ***Balancing in Roll axis***

Test date: 2017-12-10

Performed by: ***(Team WAZP)***

Purpose

Thrust the motors while the vehicle is constrained so the only rotation that can be performed is roll. This is in order to simplify the environment that the vehicle is working in to test if the controller can stabilize itself while only focusing on one axis.

In order to verify Requirements or parts of Requirements:

- REQ1.2 The vehicle shall be able to take off vertically
- REQ4.1 The software shall include a self-stabilizing system
- REQ4.2 The flight controller shall control tilt and flight

Method

Parts tested:

- Flight controller software

Testing equipment:

- Strings
- Radio controller

Procedure

Prepare the test by:

- Hang strings from the ceiling about 1-1.5m long and 4 meters apart
- Make loops that can go around the wings of the vehicle
- Thread the loops around the wings making sure that no strings are touching the EDFs
- The vehicle should now be suspended in the air and rotation should not be possible in yaw or pitch axis but only possible in roll axis

Perform startup procedure with the vehicle suspended in the air

Without arming the motors, give the vehicle a small push in roll axis

Wait for the oscillations to stop

Start the motors at about 20% thrust

Give the vehicle the same push again

Wait for the oscillations to stop
Reduce thrust to zero
Perform shutdown procedure

Expected Outcome

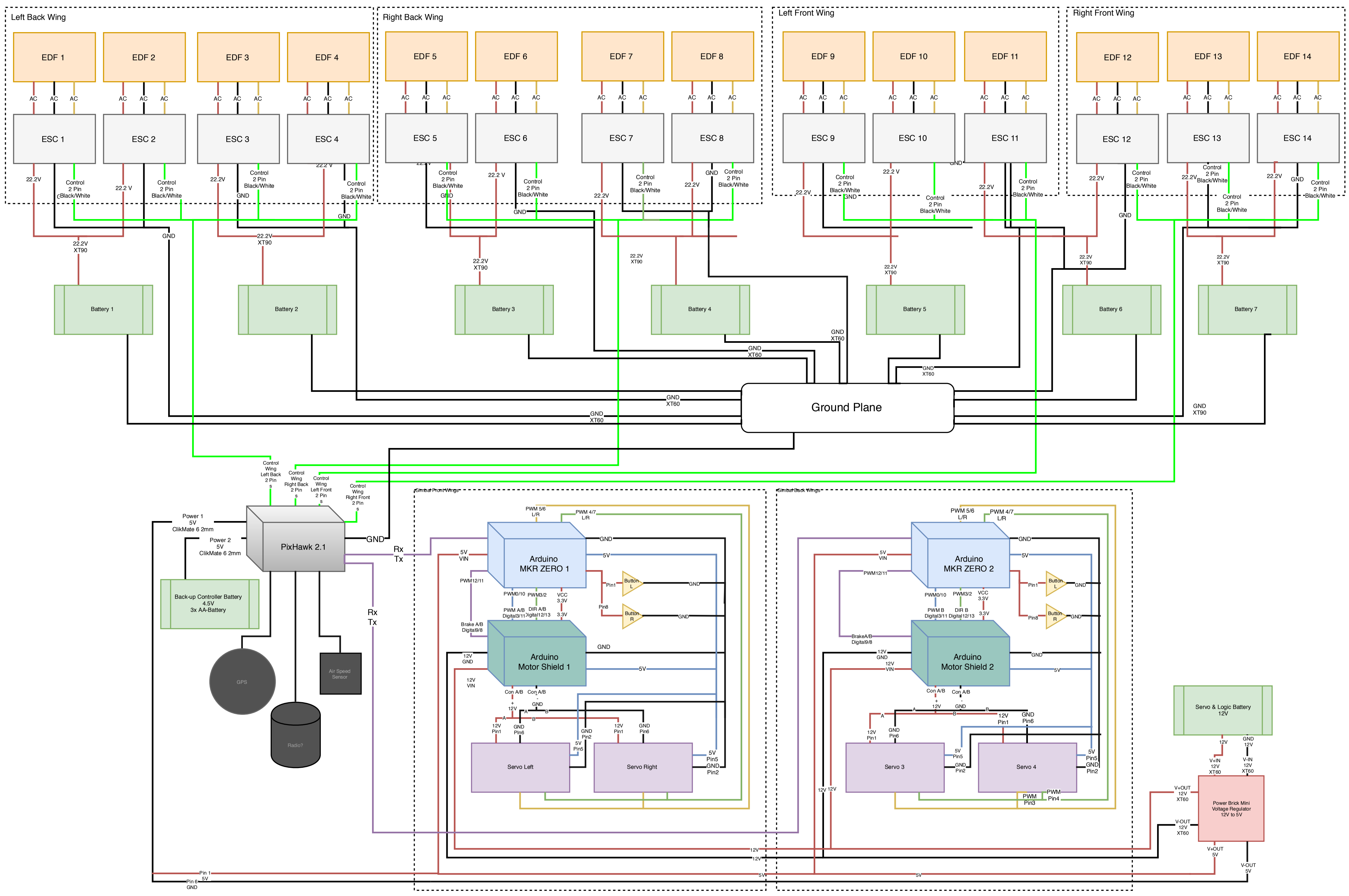
When the Flight Controller is disconnected it should take the vehicle somewhere between 5-10 seconds before the oscillations have stopped. When it is on and controlling the motors oscillations should be countered by the flight controller in significantly less time

Verdict

The controller was able to counter the disturbance really well and oscillations stopped in less than 1 second

C System Design Overview

This page intentionally left blank



Left Back Wing
Right Back Wing
Left Front Wing
Right Front Wing
EDF 1
EDF 2
EDF 3
EDF 4
EDF 5
EDF 6
EDF 7
EDF 8
EDF 9
EDF 10
EDF 11
EDF 12
EDF 13
EDF 14
AC
ESC 1
ESC 2
ESC 3
ESC 4
ESC 5
ESC 6
ESC 7
ESC 8
ESC 9
ESC 10
ESC 11
ESC 12
ESC 13
ESC 14
22.2V
Control 2 Pin Black/White
GND
22.2V XT90
Battery 1
Battery 2
Battery 3
Battery 4
Battery 5
Battery 6
Battery 7
GND XT60
GND XT90
Ground Plane
Control Wing Left Back 2 Pins
Control Wing Right Back 2 Pins
Control Wing Left Front 2 Pins
Control Wing Right Front 2 Pins
Power 1 5V ClikMate 6 2mm
Power 2 5V ClikMate 6 2mm
PixHawk 2.1
GND
Rx Tx
Back-up Controller Battery 4.5V 3x AA-Battery
GPS
Radio?
Air Speed Sensor
Gimbal Front Wings
Gimbal Back Wings
Arduino MKR ZERO 1
Arduino MKR ZERO 2
Arduino Motor Shield 1
Arduino Motor Shield 2
PWM 5/6 L/R
PWM 4/7 L/R
5V VIN
PWM12/11
PWM0/10
PWM3/2
VCC 3.3V
PWM A/B Digital3/11
DIR A/B Digital12/13
3.3V
Brake A/B Digital9/8
Pin1
Pin8
Button L
Button R
12V GND
12V VIN
Con A/B
Servo Left
Servo Right
Servo 3
Servo 4
12V Pin1
GND Pin6
5V Pin5
GND Pin2
PWM Pin3
PWM Pin4
Servo & Logic Battery 12V
V+IN 12V XT60
V-IN 12V XT60
V+OUT 12V XT60
V-OUT 12V XT60
V+OUT 5V
V-OUT 5V
Power Brick Mini Voltage Regulator 12V to 5V
Pin 1 5V
Pin 6 GND

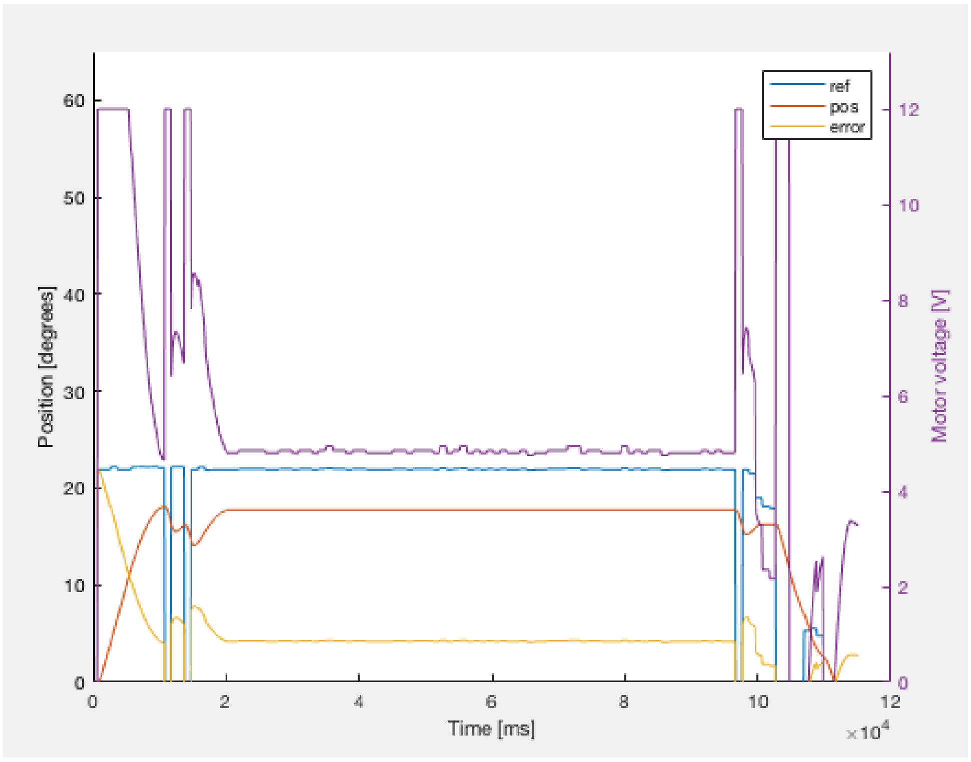


Figure 24: Controller plot: Motor Tilt System

D Results

This section contains additional figures that are related to results but are not needed in the main report.

