ROBOTICS
AND
MECHATRONICS

PROOF OF CONCEPT OF TEST SETUP FOR PORTWINGS ROBIRD

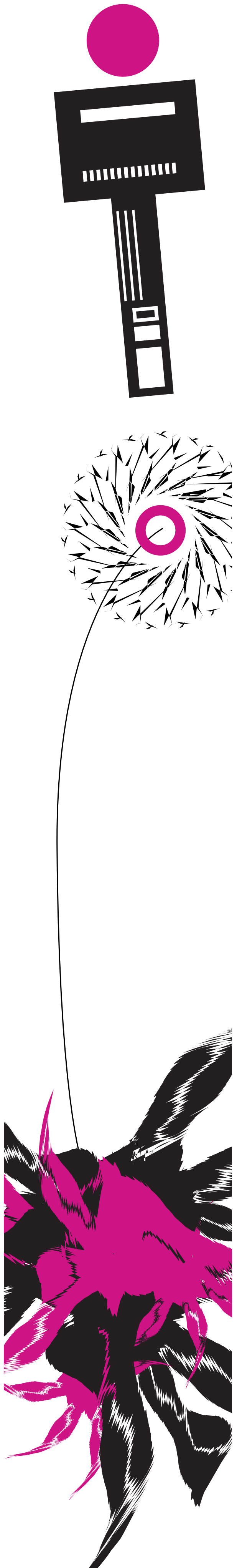
B.G. (Bo) Treur

BSC ASSIGNMENT

July, 2024

028RaM2024
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE
UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE



Summary

In the project Portwings, a robotic bird is studied. To gather data, it has to be tested in a wind tunnel. The test setup for this does not exist so a proof of concept for a test setup has to be designed. The proof of concept would need to measure the direction and tension force on a cable connected to a fixed-wing plane from a fixed point in space. The measured data would be used in a controller to keep the plane centred in the applied air stream. To realise this, a housing for a FrSky M7 gimbal was made and calibrated using code on an ESP32Wroom to measure angles in the two axes of interest. To measure the tension force, a device with a spring and time-of-flight sensor, VL6180, was constructed with a range of 0 [N] to 3 [N] and an average error of 0.089 [N]. This device was realised after multiple design cycles, which included testing. To make a control loop, a model of the test setup was created in 20-sim. The model consists of a cable model and a plane model. It was partly validated by test results, but not fully finished. To finalize the model, a few more tests have to be conducted. These tests are mostly thought out and only have to be executed. Due to time constraints, the proof of concept was not fully realised, however, initial work has been done.

Contents

1 Introduction

In the ERC project Portwings a robotic bird is studied. To gather data on this robotic bird, it has to be tested in a large wind tunnel. However, this would be hard because of the behaviour of the bird, which has to be as little constrained as possible to be optimally tested. A possible idea is to connect a cable to the nose of the bird which comes from the middle of the wind tunnel source. It should be possible to steer the bird to the centre of the stream by using the angle of the cable and its tension. To test this idea, a proof of concept has to be set up, which tests this idea on a small controllable fixed-wing plane. A cable is connected from the nose of the plane to a small mechanism that can measure the angle that the cable makes. To measure the tension force, an elastic element could be used while measuring its displacement. This setup is placed in front of a big fan which generates the air stream. The idea could be scaled up to the actual setup when this succeeds. This assignment starts by constructing a mechanism which can measure the tension force and the angle of the cable and do some initial testing.
Research questions for this assignment are defined as follows:
Main research question: How can a setup be designed which is able to measure the angle of the cable and its tension force while attached to a, to-be-chosen, small controllable fixed-wing plane?
Sub research question 1: What plane has to be chosen for the setup?
Sub research question 2: How to measure tension in a cable?
Sub research question 3: How to measure the angle of a cable from a fixed point in space which has 2 degrees of freedom?
Sub research question 4: How is a control loop for this system designed?

This report first goes into a bit of theory on modelling a plane, and then the design of the proof of concept of the setup will be explained. Following is the section about experiments conducted on the design. And lastly, an evaluation of the design can be found.

2 Theory

In this section, theory is highlighted about the modelling of a plane, which is later used in the design of the model of the system.

2.1 Flight theory for modelling

2.1.1 Basics of flight

An airfoil generates a lift and a drag force. The lift force is perpendicular and the drag force is parallel to the air stream. The lift and drag forces depend on multiple factors and among those are the coefficient of lift and the coefficient of drag. These coefficients depend on the shape of the airfoil and the angle the airfoil has with the incoming flow or Angle of Attack (AoA). These forces do not act at the same point when varying the angle of attack but move with the centre of pressure which is more complicated to model. However, for low-speed airfoil design it is possible to model the aerodynamic forces in one fixed point, the Aerodynamic Center (AC). This can only be done when adding a moment on this point which is called the Aerodynamic Moment (AM). This aerodynamic moment stays approximately constant with changing angle of attack [1], and has been proven empirically and analytically. The AC is usually located at 1/4 chord length of a low-speed airfoil [1]. This means that if the AM, drag and lift forces are measured, it is possible to make a simple model of an airfoil.
The equations for drag and lift can be seen in Equation 1 and 2, respectively. Where C_d = coefficient of drag, C_l = coefficient of lift, ρ = air density $[kg/m^3]$, u = flow velocity [m/s] and A = reference area $[m^2]$.

$$F_d = 0.5 * C_d * \rho * u^2 * A \tag{1}$$

$$F_l = 0.5 * C_l * \rho * u^2 * A \tag{2}$$

2.1.2 Control & Stability

An airplane is equipped with control surfaces such as ailerons, flaps, and a rudder. When these control surfaces are deflected, the lift coefficient (C_L) of the corresponding airfoil changes, allowing the plane to execute various maneuvers [1].

Lateral static stability refers to the stability of an aircraft in roll. This stability becomes significant when an aircraft acquires a roll angle due to an external disturbance, such as a gust of wind. When the aircraft is tilted, the lift force no longer acts directly opposite to the gravitational force, resulting in a sideways and downward force that translates into a velocity, known as sideslip. And the relative wind stream can be seen in Figure 1 as V_s.

If an aircraft has wings that are parallel to each other, they will generate the same amount of lift because both wings have the same angle of attack (AoA). In this configuration, the aircraft does not naturally return to its equilibrium position after a disturbance.

Consider an aircraft with wings mounted at an angle Γ, known as the dihedral angle, as shown in Figure 2b. In this configuration, the wings produce different amounts of lift when the aircraft experiences a sideslip, leading to a rolling motion that restores the aircraft to its equilibrium position. This phenomenon is known as the dihedral effect [1].

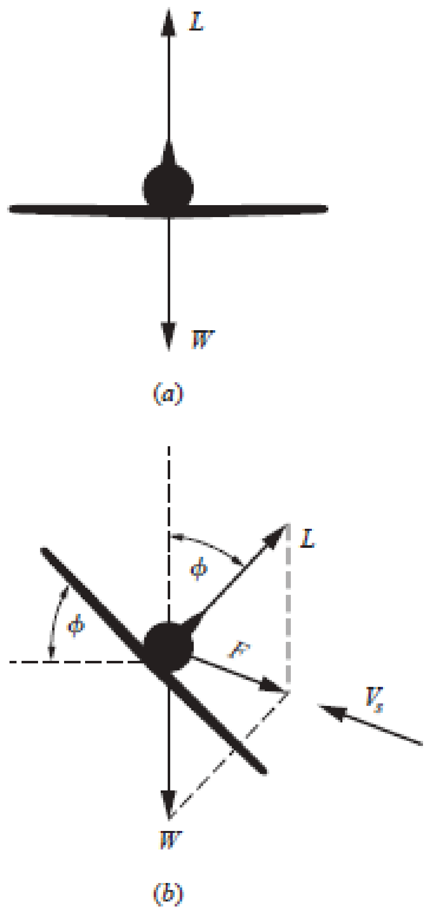


Figure 1: Generation of sideslip [1]

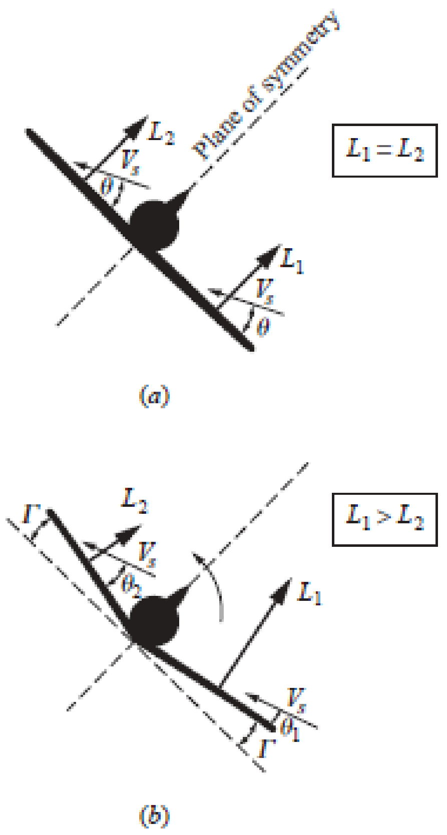


Figure 2: Effect of dihedral [1]

3 Design

The main idea of the design is a cable connected from a measuring device to the nose of a fixed-wing plane. This measuring device is attached in the middle of the ventilator. The measuring device consists of two parts; one measuring the angle of the cable, and the other one measuring the tension force. These measurements can be used for a controller to control the plane's attitude. A couple of requirements have to be known to design such a device. These requirements mainly come from the ventilator, which is not to be designed in this assignment, and the plane. The overall design could be broken up into six main parts: ventilator, plane, angle sensor, tension sensor, model and controller. Because of the lack of time, the model was not completed and not fully validated and the controller could not be designed. That is why the latter is not included in this report. A picture of the angle and tension sensor attached to the ventilator can be seen in Appendix C Figure 38.

3.1 Ventilator

The ventilator being used could go up to a wind speed of 23 [m/s] by using 19 ventilator modules consisting of a drone motor, electronics speed controller (ESC), five-bladed propeller and a laminator. The latter is needed to laminate the distorted airflow coming from the spinning propellers. If this is not done, the disrupted airflow could cause unintentional behaviour on the test setup. The entire ventilator was designed for the size of the chosen plane[1]. A picture of the ventilator is included in Appendix C Figure 34. However, it was not fully functional during this assignment.

3.2 Plane

3.2.1 Plane selection

For deciding on a plane for the test setup, the following aspects had to be considered: (1) low enough stall speed to fly using the ventilator, (2) the plane size should be small to keep the test setup up small while keeping enough space to house all the electronics, (3) amount of control surfaces has to be at least four to be able to control pitch, roll and yaw and (4) a conventional plane tail makes modelling of controlling pitch, roll, yaw and position easier. Several ideas for planes were considered from designing one to buying one, but eventually, a 3D model for a RC plane was chosen. There are several advantages of choosing a 3D printed plane over the other options such as (1) 3D printed parts are easily replaceable, (2) it has a hollow fuselage which can be used to house electronics, (3) easily modifiable and (4) cheap to produce. The chosen design was the "Micro SportCam" from 3DAeroventures.com [2] and can be seen in Appendix C Figure 35. This design

[1]Thanks to technicians at the Robotics and Mechatronics department who have designed and built this ventilator

has four controllable surfaces, a conventional plane tail and a slight dihedral angle which results in more roll stability as explained in section 2.1.2. The stall speed was not mentioned in the design. However, it is mentioned that this plane is meant for flying at low speeds and looking at the video provided it is supposed that it will fly with the ventilator. The plane was printed out of PLA and assembled as described in the assembly guide of the plane. Because the recommended servo motors did not arrive in time, different servo motors had to be chosen. The FEETECH Sub-Micro Servo FS0403 were chosen. These motors had different dimensions which led to some modifications in the servo brackets. These servo motors have dimensions of 20 x 8.3 x 19.3 [mm] compared to the recommended servo dimensions of 18.02 x 7.91 x 16.8 [mm] for which the plane was designed. Next to those changes, the plane did not get landing gear as instructed in the building guide since this will not be used. Instead, a small attachment point is glued onto the front to attach a cable. Also, a small hole is drilled in the nose so that the wires for communication and power can come through as shown in Appendix C Figure 36.

3.2.2 Electronics

In the build guide, some electronics were recommended but only the servos were acquired, this is because the plane will be connected to the angle sensor by a cable. A few lacquered copper wires from an ESP32Wroom could be twisted around the cable to be able to communicate with the plane and provide power to the electronics on board. By doing this, the plane does not need a receiver or batteries, this saves weight and also eliminates the problem of a battery running low. To make the controller design easier, a flight controller is used to keep the roll axis of the plane stable. This flight controller is the Speedybee F405 wing mini, this one was chosen because of its small size and application for fixed-wing aircraft. As mentioned before, the recommended EMAX ES9251ii (2.5g) Digital Servo did not arrive and that is why the FEETECH Sub-Micro Servos FS0403 were chosen. These servos were chosen because of their similar dimensions to the original ones. The four servos were connected to the flight controller. To communicate with the flight controller, two lacquered wires were connected to the SBUS port of the flight controller and connected to a UART port of the ESP32Wroom. An ESP32Wroom was chosen because of its many different types of ports. To power the flight controller, the other two lacquered wires were connected to the power input. These are connected to a voltage source of 7.5 [V]. Connectors were made for the SBUS input and the power input to be able to easily swap the lacquered wires running along the main cable. Because the lacquered wires do not have an isolation layer it could introduce crosstalk. When the crosstalk becomes a problem, the lacquered wires could easily be swapped out for isolated wires. However, this comes with the cost of adding weight to the cable.

3.2.3 SBUS communication and flight controller software

The fast SBUS protocol is used to communicate with the flight controller. To do this with an ESP32Wroom, the GPIO16 (RX) pin is used to send data, which corresponds to UART2. UART1 is not used because of its direct link with the serial port of the USB output. The 'bolderflight SBUS' library is used [3]. The settings for this are: inverted serial and fast SBUS turned on. Values sent are in the range of [172 1811] which corresponds to [987 2011] in the INAV flight controller software, this range is used because it is common. INAV is being used because of its many different useful settings and familiarity of one of the staff members. To connect the proper SBUS channel with its corresponding INAV channel, the Table 4 in Appendix B was found and can be used.

In the INAV software, there are options to include stable flight, this can also be done to the roll axis. The parameters of this controller can be changed to ensure the roll angle stays the same. This has to be tuned when testing in an air stream.

