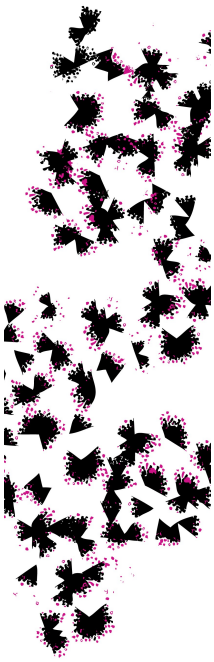

Glide-mode regulator for the Robird

J. (Jeroen) van Dorp

BSc Report

Committee:
Prof.dr.ir. S. Stramigioli
Ir. G.A. Folkertsma
Ir. W. Straatman
J.P. Schilder, MSc

June 2016

012RAM2016
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Acknowledgment

I would like to thank my daily supervisor Geert for the guidance and advice, where needed. Next to that I would like to thank Wessel for providing me this assignment in the first place and providing me with all the data and materials needed to work on this assignment.

Abstract

This report is about the research and development of a controller that enables the Robird to go into 'glide mode'. Currently there is already such a controller implemented in the Robird, but this is not very accurate nor very fast. Research on the Robird was done to determine what the possibilities were to design this controller. An embedded system, a Ramstix, was used for this purpose. Read-out and control of the system is done by the Ramstix and eventually a working controller was developed. Extensive testing of this controller in several set-ups is done to proof the concept. For the controller to work inside a flying Robird only one thing is left to be done; implementing this controller onto a smaller embedded system that fits inside the Robird. Recommendations for such an embedded system are also given in this report.

Contents

1 Introduction

The Clear Flight Solutions Robird is a robotic bird mimicking the flight of real bird of prey. The first model, a peregrine falcon, was put together by a falconer and model aeroplane enthusiast. This was done by trial-and-error. It makes the Robird an interesting research object as it already flies, but there is not yet a complete understanding about the how and why it flies. This also makes it harder to improve the Robird as it is just not yet completely known why it does what it does. During my bachelor assignment I am going to focus on improving reliability and speed of the switch between glide mode and flapping wing mode. This report will start with some general information needed to get a good understanding about what the assignment is and what needs to be done. This will lead to some requirements for the system that will be build. Next the focus will be on finding out the dynamics of the system both by measuring and modelling. After that based on the knowledge that is gathered about the system a controller will be designed and tested. At the end, implementation of the controller in the Robird will be discussed. From all this a conclusion will be drawn and there will be a discussion.

1.1 General information Robird

Now some general information of the Robird will be given just to get a bit of an overview of the robot.

Use of the Robird In general the Robird is used for bird-friendly pest bird control. This has proven to be very useful in aviation, waste management, harbours and agriculture, where birds can cause many problems[1]. The birds are not only chased away during the flight of the Robird in the area, but they also stay away from that area afterwards as they know that there is a 'predator' present in that area. Also, as far as is known, birds do no get used to this bird control method as would happen with for example methods where scarecrows or noise[2] are used to chase off birds. If birds would get used to the Robird it is very likely they are also not scared of real birds of prey any more, resulting in a much higher chance of becoming a prey. This makes it a very effective method of pest bird control compared to other methods.

Dimensions There are two models of Robirds. One resembles a peregrine falcon (see figure 1 on the following page) and the other a bald eagle. Both have a similar size and weight as their biological counterparts. This results in flight dynamics that are comparable to the real birds, making it nearly impossible for other birds to distinguish between a real bird and a Robird.

Figure 1: On the right a real peregrine falcon[3] and on the left one of the Robirds; almost indistinguishable.

Flight modes Two flight modes can be identified for the Robird. Namely the flapping-wing mode and the gliding mode. As expected in the flapping-wing mode the Robird propels itself by flapping its wings. This is a very important flight mode for the robot as this is the only way it can gain altitude. Next to that this is the mode in which it can extrude birds. For real birds of prey flapping their wings is the behaviour they show while hunting.
Next to that there is the glide mode. In this mode it can soar. The wings are locked in a certain position, the gliding position. While in this mode not much energy is used, because the wings are not flapping, therefore it is an excellent way to extend the flight duration of the Robird. Real peregrine falcons also have this 'flight mode'. They soar, as this gliding is called, at a high altitude searching for their prey[4]. This soaring also saves the real peregrine falcon valuable energy.

Control and autopilot The Robird can not yet fly autonomously, this is a goal for the future. Ideally the Robird can completely pilot itself, such that there are no human pilots needed any more for the Robird. The Robird for example would then fly to pre-programmed way-points to chase birds away. When it is done doing this it would then automatically return to its base to recharge and to be serviced if necessary. Before that can happen there are still quite some things to be done. W. Straatman however has already done a lot to improve the flight behaviour to take some load of the pilot[5]. At first the pilot had to keep the Robird stable all the time, which is a heavy task as the pilot had to be focused all the time. For example the bird is nose-heavy, because of this a slight pitch upwards is required to maintain altitude. This slight pitch was something the pilot had to do manually all the time. The Robird can now keep itself stable, thus making it easier to pilot the bird.
Before the autopilot was developed the pilot had direct control over all the channels for the ruddervators (combined rudders and elevators) and with that the control over pitch and roll. With the autopilot there is the FBWA (Fly-By-Wire A) mode. In this mode the pilot does not have direct control over ruddervators any more. Instead the pilot now controls the pitch and roll, which become a set-point for the autopilot. This autopilot subsequently controls all the separate channels to achieve the desired set-point.

Switching between flight modes is also done by the autopilot. It gets a throttle input from the pilot and when that is below a certain threshold it puts the wings in glide mode. As long as it is above the glide mode threshold the pilot still has direct throttle control.

1.2 Existing hardware Robird

It is important that the glide mode regulator that will be developed can interface with the hardware that is already present in the Robird, to ensure an easy and inexpensive implementation. Therefore it is important to find out what that hardware is and how the interfacing can be done. In figure 2 is a schematic overview of what the system with the glide mode regulator in place would look like. The regulator will get the throttle input from the autopilot. Next it will control the ESC (electronic speed controller) which drives the three phase electric motor. The motor then drives the gearbox mechanism which lets the wings flap. Attached to this gearbox there is a sensor (a hall-sensor) which measures the position of the wings and thus the motor. Currently that is connected to the autopilot and is read out with 50 Hz.

It is also possible that another kind of sensor can better be used to measure the position of the wings, because of better accuracy or less disturbance of the system. At the moment the hall-sensor is placed on a separate axle that is connected to the gearbox, because of the fact that the motor now needs to rotate more mass (the additional axle) it influences the dynamics of the complete system. Whether another sensor will be used will be looked into later on when the assignment has been started.

When the to be developed glide mode regulator is in place it will read-out the sensor instead of the autopilot. Possibly with a higher frequency as this will give the possibility for better accuracy when estimating the position of the wings and motor. For data-logging purposes the measured position can be send to the autopilot.

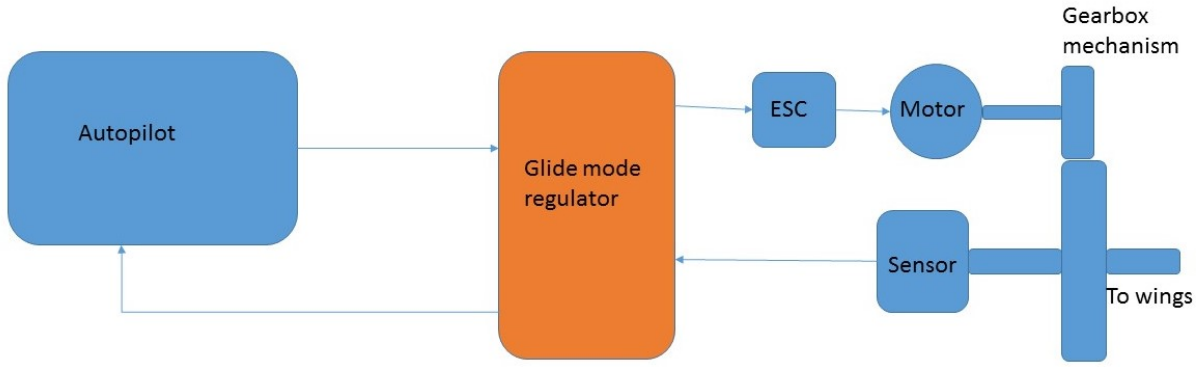

Figure 2: Schematic overview of the system with the existing hardware of the robot coloured blue.

The autopilot/brain of the Robird is a Pixhawk PX4[6], which can be used for all kinds of remote controlled vehicles. With some software modification it can also be used to pilot the Robird.