E Flight Controller Code

This section shows the function for control of the Motor Tilt System that was added to the Ardupilot code base.

```
#include "Copter.h"

#define SCHED_TASK(func, rate_hz, max_time_micros) SCHED_TASK_CLASS(Copter, &copter, f

/*
  scheduler table for fast CPUs - all regular tasks apart from the fast_loop()
  should be listed here, along with how often they should be called (in hz)
  and the maximum time they are expected to take (in microseconds)
 */
const AP_Scheduler::Task Copter::scheduler_tasks[] = {
    SCHED_TASK(rc_loop,              100,    130),
    SCHED_TASK(throttle_loop,         10,     75),
    SCHED_TASK(update_GPS,            50,    200),
#if OPTFLOW == ENABLED
    SCHED_TASK(update_optical_flow,  200,    160),
#endif
    SCHED_TASK(update_batt_compass,   10,    120),
    SCHED_TASK(servo_loop,            10,     50), // Avian servo loop
    SCHED_TASK(read_aux_switches,     10,     50),
    SCHED_TASK(arm_motors_check,      10,     50),
    SCHED_TASK(auto_disarm_check,     10,     50),
    SCHED_TASK(auto_trim,             10,     75),
    SCHED_TASK(read_rangefinder,      20,    100),
    SCHED_TASK(update_proximity,     100,     50),
    SCHED_TASK(update_beacon,        400,     50),
    SCHED_TASK(update_visual_odom,   400,     50),
    SCHED_TASK(update_altitude,       10,    100),
    SCHED_TASK(run_nav_updates,       50,    100),
    SCHED_TASK(update_throttle_hover,100,     90),
    SCHED_TASK(smart_rtl_save_position, 3,    100),
    SCHED_TASK(three_hz_loop,          3,     75),
    SCHED_TASK(compass_accumulate,   100,    100),
    SCHED_TASK(barometer_accumulate,  50,     90),
#if PRECISION_LANDING == ENABLED
    SCHED_TASK(update_precland,      400,     50),
#endif
#if FRAME_CONFIG == HELI_FRAME
    SCHED_TASK(check_dynamic_flight,  50,     75),
#endif
    SCHED_TASK(fourhundred_hz_logging,400,    50),
    SCHED_TASK(update_notify,         50,     90),
    SCHED_TASK(one_hz_loop,            1,    100),
    SCHED_TASK(ekf_check,             10,     75),
    SCHED_TASK(gpsglitch_check,       10,     50),
    SCHED_TASK(landinggear_update,    10,     75),
    SCHED_TASK(lost_vehicle_check,    10,     50),
    SCHED_TASK(gcs_check_input,      400,    180),
    SCHED_TASK(gcs_send_heartbeat,     1,    110),
    SCHED_TASK(gcs_send_deferred,     50,    550),
    SCHED_TASK(gcs_data_stream_send,  50,    550),
    SCHED_TASK(update_mount,          50,     75),
    SCHED_TASK(update_trigger,        50,     75),
    SCHED_TASK(ten_hz_logging_loop,   10,    350),
    SCHED_TASK(twentyfive_hz_logging, 25,    110),
    SCHED_TASK(dataflash_periodic,     400,    300),
    SCHED_TASK(ins_periodic,           400,     50),
    SCHED_TASK(perf_update,            0.1,     75),
    SCHED_TASK(read_receiver_rssi,     10,      75),
    SCHED_TASK(rpm_update,             10,     200),
    SCHED_TASK(compass_cal_update,    100,     100),
    SCHED_TASK(accel_cal_update,       10,     100),
#if ADSB_ENABLED == ENABLED
    SCHED_TASK(avoidance_adsb_update, 10,      100),
#endif
#if ADVANCED_FAILSAFE == ENABLED
    SCHED_TASK(afs_fs_check,           10,     100),
#endif
    SCHED_TASK(terrain_update,         10,     100),
#if GRIPPER_ENABLED == ENABLED
    SCHED_TASK(gripper_update,         10,      75),
#endif
    SCHED_TASK(winch_update,           10,      50),
#ifdef USERHOOK_FASTLOOP
    SCHED_TASK(userhook_FastLoop,     100,      75),
#endif
#ifdef USERHOOK_50HZLOOP
    SCHED_TASK(userhook_50Hz,          50,      75),
#endif
#ifdef USERHOOK_MEDIUMLOOP
    SCHED_TASK(userhook_MediumLoop,    10,      75),
#endif
#ifdef USERHOOK_SLOWLOOP
    SCHED_TASK(userhook_SlowLoop,      3.3,     75),
#endif
#ifdef USERHOOK_SUPERSLOWLOOP
    SCHED_TASK(userhook_SuperSlowLoop, 1,    75),
#endif
    SCHED_TASK(button_update,           5,     100),
    SCHED_TASK(stats_update,            1,     100),
};


void Copter::setup()
{
    // Load the default values of variables listed in var_info[]s
    AP_Param::setup_sketch_defaults();

    // setup storage layout for copter
    StorageManager::set_layout_copter();

    init_ardupilot();

    // initialise the main loop scheduler
    scheduler.init(&scheduler_tasks[0], ARRAY_SIZE(scheduler_tasks));

    // setup initial performance counters
    perf_info.reset();
    fast_loopTimer = AP_HAL::micros();

    hal.uartE->begin(57600);

}
```

```
void Copter::perf_update(void)
{
    if (should_log(MASK_LOG_PM))
        Log_Write_Performance();
    if (scheduler.debug()) {
        gcs().send_text(MAV_SEVERITY_WARNING, "PERF: %u/%u max=%lu min=%lu avg=%lu sd=
                            (unsigned)perf_info.get_num_long_running(),
                            (unsigned)perf_info.get_num_loops(),
                            (unsigned long)perf_info.get_max_time(),
                            (unsigned long)perf_info.get_min_time(),
                            (unsigned long)perf_info.get_avg_time(),
                            (unsigned long)perf_info.get_stddev_time());
    }
    perf_info.reset();
    pmTest1 = 0;
}

/*
  update AP_Stats
 */
void Copter::stats_update(void)
{
    g2.stats.update();
}

void Copter::loop()
{

   // hal.uartE->printf("hej\n");
    // wait for an INS sample
    ins.wait_for_sample();

    uint32_t timer = micros();

    // check loop time
    perf_info.check_loop_time(timer - fast_loopTimer);

    // used by PI Loops
    G_Dt                    = (float)(timer - fast_loopTimer) / 1000000.0f;
    fast_loopTimer          = timer;

    // for mainloop failure monitoring
    mainLoop_count++;

    // Execute the fast loop
    // ---------------------
    fast_loop();

    // tell the scheduler one tick has passed
    scheduler.tick();

    // run all the tasks that are due to run. Note that we only
    // have to call this once per loop, as the tasks are scheduled
    // in multiples of the main loop tick. So if they don't run on
    // the first call to the scheduler they won't run on a later
    // call until scheduler.tick() is called again
    const uint32_t loop_us = scheduler.get_loop_period_us();
    const uint32_t time_available = (timer + loop_us) - micros();
    scheduler.run(time_available > loop_us ? 0u : time_available);
}


// Main loop - 400hz
void Copter::fast_loop()
{
    // update INS immediately to get current gyro data populated
    ins.update();

    // run low level rate controllers that only require IMU data
    attitude_control->rate_controller_run();

    // send outputs to the motors library immediately
    motors_output();

    // run EKF state estimator (expensive)
    // --------------------
    read_AHRS();

#if FRAME_CONFIG == HELI_FRAME
    update_heli_control_dynamics();
#endif //HELI_FRAME

    // Inertial Nav
    // --------------------
    read_inertia();

    // check if ekf has reset target heading or position
    check_ekf_reset();

    // run the attitude controllers
    update_flight_mode();

    // update home from EKF if necessary
    update_home_from_EKF();

    // check if we've landed or crashed
    update_land_and_crash_detectors();

#if MOUNT == ENABLED
    // camera mount's fast update
    camera_mount.update_fast();
#endif

    // log sensor health
    if (should_log(MASK_LOG_ANY)) {
        Log_Sensor_Health();
    }
}

// rc_loops - reads user input from transmitter/receiver
// called at 100hz
void Copter::rc_loop()
{
    // Read radio and 3-position switch on radio
    // -----------------------------------------
    read_radio();
    read_control_switch();
}

// throttle_loop - should be run at 50 hz
// ----------------------------
void Copter::throttle_loop()
{
    // update throttle_low_comp value (controls priority of throttle vs attitude cont
    update_throttle_thr_mix();

    // check auto_armed status
    update_auto_armed();

#if FRAME_CONFIG == HELI_FRAME
    // update rotor speed
    heli_update_rotor_speed_targets();

    // update trad heli swash plate movement
    heli_update_landing_swash();
#endif

    // compensate for ground effect (if enabled)
    update_ground_effect_detector();

}

//----------------------------Avian control code-------------------------


void Copter::servo_loop(){

    uint16_t ch8 = hal.rcin->read(CH_8);
    uint16_t ch9 = hal.rcin->read(CH_9);
    uint16_t ch10 = hal.rcin->read(CH_10);
    uint16_t ch11 = hal.rcin->read(CH_11);


    if (ch11 > 1600) {



        if (ch8 > 995) {

            hal.uartE->printf("b%u\n", (unsigned int)ch8);

        } else {

            if (ch9 > 1100 && ch9 < 1400 ) // left front forward
            {
                hal.uartE->printf("e\n");
            }
            if (ch9 > 1400 && ch9 < 1600 ) // left front backward
            {
                hal.uartE->printf("f\n");
            }
            if (ch9 > 1600 && ch9 < 1900 ) // left back forward
            {
                hal.uartE->printf("i\n");
            }
            if (ch9 > 1900 && ch9 < 2100 ) // left back backwards
            {
                hal.uartE->printf("j\n");
            }

        // maybe using uartD