3.2.4 Parameters

To model the plane, a few parameters have to be known: the location of the Centre of Mass (CoM), the mass, the lift coefficient and the drag coefficient. The latter two will be found by an experiment described in section 4.6.1. The mass of the plane including electronics was measured at 185.57 [g]. To find the CoM of the plane, it was balanced along its roll axis, its pitch axis and lastly, the plane was hung from its attachment point and an imaginary line could be drawn from the attachment point straight down which coincides with the other balancing points. This point was the CoM and its location can be seen in Figure 3. The wing area was specified in the build guide as 0.0764 m^2.

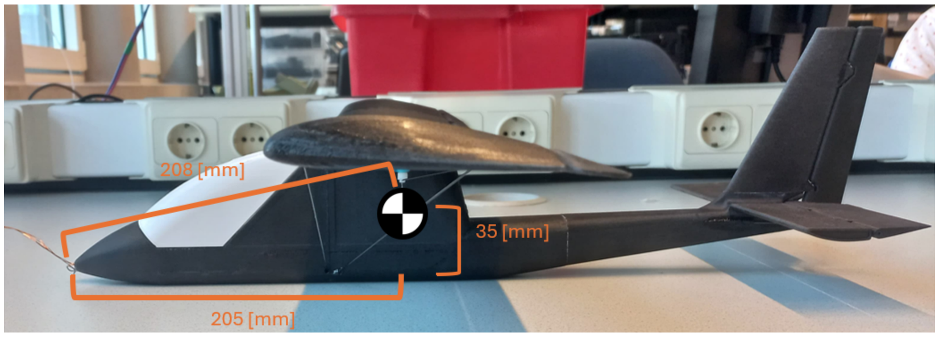


Figure 3: Location of centre of mass of plane (note: the vertical measurement is taken from the bottom of the plane)

3.3 Angle sensor

3.3.1 Joystick selection

A joystick is used to measure the angle of the cable. When choosing a joystick there was little to no data in the datasheet about the accuracy. Initially, the idea was there to test two joysticks and use the best one, but because of time shortage, a joystick was chosen. The joysticks considered were the "PS2 joystick module" and the "FrSky M7 gimbal". The latter made use of hall-effect sensors and bearings which made the joystick have very low friction. The difference in friction between the joysticks could easily be felt. That is why, the "FrSky M7 gimbal" was chosen. However, there were some concerns with the effect of the motors on the hall-effect sensors. To check if the motors affected the joystick, a small test was set up by keeping the joystick as close as it would eventually be mounted on the setup and the motor was spun. When moving the joystick around, it gave the same values as without the motor being turned on. This was only tested with one motor because of the ventilator setup not being ready at that time.

3.3.2 Housing

A housing was made to mount the joystick onto the ventilator setup. Three inserts were designed to exactly fit into specific holes on the ventilator. A screw can be used to expand the insert and secure the housing in place. Hexagonal-shaped holes were added to reduce printing time.

3.3.3 Code

To read out angles, the joystick output values have to be mapped. This is done with the same ESP32Wroom that controls the plane. A test is conducted to find the values corresponding to the angles and can be found in section 4.1. This resulted in a mapping of the X-axis from [575 3990] to [-300 300] which corresponds to [-30.0 to 30.0][°] and the same goes for Z-axis [485 2910]. In section 3.6 it becomes apparent why these axis labels are chosen.

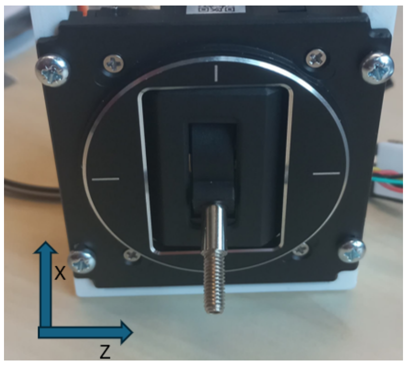


Figure 4: Joystick with arrows in which direction the movement is defined

3.4 Cable

For the cable, a 0.8 [mm] steel cable was used. This was chosen because of its availability during construction. The steel cable itself should also have a low compliance, which is easier to model because it could be neglected.

3.5 Tension sensor

The main idea of the design of the tension sensor is to measure the displacement of a spring and then obtain the force with its spring constant, this is because force cannot be directly physically measured. Two methods of obtaining the displacement were considered; using a potentiometer or using a Time of Flight (ToF) sensor. There was an idea of testing both the components and comparing them, but due to a lack of time, the time-of-flight sensor was chosen. This is because a time-of-flight sensor is small, lightweight, and requires no operating force. It can be mounted near the attachment point of the cable to the angle sensor, resulting in a shorter leverage arm and, consequently, less impact on the plane. The VL6180 was chosen because of its accuracy compared to other time-of-flight sensors available on a break-out board. The latter was preferred because of the ease of use during prototyping. This sensor has a resolution of approximately 1 [mm] when used for the range [5 200] [mm]
To choose a spring for the device, the force on the device has to be estimated. When looking through different airfoils on airfoiltools.com and looking at the airfoil that was most similar to the one in the plane design, it is possible to have an approximation of the minimal force and the maximal force. The one which looked the most similar was the NACA 23015. By looking at the coefficient of drag (C_d) at low Reynolds numbers (50000), it can be seen that the lowest C_d is around 0.25. Then taking an arbitrary low air speed of 10 m/s, the wing area of the plane of 0.0764 [m^2], air density of 1.204 [kg/m^3] (at 20 [°C]) [4] and using the formula for drag from Equation 1 the force of 0.114 [N] is calculated. In the same way, the maximum drag force is calculated using a wind speed of 25 m/s and a $C_d = 0.09$ which results in 2.587 [N]. To make sure the device will not run out of range, it will be designed for 3 [N]. A spring with a spring constant of 0.13 [N/mm] was chosen. This had the lowest spring constant of all available springs in the workshop. A low spring constant is desirable since this results in bigger displacements for the same amount of force which results in more accurate readings of the ToF sensor. This means that using this spring and ToF sensor, the device has a resolution of 0.13 [N].
To measure 3 [N] the device has to let the spring have a displacement of 23 [mm].
In the following section, the different versions of the design of the tension sensor are described.

3.5.1 Version 1

The first version was made of two main parts. One part slid into the other, this made sure that the axes would stay aligned. The spring was mounted using nylon M3 bolts and nuts as can be seen in Figure 5. The tension sensor could easily be mounted onto the angle sensor by using an M4 nut to bolt down onto the angle sensor.
The ToF sensor is mounted on the bottom side with some nylon M3 bolts and nuts and can read the distance to a plate opposite of it. This plate is designed to slide in and out as part of the overall assembly. When mounting the spring, nylon M2 bolts and nuts were initially considered. However, they proved too weak to withstand the spring's strength, leading to bending. So nylon M3 bolts and nuts were used
The design also accounted for the spring's mounting points, ensuring they were 18 mm apart. This spacing allows the spring to be mounted without any tension. The maximum distance is the maximum extension + resting distance: 23 [mm] + 18 [mm] = 41 [mm].
To mount the cable that goes to the plane, a small insert is made to route the cable and then clamp it down with another 3D-printed plate.
When the whole design was printed, it became clear that this design did not work. This was because it did not have the spring in line with the cable which caused a moment. This created more friction than desired in the sliding part of the design.

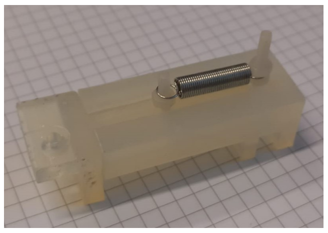


Figure 5: Version 1 tension sensor

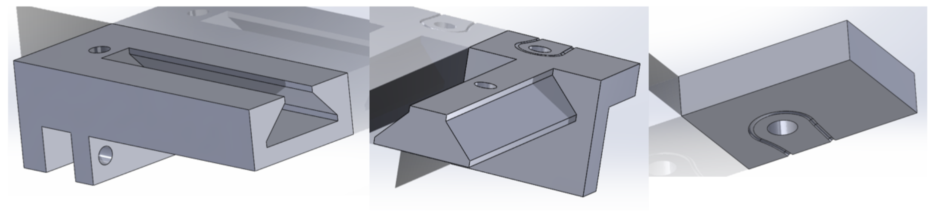


Figure 6: 3D design of version 1 of the tension sensor (left: the main part where ToF sensor mounts on the bottom side, middle: the second main part which slides into the other main part and right: part to clamp cable down)

3.5.2 Version 2

Then a second version was designed with the idea of having little friction and the spring in line with the angle sensor and the cable. The design consisted of two main parts as can be seen in Figure 7 and Figure 9. The part fixed to the angle sensor (Figure 7) has two holes at the top where the ToF sensor could be bolted down. Below those holes, there is a hole where the angle sensor could be attached using a M4 nut. Then there are two more holes which are opposing each other,

these are used to fix one end of the spring in place using an M3 bolt and nuts.
The sliding part (Figure 8) has two cylinders with holes, these holes are used to attach the other part of the spring using an M3 bolt and nuts. Then on the other side of this part, there is a space for the cable to be attached by putting an M2 bolt through the sliding part and a loop in the cable. The plate that is sticking out is used to reflect the light coming from the ToF sensor. Later during testing, it was discovered that the area of this part was too small and gave non-desirable outcomes. The reflecting plate was changed for a bigger one which can be seen in Figure 9.
The reader can also see that two small pieces are sticking up from each of the cylinders, those are there to restrict sideways movement. However, when this part was printed it did not turn out well because of the printer not being able to print such small details.
This version had an average error rate of 0.068 [N] and the test results can be found in Appendix H

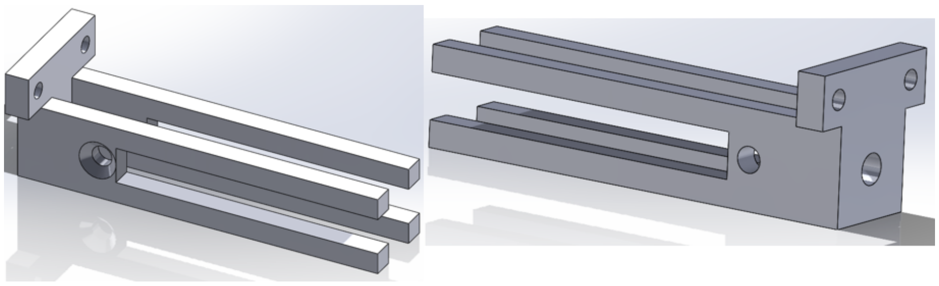


Figure 7: 3D design of version 2 of the tension sensor.

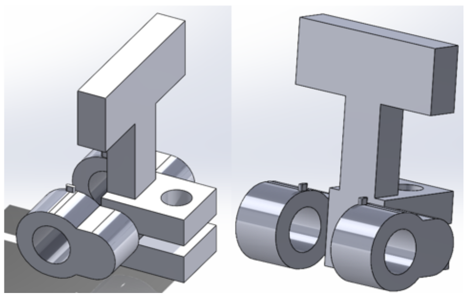


Figure 8: 3D design of the sliding part of version 2 of the tension sensor.

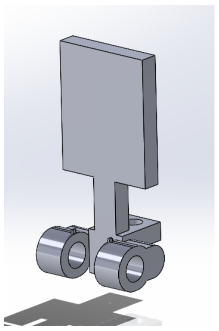


Figure 9: 3D design of the second version of the sliding part of version 2 of the tension sensor.

3.5.3 Version 3

Because version 2 did not constrain the sliding part enough, it resulted that the sliding part could rotate. This could lead to inaccuracies when measuring the displacement. In version 3, the design was altered to constrain this rotational movement. Only the sliding part was adapted and can be seen in Figure 10. Also, reflective tape was added to help the ToF sensor and the fixed part was printed in black so that less light would distort the ToF sensor. The result can be seen in Figure 10 and Figure 11. The design had a mass of 10.55 [g]. This version had an average error of 0.89 [N].

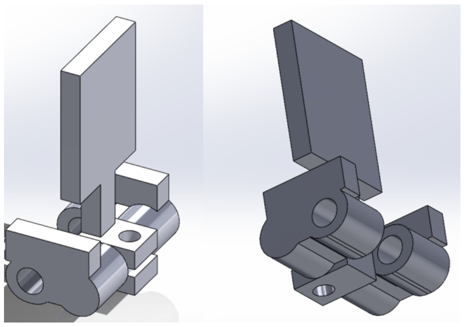


Figure 10: 3D design of the sliding part of version 3 of the tension sensor.

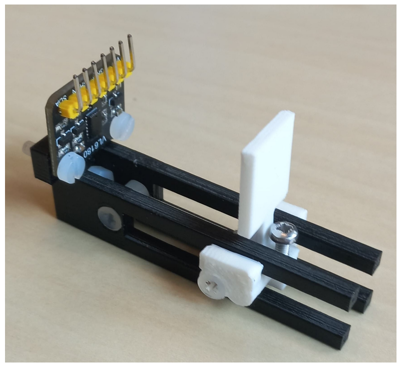


Figure 11: Realised design of version 3

3.5.4 Code

The code for measuring force is provided in Appendix G. This code includes calibration for the distance measurements. Initially, it collects 2000 samples and averages to determine the initial distance measured by the ToF sensor. 2000 samples were chosen because for less than this, the calibration was not working well enough. This measured distance is then subtracted from the reference distance used in the mapping formula to ensure accurate mapping. In different light settings, the initial distance changes, and that is why the calibration is needed. The mapping formula from section 4.2 was used to obtain the actual distance.
During testing, it was observed that the spring constant was incorrect. The correct spring constant was determined by measuring the force and displacement of the spring using a force gauge and a caliper. The recalculated spring constant is 0.171 [N/mm].
The displacement is obtained by subtracting the starting distance, this displacement is multiplied with the spring constant to obtain the force measured.
The code for the tension and angle sensor is combined into one code and can be seen in Appendix I. The running average could be removed if needed. The total run time for obtaining the values for both angles of the angle sensor and force computation is done in 11 [ms].

3.6 Model

3.6.1 Assumptions and approach of modelling

Before making a model, a few assumptions were made about the system:

1. The cable is always tense.
2. The plane itself is a rigid body, neglecting flexion in the body and wings.
3. The lift force is always perpendicular to the airflow and the drag is anti-parallel to the airflow.
4. The airflow applied to the system is a constant laminar flow.
5. The roll axis is held stable by the flight controller and can be ignored in the model.

The bodies being modelled are: (1) The cable and (2) the plane. The frames needed on each of the bodies are as follows:

- Frames for cable: cable-left-end frame, cable-CoM frame and cable-right-end frame.
- Frames for plane: plane-nose frame, plane-CoM frame.