A locking mechanism for the wings to lock them in glide-mode is already installed on the Robird (see figure 3 on the following page). This cochlea shaped wheel is on a separate axle inside the gearbox between the motor and wings. The only purpose of this axle is to be able to lock the complete gearbox in the right position for gliding. During the flapping wing mode the cochlea shaped wheel rotates counter clockwise. Putting the wings in glide mode requires the motor to be stopped such that the pawl on the wheel is somewhere in the blue area. The lift of the wings will then push the wings and the wheel that is connected to it into the locking position. Currently this cannot be done very precise, because this control is not done based on a dynamical model of the system. Next to that the measurements of the position of the wheel is only done with 50 Hz and the wheel rotates with 5-7

Hz, resulting in less than 10 position measurements during every rotation. The blue area, in which the wings need to be stopped, is already smaller than one tenth of the complete wheel, thus it can occur that multiple extra wing flaps are completed after the system should already be in glide mode, before it goes into glide mode.

1.3 State estimation

It is important to be able to know what the speed and position of the motor and wings are as precise as possible at any time. For this purpose there is already a sensor attached to the gearbox, which can be read out to determine both position and speed. This however will be done with a certain sample frequency, resulting in a non-continuous signal. In between the measurements the position and speed are unknown, but are needed to be able to control the system. What can be done to solve this is use state estimation [7;8]. Here the state of the system, so in this case speed and position, will be estimated by using what is observed. Based on the input (throttle in this case), the output (sensor that is read out) and a mathematical model of the dynamics of the motor and wings the states of the system can be estimated very precise at any time it is needed. There are several methods that can be used to do state estimation. One of them is the Kalman-filter. This particular method can be useful if there is a lot of noise in the measurements. If and what specific method of state estimation will be used will be determined later as first more research on the actual system to be controlled needs to be done.

1.4 Dynamics of the wings and motor

As mentioned in subsection 1.3 a good mathematical model of the dynamics of the wings and motor is important to be able to design a good controller. Both for possible state estimation and for testing the controller in a model before it is tested in a physical set-up. Especially the dynamics of the wings will be important as these will cause a varying load on the motor. When the wings go down the motor will rotate slower, because of the higher load, when the wings go up it is the exact opposite. This behaviour is what will make the state estimation hard. Modelling will be done using the 20-sim software package. Based on in-flight data and knowledge of the system acquired by doing measurements a suitable model will be made. This should in simulation show the same behaviour as the physical system and can therefore be verified with the help of real measurement data.

1.5 Controller design

For the controller design the same holds as for the modelling part, 20-sim will be used. When a good model is made designing a controller and testing it can be done in 20-sim. Some testing and tweaking can this way easily be done before the real-life testing is started.

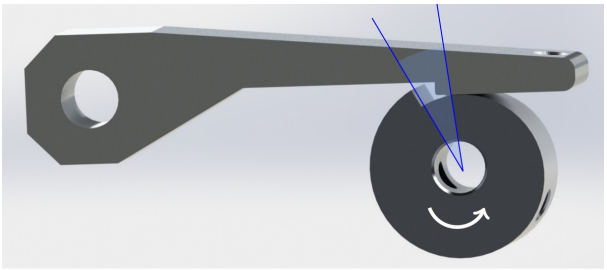

Figure 3: Passive locking mechanism for the Robird wings [5].

1.6 Embedded system

For first testing purposes a Ramstix embedded system will be used. If the modelling and controller design is done in the 20-sim software package the implementation of the controller should be quite straightforward. The Ramstix is compatible with 20-sim using the 20-sim 4c software. This will save time in the testing phase, as implementation on the embedded system should be simple. Later on, when a desired controller is developed and time permits, a smaller and lighter embedded system can be chosen that fits inside the actual Robird. This is of course a must if the controller is really going to be used in the Robird.

2 Requirements and goal of the assignment

2.1 Requirements

It is important to set up some requirements for the final system, as this makes it later on verifiable whether the assignment was successfully completed or not. There are no hard requirements from Clear Flight Solutions, thus by looking at some flight data and using common sense requirements are set. The requirements for the final system are:

1. The system will be made for the current peregrine falcon model.
2. Eventual size will be such that it fits inside the Robird.
3. After the 'command of locking' is received the locking needs to be done within two wingflaps.
4. Switching between glide and flapping mode is decided by the glide mode regulator.
5. Braking is done smoothly, no abrupt stopping of the motor.
6. The input (throttle input from the pilot) of the system will be PWM, such that it can interface with the Pixhawk.
7. The output of the system will be PWM, such that it can interface with the ESC.
8. If possible replace the sensor wing angle sensor with a contactless type.

2.2 Goal of the assignment

The assignment is to investigate the dynamics of the wings and motor, such that a controller can be designed for putting the wings in glide mode. Further analysis of the characteristics of this model will probably enable to design a better controller that can smoothly and reliably bring the wings in glide position. It is most likely also needed to determine the characteristic curve of the electric motor in order to be able to make a braking curve, such that the motor can be stopped from every position and every speed smoothly. This controller will have to be implemented on an embedded system, probably Arduino based, and will have to interface with the already existing hardware of the Robird.

3 Measurements

In this section the 20-sim model made for the measurement part of the set-up will be explained. Next to that the different measurement set-ups will be explained, together with the flaws they have compared to the real Robird.

3.1 Hardware

Before the 20-sim model for the read-out of the sensor and the control of the throttle can be made it needs to be clear how the interface between the Ramstix and the parts of the Robird should be. The Ramstix should do roughly the same as the Pixhawk does in the real Robird. In the Robird the Pixhawk gets a PWM servo signal with 50 Hz refresh-rate from the receiver. Next it is determined whether this throttle input is below or above the glide mode threshold. If it is above that threshold the Pixhawk puts out the exact same PWM servo signal at 50 Hz to the ESC (20AUBEC Hobbyking ESC). The ESC translates this to some desired frequency and puts out a three-phase signal to the motor (T-motor MT2212 KV980) which starts rotating. In the Robird this ESC is powered by a 4 cell LiPo, which directly through the ESC also powers the motor. In the Ramstix a throttle input from the pilot needs to be simulated, next it needs to put out the PWM signal to the ESC. The ESC in the test set-up will be powered by a lab power supply, this makes sure no time is lost with recharging empty LiPo's while testing. The Ramstix also needs to read-out the position sensor (AMS AS5600[9]). The sensor puts out some voltage which is between 0 and the supply voltage of the sensor. This supply voltage is on the Pixhawk 5 volt and on the Ramstix a 3.3 or 5 volt supply can be used for this. The voltage from the sensor is read-out by one of the 2 16-bit AD-converters of the Ramstix.

3.2 20-sim model for measuring

Based on what is explained in section 3.1 a 20-sim model can be made for the Ramstix. This 20-sim model that is depicted in figure 4 on the following page will be explained. All the blocks that are not depicted in this chapter can be found in the appendix.

As external input there is the `Position_sensor`. This goes into the `Map2radian` submodel, where this signal is converted into radians and translated such that the locking position is in π. This to make sure that no steps from 2π to 0 cause complications around the locking position when designing the controller. Next this position signal goes to the frequency calculator, which can later be used for data logging purposes and the controller. The second part of the 20-sim model is the throttle input part. The output of this `Throttle_input` model is between 0 and 1 where 0 is the lowest and 1 the highest throttle position. This goes through a limiter to make sure that the `Throttle_input` output to the `Servo_pwm_converter` does never exceed the intended maximum of 1. Inside the `Servo_pwm_converter` the 0 to 1 throttle signal is converted to a corresponding PWM signal and is outputted over one of the pins of the Ramstix that is equipped with servo PWM support.

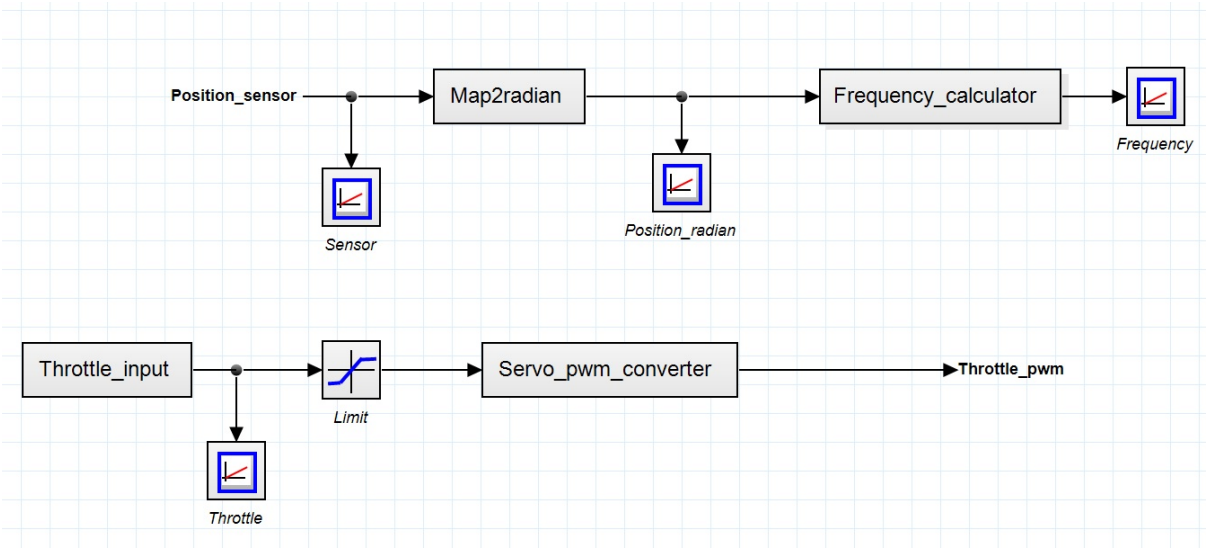

Figure 4: 20-sim model for doing the first measurements

3.3 First testing in several set-ups

With an old gearbox mechanism from CFS the 20-sim model and Ramstix set-up can be tested. This is the phase in which the dynamics of the system can be explored based on the measurement data. Most interesting will be the throttle versus wing flap frequency and the 'non linearity' in the position graph.