            if (ch10 > 1100 && ch10 < 1400 ) // right back forward
            {
                hal.uartE->printf("k\n");
            }
            if (ch10 > 1400 && ch10 < 1600 ) // right back backward
            {
                hal.uartE->printf("m\n");
            }
        //Add a comment to this line
            if (ch10 > 1600 && ch10 < 1900 ) // right front forward
            {
                hal.uartE->printf("g\n");
            }
            if (ch10 > 1900 && ch10 < 2100 ) // right front backwards
            {
                hal.uartE->printf("h\n");
            }

        }
    }

}


// update_mount - update camera mount position
// should be run at 50hz
void Copter::update_mount()
{
#if MOUNT == ENABLED
    // update camera mount's position
    camera_mount.update();
#endif
}

// update camera trigger
void Copter::update_trigger(void)
{
#if CAMERA == ENABLED
    camera.update_trigger();
#endif
}

// update_batt_compass - read battery and compass
// should be called at 10hz
void Copter::update_batt_compass(void)
{
    // read battery before compass because it may be used for motor interference comp
    read_battery();

    if(g.compass_enabled) {
        // update compass with throttle value - used for compassmot
        compass.set_throttle(motors->get_throttle());
        compass.read();
        // log compass information
        if (should_log(MASK_LOG_COMPASS) && !ahrs.have_ekf_logging()) {
            DataFlash.Log_Write_Compass(compass);
        }
    }
}

// Full rate logging of attitude, rate and pid loops
// should be run at 400hz
void Copter::fourhundred_hz_logging()
{
    if (should_log(MASK_LOG_ATTITUDE_FAST)) {
        Log_Write_Attitude();
    }
}

// ten_hz_logging_loop
// should be run at 10hz
void Copter::ten_hz_logging_loop()
{



    // log attitude data if we're not already logging at the higher rate
    if (should_log(MASK_LOG_ATTITUDE_MED) && !should_log(MASK_LOG_ATTITUDE_FAST)) {
        Log_Write_Attitude();
        Log_Write_EKF_POS();
    }
    if (should_log(MASK_LOG_MOTBATT)) {
        Log_Write_MotBatt();
    }
    if (should_log(MASK_LOG_RCIN)) {
        DataFlash.Log_Write_RCIN();
        if (rssi.enabled()) {
            DataFlash.Log_Write_RSSI(rssi);
        }
    }
    if (should_log(MASK_LOG_RCOUT)) {
        DataFlash.Log_Write_RCOUT();
    }
    if (should_log(MASK_LOG_NTUN) && (mode_requires_GPS(control_mode) || landing_with_
        Log_Write_Nav_Tuning();
    }
    if (should_log(MASK_LOG_IMU) || should_log(MASK_LOG_IMU_FAST) || should_log(MASK_I
        DataFlash.Log_Write_Vibration(ins);
    }
    if (should_log(MASK_LOG_CTUN)) {
        attitude_control->control_monitor_log();
        Log_Write_Proximity();
        Log_Write_Beacon();
    }
#if FRAME_CONFIG == HELI_FRAME
    Log_Write_Heli();
#endif
}
```

```
// twentyfive_hz_logging - should be run at 25hz
void Copter::twentyfive_hz_logging()
{
#if HIL_MODE != HIL_MODE_DISABLED
    // HIL for a copter needs very fast update of the servo values
    gcs().send_message(MSG_SERVO_OUTPUT_RAW);
#endif

#if HIL_MODE == HIL_MODE_DISABLED
    if (should_log(MASK_LOG_ATTITUDE_FAST)) {
        Log_Write_EKF_POS();
    }

    // log IMU data if we're not already logging at the higher rate
    if (should_log(MASK_LOG_IMU) && !should_log(MASK_LOG_IMU_RAW)) {
        DataFlash.Log_Write_IMU(ins);
    }
#endif

#if PRECISION_LANDING == ENABLED
    // log output
    Log_Write_Precland();
#endif
}

void Copter::dataflash_periodic(void)
{
    DataFlash.periodic_tasks();
}

void Copter::ins_periodic(void)
{
    ins.periodic();
}

// three_hz_loop - 3.3hz loop
void Copter::three_hz_loop()
{
    // check if we've lost contact with the ground station
    failsafe_gcs_check();

    // check if we've lost terrain data
    failsafe_terrain_check();

#if AC_FENCE == ENABLED
    // check if we have breached a fence
    fence_check();
#endif // AC_FENCE_ENABLED

#if SPRAYER == ENABLED
    sprayer.update();
#endif

    update_events();

    // update ch6 in flight tuning
    tuning();
```

```
}

// one_hz_loop - runs at 1Hz
void Copter::one_hz_loop()
{
    if (should_log(MASK_LOG_ANY)) {
        Log_Write_Data(DATA_AP_STATE, ap.value);
    }

    arming.update();

    if (!motors->armed()) {
        // make it possible to change ahrs orientation at runtime during initial conf
        ahrs.set_orientation();

        update_using_interlock();

        // check the user hasn't updated the frame class or type
        motors->set_frame_class_and_type((AP_Motors::motor_frame_class)g2.frame_class.

#if FRAME_CONFIG != HELI_FRAME
        // set all throttle channel settings
        motors->set_throttle_range(channel_throttle->get_radio_min(), channel_throttle
#endif
    }

    // update assigned functions and enable auxiliary servos
    SRV_Channels::enable_aux_servos();

    check_usb_mux();

    // log terrain data
    terrain_logging();

    adsb.set_is_flying(!ap.land_complete);

    // update error mask of sensors and subsystems. The mask uses the
    // MAV_SYS_STATUS_* values from mavlink. If a bit is set then it
    // indicates that the sensor or subsystem is present but not
    // functioning correctly
    update_sensor_status_flags();
}

// called at 50hz
void Copter::update_GPS(void)
{
    static uint32_t last_gps_reading[GPS_MAX_INSTANCES];   // time of last gps messag
    bool gps_updated = false;

    gps.update();

    // log after every gps message
    for (uint8_t i=0; i<gps.num_sensors(); i++) {
        if (gps.last_message_time_ms(i) != last_gps_reading[i]) {
            last_gps_reading[i] = gps.last_message_time_ms(i);

            // log GPS message
            if (should_log(MASK_LOG_GPS) && !ahrs.have_ekf_logging()) {
                DataFlash.Log_Write_GPS(gps, i);
            }

            gps_updated = true;
        }
    }

    if (gps_updated) {
        // set system time if necessary
        set_system_time_from_GPS();

#if CAMERA == ENABLED
        camera.update();
#endif
    }
}

void Copter::init_simple_bearing()
{
    // capture current cos_yaw and sin_yaw values
    simple_cos_yaw = ahrs.cos_yaw();
    simple_sin_yaw = ahrs.sin_yaw();

    // initialise super simple heading (i.e. heading towards home) to be 180 deg from
    super_simple_last_bearing = wrap_360_cd(ahrs.yaw_sensor+18000);
    super_simple_cos_yaw = simple_cos_yaw;
    super_simple_sin_yaw = simple_sin_yaw;

    // log the simple bearing to dataflash
    if (should_log(MASK_LOG_ANY)) {
        Log_Write_Data(DATA_INIT_SIMPLE_BEARING, ahrs.yaw_sensor);
    }
}

// update_simple_mode - rotates pilot input if we are in simple mode
void Copter::update_simple_mode(void)
{
    float rollx, pitchx;

    // exit immediately if no new radio frame or not in simple mode
    if (ap.simple_mode == 0 || !ap.new_radio_frame) {
        return;
    }

    // mark radio frame as consumed
    ap.new_radio_frame = false;

    if (ap.simple_mode == 1) {
        // rotate roll, pitch input by -initial simple heading (i.e. north facing)
        rollx = channel_roll->get_control_in()*simple_cos_yaw - channel_pitch->get_co
        pitchx = channel_roll->get_control_in()*simple_sin_yaw + channel_pitch->get_c
    }else{
        // rotate roll, pitch input by -super simple heading (reverse of heading to ho
        rollx = channel_roll->get_control_in()*super_simple_cos_yaw - channel_pitch->g
        pitchx = channel_roll->get_control_in()*super_simple_sin_yaw + channel_pitch->
    }

    // rotate roll, pitch input from north facing to vehicle's perspective
    channel_roll->set_control_in(rollx*ahrs.cos_yaw() + pitchx*ahrs.sin_yaw());
    channel_pitch->set_control_in(-rollx*ahrs.sin_yaw() + pitchx*ahrs.cos_yaw());
}

// update_super_simple_bearing - adjusts simple bearing based on location
// should be called after home_bearing has been updated
void Copter::update_super_simple_bearing(bool force_update)
{
    // check if we are in super simple mode and at least 10m from home
    if(force_update || (ap.simple_mode == 2 && home_distance > SUPER_SIMPLE_RADIUS))
        // check the bearing to home has changed by at least 5 degrees
        if (labs(super_simple_last_bearing - home_bearing) > 500) {
            super_simple_last_bearing = home_bearing;
            float angle_rad = radians((super_simple_last_bearing+18000)/100);
            super_simple_cos_yaw = cosf(angle_rad);
            super_simple_sin_yaw = sinf(angle_rad);
        }
    }
}

void Copter::read_AHRS(void)
{
    // Perform IMU calculations and get attitude info
    //-----------------------------------------------------------
#if HIL_MODE != HIL_MODE_DISABLED
    // update hil before ahrs update
    gcs_check_input();
#endif

    // we tell AHRS to skip INS update as we have already done it in fast_loop()
    ahrs.update(true);
}

// read baro and rangefinder altitude at 10hz
void Copter::update_altitude()
{
    // read in baro altitude
    read_barometer();

    // write altitude info to dataflash logs
    if (should_log(MASK_LOG_CTUN)) {
        Log_Write_Control_Tuning();
    }
}

AP_HAL_MAIN_CALLBACKS(&copter);
```