Each of these frames is fixed to the corresponding body and can be seen in Figure 12. These frames can be translated to the inertial frame. The inertial frame itself is subdivided into two domains: the rotational domain and the translational domain. In the rotational domain, all rotational velocities and moments are modelled and in the translational domain, all translational velocities and forces are modelled.
All of the modelling is down in 20-sim.

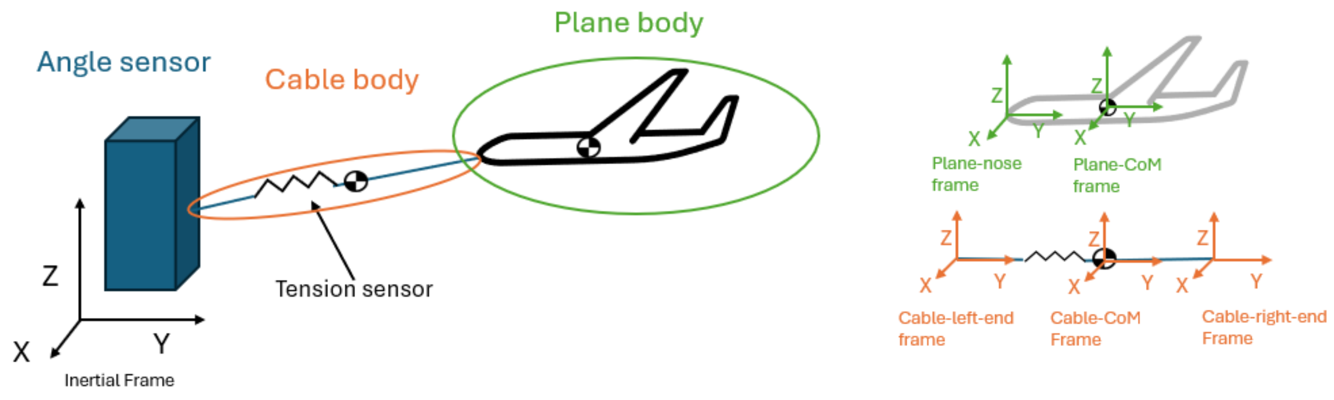


Figure 12: Modelling approach and frames on bodies depicted

3.6.2 Rotational Transformer

Throughout the model, the Rotational Transformer (RTF) element is used, this is a modulated transformer including rotation matrices. Because this model does not consider roll, which is rotation around the Y-axis in the model, only the rotation matrices for the X and Z axis are considered and can be seen below. The code for the RTF element can be seen in Appendix D Figure 39.

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x \\ 0 & sin\theta_x & cos\theta_x \end{bmatrix}, \quad R_z(\theta_z) = \begin{bmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} [5]$$

3.6.3 Cable model

First, a gravity model is added and can be seen in Figure 14 and at the bottom of Figure 13. This model acts in the inertial frame as an effort source, for the gravitational force, and an I-element, for the mass of the cable, at the CoM of the cable.
To translate from the translational domain in the inertial frame at the CoM to the cable-CoM frame, an RTF is used and goes to a 1-junction which is the 1-junction between the left and the right cable end models, this 1-junction represents the origin of the cable-CoM frame.

$$I_{mass\_cable} = \begin{bmatrix} mass\ of\ cable & 0 & 0 \\ 0 & mass\ of\ cable & 0 \\ 0 & 0 & mass\ of\ cable \end{bmatrix}, Se_{gravity} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -g{*}mass\ of\ cable \end{bmatrix}$$

Where 'mass of cable' is the mass of the whole cable body including the tension sensor. And where g=9.81 $[m/s^2]$.

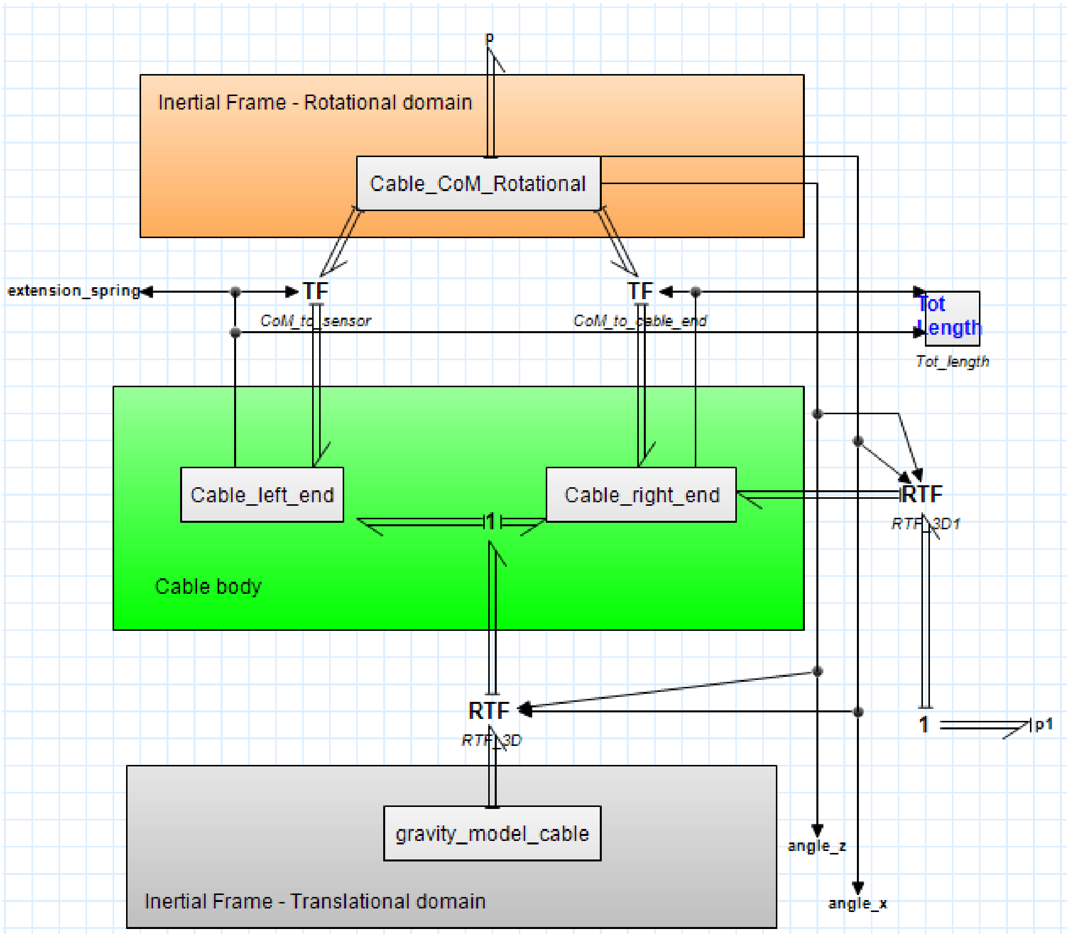


Figure 13: Cable model in 20-sim

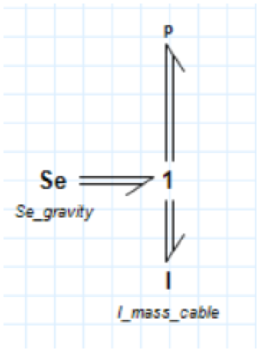


Figure 14: Cable gravity model in 20-sim

To model the rotational velocities acting at the CoM of the cable in the inertial frame, the model in Figure 15 is added and is called 'Cable_CoM_Rotational'. This model only has the rotational

velocities around the X-axis and the Z-axis of the inertial frame. The Y-axis is ignored since this is held stable by the plane's flight controller when connected to the plane.
This model provides the angle input for the RTFs by integrating the rotational velocities. The moment of inertia of the whole cable body and the friction of the angle sensor are modelled in this model. The latter can be added because the friction in the angle sensor only acts on the difference in rotational velocities of the cable with respect to the inertial frame.
The 'p2'-port goes out of the cable model to eventually be used in taking the difference between the plane's rotational velocity and the cable's rotational velocity to model the joint friction between them which will be seen later. The values for $I_{moment\_inertia\_cable}$ are found in section 4.4.

$$I_{moment\_inertia\_cable} = \begin{bmatrix} moment\ of\ inertia\ X & 0 \\ 0 & moment\ of\ inertia\ Z \end{bmatrix}, R_{friction\_fixed\_joint} = \begin{bmatrix} friction\ X & 0 \\ 0 & friction\ Z \end{bmatrix}$$

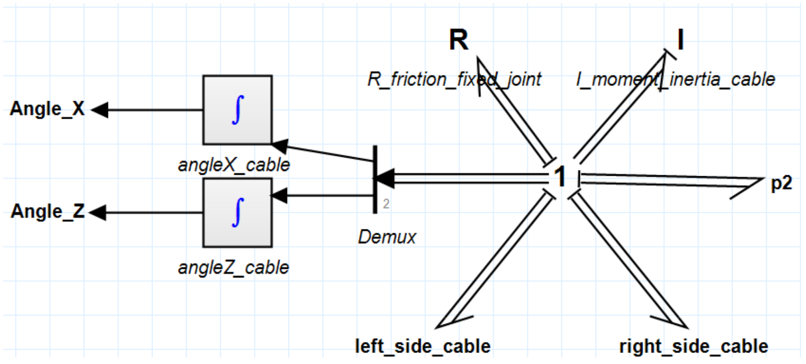


Figure 15: Rotational velocities of cable CoM in inertial frame

In Figure 12 can be seen that the tension sensor is between the cable-left-end frame and the cable-CoM frame. Meaning that there is a velocity difference between the frames because of the spring inside the tension sensor.
To model this, the structure in Figure 16 is used. The spring and the friction of the tension sensor act on this velocity difference and that is why a C-element and a R-element are connected to the 1-junction representing the velocity difference as can be seen in Figure 17.
When subtracting this difference from the velocity at the CoM in the cable-CoM frame, the velocity in the cable-left-end frame due to translation in the inertial frame is found. The velocities due to the rotation of the cable in the inertial frame have to be added to complete the velocities at the origin of the cable-left-end frame. First, the rotational velocities in the CoM in the inertial frame are transformed into velocities in the cable-left-end frame and then added to the cable-left-end velocities due to translation in the inertial frame. The "V_left_cable_end" represents the velocities in the cable-left-end frame at the origin. These velocities have to be set to zero to fix the left end of the cable as in the actual setup. The transformer that translates the rotational velocities on the

cable in the inertial frame to velocities at the cable-left-end frame is dependent on the extension of the spring in the tension sensor, since the length of the body changes.

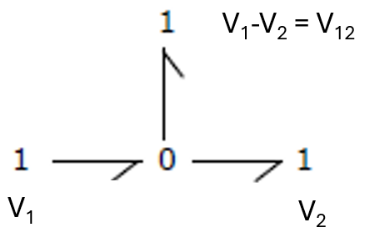
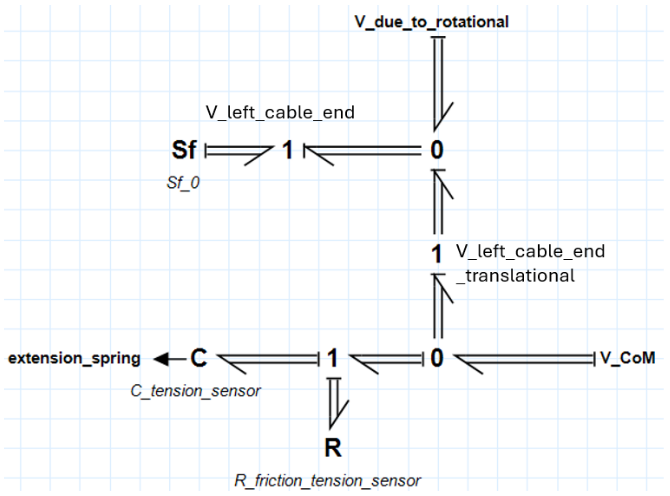


Figure 16: Structure to take the difference in effort or flow from "Introduction to Physical Systems Modelling with Bond Graphs"[6]



$$C_{tension\_sensor} = \begin{bmatrix} Compliance\ X & 0 & 0 \\ 0 & Compliance\ Y & 0 \\ 0 & 0 & Compliance\ Z \end{bmatrix}$$

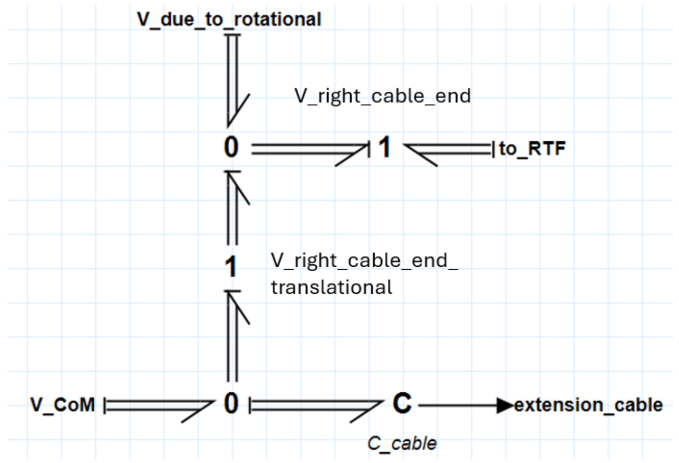
$$R_{friction\_tension\_sensor} = \begin{bmatrix} Friction\ X & 0 & 0 \\ 0 & Friction\ Y & 0 \\ 0 & 0 & Friction\ Z \end{bmatrix}$$

Figure 17: Cable left end model

The compliance Y of the C-element has to be the compliance of the spring used, and the others have to be close to 0 (10^{-11} is used). For the R-element, the tension sensor's friction has to be modelled as friction Y and the others have to be set to close to 0. The reason for modelling it in the Y values is because of how the tension sensor moves with respect to the cable-left-end frame and the cable-CoM frame. Note that to be able to run the model, 0 cannot be filled in an element.