Measurements on mechanism without wings From the first measurement data as shown in figure 5 on the next page several things can be derived. First of all the position is much more accurate over time than what is logged by the Pixhawk (figure 6 on the following page). This can be explained by the higher refresh rate. The Pixhawk can only log at 50 Hz where the Ramstix can easily log at 1 kHz as was done during this measurement. The frequency that is calculated has two problems. The first being that it is off by a factor of two. This was easily solved later on by correcting a fault in the frequency calculator. The other thing is the noise that is present in the channel, which is significant. This is mainly due to small noise that is present in the position sample, which becomes large if some differential is taken of that.

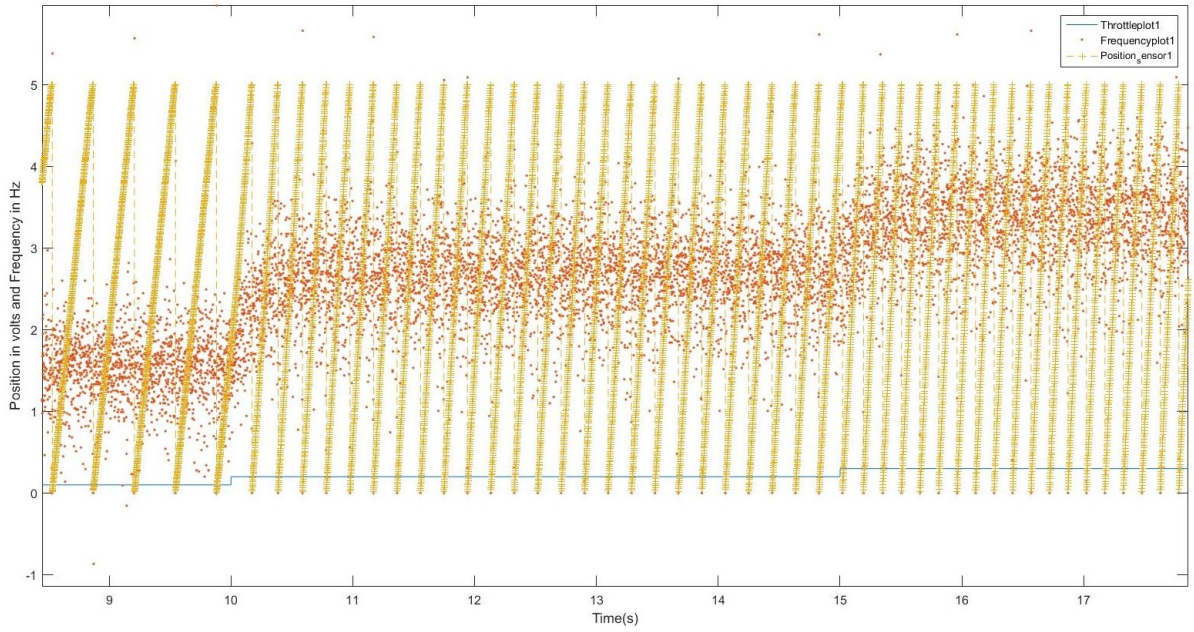

Figure 5: First measurement data on the x-axis time in seconds on the y-axis the read-out of the sensor and the frequency in Hertz. The sawtooth is the position sensor, the dots are the calculated frequencies and at the bottom the throttle level is shown.

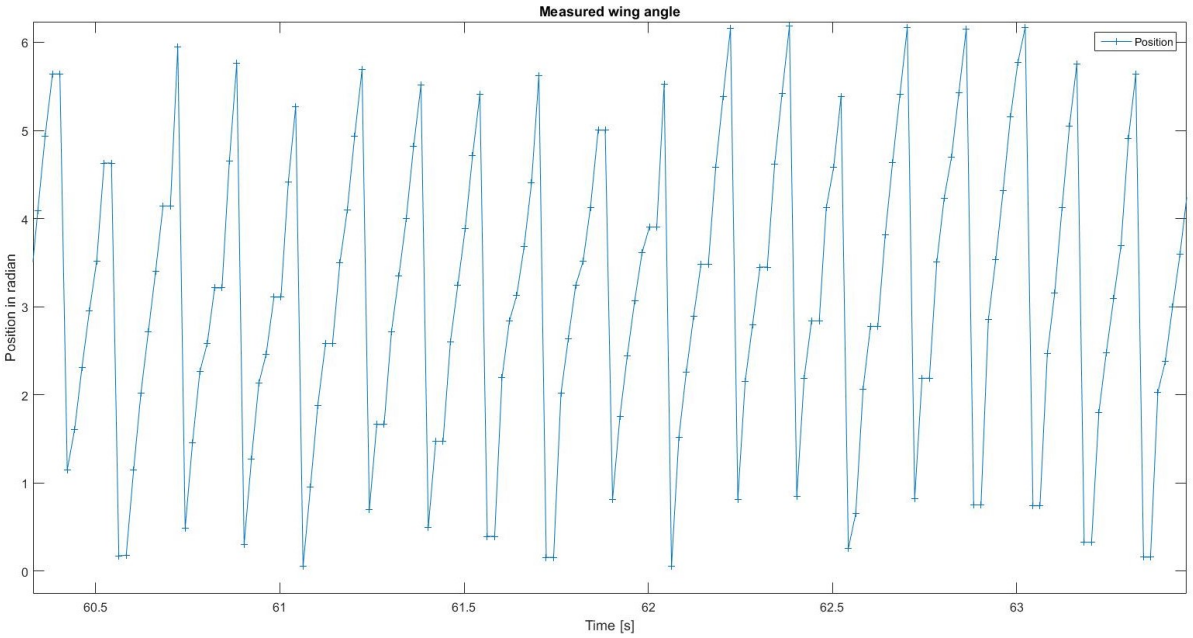

Figure 6: Flight data logged at 50 Hz. Only the position sensor read-out is shown here. All the data points are the '+' in the graph.

Measurements with wings The dynamics that are found in the Robird cannot be simulated by the set-up without wings. In order to get an insight in these dynamics it would be best to put the Ramstix in a flying Robird, due to size constraints this is not possible. Therefore a static test set-up from Clear Flight Solutions is used. Important to notice is that in this case the mechanism is hold steady and the wings flap. Thus on a down-flap gravity is positively acting on the motion and air drag negatively.

When the wings go up, gravity is acting negatively on the motion of the wings as well as air drag. Some data acquired in this set-up can be seen in figure 7 and figure 19 on page 32. Here clearly can be seen that it is not a nice sawtooth any more, as was the case when no wings were attached. The mechanism clearly speeds up and slows down at certain points because of the wings. This was the first time that these dynamics were measured with this precision/speed.