F Motor Tilt System code

This section shows the function for control of the Motor Tilt System that was added to the Ardupilot code base.

```
//MKR Zero code for controlling servos on right and left side, can be used both front
//Encoder does 11100 tics (one encoder channel) for 1 full revolution, 2775 for 90 de

//Structure for reading and printing functions taken from https://forum.arduino.cc/in
//DATASHEET: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/Atmel-42181-SAM-D
//p.383 how to initialize


//LEFT YELLOW ON GPIO5, PB11
//RIGHT YELLOW ON GPIO6, PA20

//LEFT ON CHANNEL A, RIGHT ON CHANNEL B
//USED PINS:
//ARDUINO_PIN -> SHIELD_PIN(VARIABLE_NAME):
//0->3(PWM_LEFT_PIN);                          10->11(PWM_RIGHT_PIN)
//5->servo_left_yellow(ENC_LEFT_Y_PIN);        6->servo_right_yellow(ENC_RIGHT_Y_PIN)
//4->servo_left_green(ENC_LEFT_G_PIN);         7->servo_right_green(ENC_RIGHT_G_PIN)
//3->12(DIRECTION_LEFT_PIN);                   2->13(DIRECTION_RIGHT_PIN)
//12->9(BREAK_LEFT_PIN);                       11->8(BREAK_RIGHT_PIN)
//1->button_L(ZEROING_LEFT_PIN);               8->button_R(ZEROING_RIGHT_PIN)

//External software libraries needed
//Install board file for MKR Zero (by searching for "samd" in the boards manager unde




#include <Arduino.h>

#define NOT_AN_INTERRUPT -1


//GO HERE TO CHANGE PINS, REMEMBER TO CHANGE IN DESCRIPTION AS WELL!
#define PWM_LEFT_PIN 0
#define PWM_RIGHT_PIN 10

#define ENC_LEFT_Y_PIN 5
#define ENC_LEFT_G_PIN 4

#define ENC_RIGHT_Y_PIN 6
#define ENC_RIGHT_G_PIN 7

#define DIRECTION_LEFT_PIN 3
#define DIRECTION_RIGHT_PIN 2

#define BREAK_LEFT_PIN 12
#define BREAK_RIGHT_PIN 11

#define ZEROING_LEFT_PIN 1
#define ZEROING_RIGHT_PIN 8

//TODO Check that I can really use this PIN
//#define HEARTBEAT_PIN 9
```

```
#define PWM_INPUT_PIN 9



//Define directions on how to turn the different rods. This is correct 100%
#define DIRECTION_INCREASE_RIGHT LOW
#define DIRECTION_INCREASE_LEFT HIGH
#define DIRECTION_DECREASE_RIGHT HIGH
#define DIRECTION_DECREASE_LEFT LOW

//Based on direction_increase and angle sketch
#define DIRECTION_TICKS_DECREASE_RIGHT HIGH
#define DIRECTION_TICKS_INCREASE_RIGHT LOW

#define DIRECTION_TICKS_DECREASE_LEFT LOW
#define DIRECTION_TICKS_INCREASE_LEFT HIGH

//Constant for changing encoder ticks to degrees
#define DEGREES_TO_TICKS 31


//Delay time for the main loop in milliseconds
#define MAIN_LOOP_DELAY 10
//Average execution time of the tasks in the main loop, in milliseconds. Measurement
#define MAIN_LOOP_EXECTIME 1
//Approximate time delta in seconds between each controller calculation (it is only a
#define TIME_DELTA (MAIN_LOOP_DELAY - MAIN_LOOP_EXECTIME)

#define MAX_ANGLE 60
#define MAX_TICKS (MAX_ANGLE * DEGREES_TO_TICKS)

//Control variables
#define CONTROL_SIGNAL_MAX 255
#define KP 0.7
#define KI 0.001

//Activate this to measure (and print) the execution time of mainloop. Set MAIN_LOOP_
#define TIMING

//Activate this to enable automatic calibration, manual calibration via hand-controll
//#define AUTOMATIC_CALIBRATION

//Activate this to print debug logs, deactivate this when running the program for rea
//#define DEBUG

//More verbose debugging
//#define VERBOSE

//Active this to run only on computer connected (i.e. the program expects commands to
//#define TESTMODE

//Use this when programming the front arduino
//#define FRONT

#ifndef FRONT
  #define BACK
#endif
```

```
//Command buildup for front and back mode
#ifdef FRONT
  #define RIGHT_INCREMENT_COMMAND 'g'
  #define RIGHT_DECREMENT_COMMAND 'h'
  #define LEFT_INCREMENT_COMMAND 'e'
  #define LEFT_DECREMENT_COMMAND 'f'
#endif

//
#ifdef BACK
  #define RIGHT_INCREMENT_COMMAND 'k'
  #define RIGHT_DECREMENT_COMMAND 'm'
  #define LEFT_INCREMENT_COMMAND 'i'
  #define LEFT_DECREMENT_COMMAND 'j'
#endif

//Other commands
#define CONTROLLER_DEACTIVATE_COMMAND 'n'
#define CONTROLLER_ACTIVATE_COMMAND 'o'

#define MOTOR_MOVE_DELAY 30
#define MOTOR_STOP_DELAY 500


//Activate this (and activate TESTMODE, and DEACTIVATE DEBUG and VERBOSE) to output
//#define CONTROL_OUTPUT

#define PI_CONTROL_OUTPUT


//LOOKUP TABLE
static int basic_lookup[] = {-1, 1}; //If this works im happy. And it did, he was hap

//START POSITION
int current_pos_left = 0; //Counted in encoder tics
int current_pos_right = 0;

//Zeroing variable
boolean zeroed_left = false;
boolean zeroed_right = false;
boolean controller_activated = true;

//READING VARIABLES FROM SERIAL
int received_val = 0;
int received_val_left = 0;
int received_val_right = 0;
boolean newData = false;
boolean negative = false;
boolean message_left = false;
boolean message_right = false;
boolean message_recognized = false;
boolean first_char_read = false;
boolean convert = false;


//CONTROLL VARIABLES
int ref_pos_left_ticks = 0;
int ref_pos_right_ticks = 0;

int error_pos_left = 0;
int error_pos_right = 0;



//Heartbeat state
bool heartbeat_state = LOW;

//PWM measurement values
volatile int pwm_value = 0;
volatile int prev_time = 0;
int ref_test = 0;

//Variables for measuring time of main loop
unsigned long time_mainloop = 0;

//Variables for serial commands, +1 for null terminator
char command;

/* SPEED & DIRECTION INFO
SET SPEED (0-255)
SET DIRECTION (LOW = Motor CCW and Shaft CW, HIGH = Motor CW and Shaft CCW. HIGH is p
SET BREAK (LOW = No breaks, HIGH = Breaks on. Breaks means that speed = 0, no actual
*/

//INITIAL SPEED AND DIRECTION, CHANGE VALUE IN LOOP FOR TESTING
//LEFT
int control_signal_left = 0;  //int for setting the speed, output for PWM
bool direction_wanted_left = LOW;  //bool for tracking the direction
bool break_wanted_left = LOW; //bool for break level


//RIGHT
int control_signal_right = CONTROL_SIGNAL_MAX;  //int for setting the speed, output f
bool direction_wanted_right = LOW;  //bool for tracking the direction
bool break_wanted_right = LOW; //book for break level








void setup()
  {
    //USB serial interface
    Serial.begin(9600);

    //RXTX Hardware serial
    Serial1.begin(57600);

    //Setup Left Servo, setting up encoders with one channel RISING
    pinMode(PWM_LEFT_PIN, OUTPUT);
    pinMode(DIRECTION_LEFT_PIN, OUTPUT); //DIRECTION PIN IS OUTPUT
    pinMode(BREAK_LEFT_PIN, OUTPUT); //BREAK PIN IS OUTPUT
    pinMode(ENC_LEFT_Y_PIN, INPUT); //Yellow wire encoder right
pinMode(ENC_LEFT_G_PIN, INPUT); //Green wire encoder right

digitalWrite(DIRECTION_LEFT_PIN, direction_wanted_left);
digitalWrite(BREAK_LEFT_PIN, LOW); //Default no brakes

attachInterrupt(digitalPinToInterrupt(ENC_LEFT_G_PIN), encoder_ISR_left, RISING);
//attachInterrupt(digitalPinToInterrupt(ENC_LEFT_Y_PIN), encoder_ISR_left, RISING



//Setup Right Servo, setting up encoders with one channel RISING
pinMode(PWM_RIGHT_PIN, OUTPUT);
pinMode(DIRECTION_RIGHT_PIN, OUTPUT); //DIRECTION PIN IS OUTPUT
pinMode(BREAK_RIGHT_PIN, OUTPUT); //BREAK PIN IS OUTPUT
pinMode(ENC_RIGHT_Y_PIN, INPUT); //Yellow wire encoder right
pinMode(ENC_RIGHT_G_PIN, INPUT); //Green wire encoder right

digitalWrite(DIRECTION_RIGHT_PIN, direction_wanted_right);
digitalWrite(BREAK_RIGHT_PIN, LOW); //Default no brakes

attachInterrupt(digitalPinToInterrupt(ENC_RIGHT_G_PIN), encoder_ISR_right, RISING
//attachInterrupt(digitalPinToInterrupt(ENC_RIGHT_Y_PIN), encoder_ISR_right, RISIN

//Setup Button left and right
pinMode(ZEROING_LEFT_PIN, INPUT_PULLUP); //INPUT_PULLUP, PIN starts at 1, when co
pinMode(ZEROING_RIGHT_PIN, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(ZEROING_LEFT_PIN), button_ISR_left, FALLING
attachInterrupt(digitalPinToInterrupt(ZEROING_RIGHT_PIN), button_ISR_right, FALLIN

//Configure braking to be disabled
digitalWrite(BREAK_RIGHT_PIN, break_wanted_right); //Default no brakes
digitalWrite(BREAK_LEFT_PIN, break_wanted_left); //Default no brakes

//Configure BUILTIN LED
pinMode(LED_BUILTIN, OUTPUT);
digitalWrite(LED_BUILTIN, LOW);

//Configure Heartbeat PIN
#ifdef HEARTBEAT_PIN
  pinMode(HEARTBEAT_PIN, OUTPUT);
  digitalWrite(HEARTBEAT_PIN, LOW);
#endif

#ifdef PWM_INPUT_PIN
  pinMode(PWM_INPUT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(PWM_INPUT_PIN), pwm_ISR_rising, RISING);

#endif


#ifdef CONTROL_OUTPUT
  //Controller debug
  //Serial.print(ref_pos_left);
  //Serial.print(",");
  //Serial.print(current_pos_left);
  //Serial.print(",");
  //Serial.println(control_signal_left);
  Serial.println("ref,pos,cnt");
```

```
    #endif

  }


void pwm_ISR_rising(){

  prev_time = micros();

  //Change so that we trigger on falling edge instead
  attachInterrupt(digitalPinToInterrupt(PWM_INPUT_PIN), pwm_ISR_falling, FALLING);


}

void pwm_ISR_falling(){

  //Calculate how long we were on top
  pwm_value = micros() - prev_time;
  prev_time = 0;

  //Change so that we trigger on rising edge
  attachInterrupt(digitalPinToInterrupt(PWM_INPUT_PIN), pwm_ISR_rising, RISING);

}


//Interrupt routine for left encoder
void encoder_ISR_left()
  {