The same model is used but then for the right end of the cable and can be seen in Figure 18. The only difference is that there is no friction acting on the velocity difference between the cable-CoM frame and the cable-right-end frame. The compliance of the cable is modelled, this could be neglected but is included for when the cable gets changed to a more compliant cable. The "V_right_cable_end" represents the velocities in origin of the cable-right-end frame. This 1-junction goes to an RTF which translates the cable-right-end frame velocities to the inertial frame in the translational domain

at the right end of the cable. These velocities are later used to relate the right end of the cable and the plane's nose.
The TF used to translate the rotational velocities on the cable in the inertial frame to velocities at the cable-right-end frame depends on the extension of the cable. Also note that there is a block called "Tot Length" in Figure 13 which represents the total length of the cable which is used in the 3D simulation window as a reference.



$$C_{cable} = \begin{bmatrix} Compliance\ X & 0 & 0 \\ 0 & Compliance\ Y & 0 \\ 0 & 0 & Compliance\ Z \end{bmatrix}$$

Figure 18: Cable right end

3.6.4 Plane model

The plane is modelled in a similar way as the cable and can be seen in Figure 19. The gravity model works exactly as the one in Figure 14, however, 'mass of cable' is changed to the value corresponding to the mass of the plane.
A noticeable difference is that there are more forces modelled in the inertial frame of this model. These are the moments and forces that result from the plane being placed in an airflow. Modulated effort sources are used which are dependent on the windspeed, elevator input, rudder input, angle of attack and yaw angle and can be seen in Figure 21 and Figure 22. The moments acting on the plane are modelled in the rotational domain and the force acting on the plane are modelled in the translational domain. The formula inside of the modulated effort sources has to be determined empirically and is described in section 4.6.1. The reason for the empirical modelling approach is because of the lack of information on the plane. To get data on the forces and moments acting on the plane, it has to be tested. The inspiration of doing it in this way came from the theory about the aerodynamic centre in section 2.1.1
The model for the rotational velocities at the CoM of the plane in the inertial frame can be seen in Figure 20. The 'p1' port is there to be able to take the difference between the rotational velocities of the cable body and the plane body in the inertial frame to model the joint friction between them

and will be shown later. The ‘moments’ port comes from the ’rotational_plane_forces’ model and is shown in Figure 21.
The location of the velocities at the CoM of the plane in the inertial frame for the translational domain can be seen as the highlighted ‘CoM’ 1-junction in Figure 19. This gets translated to the plane-CoM frame by an RTF. When adding up the velocities due to translation and rotation in the inertial frame on the plane nose, the velocities at the plane-nose frame are obtained, which is highlighted as the ‘plane nose’ 1-junction. This 1-junction gets translated to the inertial frame through an RTF. This will later be used to relate the plane’s nose to the right end of the cable.

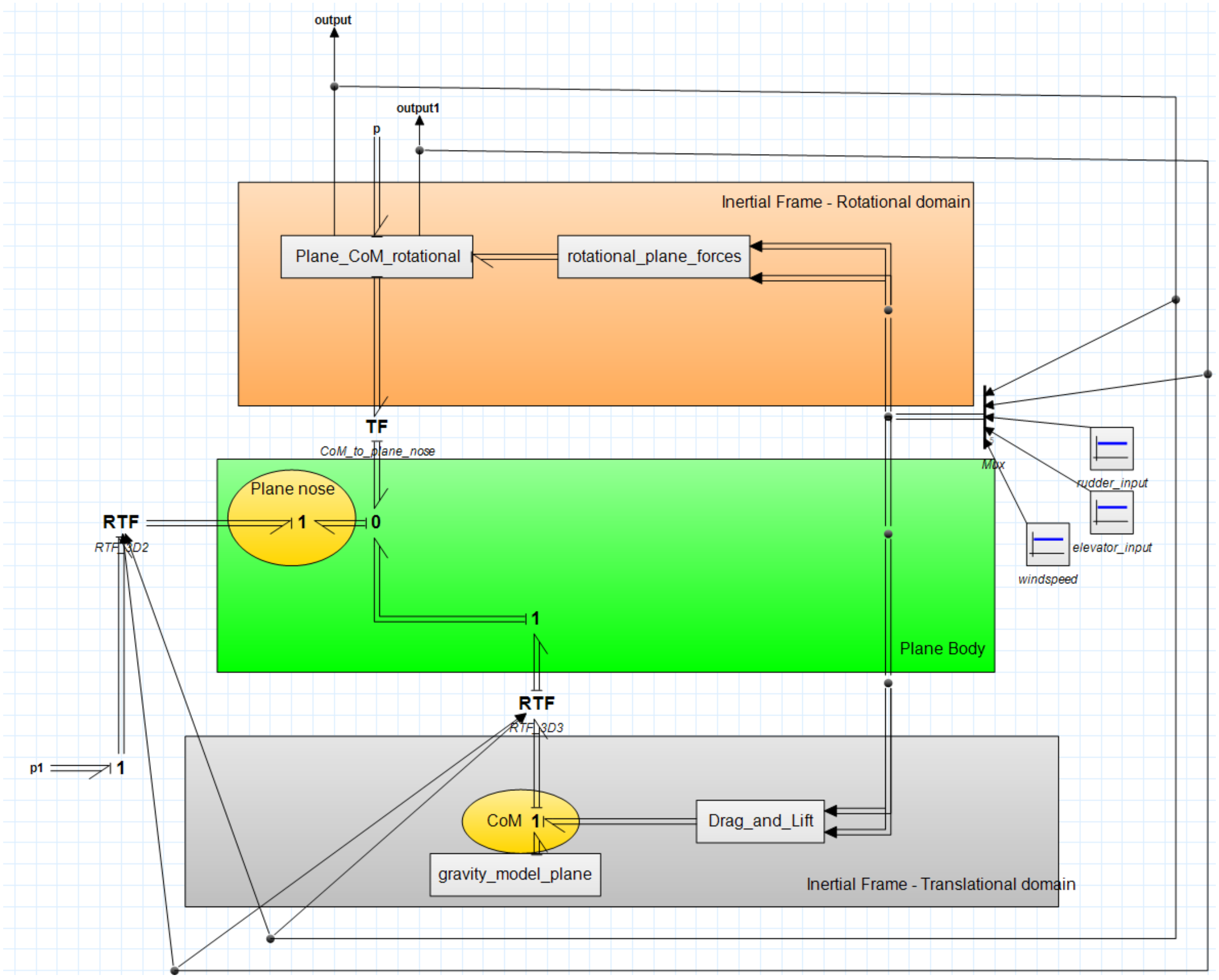


Figure 19: Plane model

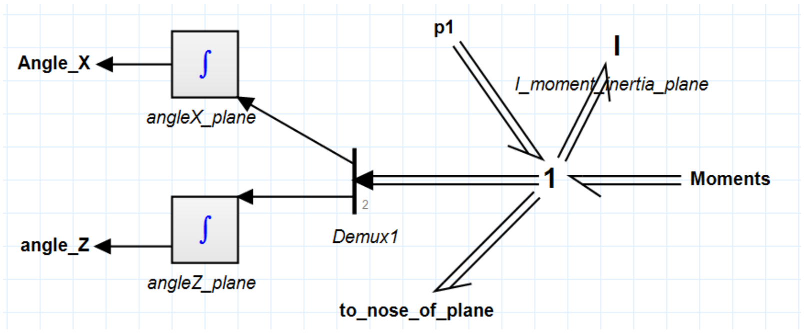


Figure 20: Rotational velocities of plane CoM in inertial frame

$$I_{moment\_inertia\_plane} = \begin{bmatrix} moment\ of\ inertia\ X & 0 \\ 0 & moment\ of\ inertia\ Z \end{bmatrix}$$

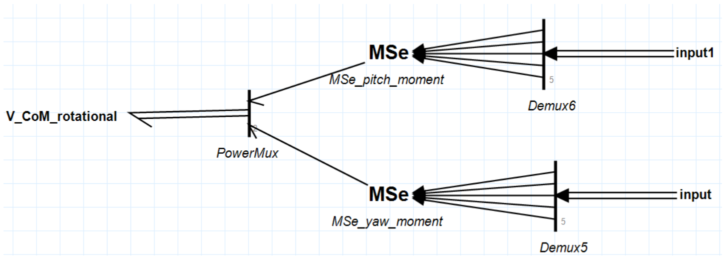


Figure 21: Plane moment model

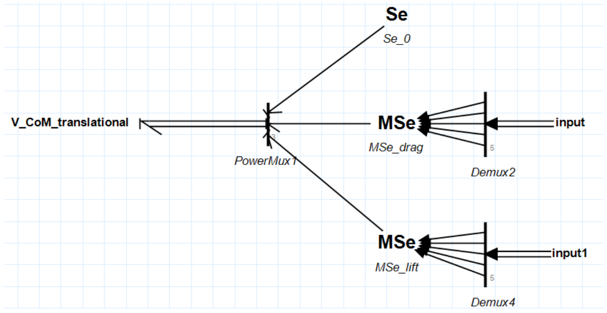


Figure 22: Plane force model

3.6.5 Overall model

In the overall model, the interactions between the cable body and plane body are modelled and be seen in Figure 23. The difference in rotational velocities in the inertial frame of both bodies is taken in the top multi-bonds. The joint friction between the plane nose and the right cable end acts on this difference. This is modelled as an R-element. The single bonds are there to get the difference between the angles and are used in the 3D simulation window as references.
The bottom multi-bonds represent the velocities of the cable-right-end frame and plane-nose frame expressed in the inertial frame. In the actual setup, the right cable end and the plane nose are linked together, this means that there is no translational velocity difference in inertial between these points. To model this, the difference is taken with the 0-junction and the flow source of 0 velocities is added, which results in the difference in translational velocities in inertial between the two frames being 0.
Values of the model are mapped by gain blocks to realistic sensor values, which can be seen in the top left. Later this could be used to create a controller for the system.

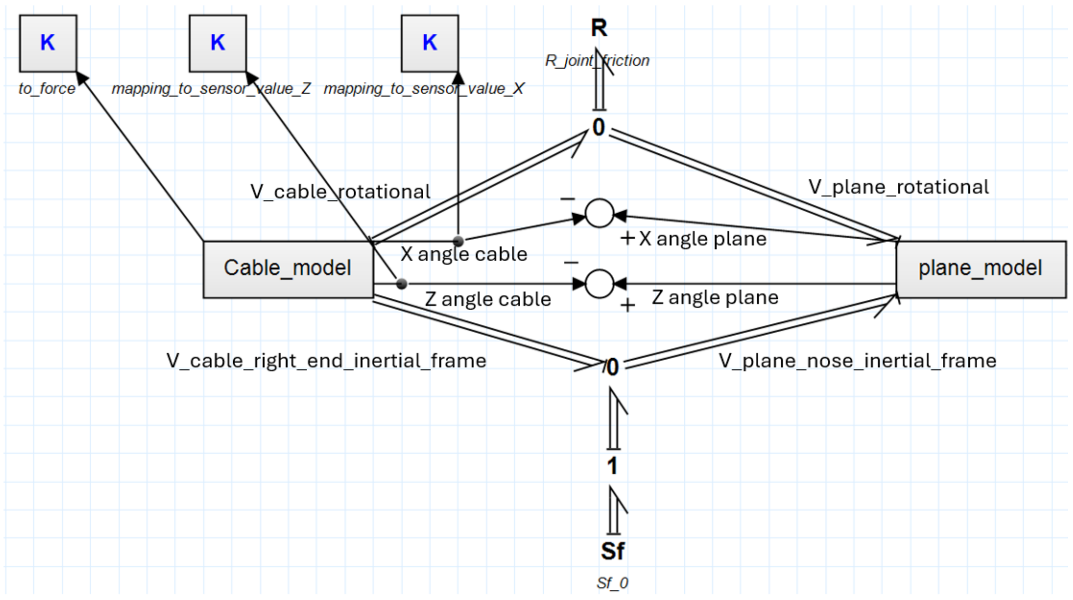


Figure 23: Overall model

4 Experiment procedures and results

4.1 Angle sensor

To obtain the joystick values which corresponds to a certain amount of degrees in an axis, a test has to be conducted.

4.1.1 Method

Materials:

- ESP32Wroom
- M4 bolt
- Triangle ruler
- Joystick
- Angle sensor housing
- Tape
- Wires

The materials were set up as shown in Figure 24. One axis is fixed by tape as shown in the figure, while the other one is tested and vice versa. The axis of rotation of the corresponding axis has to be placed at 0 on the triangle ruler. Only then it is possible to read out the angle that the joystick makes from the triangle ruler. The joystick's base has to be set up 11 [mm] over the ruler when measuring the X-axis to align the rotation point and the 0 on the ruler. The same goes when measuring the Z-axis for the value of 9 [mm]. These values were found by measuring the distance from the surface of the joystick to the rotation axis.
The joystick's wires are connected to an ESP32Wroom as follows: The power wires (red) to a 3.3 [V] pin, ground wires (black) to GND, green wire to GPIO 34 and yellow wire to GPIO 35. Then the code in Appendix E is run and values are read out and noted down for the angles [-30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30][°].

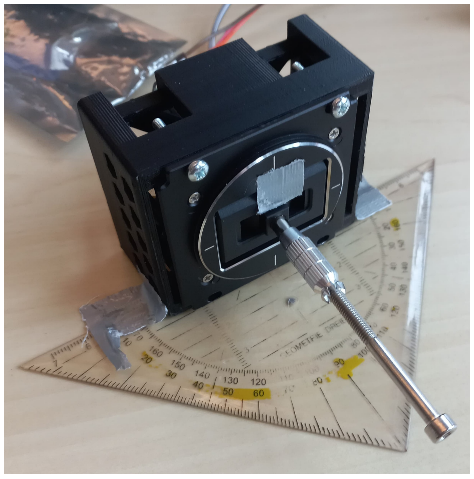


Figure 24: Test setup for testing values and their corresponding angles

4.1.2 Results

Angle [°]	-30	-25	-20	-15	-10	-5	0	5	10	15	20	25	30
X-axis Value	687	929	1150	1360	1635	1980	2287	2470	2790	3070	3405	3849	4060
Z-axis Value	443	680	870	1195	1283	1500	1687	1885	2135	2265	2500	2720	2905

Table 1: Results of angle sensor testing

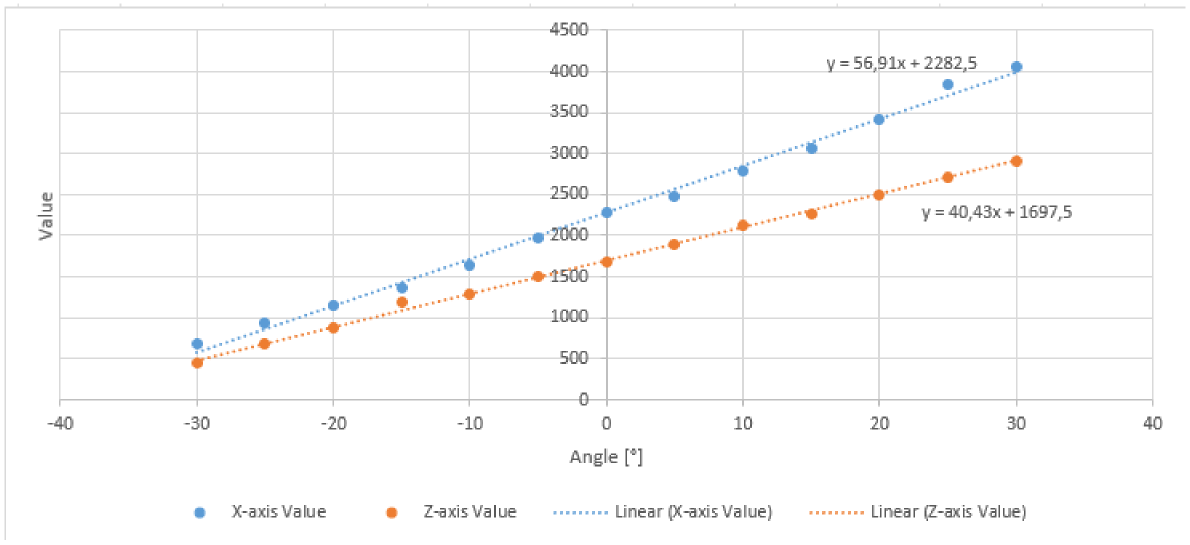


Figure 25: Joystick values for both axes to its corresponding angle and linear trendline

When the data points were plotted, a linear relation could be observed. The formulae for the trendlines are:

$$X_{value} = 56.91 * \alpha + 2282.5 \tag{3}$$

$$Z_{value} = 40.43 * \alpha + 1697.5 \tag{4}$$

Where α is the angle that the joystick makes for the corresponding axis. These formulae are used to obtain the mapping for the joystick and can be seen in section 3.3.3.