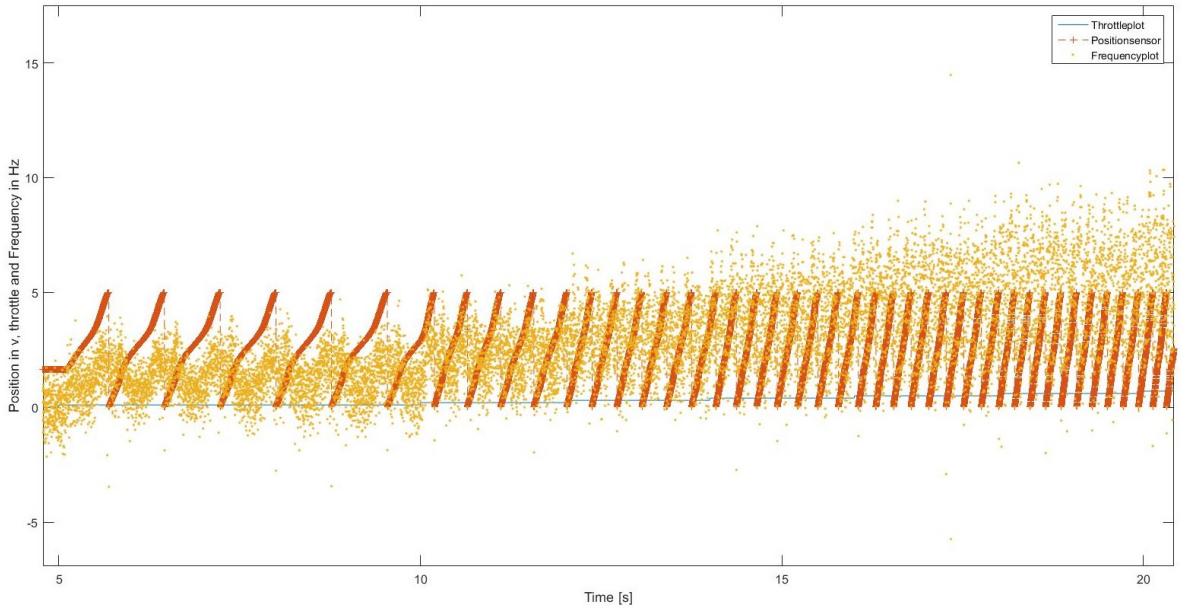

Figure 7: Measurement data in set-up with wings the non-linear behaviour of the sawtooth can now be seen. A zoomed in graph on the higher frequency part can be seen in figure 19 on page 32. The dots that are spread all over the graph are measurements of the unfiltered frequency, clearly there is still a lot of noise.

Logged flight data Logged flight data was also a source of important information about the behaviour of the system. This data can be seen in figure 6 on the previous page. Important here is to notice that the refresh rate was only 50 Hz as this is the highest frequency that can be reached by the Pixhawk. Although this is the only data of the bird flying that is available it does not give as much insight in the dynamics of the system as the measurement done with the wings in the CFS set-up.

3.4 Knowledge acquired

The dynamics in the real Robird are still not completely mapped due to a lack in accuracy. This accuracy was however there for the measurements with the static mechanism with and without wings. Based on that data and knowledge a mathematical model can be made describing the behaviour of the system, which will be explained in chapter 5 on page 18. The next thing that was learnt from the first measurement was that the position sensor seems precise and without a lot of noise. Combined with the fact that the read-out of this sensor can easily be done at 1kHz the need of state estimation for a controller later on seems unnecessary. Important for a test set-up to test the glide mode regulator is that the wings are being pulled upward, otherwise they do no lock. The thoughts and development of a simple test set-up which is capable of doing this are explained in chapter 4 on page 16. Lastly from the data from the measurement without the wings a throttle versus frequency curve can be extracted, as can be seen in figure 17 on page 30. This will be useful for the mathematical models.

The following relation is found for the throttle and frequency:

$$f = -18.221 * \text{throttle}^4 + 60.014 * \text{throttle}^3 - 72.009 * \text{throttle}^2 + 39.529 * \text{throttle} \quad (1)$$

4 Test set-up

It was decided to design a simple test set-up in order to test the glide-mode regulator. Erik Landman, another Advanced Technology student, would in the mean time also be busy with designing and building a test set-up which could house the complete Robird in which the dynamics would be as close as possible to the ones found during flight. Eriks set-up would probably not be finished in time, therefore a simple set-up is designed for this assignment.

4.1 Requirements of the test set-up

The most important requirement for the test set-up is that it can exert a force on the mechanism similar to the force that is normally acting on the wings during flight. On average this force will be the weight of the Robird with gravity acting equally on both wings, because if this is the case the average altitude will stay the same, hence the Robird flies. So we get:

$$F_{\text{gravity}} - F_{\text{wings}} = m_{\text{Robird}} * a_{\text{average}} \tag{2}$$

where $a_{average} = 0$, thus $F_{gravity} = F_{wings}$. The Robird has a weight of approximately 0.730 kg.

$$F_{\text{gravity}} = m_{\text{Robird}} * g = 0.730 * 9.81 \approx 7.2\,\text{N} \tag{3}$$

I assume that half this force acts on both wings at half the length of the wings. The wings have a length of approximately 50 cm, thus the force would act at 25 cm from the shoulders. So in both shoulders there is a moment of

$$M = FR = 7.2/2 * \times 0.25 = 0.9\,\text{Nm} \tag{4}$$

There needs to be something in the test set-up that creates this moment in the shoulders.

4.2 Design of the test set-up

The design is as follows; the gearbox is fixed with rods instead of wings inserted. Attached to those rods are springs. To make sure the set-up does not become to big for the laser cutter (which would be used to make the parts quickly) it cannot be much larger than 50 cm. The rods are made such that the springs can be attached at 15 cm from the shoulder. With this distance the force of the springs can be determined. $F = M/R = 0.9/0.18 = 5N$ would be the force that is needed. Added to that is that over a change in length of the spring, a wingflap, it should not change too much, because this also does not happen in the real Robird. In one wingflap the spring length changes with approximately 20 cm and the change in force over this distance should not be too large. Finding springs that meet these demands is nearly impossible. Using rubber bands was a good alternative. In the set-up which can be seen in figure 8 on the next page and figure 9 on the following page it can be seen that the mounting point of the rubber bands is variable, this makes that the tension in the rubber bands can be varied to get the required force from the rubber bands. Although this set-up is not very precise it does pull the wings in glide mode, thus it can be used to test the glide mode regulator later on.