    //REG_POST_IN1 & PORT_PB11 checks if the value on GPIO 5 is HIGH, >> 11 shifts it
    //Change to minus to swap direction, so that both
    current_pos_left = current_pos_left - basic_lookup[(REG_PORT_IN1 & PORT_PB11) >>

  }


//Interrupt routine for right encoder
void encoder_ISR_right()
{

  //REG_POST_IN0 & PORT_PA20 checks if the value on GPIO 6 is HIGH, >> 20 shifts it t
  current_pos_right = current_pos_right + basic_lookup[(REG_PORT_IN0 & PORT_PA20) >> 2

}


//Interrupt routine for left button
void button_ISR_left()
{
  current_pos_left = 0;
  ref_pos_left_ticks = 0;
  control_signal_left = 0;

  //Set zeroed variable
  zeroed_left = true;
  #ifdef DEBUG
    Serial.println("Left␣servo␣zeroed");
  #endif

}


//Interrupt routine for right button
void button_ISR_right()
{
  current_pos_right = 0;
  ref_pos_right_ticks = 0;
  control_signal_right = 0;

  //Set zeroed variable
  zeroed_right = true;

  #ifdef DEBUG
    Serial.println("Right␣servo␣zeroed");
  #endif

}

//Increment the left motor gimbal slightly, and set new zero
void left_increment(){

  //Deactivate the controller
  controller_activated = false;

  //Zero the controller values
  controller_PI();

   //Actuate the value, i e move the motor a bit
   direction_wanted_left = DIRECTION_TICKS_INCREASE_LEFT;
   control_signal_left = CONTROL_SIGNAL_MAX;
   actuate();

   //Wait a bit, to allow the motor to get somewhere
   delay(MOTOR_MOVE_DELAY);

   //Stop the motor
   control_signal_left = 0;
   actuate();

   //Let the motor stop
   delay(MOTOR_STOP_DELAY);

    //Zero the controller values again before activating the controller
    controller_PI();

    //Set the zeroed ledf
    zeroed_left = true;

   //Activate the controller again
   controller_activated = true;

}
```

```
void left_decrement(){

  //Deactivate the controller
  controller_activated = false;

  //Zero the controller values
  controller_PI();

   //Actuate the value, i e move the motor a bit
   direction_wanted_left = DIRECTION_TICKS_DECREASE_LEFT;
   control_signal_left = CONTROL_SIGNAL_MAX;
   actuate();

   //Wait a bit, to allow the motor to get somewhere
   delay(MOTOR_MOVE_DELAY);

   //Stop the motor
   control_signal_left = 0;
   actuate();

   //Let the motor stop
   delay(MOTOR_STOP_DELAY);

    //Zero the controller values again before activating the controller
    controller_PI();

    //Set the zeroed flag
    zeroed_left = true;

   //Activate the controller again
   controller_activated = true;

}

void right_increment(){


  //Deactivate the controller
  controller_activated = false;

  //Zero the controller values
  controller_PI();

   //Actuate the value, i e move the motor a bit
   direction_wanted_right = DIRECTION_TICKS_INCREASE_RIGHT;
   control_signal_right = CONTROL_SIGNAL_MAX;
   actuate();

   //Wait a bit, to allow the motor to get somewhere
   delay(MOTOR_MOVE_DELAY);

   //Stop the motor
   control_signal_right = 0;
   actuate();

   //Let the motor stop
   delay(MOTOR_STOP_DELAY);
```

```
      //Zero the controller values again before activating the controller
      controller_PI();

      //Set the zeroed flag
      zeroed_right = true;

     //Activate the controller again
     controller_activated = true;

}

void right_decrement(){

    //Deactivate the controller
    controller_activated = false;

    //Zero the controller values
    controller_PI();

     //Actuate the value, i e move the motor a bit
     direction_wanted_right = DIRECTION_TICKS_DECREASE_RIGHT;
     control_signal_right = CONTROL_SIGNAL_MAX;
     actuate();

     //Wait a bit, to allow the motor to get somewhere
     delay(MOTOR_MOVE_DELAY);

     //Stop the motor
     control_signal_right = 0;
     actuate();

     //Let the motor stop
     delay(MOTOR_STOP_DELAY);

      //Zero the controller values again before activating the controller
      controller_PI();

      //Set the zeroed flag
      zeroed_right = true;

     //Activate the controller again
     controller_activated = true;

}



//Calibrate both sides function
void calibrate()
{

    //Inside calibrat
    #ifdef DEBUG
      Serial.print("Inside␣calibrate");
    #endif

      //Set speed
      if(zeroed_left == false)
      {
        #ifdef DEBUG
          Serial.println("Zeroing␣L");
        #endif
        control_signal_left = CONTROL_SIGNAL_MAX;
      }

      else
      {
        //Reset control and ref to not move away from 0 instantly (if we had previous r
        control_signal_left = 0;
        ref_pos_left_ticks = 0;
      }

      if(zeroed_right == false)
      {
        #ifdef DEBUG
          Serial.println("Zeroing␣R");
        #endif
        control_signal_right = CONTROL_SIGNAL_MAX;
      }

      else
      {
        //Reset control and ref to not move away from 0 instantly (if we had previous r
        control_signal_right = 0;
        ref_pos_right_ticks = 0;
      }

      //Set the direction
      direction_wanted_left = DIRECTION_TICKS_DECREASE_LEFT;
      direction_wanted_right = DIRECTION_TICKS_DECREASE_RIGHT;

      //Actuate it
      actuate();

      if(zeroed_left == false || zeroed_right == false)
      {
        calibrate();
      }
      else
      {

        #ifdef DEBUG
          Serial.println("Calibration␣done!");
        #endif

        //Send the calibration done command
        Serial1.println('d');

      }

      //TODO WRITE 'd' to the bus for calibration done

}

void receive_data_new(){
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData == false)
    {

      //Read into a char
      rc = Serial.read();

      //If we are not at the end we have good information
      if (rc != endMarker)
      {


      }

      //If we are at the end mark the information as new to quit the loop
      else
      {
        newData = true;
      }


    }

}


//code taken and modified from https://forum.arduino.cc/index.php?topic=288234.0
//Expected messages: "chars\n" where chars in this case are numeric representations o
void receive_data() {
    char endMarker = '\n';
    char rc;

    first_char_read = true;

    while (Serial.available() > 0 && newData == false)
    {
      //Reads into a char
      rc = Serial.read();


      #ifdef DEBUG
        //See what we have read
        Serial.print("Message read: ");
        Serial.println(rc);
      #endif


      if (rc != endMarker)
      {

        #ifdef DEBUG
          //Check that we have new data
          Serial.print("New data detected: ");
          Serial.println(rc);
        #endif
```

```
//Decide what to do with the data

//Calibrate
if (rc == 'c'){


  //Flush the buffer to not read in junk characters that are displayed afterwar
  //Serial1.flush();

  #ifdef DEBUG
    //Check that we have new data
    Serial.println("Calibrate cmd");
  #endif

  //Turn OFF the builting led (yellow)
  digitalWrite(LED_BUILTIN, LOW);

  //Reset the zeroing flags to allow calibration
  zeroed_left = false;
  zeroed_right = false;

  //Initiate calibration
  #ifdef AUTOMATIC_CALIBRATION
  calibrate();
  #endif

  convert = false;
  message_recognized = true;

}

//Set the left angle
else if (rc == 'l'){

  #ifdef DEBUG
    //Check that we have new data
    Serial.println("Left angle cmd");
  #endif

  //Set the flag to signify that the number coming after is for left servo
  convert = false;
  message_recognized = true;
  message_left = true;

  //Turn OFF the builting led (yellow)
  digitalWrite(LED_BUILTIN, LOW);

  //TODO Set left reference value here
}

//Set the right angle
else if (rc == 'r'){

  #ifdef DEBUG
    //Check that we have new data
    Serial.println("Right angle cmd");
  #endif
```

```
    //Set the flag to signify that number coming after is for right servo
    convert = false;
    message_recognized = true;
    message_right = true;

    //Turn OFF the builting led (yellow)
    digitalWrite(LED_BUILTIN, LOW);

    //TODO Set right reference value here

  }

  //Heartbeat, toggle the heartbeat pin
  else if (rc == RIGHT_DECREMENT_COMMAND){

    #ifdef DEBUG
      //Check that we have new data
      #ifdef HEARTBEAT_PIN
        Serial.println("Heartbeat cmd");
      #endif
    #endif

    convert = false;
    message_recognized = true;

    //Toggle the heartbeat pin, and change the state
    #ifdef HEARTBEAT_PIN
    digitalWrite(HEARTBEAT_PIN, !heartbeat_state);
    heartbeat_state = !heartbeat_state;
    #endif

  }

  //If we didn't match any of the above statements we didn't recognize the comman
  else {

    if ( newData == false && !message_recognized) {
    #ifdef DEBUG
      //Check that we have new data
      Serial.println("Didnt recognize cmd");
    #endif

    convert = false;
    message_recognized = false;
    message_right = false;
    message_left = false;


    //Turn ON the builting led (yellow)
    digitalWrite(LED_BUILTIN, HIGH);

    }

  }