4.2 Time of Flight sensor testing

Testing has to be done to see if the ToF is giving the correct values in the environment where it is used in the tension sensor. These tests were done with versions 2 and 3 of the tension sensor. The test results for version 2 can be seen in Appendix H and the results for version 3 can be seen in the next section.

4.2.1 Method

Materials:

- ESP32Wroom
- Caliper
- Tension sensor housing
- Wires
- VL6180 ToF sensor

The ToF sensor is mounted on the tension sensor housing and wired to an ESP32Wroom as follows: SDA pin to GPIO 21, SCL pin to GPIO 22, Vin to Vin and GND to GND.
The code in Appendix F is used to read out the values. Note that in the code a rolling average of 100 samples is used to obtain better readings for this test. For the tension sensor's use case, only the difference in starting distance and end distance matters. Meaning that with this test, only the linearity of the sensor has to be tested. This is done by measuring the distance from a reference point to the reflection plate and comparing it with the value that the ToF is giving out. In this experiment, the reference point from where the caliper measures the distance to the reflecting plate is the head of the bolt where the ToF is mounted (see Figure 26).

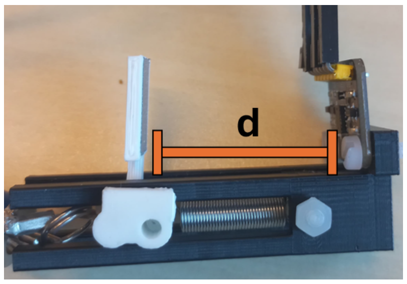


Figure 26: Reference distance d

4.2.2 Results

Distance d [mm]	34	36	38	40	42	44	46	48	50	52
Distance measured ToF [mm]	30,5	32,2	34	35,9	36,9	38,3	40,5	42	43,9	46

Table 2: Results of testing ToF mounted on version 3

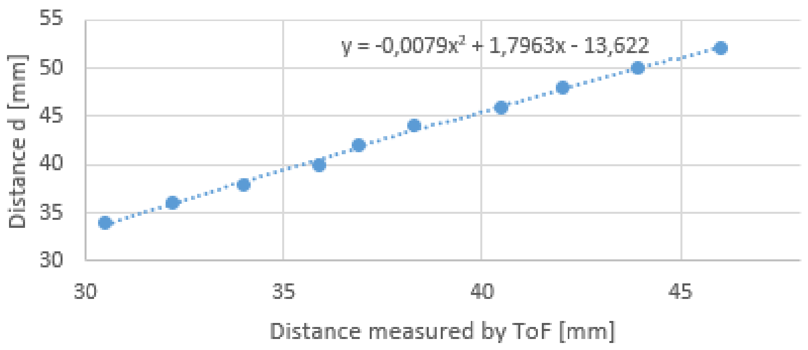


Figure 27: ToF measured distance vs distance d

When trying different trendlines to obtain a trendline which corresponds the most with the data, a second-order polynomial was chosen:

$$d_{actual} = -0.0079 * d_{ToF}^2 + 1.7963 * d_{ToF} - 13.622 \tag{5}$$

This was put into the code to map the values from the ToF sensor to the correct distance.

4.3 Tension sensor force testing

When the calibration code was written and the code for giving the right distance, the tension sensor had to be tested. It was noticed that the spring constant had to be changed to give the right values and was changed to 0.171 [N/mm]. This test is conducted on both version 2 and version 3. The test results for version 2 can be found in Appendix H.

4.3.1 Method

Materials:

- Force gauge of 5[N]
- Tension sensor
- ESP32Wroom
- Wires

A force gauge is attached to the tension sensor. While one part of the tension sensor is held in place, a force is applied to the other side through the force gauge (see Figure 28). The tension sensor's force values are read out using a code that uses a rolling average of 100 samples. This is then compared to the force gauge for the values of [0 0.5 1 1.5 2 2.5 3] [N]. The code for reading the values can be found in Appendix G.

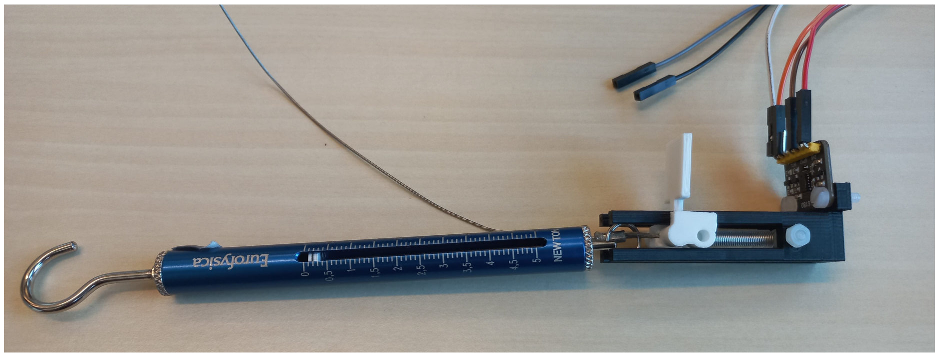


Figure 28: Test setup for testing the tension sensor

4.3.2 Results

Force applied [N]	Output of Tensions sensor [N]			Average error [N]
	Measurement 1	Measurement 2	Measurement 3	
0	-0,07	0	-0,05	0,04
0,5	0,3	0,4	0,35	0,15
1	0,95	0,9	0,95	0,067
1,5	1,55	1,5	1,5	0,017
2	2,11	2	2	0,037
2,5	2,78	2,6	2,6	0,16
3	3,3	3	3,15	0,15

Table 3: Force measured from tension sensor compared to force applied

The total average error is 0.089 [N].

4.4 Plane rotational inertia

A test has to be conducted to obtain the moment of inertia of the plane for both pitch and yaw.

4.4.1 Method

Materials:

- Plane with electronics
- cable to be able to let the plane swing

To obtain the moment of inertia a formula is used.

$$I = \frac{T^2 * m * g * d}{4\pi^2}[7] \tag{6}$$

Where I = moment of inertia [$kg * m^2$], T = period in [s], m = mass of the object [kg], g = 9.81 [m/s^2] and d is the distance from point of rotation to CoM [m]. This formula only holds for small angles from the neutral hanging point. To measure the period, the plane was hung from a cable and swung. The cable itself was fixed as close as possible to the attachment point of the plane so that is would not influence the measurements. The IMU from the flight controller measures acceleration data. This can be plotted in Matlab to see the period. The flight controller itself can only measure every 10 [ms]. The accelerometer data from the Z-axis of the flight controller is used for the pitch and data from the Y-axis for the yaw. These accelerometers corresponds with these axis because of the orientation of the flight controller inside the plane.

4.4.2 Results

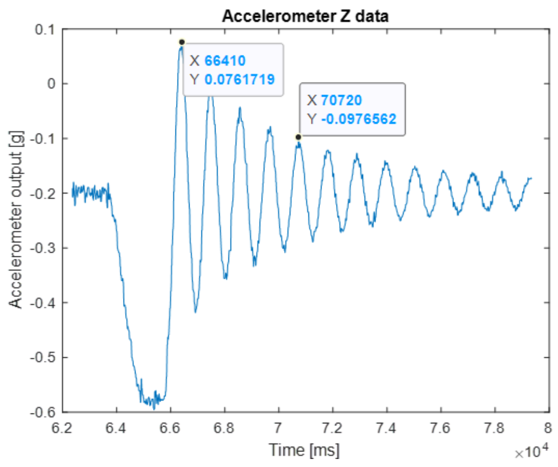


Figure 29: Test results for pitch

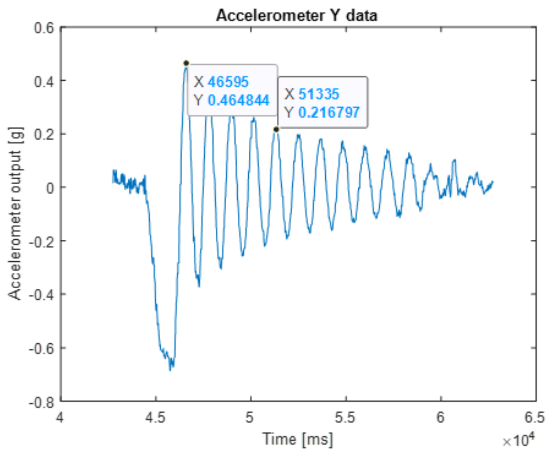


Figure 30: Test results for yaw

When obtaining the period, an average was used over the first 4 periods. Then the rest of the values where filled in to the formula: m_{plane}=0.18557 [kg], d = 0.208 [m], T_{pitch}=1.0775[s] and T_{yaw}=1.1850[s]. This resulted in $I_{pitch} = 0.0111$ $[kg * m^2]$ and $I_{yaw} = 0.0135$ $[kg * m^2]$.

4.5 Model validation

To test the values obtained from the experiment in section 4.4, the values can be put into the model and compared to how the model behaves compared to the actual IMU data of the plane.

4.5.1 Method

In the model, the cable is fixed in its original horizontal position and the compliances of the cable and tension sensor spring are put to values close to 0 (10^{-11})[m/N]. Another method was also tried by setting the velocities of the nose of the plane to 0 but this gave errors in the model. Then the plane was positioned in the model so that it would swing around the correct axis, one simulation with pitch and another simulation with yaw. The amplitudes of the accelerometer and gyroscope data will be smaller, this is because in the model the air resistance of the plane is not being modelled. However, this does not matter because the period is only needed for validation. When the periods do not match up, the moment of inertia can be adjusted till it matches the actual data

4.5.2 Results

In the initial results, the period of the actual data and the one simulated did not match. The first few oscillations of the validation data are the most accurate, because other rotational velocities play a part in the data after this period due to the setup not being able to fully constrain the axis of movement. It can be observed that more frequencies are included in Figure 32 for the model. This is due to the CoM not exactly being aligned with the rotational axis in the model, which caused a non-zero velocity in the pitch axis. To match the period, the moment of inertia was changed to:

$$I_{plane\ rotational} = \begin{bmatrix} 0.0026 & 0 \\ 0 & 0.0029 \end{bmatrix} [kg/m^2]$$

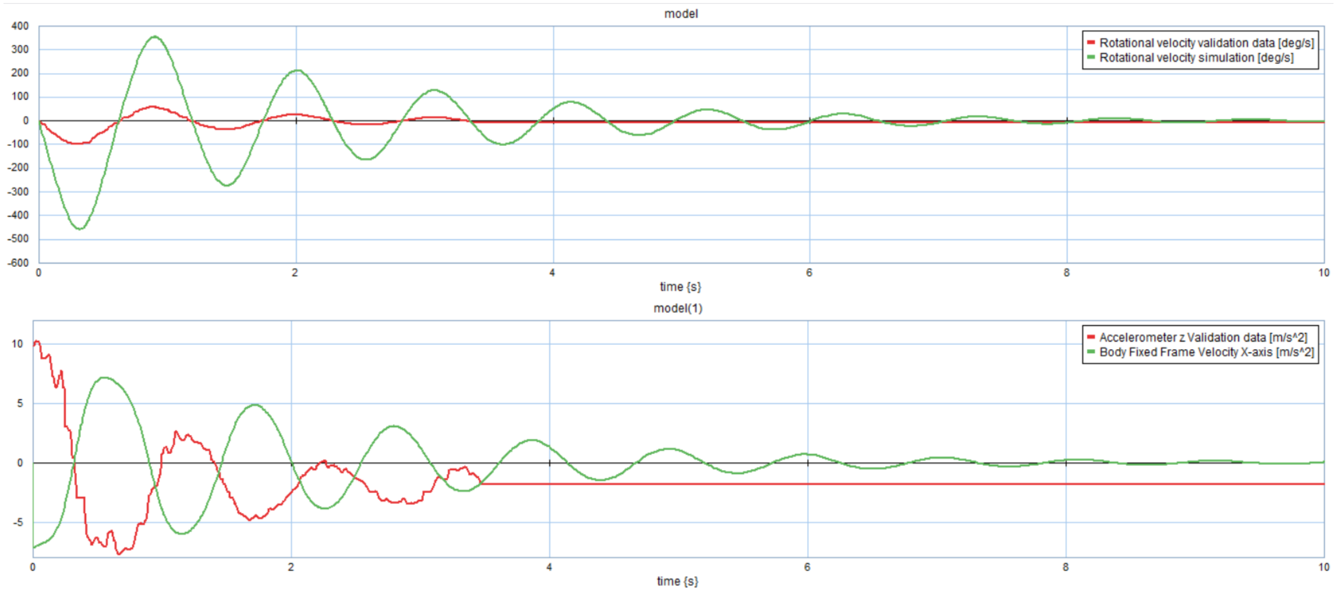


Figure 31: Simulation results for swinging about the pitch axis compared to actual IMU data after adjusting to I_X=0.0026 $[kg/m^2]$