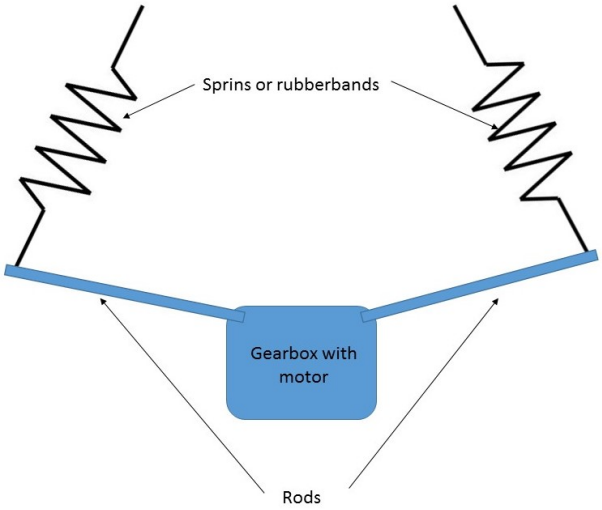

Figure 8: Schematic drawing of the test-set-up

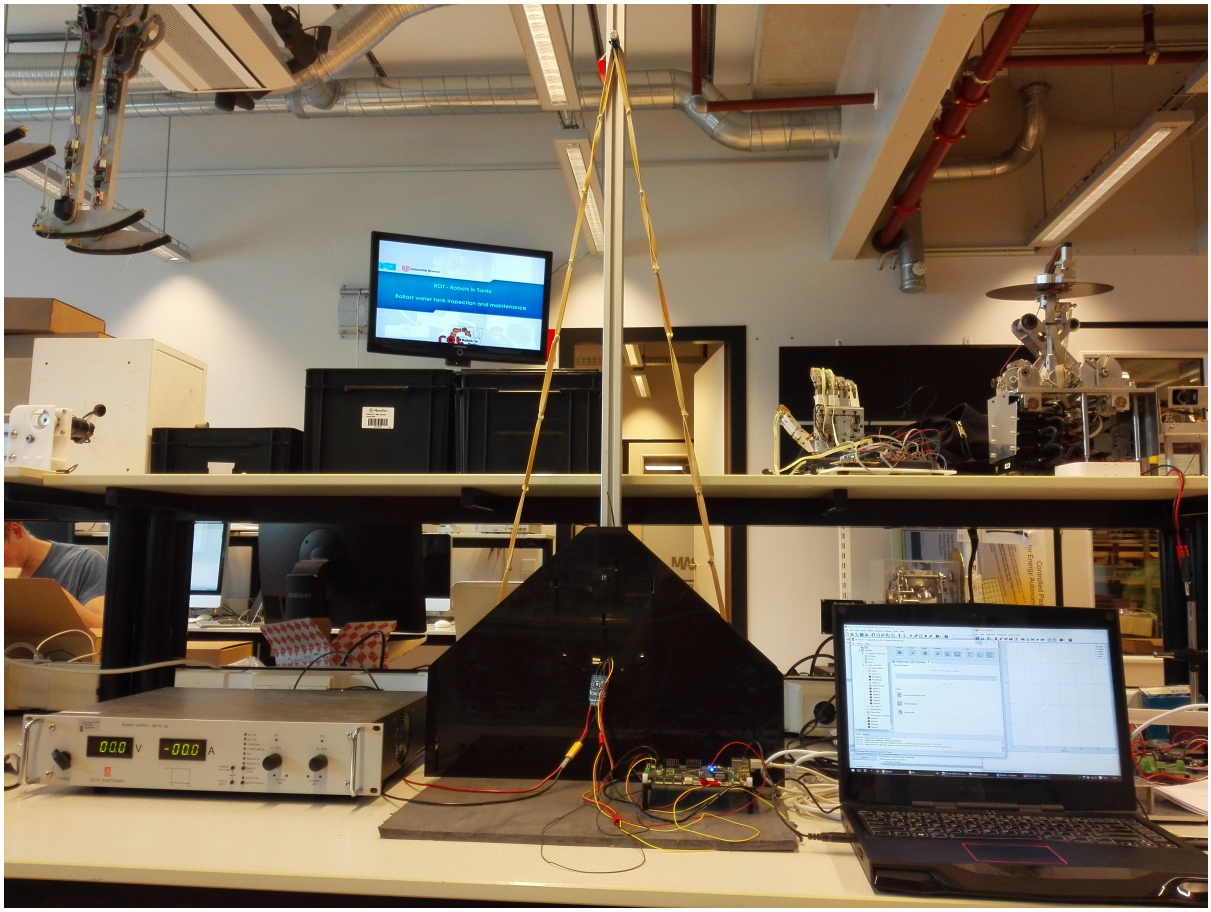

Figure 9: The test-setup where the rubberband mounting point can clearly be seen at the top of the picture. On the left there is the lab power supply, in front of the set-up there is the Ramstix and on the right the laptop to control it all.

5 Modelling

In this section the models that are made describing the behaviour of the system will be explained. Several different models are made for several different situation. Each of them will be explained piece by piece. All models will be made using bond graphs, a way off graphically making mathematical models. It is possible that the reader of this report is not acquainted with bond graphs and therefore might have some problems understanding the models. If this is the case I would like to refer to the lecture notes of the course Engineering system Dynamics [10] where the use of bond graphs is introduced and explained. Furthermore, all the code of the sub-models that are described and explained in this chapter can be found in the appendix, 11.2.1.

5.1 Model for the set-up without wings

The model as seen in figure 20 on page 33 should show the same behaviour as the mechanism without wings as found in section 3.3 on page 12. The model starts with an MSE (modulated source of effort) which goes to a 1-junction, the rotational velocity of the motor. On this 1-junction the motor-inertia works and a TF (transformer) with a ratio of 1/19. This is the ratio between the motor axle rotation and the rotation of the main shaft to which the wings are attached. This TF goes to another 1-junction, which is the rotational velocity of the main shaft. On this rotational velocity an inertia of the gearbox and a resistance of the gearbox work. This main shaft velocity goes through an AD block after which it is integrated and mapped to one rotation by the `Map2one_rotation` block in order to get the position of the main shaft. From this the frequency can be calculated with the `Frequency_calculator` block. Next there is the `ESC` block, which gets an input for throttle and an input for the frequency. This block checks whether the frequency that should be reached with a certain throttle level is reached. If the measured frequency is lower the motor goes on full power. When the throttle and frequency correspond the motor does not put any power into the system. In the case the frequency is higher than desired the ESC brakes the motor with full power until the correct frequency is reached.

5.2 Model for the set-up with wings

The model as seen in figure 21 on page 34 is a model for the case where the mechanism is put in the CFS wing balancing set-up as mentioned in section 3.3 on page 13. Some small changes have been made compared to the model without the wings. First of all the `RAD2RPM` block is added, because this makes it easy to compare the motor RPM to the specifications of the motor [11]. Next to that also the integration of the main shaft rotation is now done with a continuous integral instead of a discrete one. Also the mapping to one rotation is done continuous. Only after that the signal is converted to a digital signal with which the frequency of the rotation is calculated. This is done to get a better representation of the physical system. Several other things are added compared to the previous model. There is the `Motion_of_wing` sub-model. This is to transform the motion of the main shaft into the motion of the middle of the wing. Inside that block there is a cosine and a gain which represents the distance from the shoulder (rotation) point to the middle of the wing. The output of the `Motion_of_wing` sub-model comes together with the rotational velocity of the main shaft in an MTF (modulated transformer) that translates the rotation of the main shaft in a motion of the middle of the wing. This is represented by the 1-junction at the right. On this 1-junction, the speed of the middle of the wing, several things work. There is an inertia (I) of the mass of both wings and there is the gravity (SE) acting on the wings. Next to that there is also the airdrag force that is acting on the wings. The magnitude of the force is determined by the `Airdragforce`.

5.3 Model for the test set-up with rubber bands

For the built test set-up with rubber bands as explained in chapter 4 on page 16 also a model is made. In this model an addition to the ESC was done to make it 50 Hz as the ESC is normally controlled by a 50 Hz servo pwm signal. This could affect how well the controller works in the mathematical model and the physical system. The rest of the model is basically the same as the one with the wings. An extra sub-model `rubberband` is added which translates the motion of the rods into some force that is exerted by the rubber bands. On the 1-junction representing the movement of the rods now act the inertia of the rods, the Se for the gravity on the rods and the extra inertia that will possibly be added to the rods. For now the extra inertia is kept 0, but could be altered in the future to see how much it would influence the behaviour of the test set-up theoretically if mass was added to the rods.

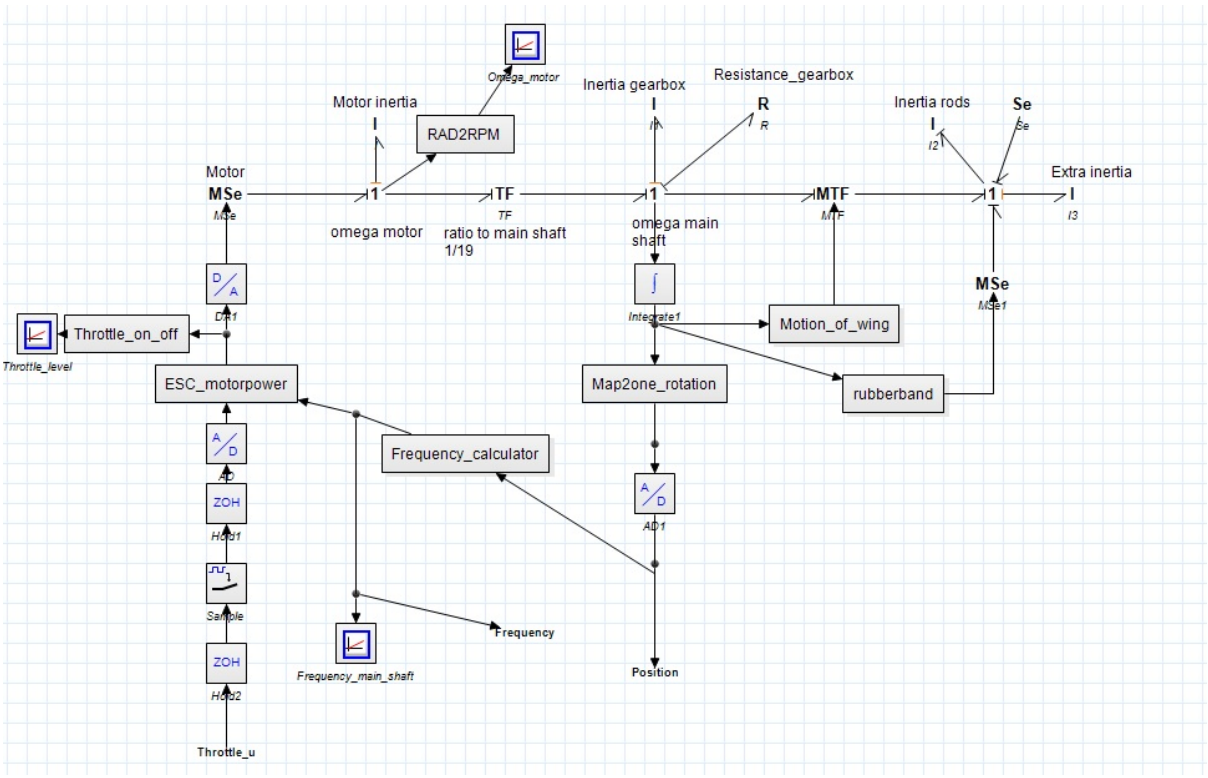

Figure 10: 20-sim model of the set-up with rubber bands

6 Controller design

6.1 Basic ideas

Now that several models and a test set-up have been made the design of the controller can start. From the start of this assignments I had several ideas for the design of the controller. For the design of the controller it is very important to notice that the motor can only rotate in one direction, thus if the motor is past the locking position it needs to make another rotation to get there again. Next to that the only thing that can be controlled is the frequency set-point for the ESC. The controller is not meant to control the frequency, but to control the position. Keeping this in mind it is not as straight forward as it seems to make the glide mode regulator.