  //If we have ran through all of the known commands it must be time for conversio
  if(message_recognized == true && convert == true)
```

```
 {

   //Set the value negative
   if(rc == '-')
   {
     negative = true;

     #ifdef DEBUG
       Serial.println("Negative␣value␣found");
       Serial.println(rc);
     #endif

     convert = false;

   }

//   if(convert == true)

   if(negative == true)
   {


       if (message_left == true){

         #ifdef DEBUG
           Serial.print("calc␣neg␣left:␣");
           Serial.println(rc);
         #endif

         //ASCII conversion
         rc = rc - 48;

         received_val_left = received_val_left*10 - rc;

       }



       if (message_right == true){

         #ifdef DEBUG
           Serial.print("calc␣neg␣right:␣");
           Serial.println(rc);

         #endif

         received_val_right = received_val_right*10 - rc;

       }

     }

   //If not negative it is positive
   else
   {

     if (message_left == true){
```

```
              #ifdef DEBUG
                Serial.print("calc␣pos␣left:␣");
                Serial.println(rc);
              #endif

              received_val_left = received_val_left*10 + rc;

            }

            if (message_right == true){

              #ifdef DEBUG
                Serial.print("calc␣pos␣right:␣");
                Serial.println(rc);
              #endif

              received_val_right = received_val_right*10 + rc;

            }



          }


        }

        //If message is not recognized we will not convert the subsequent stuff to in
        else{
          Serial.println("message␣not␣recognized␣(or␣conversion␣not␣selected),␣␣no␣co
        }



      }

    if(message_recognized == true && convert == false){
      convert = true;
    }


    //else
    if(rc == endMarker)
    {
      newData = true;
      negative = false;

      message_right = false;
      message_left = false;
    }
  }

  first_char_read = false;

}


//Receive data on the old way
```

```
void receive_data_old(){

  char endMarker = '\n';
  char rc;

  while (Serial.available() > 0 && newData == false)
    {
      //Reads into a char
      rc = Serial.read();

      #ifdef DEBUG
        //See what we have read
        Serial.print("Message read: ");
        Serial.println(rc);
      #endif


      if (rc != endMarker)
      {

        if (rc == 'c'){

        #ifdef DEBUG
          //Check that we have new data
          Serial.println("Calibrate cmd");
        #endif

        //Turn OFF the builting led (yellow)
        digitalWrite(LED_BUILTIN, LOW);

        //Reset the zeroing flags to allow calibration
        zeroed_left = false;
        zeroed_right = false;

        //Initiate calibration
        #ifdef AUTOMATIC_CALIBRATION
          calibrate();
        #endif

        }


        //Set the value negative
        else if(rc == '-')
        {
          negative = true;

          #ifdef DEBUG
            Serial.println("Negative value found");
            Serial.println(rc);
          #endif

        }

        else
        {

          //ASCII conversion
```

```
            rc = rc - 48;

            if(negative)
            {
              received_val = received_val*10 - rc;
            }
            else
            {
              received_val = received_val*10 + rc;
            }
          }

      }

      else
      {
        newData = true;
        negative = false;
      }

      }

  }


  //Parse the data received
  void parse_data_old()
  {
    if (newData == true)
    {



        #ifdef DEBUG
          //Check that we have new data
          Serial.println("New reference command received");
        #endif

      set_servo_position(received_val, 'l');
      set_servo_position(received_val, 'r');

      received_val = 0;
      received_val = 0;
      newData = false;


    }

    else
    {
      #ifdef DEBUG
        //We didn't have new data
        //Serial.print("Not new data");
      #endif

    }

  }
```

```
//Calculate wanted position
void set_servo_position(int change_pos, char side)
{
  if (side == 'l')
  {
    ref_pos_left_ticks = change_pos * DEGREES_TO_TICKS; //change_pos is degrees, 1 de

    #ifdef DEBUG
      //Set reference position
      Serial.print("L␣ref␣set:␣");
      Serial.println(ref_pos_left_ticks);
    #endif

  }

  else if (side == 'r')
  {
    ref_pos_right_ticks = - change_pos * DEGREES_TO_TICKS; //change_pos is degrees, 1

    #ifdef DEBUG
      //Set reference position
      Serial.print("R␣ref␣set:␣");
      Serial.println(ref_pos_right_ticks);
    #endif

  }

}


//PI controller
void controller_PI()
{

    double error_integral_pos_left = 0;
    double error_integral_pos_right = 0;

    static double error_pos_left_old = 0;
    static double error_pos_right_old = 0;

    static int control_signal_left_old = 0;
    static int control_signal_right_old = 0;

    //Controller variables (used in the controller_PI function)
    double p_part_left = 0;
    double p_part_right = 0;
    double i_part_left = 0;
    double i_part_right = 0;


  if (controller_activated == true)
  {

  #ifdef VERBOSE
    Serial.println("Calc␣cnt");
    Serial.print("r:␣");
    Serial.println(ref_pos_left_ticks);
    Serial.print("y:␣");
    Serial.println(current_pos_left);
    Serial.print("e:␣");
    Serial.println(error_pos_left);
  #endif

    //Control variables (for control_PI() function)


    //Calculate the error
    error_pos_left = ref_pos_left_ticks - current_pos_left;
    error_pos_right = ref_pos_right_ticks - current_pos_right;

    //Integrate the error
    error_integral_pos_left = error_pos_left_old * TIME_DELTA;
    error_integral_pos_right = error_pos_right_old * TIME_DELTA;

    //Calculate the P part
    p_part_left = KP * error_pos_left;
    p_part_right = KP * error_pos_right;

    //Calculate the I part
    i_part_left = KI * error_integral_pos_left;
    i_part_right = KI * error_integral_pos_right;

    //Calculate the control signal, leave out KI for now
    control_signal_left =  (int) p_part_left + (int) i_part_left;
    control_signal_right =  (int) p_part_right + (int) i_part_right;


  //Set the proper directions

    //LEFT
    //Same as alpha controller but with control signal
    if(control_signal_left > 0)
    {
      direction_wanted_left = DIRECTION_TICKS_INCREASE_LEFT;

    }
    else
    {
      direction_wanted_left = DIRECTION_TICKS_DECREASE_LEFT;
    }
    //Saturate the left signal
    if(abs(control_signal_left) > CONTROL_SIGNAL_MAX) control_signal_left = CONTROL_SI


    //RIGHT
    //Same as alpha controller but with control signal
    if(control_signal_right > 0)
    {
      direction_wanted_right = DIRECTION_TICKS_INCREASE_RIGHT;

    }
    else
    {
      direction_wanted_right = DIRECTION_TICKS_DECREASE_RIGHT;
    }
    //Saturate the right signal
    if(abs(control_signal_right) > CONTROL_SIGNAL_MAX) control_signal_right = CONTROL_

  //Look at the values
  #ifdef PI_CONTROL_OUTPUT
      Serial.print(ref_pos_left_ticks);
      Serial.print(",");
      Serial.print(current_pos_left);
      Serial.print(",");
      Serial.print(error_pos_left);
      Serial.print(",");
      Serial.print((int) p_part_left);
      Serial.print(",");
      Serial.print(i_part_left);
      Serial.print(",");
      Serial.print(control_signal_left);
      Serial.print(",");
      Serial.println(direction_wanted_left);


  #endif

   #ifdef VERBOSE
      Serial.print("L err: ");
      Serial.print(error_pos_left);
      Serial.print("\tR err: ");
      Serial.println(error_pos_right);
      Serial.print("L err int: ");
      Serial.print(error_integral_pos_left);
      Serial.print("\tR err int: ");
      Serial.println(error_integral_pos_right);
      Serial.print("L cntrl: ");
      Serial.print(control_signal_left);
      Serial.print("\tR cntrl: ");
      Serial.println(control_signal_right);
      Serial.println();
    #endif

   //Save the values for next run
   error_pos_left_old = error_pos_left;
   error_pos_right_old = error_pos_right;
   control_signal_left_old = control_signal_left;
   control_signal_right_old = control_signal_right;

  }

  //If the controller has been deactivated, zero all of the variables
  else
  {

    #ifdef DEBUG
      Serial.println("cnt off");
    #endif

    error_integral_pos_left = 0;
    error_integral_pos_right = 0;
    error_pos_left_old = 0;
    error_pos_right_old = 0;

    control_signal_left_old = 0;
    control_signal_right_old = 0;

    //Controller variables (used in the controller_PI function)
    p_part_left = 0;
    p_part_right = 0;
    i_part_left = 0;
    i_part_right = 0;

    error_pos_left_old = 0;
    error_pos_right_old = 0;

    error_pos_left = 0;
    error_pos_right = 0;

    control_signal_left = 0;
    control_signal_right = 0;

    current_pos_left = 0;
    current_pos_right = 0;


    #ifdef VERBOSE