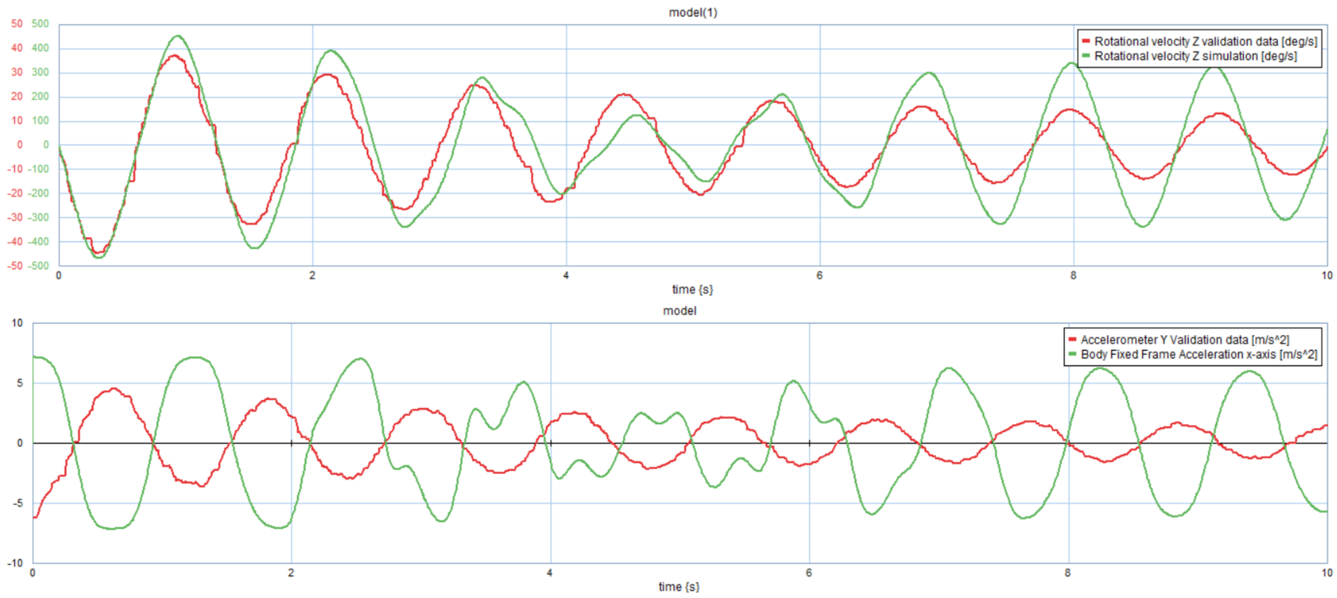


Figure 32: Simulation results for swinging about the yaw axis compared to actual IMU data after adjusting to I_Z=0.0029 $[kg/m^2]$

4.6 Future experiments

4.6.1 Plane parameters

To model the plane, it is necessary to test the force and moments that occur because of an airflow acting on the plane.
Materials:

- Plane
- ATI 40 mini SI-80-4
- Setup to fix the plane to the F/T sensor
- Ventilator (has to be able to make an airflow as big as the plane)

The materials are set up as shown in Figure 33. To ensure most configurations for angle of attack, yaw angle, flap input, and rudder input are accurately modelled, it is necessary to vary these parameters during testing and measure their effects. For every configuration of AoA and yaw angle, the input of flaps and rudder have to be tested. During this test it is assumed that the rudder and flap input are independent, this is done to perform less measurements. Then looking at the AoA that is most relevant from the NACA 23015 it comes down to -10°to 10°. The yaw angle is tested up to 8 °, this is because when exceeding this, the plane will probably drift too fast out of the air stream. The values of the input of the elevator and rudder are chosen in such a way that they resemble full negative to full positive actuation. The following values are tested:

$$\text{AoA} = [-10\ \ -7\ \ -3\ 0\ 3\ 7\ 10][^\circ] \tag{7}$$

$$\text{Yaw angle} = [0\ 2\ 5\ 8][^\circ] \tag{8}$$

$$\text{Elevator input} = [172\ 582\ 992\ 1401\ 1811](\text{from ESP32Wroom}) \tag{9}$$

$$\text{Rudder input} = [172\ 582\ 992\ 1401\ 1811](\text{from ESP32Wroom}) \tag{10}$$

This means that 280 measurements have to be conducted. This could be reduced by making the amount of measurement less but this does come with the trade of having a less accurate model.
To speed things up, a code is written to control the rudder and elevator. This code can be found in Appendix J.
The torques and forces in three different axes are measured with the ATI mini40. To obtain the forces and moments acting on the plane, a transformation matrix is needed. This transformation has to transform the moments acting in the frame of the sensor to the frame of the plane, the same goes for the forces. The easiest way to do this is using the concept of wrench. However, due to time constraints, the concept could not be fully understood to come up with the transformation matrix needed for transforming the moments and forces from the sensor to the CoM.
When conducting the test, the ATI mini40 has to be set to zero when the holder for the plane and the plane is placed on top of the sensor. This way the weight of both will not be taken into account during the measurement.

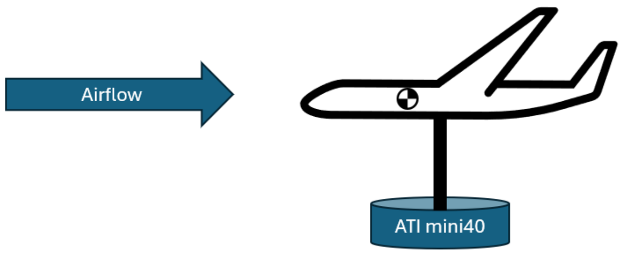


Figure 33: Test setup for obtaining plane parameters

4.6.2 Moment of inertia cable

The moment of inertia of the cable can be obtained in the same manner as described in section 4.4 and validated. Instead of using IMU data, the angle sensor can be used. After the cable length is known for the overall test setup, the mass can be measured and the CoM can be found. Then by hanging the cable and tension sensor upside down from the angle sensor, it is possible to let the cable and tension sensor swing.

4.6.3 Validation of flying plane

When the plane is put in a known constant windspeed, the angles and the tension force from the sensors can be compared to the values that the model gives when the same constant windspeed is put in the model. This way the model can be tuned to correspond more with the actual setup.

4.6.4 Validation of actuators

A known input signal can be given to the rudder input or elevator input, and then the IMU in the flight controller can record data of the plane. The same input signal can be put in the model and the data of the actual setup can be compared. The values could be tuned to make the model more accurate.

5 Evaluation

5.1 Discussion

This assignment was the beginning of making a proof of concept of a test setup. This also means that there were quite a lot of unforeseen events and improvement points.
The angle sensor itself worked quite nicely but is situated in the windstream of the ventilator which could cause a non-laminar flow. This could cause behaviours that are not modelled in the model. The tension sensor had some issues with the reflecting plate being able to rotate, even after the changes made resulting in version 3. This is due to the guiding part not being stiff enough. Recommendations for the angle sensor and tension sensor can be seen in section 5.3. Next to that, the wires used to connect the ToF sensor were too stiff to easily move with the changing angle of the cable when attached to the angle sensor, which will influence the tension measurements. This can easily be improved by using more flexible wires.
The force gauge used during the experiment with the tension sensor made it hard to accurately read off values. This could have caused inaccuracies in the data. The tension sensor could be measured again, but then with a more accurate way of reading off the values. It can also be observed that version 2 is slightly more accurate than version 3, this could be because of the inaccurate reading of the force gauge.
The same goes for the angle sensor. The experiment for obtaining the values for the corresponding angles could be done more accurately since reading the angles from a triangle ruler brings a significant margin of error. Another way to do this would be by attaching a cable or rod to the joystick. Then measuring the distance from the cable to a plane that is parallel to the ground and passes through the middle of the angle sensor, the angles can be determined more accurately using geometry.
The model itself could not be fully completed and validated due to a lack of time as well as the ventilator not being ready in time. This does mean that there is some uncertainty in how accurate the model will be.
At the beginning of the assignment, there was the understanding that the ventilator would be done earlier to do measurements, which changed the planning during the assignment. This resulted in some parts of experiments being prepared but not further executed.

5.2 Conclusion

It could be seen that most of the sub research questions in section 1 have been answered, except for question 4. A 3D printed plane was chosen which has a wing span of 695 [mm] and should be able to fly in a generated windstream using a F405 wing mini flight controller which is controlled by an ESP32Wroom. This ESP32Wroom can later be used as the controller of the whole setup.

The tension force is being measured using a device which measures the displacement in a spring using a time of flight sensor. This device had an average error of 0.089 [N]. There are improvement points in the design, but the concept of measuring the tension force on the cable has been proven. To make the control loop more stable, the sensor's average error has to be brought down more.
For measuring the angle of the cable, the FrSky M7 gimbal was chosen. A housing has been designed to mount the joystick onto the setup. After testing, the device has been calibrated and can read the angle of the cable in two axes.
The design of the control loop was not completed, however a model was made of the test setup. This model is not fully completed but is partly validated by data obtained during testing but has to be fully validated and completed in later research.

5.3 Recommendations

5.3.1 Angle sensor

The angle sensor could be placed more to the sides of the ventilator, this will reduce the amount of disrupted air generated by the sensor being in the middle of the wind stream. The angle sensor itself could be reduced in size to disrupt less airflow.

5.3.2 Tension sensor

The tension sensor could be improved by using a ToF sensor which has a smaller resolution than the current 1 [mm]. Another improvement would be by using a different spring which has a lower spring constant. Both result in a better resolution of the device.
For the mechanism itself, there are also a few improvements to be made. Now the reflecting plate can make small rotations which will cause inaccurate readings of the ToF sensor. This could be mitigated by using a stiffer construction, which will not bend when a force is applied to the reflecting plate. The stiffer construction could exist out of carbon fibre rods, together with a stiff plate which holds the construction in place. Another way of making the construction more stiff is by making a plate on the open side of the tension sensor. This plate could restrict flexion in the guiding part of the design and could solve the problem.
A whole different approach would be by routing the cable through the angle sensor to a mechanism which works in the same way but does not have to be in the airflow itself. This could reduce the amount of inaccurate data being read because of the sensor being in a more controlled environment. However for this idea, a new mechanism has to be made which alters the gimbal used for the angle sensor directly.

5.3.3 Model and validation

In the section 4.6, future experiments are discussed to be able to validate the model.

6 Acknowledgements

I would like to express my gratitude to my supervisor, Stefano Stramigioli, for his guidance and support throughout the course of my assignment. His expertise and insightful feedback have been very helpful to this work.

I am also deeply thankful to my daily supervisors, Sander Smits and Artur Gąsienica, for their continuous support, patience, and assistance. Their dedication and commitment have been instrumental in this project.

A special thanks to the technical staff at RaM, whose assistance and expertise were vital in the construction of the ventilator. Next to that, their support ensured that I had the necessary resources and tools to conduct my experiments effectively.

Thank you all for your contributions and support.

References

[1] J. D. Anderson Jr., *Introduction to Flight.* New York, NY: McGraw-Hill Education, 8th ed., 2016.

[2] 3D AeroVentures, "Microsportcam." https://www.3daeroventures.com/microsportcam. Accessed: June 2024.

[3] B. F. Systems, "Sbus library for arduino." https://github.com/bolderflight/sbus. Accessed: June 2024.

[4] Princeton University, "Volume, density and specific gravity." `https://www.princeton.edu/~maelabs/hpt/mechanics/mecha_36.htm`. Accessed: 2024-06-18.

[5] P. Burzynski, "Mathematics for game developers: 4.6: Rotation matrices in 3-dimensions." `https://math.libretexts.org/Bookshelves/Applied_Mathematics/Mathematics_for_Game_Developers_(Burzynski)/04%3A_Matrices/4.06%3A_Rotation_Matrices_in_3-Dimensions`, 2019. Accessed: 2024-06-26.

[6] J. F. Broenink, "Introduction to physical systems modelling with bond graphs," 2000.

[7] OpenStax, "University physics i - mechanics, sound, oscillations, and waves: 15.5: Pendulums." `https://phys.libretexts.org/Bookshelves/University_Physics/University_Physics_(OpenStax)/Book%3A_University_Physics_I_-_Mechanics_Sound_Oscillations_and_Waves_(OpenStax)/15%3A_Oscillations/15.05%3A_Pendulums`, 2016. Accessed: 2024-06-26.

A AI Declaration

During the preparation of this work the author used ChatGPT based on GPT-3.5 to rewrite parts of the text for the report and to come up with parts of Arduino code that have been used for obtaining measurements during experimenting. During the preparation of this work the author used Grammerly in order to spell- and grammer-check written text. After using these tools/services, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

B INAV channel table

SBUS channel	**Corresponding INAV channel**
Ch0	Roll
Ch1	Pitch
Ch2	Throttle
Ch3	Yaw
Ch4	Ch5
...	...
Ch15	Ch14

Table 4: SBUS channels with their corresponding INAV channels.