6.2 PID control

The first idea was to use a PID controller with as input the position of the wings and with the output being the throttle level, corresponding to a set-point frequency for the motor. The set-point for the controller is π. There is the proportional part of the controller that reacts on the error between the set-point and the measured value, becoming smaller as the error becomes smaller. Next there is the differential part of the control that will brake faster or slower depending on what the proportional control already does. The integral part would give the motor some extra throttle input to rotate further if a steady state error arises (the motor is stopped too early). However, after further examination of what would happen, the conclusion was that it was not as straight forward as initially thought. The position could be measured, but controlling the position cannot be done directly. Only the throttle level, thus the rotating speed of the motor can be controlled. The proportional part of the controller will thus decrease the throttle if the set-point offset becomes smaller. This would already act as a braking mechanism. The differential part of the controller would work on the differential of the position, thus in this case the deceleration. What happens is that with a fast deceleration with a positive gain the motor would be braked even faster and with a negative gain the controller would do the opposite. It would be better to let the differential part work on the decrease or increase of the frequency. So if the motor frequency decreases too fast it would throttle up a bit and if it decreases too slow it would slow it down faster. The integral part will in theory still do what it was intended to do. If however the proportional and differential gain are chosen correctly the integral part would be superfluous.

6.3 PD control

Based upon the idea of the PID control where the integral part was superfluous a PD controller seemed a good option. The proportional part working with the position as input and the throttle level as output. The differential part would then need to work on the frequency as calculated by the frequency calculator that was already in place. The signal from the frequency calculator without a filter responds fast to changes in the position, however due to noise using this output does not provide useful feedback. In any case it is needed to let this signal go through a low-pass filter before it can be used for the differential part of the controller. This will be further examined and explained in chapter 7 on page 22.

6.4 P control

Another possibility is using P control, the same way as the PD control would be realised only then without the differential part. Only a suitable gain needs to be found for the error in the position which will be converted into a throttle output. However the chance of overshoots for example will be larger as there is no feedback from the change in frequency/speed of the motor.

6.5 Braking curve based on frequency

An easy way for smooth braking seems to be determining a braking curve based upon the rotating frequency of the motor or the wingflap frequency. This means that the output of the frequency calculator is used. Next there is calculated how much time is needed go from the current frequency and throttle level to stand still in two wingflaps. This calculation gives a decrease in throttle per time step. The internals of this controller can be seen in figure 27 on page 40. It is very hard to predict the actual frequency at any time for every throttle level, which will make it nearly impossible to make a braking curve only based on the initial frequency and position that stops at exactly the right position. Because of this it will still be necessary to do the last part of the braking with a controller that is based on the position. This last part is the last half rotation. For example the P or PD controller as described earlier with a set-point in π and letting it rotate from 0 to π can be used.

7 Results

The test results of the controllers are discussed in this section. The basic structure of all the controllers is the same and can be seen in figure 11. The sensor read-out, pilot input and output to ESC are basically the same as the ones in section 3 on page 11. The addition here is the controller part. There is the block `Lock_controller`, this is the actual controller with one of the controllers as explained in section 6 on page 20 inside. Next there is the `Lock_switch` this is the block that makes sure there is switched between the direct pilot input for the throttle or that the `Lock_controller` output goes to the `start_up` block. This `start_up` block only has one task; for the first three seconds the controller is started and the ESC is turned on the throttle output to the ESC will be zero. This is necessary otherwise the ESC will not work. After these seconds the `start_up` acts again as a normal connection, thus letting the throttle from `Lock_switch` go directly to the `Limit`.

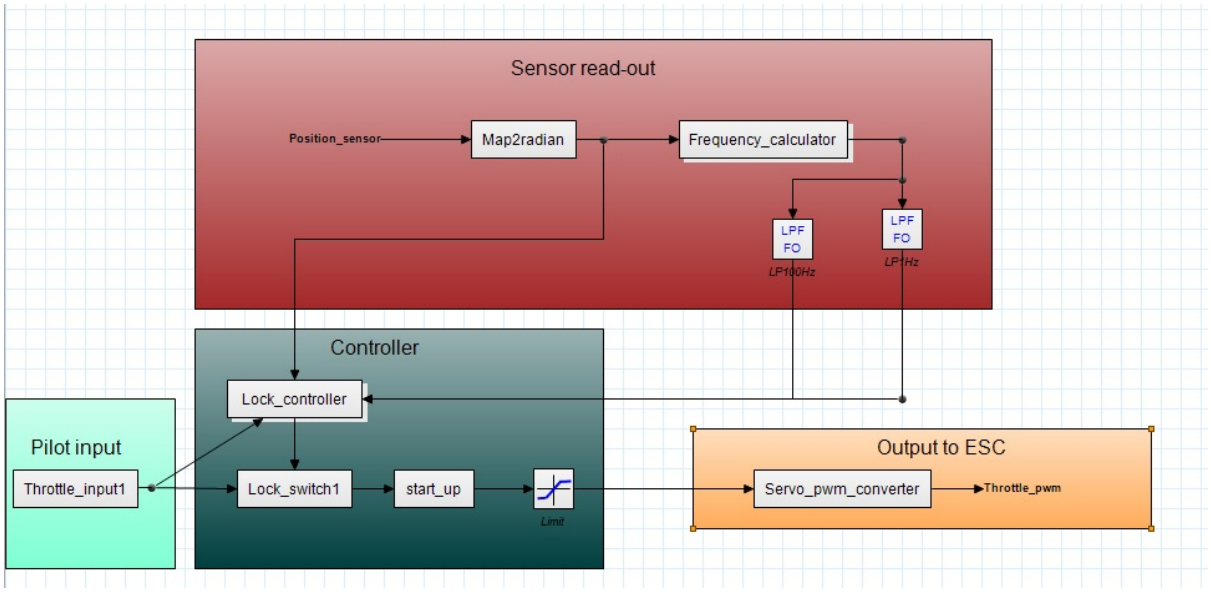

Figure 11: 20-sim model of the controller

7.1 Test in mathematical model

All three controllers (P, PD and braking curve) were tested in the mathematical model before testing them in the physical set-up. Here they all worked; this can be seen in figure 14 on page 29, figure 15 on page 29 and figure 16 on page 30, therefore it was decided to start testing them in the real set-up.

7.2 PD control

As mentioned in the section 6 on page 20 for the PD control to work it is necessary to have a frequency signal that responds fast and is without too much noise. For this a low-pass filter with a cut-off frequency at 50 Hz is used. As can be in figure 12 on the next page this filter reacts too slow and still has too much noise. This means the D action from the PD controller cannot really contribute to the controller. Taking a higher cut-off frequency would let through more noise and taking a lower would make the response even slower, both making that the D action would always not contribute.

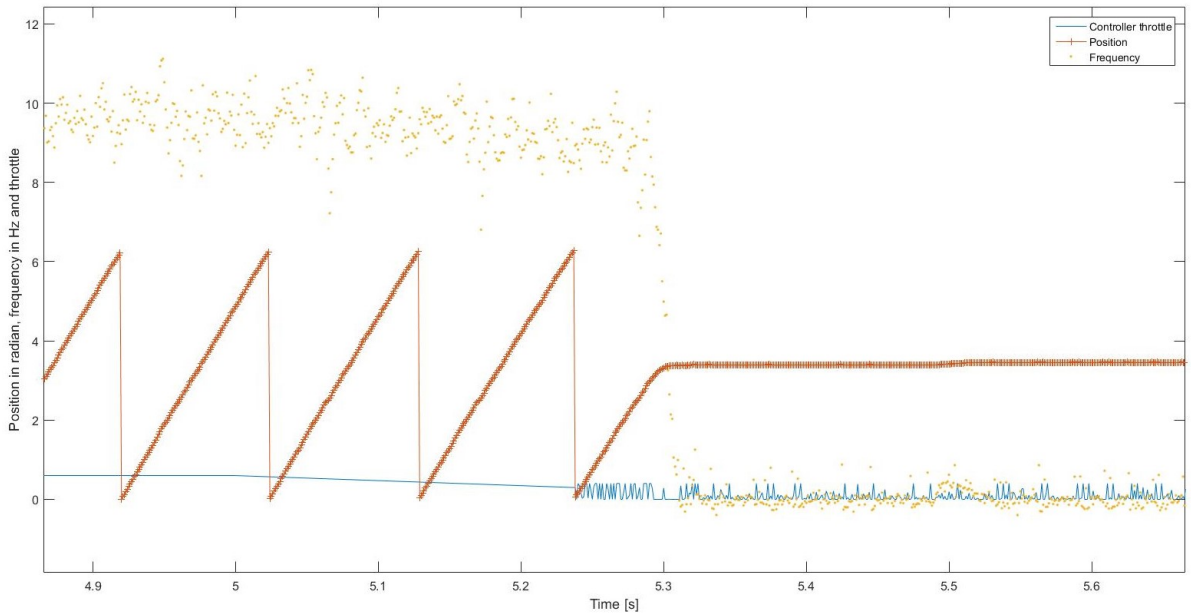

Figure 12: In this graph the position sensor, the sawtooth, the frequency after a low-pass filter of 50 Hz and the throttle level from the controller can be seen. Clearly the frequency signal still has too much noise and reacts too slow. The only thing that the D action in this case will do, as can bee seen in the throttle signal, is add some noise to that signal.