    Serial.println("After␣zero:␣\n");

    Serial.println("Calc␣cnt");
    Serial.print("r:␣");
    Serial.println(ref_pos_left_ticks);
    Serial.print("y:␣");
    Serial.println(current_pos_left);
    Serial.print("e:␣");
    Serial.println(error_pos_left);

      Serial.print("L␣err:␣");
      Serial.print(error_pos_left);
      Serial.print("\tR␣err:␣");
      Serial.println(error_pos_right);
      Serial.print("L␣err␣int:␣");
      Serial.print(error_integral_pos_left);
      Serial.print("\tR␣err␣int:␣");
      Serial.println(error_integral_pos_right);
      Serial.print("L␣cntrl:␣");
      Serial.print(control_signal_left);
      Serial.print("\tR␣cntrl:␣");
      Serial.println(control_signal_right);
      Serial.println();

    #endif

  }


}
```

```
//Calculate control signal, super blunt P controller
void controller()
{


if(controller_activated == true)
  {

    //Control signal calculation

      error_pos_left = ref_pos_left_ticks - current_pos_left;
      error_pos_right = ref_pos_right_ticks - current_pos_right;

      //UPDATING LEFT MOTOR, decide direction
      if (error_pos_left > 0)
      {

        //Error positive --> we have gone too far, check paper
        //direction_wanted_left = DIRECTION_DECREASE_LEFT;

        //Testing this
        direction_wanted_left = DIRECTION_TICKS_INCREASE_LEFT;


      }
      else
      {

        //direction_wanted_left = HIGH;

        //Ticks and angles are inverse for the left motors
        //direction_wanted_left = DIRECTION_INCREASE_LEFT;

        //Testing this
        direction_wanted_left = DIRECTION_TICKS_DECREASE_LEFT;
      }


      //Calculate control signal for left motor
      if (abs(error_pos_left) > CONTROL_SIGNAL_MAX)
      {
        control_signal_left = CONTROL_SIGNAL_MAX;
      }
      else if (abs(error_pos_left) < 100)
      {
        control_signal_left = 0;
      }
      else if (abs(error_pos_left) < CONTROL_SIGNAL_MAX)
      {
        control_signal_left = abs(error_pos_left);
      }

      //Calculate control signal for right motor
      if ((error_pos_right) > 0)
      {

        //direction_wanted_right = DIRECTION_INCREASE_RIGHT;
        //Testing this
        direction_wanted_right = DIRECTION_TICKS_INCREASE_RIGHT;
      }
      else
      {
        //direction_wanted_right = DIRECTION_DECREASE_RIGHT;

        //Testing this
        direction_wanted_right = DIRECTION_TICKS_DECREASE_RIGHT;

      }

      if (abs(error_pos_right) > CONTROL_SIGNAL_MAX)
      {
        control_signal_right = CONTROL_SIGNAL_MAX;
      }
      else if (abs(error_pos_right) < 100)
      {
        control_signal_right = 0;
      }
      else if (abs(error_pos_right) < CONTROL_SIGNAL_MAX)
      {
        control_signal_right = abs(error_pos_right);
      }

    //Control signal calculation END


    #ifdef DEBUG
      Serial.print("L r: ");
      Serial.print(ref_pos_left_ticks);
      Serial.print("\t\tR r: ");
      Serial.println(ref_pos_right_ticks);
      Serial.print("L y: ");
      Serial.print(current_pos_left);
      Serial.print("\t\tR y: ");
      Serial.println(current_pos_right);
      Serial.print("L err: ");
      Serial.print(error_pos_left);
      Serial.print("\tR err: ");
      Serial.println(error_pos_right);
      Serial.print("L cntrl: ");
      Serial.print(control_signal_left);
      Serial.print("\tR cntrl: ");
      Serial.println(control_signal_right);
      Serial.println();
    #endif

  }

}

//Set the direction and change the PWM frequency
void actuate(){

    //Change the PWM signal
    analogWrite(PWM_LEFT_PIN, control_signal_left);
    analogWrite(PWM_RIGHT_PIN, control_signal_right);

    //Set the direction
    digitalWrite(DIRECTION_LEFT_PIN, direction_wanted_left);
    digitalWrite(DIRECTION_RIGHT_PIN, direction_wanted_right);

}


void parse_data_PWM(){

  //Actually the below was not true after restarting the controller. The values were
  //0% (or 988 us on the controller) = pwm_value 17 000 on the arduino
  //100% (or 2000 us on the controller) = pwm_value 16 000 on the arduino

  //This map was for the simulation from the arduino
  ref_test = map(pwm_value, 1000, 2000, 0, MAX_TICKS);

  //This map was for the simulation from the contrlller
  //ref_test = map(pwm_value, 17000, 16000, 0, 80);

  //Protect from overflow
  if(ref_test > 80) ref_test = 80;
  if(ref_test < 0) ref_test = 0;


  #ifdef DEBUG
    Serial.print("pwm: ");
    Serial.println(pwm_value);

    Serial.print("rt: ");
    Serial.println(ref_test);
  #endif


  //Set the reference signal
  ref_pos_left_ticks = ref_test * DEGREES_TO_TICKS;
  ref_pos_right_ticks = - ref_test * DEGREES_TO_TICKS;


}

//Function to debug serial communication
void print_message(){


  char endMarker = '\n';
  char received_byte;

    #ifdef DEBUG
      //Serial.print("received message: ");
    #endif


  //New way of parsing entire message

#ifdef TESTMODE
  while (Serial.available() > 0)
```

```
#else
  while (Serial1.available() > 0)
#endif
  {




    //Parse entire message and put it in message buffer
#ifdef TESTMODE
    received_byte = Serial.read();
#else
    received_byte = Serial1.read();
#endif

    if(received_byte != endMarker)
    {

        #ifdef DEBUG
          //Serial.print("recb: ");
          Serial.print(received_byte);
        #endif
    }
    else
    {

        #ifdef DEBUG
          //Serial.print("recb: ");
          Serial.print(endMarker);
        #endif

    }



  }

}


//Send message on serial communication
void send_message(){

  #ifdef DEBUG

    Serial.println("sending: d");

  #endif

    Serial1.println('d');

}


//Get the number from the Serial buffer
int get_number(){

  #ifdef VERBOSE
```

```
    Serial.print("Inside get_number");
  #endif

  char endMarker = '\n';
  char received_byte;
  int calculated_number = 0;
  negative = false;

#ifdef TESTMODE
    while (Serial.available() > 0){
#else
    while (Serial1.available() > 0){
#endif

#ifdef TESTMODE
    received_byte = Serial.read();
#else
    received_byte = Serial1.read();
#endif

    if(received_byte != endMarker){

      if(received_byte == '-')
      {
        negative = true;
      }

      else
      {
        //Convert from ascii number to integer and add it to the previous values
        calculated_number = calculated_number * 10 +  (received_byte - 48);
      }

    }

    else{

      break;

    }



  }

  //Make the number negative
  if(negative == true) calculated_number = calculated_number * -1;

  //Bugfix for
  if(calculated_number > 8000) calculated_number = calculated_number / 10;


  #ifdef DEBUG
    Serial.print("extracted number: ");
    Serial.println(calculated_number);
  #endif
```

```
    //Mapping to change range of 1000 to 2000 to 0 to 90 degrees
    calculated_number = map(calculated_number, 1000, 2000, 0, MAX_TICKS);

    if(calculated_number > MAX_TICKS)calculated_number = MAX_TICKS;
    if(calculated_number < 0)calculated_number = 0;



    return calculated_number;


}


//New function for receiving custom serial messages
//Read data from Serial1 interface (Pin 13 and 14)
void receive_custom_data(){

    #ifdef VERBOSE
      Serial.println("Inside␣receive_custom_data");
    #endif

    //First reader which was used to confirm the connection to the Pixhawk with RX / TX
    /*
    while(Serial1.available()){
      char inByte = Serial1.read();
      Serial.print("got byte: ");
      Serial.println(inByte);
    }
    */

    char endMarker = '\n';
    char received_byte;

    //old values stored to avoid get
    static int received_val_left_old = 0;


    //New way of parsing entire message

#ifdef TESTMODE
    while (Serial.available() > 0)
#else
    while (Serial1.available() > 0)
#endif
    {

      #ifdef VERBOSE
        Serial.println("reading␣byte");
      #endif


      //Parse entire message and put it in message buffer
#ifdef TESTMODE
      received_byte = Serial.read();
#else
      received_byte = Serial1.read();
#endif
```

```
if(received_byte != endMarker)
{

    #ifdef DEBUG
      Serial.print("recb: ");
      Serial.println(received_byte);
    #endif

    //Command for calibration
    if(received_byte == 'c'){

      #ifdef DEBUG
        Serial.print("received c cmd: ");
        Serial.println(received_byte);
      #endif

      //Turn OFF the builting led (yellow)
      digitalWrite(LED_BUILTIN, LOW);

      command = 'c';
      received_val = 0;
      newData = true;

    }

    //Command for right motors
    else if(received_byte == 'r'){

      #ifdef DEBUG
        Serial.print("received r cmd: ");
        Serial.println(received_byte);
      #endif

      //Turn OFF the builting led (yellow)
      digitalWrite(LED_BUILTIN, LOW);

      command = 'r';
      received_val_right = get_number();
      newData = true;

    }

    //Command for left motors
    else if(received_byte == 'l'){