C Photos of design

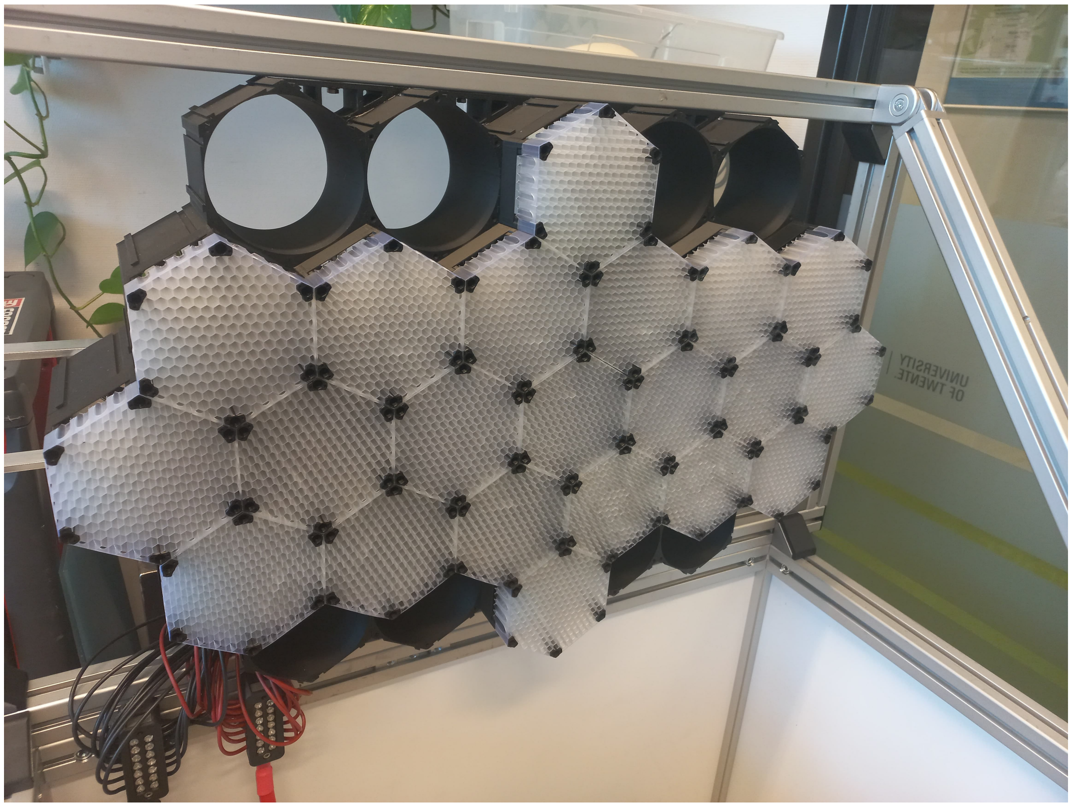


Figure 34: Ventilator used for the test setup

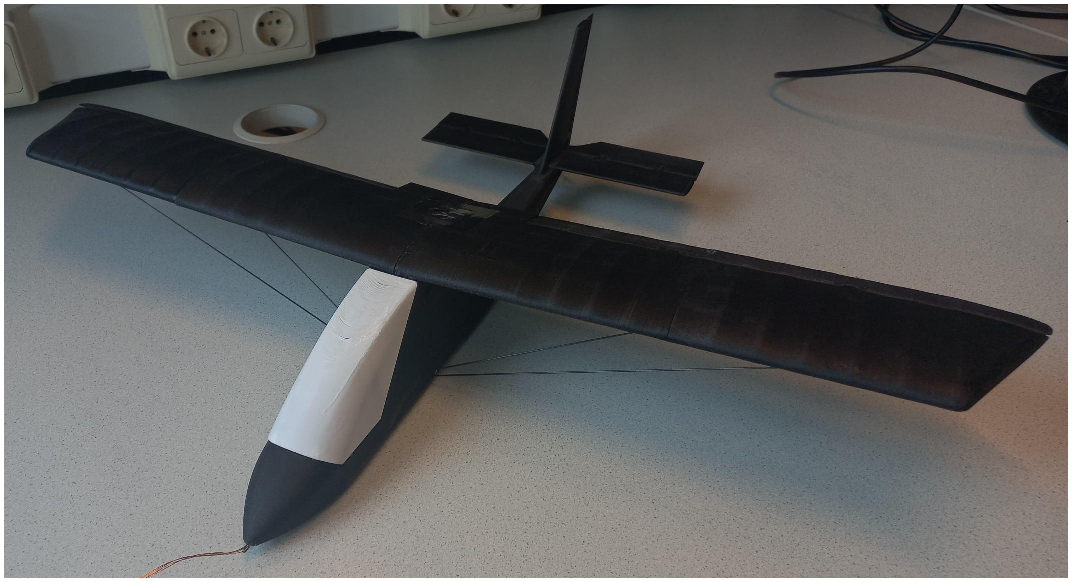


Figure 35: Micro sportcam plane printed in PLA assembled

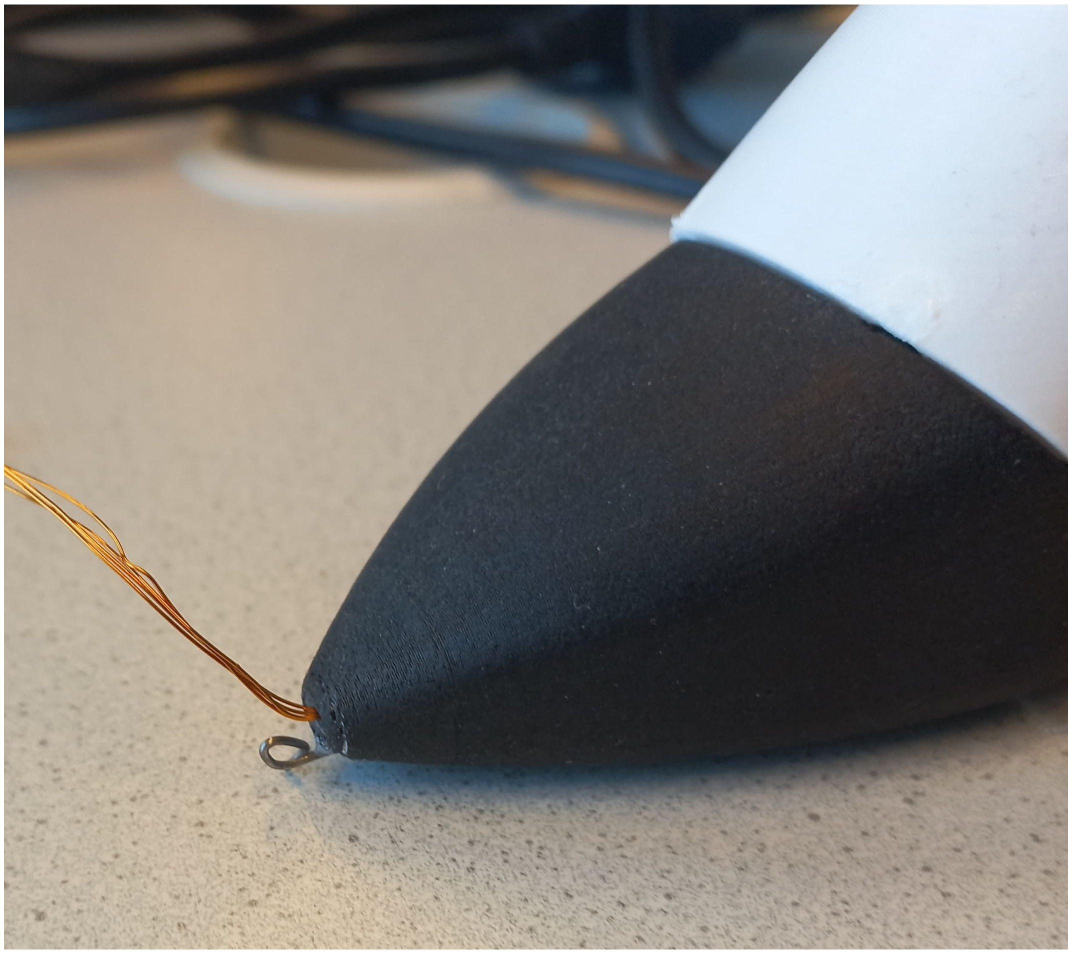


Figure 36: Nose attachment for cable and hole in nose for routing wires.

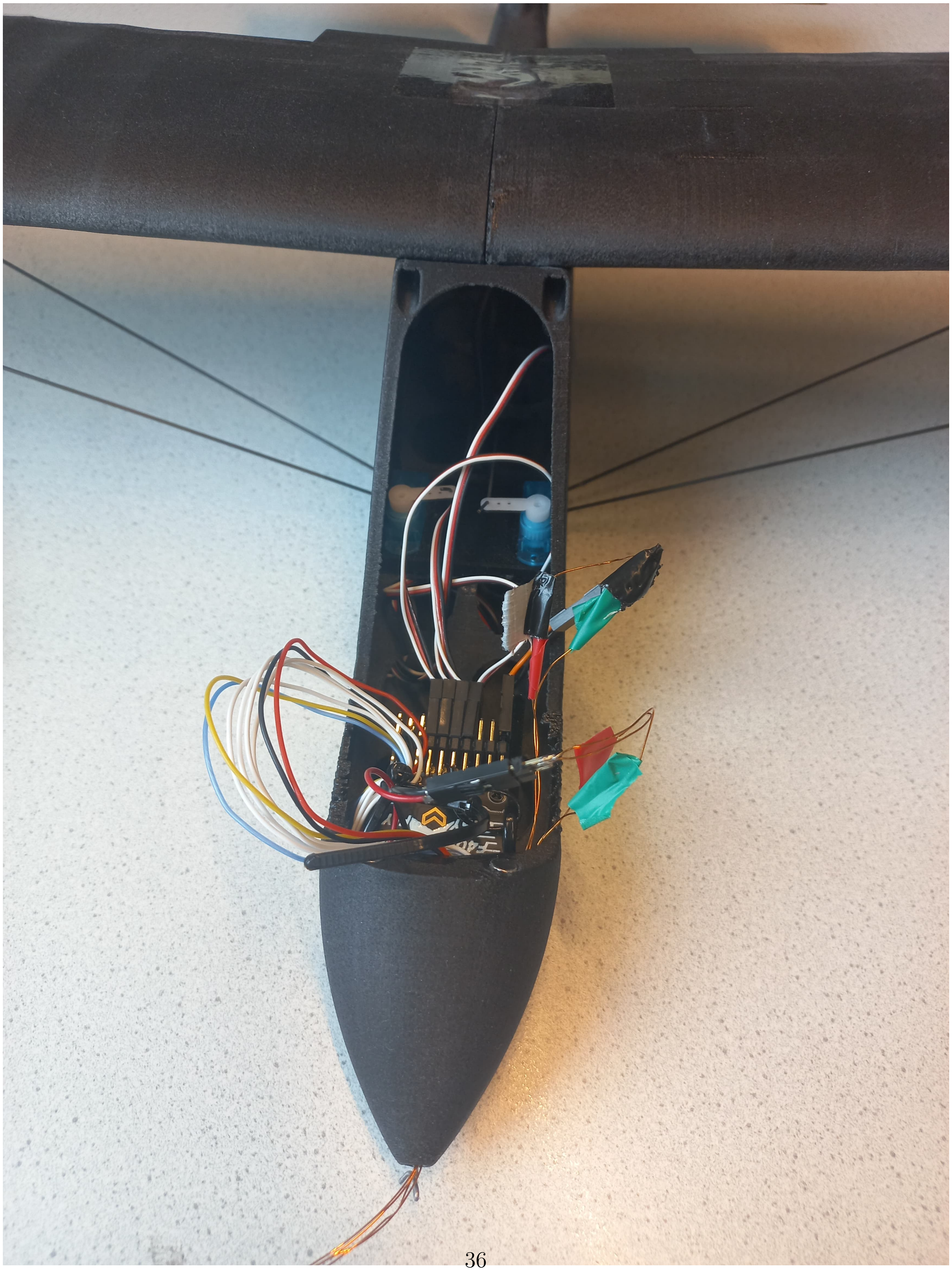


Figure 37: Electronics being housed inside the plane

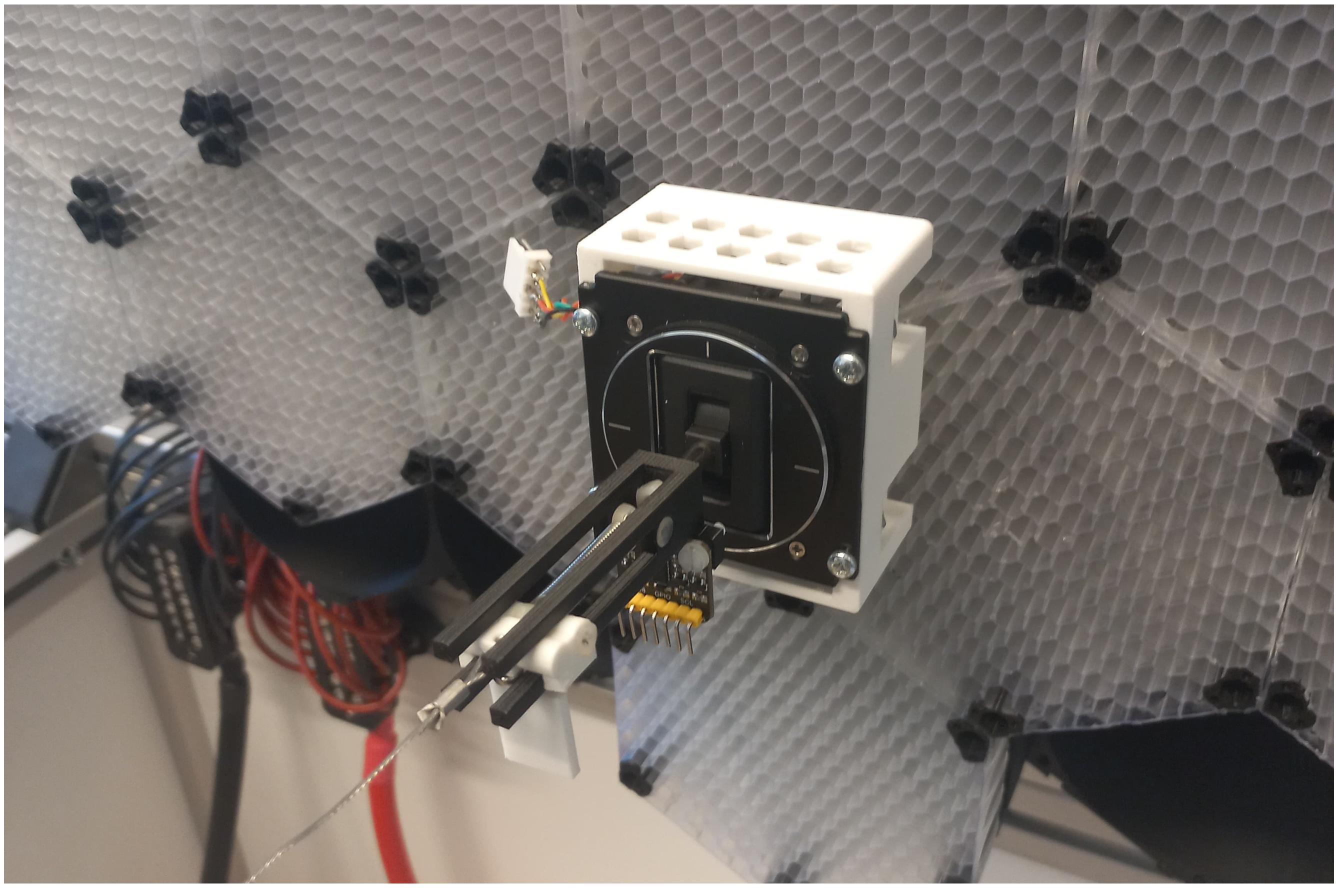
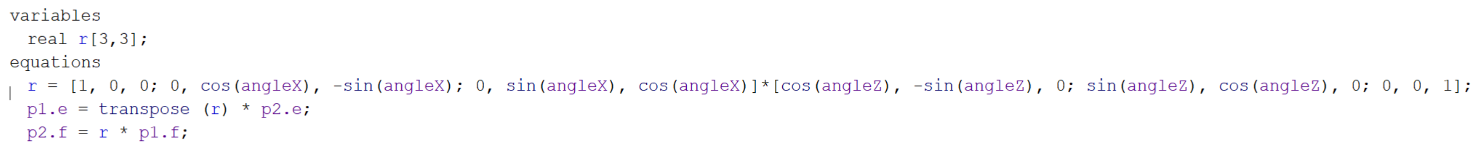