7.3 P control

Good results are achieved using only P control. Some tweaking of the gain needs to be done in the different set-ups because otherwise wings are stopped too early or there is always a too large overshoot. Other than that the P control works good. An example can be seen in figure 13 on the following page. Occasionally it takes an extra wingflap to stop as can be seen in the second attempt to stop. This is mainly caused by the fact that the ESC throttle input refresh rate was 50 Hz which was occasionally not fast enough. With some slight modifications in the Ramstix software and the `Servo_pwm_converter` the ESC refreshrate was altered to 300 Hz.

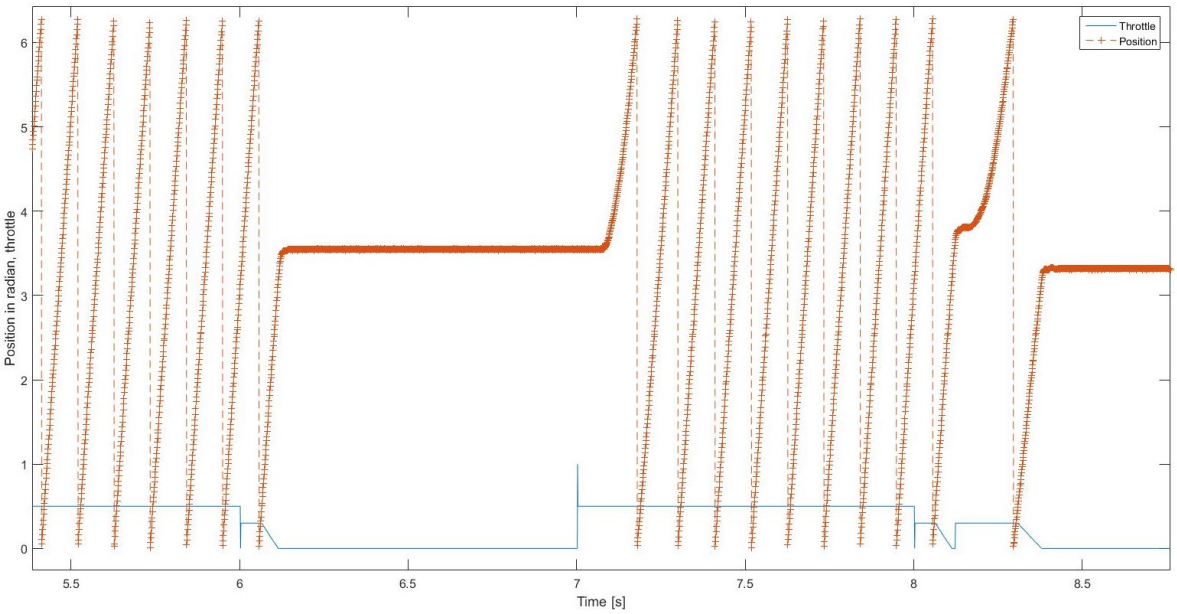

Figure 13: Above a graph can be seen with again the sawtooth being the position of the wings. The line at the bottom is the throttle level as received by the motor, partially generated by the pilot input and partially by the controller. In this picture locking can be seen twice, the first at 8 seconds and the second at 10 seconds.

7.4 Braking curve based on frequency

Also the controller with the braking curve was tested. However not much was added with this controller, apart from the fact that the controller becomes more complicated. A comparison with the P control can be seen infigure 18 on page 31. No clear distinction between the two can be seen, especially not that one has smoother behaviour than the other. A more gradual decrease in speed should be visible in the upper graph in the beginning phase of the braking, starting at 4.2 seconds, this is however not the case. Only the last part of the braking is really relevant and this is in both cases done by a P control, albeit with a different gain in this case. Because of this only extensive testing was done with the P control controller.

7.5 Extensive testing

The 'internals' of the `Lock_controller` based on P control can be seen in figure 26 on page 37. The results after repeatedly testing can be seen in table 1. Here at every frequency 50 or more lockings were tried from several initial throttle levels ranging from 30% till 80%. For the set-up with the wings the throttle level was from 30% till 50% as at higher throttle levels the set-ups starts flying away. The occasional extra wingflap or not locking correctly is completely solved by the higher refresh rate of the throttle input of the ESC as can be seen in the table.

	50 Hz	**300 Hz**	**50 Hz r**	**300 Hz r**	**50 Hz w**	**300 Hz w**
Percentage within requirements	76.7	100	96.7	100	100	100

Table 1: The percentage of locking within the requirements can be seen above. The refresh rate of the throttle input of the ESC can be seen and in which setup it is tested. *r* for the set-up with rubber bands, *w* for the set-up with wings and the first two for the mechanism with only the rods inserted.

8 Implementation in Robird

So far the controller only runs on the Ramstix, which is too large to put inside the Robird. For this reason another embedded system needs to be chosen. A very good option seems an Arduino (or Genuino) Micro[12].

Physical requirements It will fit inside the Robird with only dimensions of 48 mm by 18 mm. The weight of 13 gram should also not be a problem.

Interfacing and hardware The servo control as is done with the Ramstix can also easily be done with the Arduino servo library[13]. With some slight modification this should also be usable to get the input from the Pixhawk. For the input from the position sensor there are multiple 10-bit analog pins available. This is considerably less precise than the 16-bit analog pins of the Ramstix. However with $(3.6-\pi)/(2\pi/2^{10}) \approx 75$ measurable values in the locking area this should be more than enough. For data logging purposes the Arduino can put out the same received signal on one of the analog output pins to the Pixhawk. On the Arduino there is also a power supply of 3.3V which can be used to power the position sensor. The only problem that still needs a solution is the fact that an Arduino needs a power supply voltage between 6 and 20 volt which is not available in the Robird. The Pixhawk can supply 5 volt, which is not enough. Possibly the Arduino can be connected in some way with the LiPo, this however needs to looked into by the technicians at CFS.

Software The software side of the implementation of the controller on an Arduino can be done in several ways. 20-sim can export its models to C-code that is prepared to run on an Arduino, only certain functions it cannot export. This needs to be tested, but if it works implementation is done fairly easy. If it does not work all the submodels can be exported piece by piece in C or written again in the Arduino software that is available on their website[14].

9 Conclusion

To see whether the assignment was successfully completed it can be checked whether all requirements are met. This is the same list of requirements as found insection 2.1 on page 10.

1. The system works for the mechanism, motor, ESC and position sensor of the peregrine falcon model.
2. The size of the Ramstix is still too large to fit inside the Robird, this can however easily be solved by putting the controller on an Arduino.
3. After the 'command of locking' the locking position is always done within two wingflaps, as long as the ESC refresh rate is kept at 300 Hz.
4. Braking is theoretically done smootly, however no real difference can be seen with the eye.
5. Switching between the glide and flapping mode is decided by the glide mode regulator based on the throttle it receives from the pilot.
6. The input is not yet PWM, but this can be done on the Arduino.
7. The output of the system is PWM, such that it can interface with the ESC. The ESC can be controlled, so this is met.
8. Replacing of the sensor was not necessary for this controller and no better alternative has been found.

All in all five of the eight requirements are met. Requirements two and six will be solved when the controller is implemented on an Arduino and the replacing of the sensor has been looked into but not found useful to spend more time on.

10 Discussion

There are a few things that can be improved on the controller. One is some self-learning gain that based on overshoot or not reaching the locking point at all adjusts the gain on the fly. Another thing is a fail safe in case the wings are stopped just before the locking position is reached. Right now this can be solved by letting the pilot give some extra throttle input, however it would be nice if the controller would do that itself. Other than that is has not been proven yet that the controller will actually work in the Robird. Before implementing the controller on another embedded system it will be nice to test the controller in set-up of Erik Landman. Next the controller needs to be put on an Arduino, after which it can again be tested in Eriks set-up. If all test are completed successfully test flights with the real Robird can be done.

References

[1] http://clearflightsolutions.com/applications/, "Applications," 2015.

[2] M. Bomford and P. H. O'Brien, "Sonic deterrents in animal damage control: a review of device tests and effectiveness," *Wildlife Society Bulletin*, pp. 411–422, 1990.

[3] http://wonderopolis.org/wonder/which-bird-flies-the fastest, "Which bird flies the fastest?," 2015.