      #ifdef DEBUG
        Serial.print("received l cmd: ");
        Serial.println(received_byte);
      #endif

      //Turn OFF the builting led (yellow)
      digitalWrite(LED_BUILTIN, LOW);

      command = 'l';
      received_val_left = get_number();
      newData = true;
```

```
}

else if(received_byte == 'b'){

  #ifdef DEBUG
    Serial.print("received␣b␣cmd:␣");
    Serial.println(received_byte);
  #endif

  //Turn OFF the builting led (yellow)
  digitalWrite(LED_BUILTIN, LOW);

  command = 'b';

  //Get the number of ticks to go to
  received_val_left = get_number();

  //Check what number we got

  //Check that we suddenly didn't get zero
  if(received_val_left == 0){
    received_val_left = received_val_left_old;
  }
  else{
    received_val_left_old = received_val_left;
  }


  //Look at the number instead if get the number
  //print_message();

  received_val_right = received_val_left;
  newData = true;

}

else if(received_byte == LEFT_INCREMENT_COMMAND){


  #ifdef DEBUG
    Serial.print("received␣e␣cmd:␣");
    Serial.println(received_byte);
  #endif

  command = LEFT_INCREMENT_COMMAND;
  newData = true;

}

else if(received_byte == LEFT_DECREMENT_COMMAND){


  #ifdef DEBUG
    Serial.print("received␣f␣cmd:␣");
    Serial.println(received_byte);
  #endif

  command = LEFT_DECREMENT_COMMAND;
```

```
    newData = true;

  }


  else if(received_byte == RIGHT_INCREMENT_COMMAND){


    #ifdef DEBUG
      Serial.print("received g cmd: ");
      Serial.println(received_byte);
    #endif

    command = RIGHT_INCREMENT_COMMAND;
    newData = true;

  }



  else if(received_byte == RIGHT_DECREMENT_COMMAND){


    #ifdef DEBUG
      Serial.print("received h cmd: ");
      Serial.println(received_byte);
    #endif

    command = RIGHT_DECREMENT_COMMAND;
    newData = true;

  }

  else if(received_byte == CONTROLLER_ACTIVATE_COMMAND){


    #ifdef DEBUG
      Serial.print("received cnt activate cmd: ");
      Serial.println(received_byte);
    #endif

    command = CONTROLLER_ACTIVATE_COMMAND;
    newData = true;

  }

  else if(received_byte == CONTROLLER_DEACTIVATE_COMMAND){


    #ifdef DEBUG
      Serial.print("received cnt deactivate cmd: ");
      Serial.println(received_byte);
    #endif

    command = CONTROLLER_DEACTIVATE_COMMAND;
    newData = true;

  }
```

```
          //We didntt recognize the command
          else{

            #ifdef DEBUG
              Serial.print("not recognized cmd: ");
              Serial.println(received_byte);
            #endif

            //Turn ON the builting led (yellow)
            digitalWrite(LED_BUILTIN, HIGH);

            //Testing this
            //received_val = 0;

            //newData = false;

          }
      }

      //Here we have detected the endMarker
      else{

          #ifdef DEBUG
            Serial.print("reached endmarker");
            Serial.println(received_byte);
          #endif

          //Flush the buffer
          //Serial1.flush();

      }

    }




}


//New function for parsing given string and decide what to do
void parse_custom_data(){

  #ifdef VERBOSE
    Serial.println("Inside parse_custom_data");
  #endif

  if(newData == true){

      //Calibrate command
      if(command == 'c'){

        #ifdef DEBUG
          Serial.println("calibrate");
        #endif

        //Run the calibration
    zeroed_left = false;
    zeroed_right = false;

    //Automatic calibration
    #ifdef AUTOMATIC_CALIBRATION
      calibrate();
    #endif

    //We have acted on the data, so we set it to false
    newData = false;

  }

  //Right motor reference command
  else if(command == 'r'){

    #ifdef DEBUG
      Serial.println("right");
    #endif

    //Set the reference signal
    ref_pos_right_ticks = received_val_right;

    //We have acted on the data, so we set it to false
    newData = false;


  }

  //Left motor reference command
  else if(command == 'l'){

    #ifdef DEBUG
      Serial.println("left");
    #endif

    //Set the reference signal
    ref_pos_left_ticks = received_val_left;

    //We have acted on the data, so we set it to false
    newData = false;

  }

  //Both motors reference command
  else if(command == 'b'){

    #ifdef DEBUG
      Serial.print("both,␣");
      Serial.print("l:␣");
      Serial.print(received_val_left);
      Serial.print("r:␣");
      Serial.println(received_val_right);
    #endif

    newData = false;

    //Set the reference signal
    ref_pos_left_ticks =  received_val_left;
    ref_pos_right_ticks = received_val_right;

    //Activate the controller, TESTING THIS, should actually be in the increment/
    //controller_activated = true;

  }

  else if(command == LEFT_INCREMENT_COMMAND){

    //We have read the message
    newData = false;

    //Increment the left motor
    left_increment();


  }

  else if(command == LEFT_DECREMENT_COMMAND){

    //We have read the message
    newData = false;

    //Decrement the left motor
    left_decrement();


  }

  else if(command == RIGHT_INCREMENT_COMMAND){

    //We have read the message
    newData = false;

    //Increment the right motor
    right_increment();


  }

  else if(command == RIGHT_DECREMENT_COMMAND){

    //We have read the message
    newData = false;

    //Decrement the right motor
    right_decrement();


  }

  //Deactivate the controller
  else if(command == CONTROLLER_DEACTIVATE_COMMAND){

    #ifdef DEBUG
      Serial.println("cnt␣deactivate");
    #endif
```

```
        //We have read the message
        newData = false;

        //Deactivate the controller
        controller_activated = false;

        //Zero the values
        controller_PI();


      }


      //Activate the controller
      else if(command == CONTROLLER_ACTIVATE_COMMAND){

        #ifdef DEBUG
          Serial.println("cnt␣activate");
        #endif

        //We have read the message
        newData = false;

        //Deactivate the controller
        controller_activated = true;

        //Zero the values
        controller_PI();

      }


    }

  else{

      #ifdef VERBOSE
        Serial.println("No␣new␣data␣to␣parse");
      #endif

  }


}

//Main loop
void loop()
{

  //Check the time before running everything, can be commented out
  #ifdef TIMING
    time_mainloop = millis();
  #endif

  //Parse and set reference via Serial (NOT to be combined with parse_data_PWM or pars
  //receive_data_old(); //Receives a value from serial, stores it as a received_val (
  //parse_data_old(); //Changes position that is aimed for (set the reference signal)
```

```
//Parse and set reference via PWM (NOT to be combined with receive_data_old() and p
//parse_data_PWM(); //Map the PWM signal to a reference and print the reference (Re

//Parse and set reference via serial connection, new implementation
receive_custom_data();
parse_custom_data();

//Function to debug serial communication
//print_message();

//Send a fixed message on Serial1 to the pixhawk
//send_message();

//controller(); //Calculate the control signal (very simple P controller)

controller_PI(); //Calculate the control signal (PI controller containing vast ammo

actuate(); //Actuate the wanted values



//Debug print
#ifdef DEBUG
/*
  Serial.print("L pos: ");
  Serial.print(current_pos_left);
  Serial.print("\tR pos: ");
  Serial.println(current_pos_right);
  Serial.print("L ref pos: ");
  Serial.print(ref_pos_left_ticks);
  Serial.print("\tR ref pos: ");
  Serial.println(ref_pos_right_ticks);
  Serial.print("L cntrl: ");
  Serial.print(control_signal_left);
  Serial.print("\tR cntrl: ");
  Serial.println(control_signal_right);
  Serial.println();
  */
#endif

#ifdef CONTROL_OUTPUT
  //Controller debug

  #ifndef PI_CONTROL_OUTPUT

    Serial.print(ref_pos_left_ticks / DEGREES_TO_TICKS);
    Serial.print(",");
    Serial.print(current_pos_left / DEGREES_TO_TICKS);
    Serial.print(",");
    Serial.println((control_signal_left * 100) / CONTROL_SIGNAL_MAX);

  #endif

#endif
```

```
  //Delay to create some sort of periodicity, why do we have this?
  delay(MAIN_LOOP_DELAY);

    //Print time the mainloop took, to be used without debug prints
//#ifdef TIMING
//   Serial.print("t: ");
//   Serial.println(millis() - time_mainloop);
//#endif
}
```

G Weight Budget

This page intentionally left blank

Parts

	Amount	Weight (g) 1st	Total weight (g)	
Flight Controller				
Pixhawk 2.1	1	39	39	
			0	
Motor			0	
Dr. Mad Thrust 70mm 11-Blade Alloy EDF 1900kv Motor - 1900watt (6S) (Counter Rotating)	14	230	3220	
			0	
ESC (Electronic speed controller)			0	
Turnigy AE-80A Brushless ESC	14	78	1092	
			0	
Batteries			0	
Turnigy 5000mAh 6S 20C Lipo Pack w/XT60 Connector	7	765	5355	
			0	
Radio			0	
Taranis XD9 Plus (including rec)	1	0	0	
Antenna	1	11,5	11,5	
FrSky X8R Reciver	1	17	17	
Camera	1	35	35	
Camera transmitter	1	35	35	
			0	
GPS			0	
GNSS GPS Module	1	12	12	