Figure 38: Tension and angle sensor attached to the ventilator

D Code for RTF



```
variables
  real r[3,3];
equations
  r = [1, 0, 0; 0, cos(angleX), -sin(angleX); 0, sin(angleX), cos(angleX)]*[cos(angleZ), -sin(angleZ), 0; sin(angleZ), cos(angleZ), 0; 0, 0, 1];
  p1.e = transpose (r) * p2.e;
  p2.f = r * p1.f;
```

Figure 39: Code that is written in 20-sim for the RTF

E Code for angle sensor experiment

```
// Define analog pins for joystick
const int joyXPin = 34;  //(green)
const int joyZPin = 35;  //(yellow)

void setup() {
  // Initialize serial communication at 115200 baud rate
  Serial.begin(115200);

  // Configure joystick pins as input
  pinMode(joyXPin, INPUT);
  pinMode(joyZPin, INPUT);
}

void loop() {
  // Print the values to the Serial Monitor
  Serial.print("Joystick X: ");
  Serial.print(joyXValue);
  Serial.print("\tJoystick Z: ");
  Serial.println(joyZValue);

  // Add a small delay to make the Serial Monitor more readible
  delay(100);
}
```

Listing 1: Arduino Code for Joystick

F Code for distance ToF sensor experiment

```
#include <Wire.h>
#include <Adafruit_VL6180X.h>

#define NUM_SAMPLES 100

// Create an instance of the VL6180X sensor
Adafruit_VL6180X vl = Adafruit_VL6180X();

int range, status;

int sensorValues[NUM_SAMPLES];
int currentIndex = 0;
float rollingAverage =0;

unsigned long startTime, endTime, elapsedTime;

void setup() {
  // Initialize serial communication
  Serial.begin(115200);

  // Initialize I2C communication
  Wire.begin(21, 22); // SDA, SCL

  // Initialize the VL6180X sensor
  if (!vl.begin()) {
    Serial.println("Failed to find VL6180X chip");
    while (1);
  }
  Serial.println("VL6180X chip found");

  for (int i = 0; i < NUM_SAMPLES; i++) {
      sensorValues[i] = 0;
   }

}

void loop() {
  // Read the range (distance)
  startTime = millis();
  range = vl.readRange();
  endTime = millis();
  elapsedTime = endTime - startTime;
  // Read the status of the range reading
  status = vl.readRangeStatus();

  sensorValues[currentIndex] = range;

  // Update the rolling average
  rollingAverage = calculateRollingAverage();

  // Print the rolling average
  Serial.print("Rolling Average: ");
  Serial.println(rollingAverage);
  Serial.println(elapsedTime);

  // Increment the currentIndex and wrap around if necessary
  currentIndex = (currentIndex + 1) % NUM_SAMPLES;

  // Print the range and status
  if (status == VL6180X_ERROR_NONE) {

  } else {
    Serial.print("Error reading range: ");
    Serial.println(status);
  }

}

float calculateRollingAverage() {
  float total = 0;

  // Calculate the sum of all values in the sensorValues array
  for (int i = 0; i < NUM_SAMPLES; i++) {
    total += sensorValues[i];
  }

  // Calculate the average
  float average = total / NUM_SAMPLES;
  return average;
}
```

Listing 2: ESP32Wroom code for tension sensor distance

G Tension sensor code

```
#include <Wire.h>
#include <Adafruit_VL6180X.h>

// Create an instance of the VL6180X sensor
Adafruit_VL6180X vl = Adafruit_VL6180X();

// variables for rolling average
const int bufferSize = 100; // Number of readings to average
float forceBuffer[bufferSize]; // Circular buffer to store force values
int bufferIndex = 0; // Current index in the circular buffer
float totalForce = 0; // Sum of the values in the buffer
float rollingAverage = 0;

//variables for range
float offset_range ; //offset that the sensor value has with respect to the
    values used to obtain the mapping formula
float offset_measurement = 34; // 34 mm after the mapping formula corresponds
    to the point of 0 [N] and is used to obtain the difference
float force;

int range, status; //range read by sensor and status of sensor
float dist; // distance after corrective mapping formula

const int numSamples_cal=2000; //calibartion samples
long sum =0;

void setup() {
  // Initialize serial communication
  Serial.begin(115200);

  // Initialize I2C communication
  Wire.begin(21, 22); // SDA, SCL

  // Initialize the VL6180 sensor
  if (!vl.begin()) {
    Serial.println("Failed to find VL6180 chip");
    while (1);
  }
  Serial.println("VL6180 chip found");

  // Initialize the force buffer
  for (int i = 0; i < bufferSize; i++) {
    forceBuffer[i] = 0;
  }
  Serial.println("Calibrating in 3 sec");
  delay(1000);
  Serial.println("Calibrating in 2 sec");
  delay(1000);
  Serial.println("Calibrating in 1 sec");
  delay(1000);
  Serial.println("Calibrating.....");
  for (int i = 0; i < numSamples_cal; i++) {
    range = vl.readRange();  // Read the sensor value
    sum += range;  // Add the sensor value to the sum
  }
  Serial.println("Calibration done!");
  offset_range= sum / (float)numSamples_cal;  // Calculate the average
  Serial.println(offset_range);
  offset_range = 30.5-offset_range; //subtract measured range from starting
     point of mapping formula (30.5) to obtain offset needed to have correct
     mapping
  Serial.print("Offset sensor value: ");
  Serial.println(offset_range);  // Print the average value

}

void loop() {
  // Read the range (distance)
  range = vl.readRange();

  // Read the status of the range reading
  status = vl.readRangeStatus();

  // Print the distance being read by sensor
  Serial.print("Distance measured: ");
  Serial.print(range);
  Serial.println(" mm");

  //Convert distance read by sensor to actual distance by mapping formula
  dist = -0.0079 * pow((range+offset_range), 2) + 1.7963 * (range+offset_range)
      - 13.622;

  //force calculation
  force = 0.171 * (dist - offset_measurement);

  // Update the rolling average
  totalForce -= forceBuffer[bufferIndex]; // Subtract the oldest value
  forceBuffer[bufferIndex] = force; // Add the new value to the buffer
  totalForce += force; // Add the new value to the total
  bufferIndex = (bufferIndex + 1) % bufferSize; // Move to the next index
  rollingAverage = totalForce / bufferSize; // Calculate the average

  // Print the force and rolling average
  //Serial.print("Force: ");
 // Serial.println(force);
  Serial.print("Rolling Average Force: ");
  Serial.println(rollingAverage);
  Serial.print("Distance");
  Serial.println(dist);

  // Print the range and status
  if (status != VL6180X_ERROR_NONE) {
    Serial.print("Error reading range: ");
    Serial.println(status);
  }

  delay(10); //for readablity
}
```

Listing 3: ESP32Wroom code for rudder and elevator control during parameter testing

H Test results version 2

Force applied [N]	Force measured [N]						Average error [N]
	Measurement 1	Measurement 2	Measurement 3	Measurement 4	Measurement 5	Measurement 6	
0	-0,03	-0,15	-0,12	0	0	-0,12	0.07
0,5	0,4	0,38	0,35	0,25	0,25	0,37	0.17
1	0,9	1	0,85	1	0,8	1,03	0.08
1,5	1,6	1,55	1,62	1,5	1,35	1,42	0.08
2	2,05	2,19	2	2	1,93	1,94	0.06
2,5	2,63	2,75	2,55	2,55	2,43	2,5	0.09
3	3,15	3,2	3,11	3,11	2,93	2,96	0.11

Table 5: Force measured from tension sensor compared to the force applied for version 2

The total average error came down to 0.068 [N]

I Device code

```
#include <Wire.h>
#include <Adafruit_VL6180X.h>

// Create an instance of the VL6180X sensor
Adafruit_VL6180X vl = Adafruit_VL6180X();

const int joyXPin = 34; // green wire of joystick
const int joyZPin = 35; // yellow wire of joystick

int angleX = 0 ; //angle after mapping of joystick
int angleZ = 0 ;

const int bufferSize = 100; // Number of readings to average
float forceBuffer[bufferSize]; // Circular buffer to store force values
int bufferIndex = 0; // Current index in the circular buffer
float totalForce = 0; // Sum of the values in the buffer
float rollingAverage = 0;

float offset_range ; //offset that the sensor value has with respect to the
    values used to obtain the mapping formula
float offset_measurement = 34; // 34 mm after the mapping formula corresponds
    to the point of 0 [N] and is used to obtain the difference
float force;

int range, status; //range read by sensor and status of sensor
float dist; // distance after corrective mapping formula

const int numSamples_cal=2000; //calibartion samples
long sum =0;

unsigned long startMillis, endMillis;

void setup() {
  // Initialize serial communication
  Serial.begin(115200);
  pinMode(joyXPin, INPUT);
  pinMode(joyZPin, INPUT);

  // Initialize I2C communication
  Wire.begin(21, 22); // SDA, SCL

  // Initialize the VL6180 sensor
  if (!vl.begin()) {
    Serial.println("Failed to find VL6180 chip");
    while (1);
  }
  Serial.println("VL6180 chip found");

  // Initialize the force buffer
  for (int i = 0; i < bufferSize; i++) {
    forceBuffer[i] = 0;
  }
  Serial.println("Calibrating in 3 sec");
  delay(1000);
  Serial.println("Calibrating in 2 sec");
  delay(1000);
  Serial.println("Calibrating in 1 sec");
  delay(1000);
  Serial.println("Calibrating.....");
  for (int i = 0; i < numSamples_cal; i++) {
    range = vl.readRange();  // Read the sensor value
    sum += range;  // Add the sensor value to the sum
  }
  Serial.println("Calibration done!");
  offset_range= sum / (float)numSamples_cal;  // Calculate the average
  Serial.println(offset_range);
  offset_range = 30.5-offset_range; //subtract measured range from starting
      point of mapping formula (30.5) to obtain offset needed to have correct
      mapping
  Serial.print("Offset sensor value: ");
  Serial.println(offset_range);  // Print the average value

}

void loop() {
  startMillis = millis();
```

```
// Read the range (distance)
range = vl.readRange();

// Read angle
angleX = map(analogRead(joyXPin),575,3990,-300,300);
angleZ= map(analogRead(joyZPin),485,2910,-300,300);



// Read the status of the range reading
status = vl.readRangeStatus();

// Print  status
if (status != VL6180X_ERROR_NONE) {
  Serial.print("Error reading range: ");
  Serial.println(status);
}

//Convert distance read by sensor to actual distance by mapping formula
dist = -0.0079 * pow((range+offset_range), 2) + 1.7963 * (range+offset_range)
     - 13.622;

//force calculation
force = 0.171 * (dist - offset_measurement);
endMillis = millis();

// Update the rolling average
totalForce -= forceBuffer[bufferIndex]; // Subtract the oldest value
forceBuffer[bufferIndex] = force; // Add the new value to the buffer
totalForce += force; // Add the new value to the total
bufferIndex = (bufferIndex + 1) % bufferSize; // Move to the next index
rollingAverage = totalForce / bufferSize; // Calculate the average

// Print the  rolling average force and angle
Serial.print("Rolling Average Force: \t ");
Serial.print(rollingAverage);
Serial.print("\t Angle X: \t");
Serial.print(angleX);
Serial.print("\t Angle Z: \t");
Serial.println(angleZ);
```

```
  Serial.println(endMillis - startMillis);



  delay(10); //for readablity
}
```

Listing 4: ESP32Wroom code for combining tension and angle sensor

J Code of experiment TF test

```
#include <sbus.h>
#include <arduino.h>

bfs::SbusTx sbus_tx(&Serial1, 16, 17, true, true); //rx pin 16, tx pin 17, on
    different serial, inv set true, fast_SBUS set true
bfs::SbusData data;

int pitch = 0;
int yaw = 0;

void setup() {
  /* Serial to display data */
  Serial.begin(115200);

  while (!Serial) {}
  /* Begin the SBUS communication */
  sbus_tx.Begin();

  for (int8_t i = 0; i < data.NUM_CH; i++){ //initializes all 16 channels to
      900
    data.ch[i] = 992;
  }
}

void loop() {
  if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n'); // Read input from the Serial
        Monitor until newline character
    Serial.println("Received input: " + input);

    if (input.startsWith("p:")) {
      pitch = input.substring(2).toInt(); // Get the integer value after "p:"
      Serial.println("Parsed pitch: " + String(pitch));
    } else if (input.startsWith("y:")) {
      yaw = input.substring(2).toInt(); // Get the integer value after "y:"
      Serial.println("Parsed yaw: " + String(yaw));
    } else {
      Serial.println("Invalid input format. Please use 'p:<value>' or 'y:<value
         >'.");
    }
  }


  Serial.println("pitch: " + String(pitch));
  Serial.println("yaw: " + String(yaw));

  if (pitch == 0) {
    data.ch[1] = map(0, 0, 200, 172, 1811); // -100%
  }
  else if (pitch == 50) {
    data.ch[1] = map(50, 0, 200, 172, 1811); // -50%
  }
  else if (pitch == 100) {
    data.ch[1] = map(100, 0, 200, 172, 1811); // 0%
  }
  else if (pitch == 150) {
    data.ch[1] = map(150, 0, 200, 172, 1811); // 50%
  }
  else {
    data.ch[1] = map(200, 0, 200, 172, 1811); // 100%
  }

  if (yaw == 0) {
    data.ch[3] = map(0, 0, 200, 172, 1811); // -100%
  }
  else if (yaw == 50) {
    data.ch[3] = map(50, 0, 200, 172, 1811); // -50%
  }
  else if (yaw == 100) {
    data.ch[3] = map(100, 0, 200, 172, 1811); // 0%
  }
  else if (yaw == 150) {
    data.ch[3] = map(150, 0, 200, 172, 1811); // 50%
  }
  else {
    data.ch[3] = map(200, 0, 200, 172, 1811); // 100%
  }

  for (int8_t i = 0; i < data.NUM_CH; i++) {
    Serial.print(data.ch[i]);
    Serial.print("\t");
  }
  Serial.print(data.lost_frame);
  Serial.print("\t");
  Serial.println(data.failsafe);

  sbus_tx.data(data);
  sbus_tx.Write();
  delay(100);
}
```

Listing 5: Arduino Code for rudder and elevator control during parameter testing

K List of Abbreviations

AC Aerodynamic Center. 2

AM Aerodynamic Moment. 2

AoA Angle of Attack. 2

CoM Centre of Mass. 6

RTF Rotational Transformer. 12

ToF Time of Flight. 8