[4] R. Loon, *Birds: The Inside Story*. 2005.

[5] W. Straatman, "Developing an autopilot for the peregrine falcon robird," Master's thesis, University of Twente, 2014.

[6] https://pixhawk.org/modules/pixhawk, "Pixhawk autopilot," 27-01-2016.

[7] K. J. Ånström and R. M. Murray, *Feedback Systems, An Introduction for Scientists and Engineers*, ch. 7 Output Feedback. Princeton University Press, 2008.

[8] K. Ogata, *Modern Control Engineering*, ch. 10, Control Systems Design in State Space. Prentice Hall, 2010.

[9] AMS, "http://ams.com/eng/products/magnetic-position-sensors/angle-position-on-axis/as5600," 2016.

[10] P. Breedveld, "Integrated modeling of physical systems, dynamic systems part 1 and 2 (reader 216 in the union shop of the university of twente)," 2013.

[11] Tmotor, "http://www.rctigermotor.com/html/2013/professional_0912/44.html," 2016.

[12] Arduino, "https://www.arduino.cc/en/main/arduinoboardmicro," 2016.

[13] Arduino, "http://playground.arduino.cc/componentlib/servo," 2016.

[14] Arduino, "https://www.arduino.cc/en/main/software," 2016.

11 Appendix

11.1 Measurement data

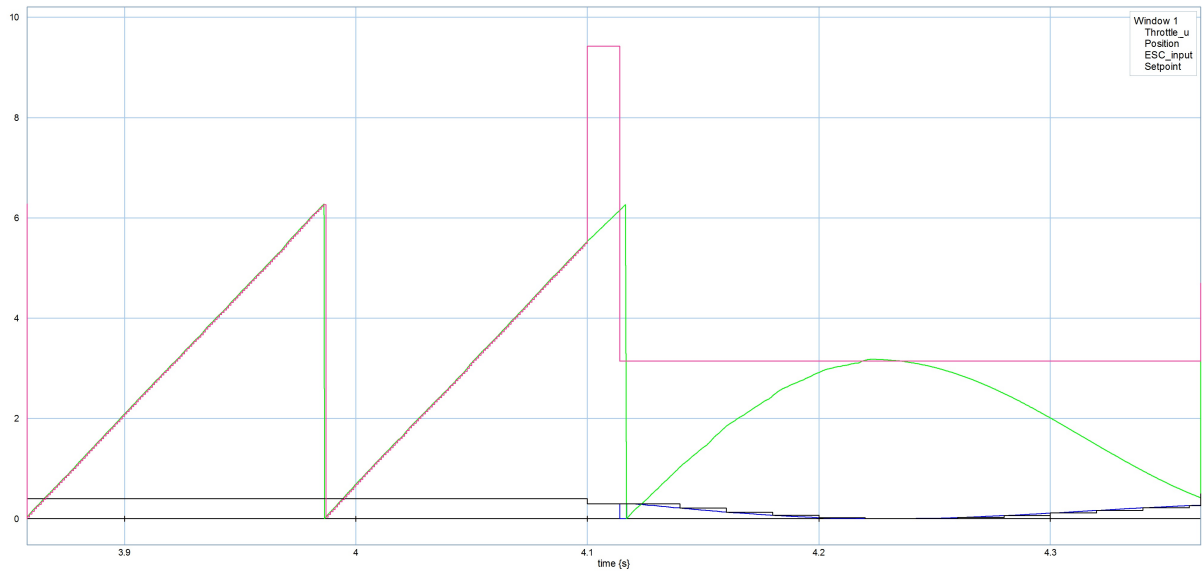

Figure 14: The P control tested in the the model for the rubber bands. At the bottom the throttle of the controller and the throttle sampled with 50 Hz. Next to that again the position and the set-point of the controller is plotted. Starting at 3π and after passing 2π goes to π. The controller stops nicely just above the setpoint.

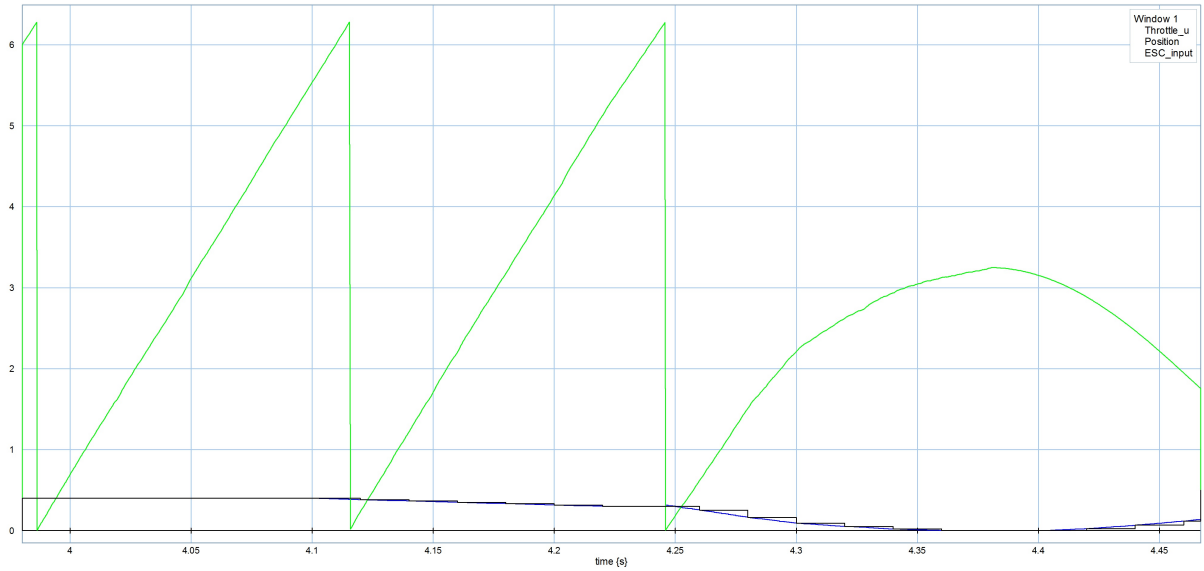

Figure 15: The P control with a braking curve based on the frequency tested in the the model for the rubber bands. At the bottom the throttle of the controller and the throttle sampled with 50 Hz. The controller stops nicely just above the setpoint π.

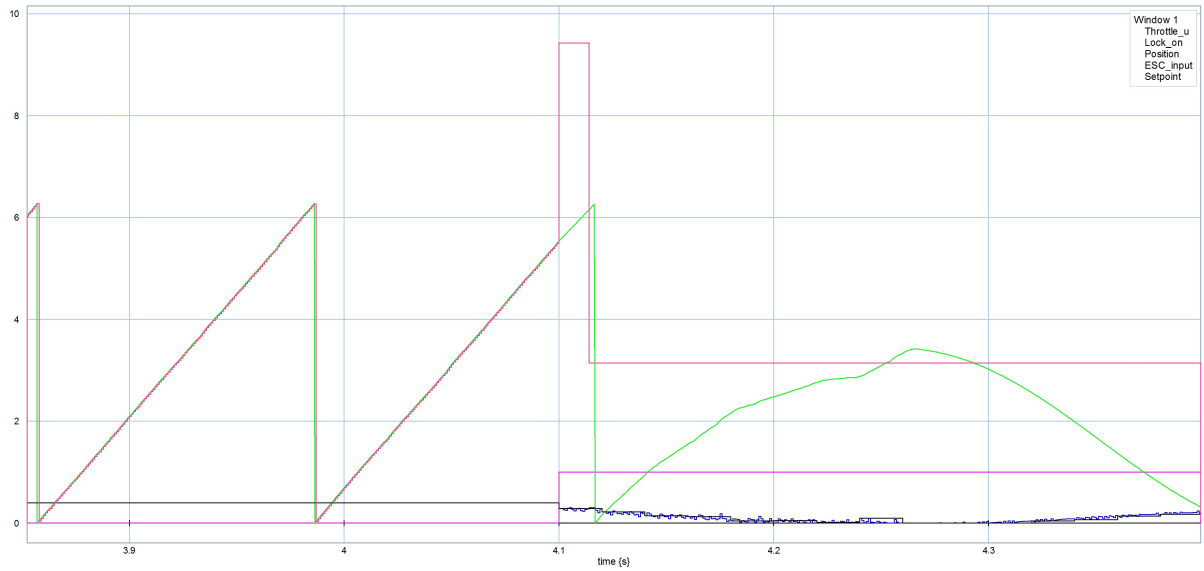

Figure 16: Above the PD control can be seen. The gain of the D part is kept at 0.00005, because otherwise the noise it introduces becomes too large and does not really contribute anything to the control. Even now it can be seen that although the controller still nicely stops at the right position, this is definitely not because of the D control.

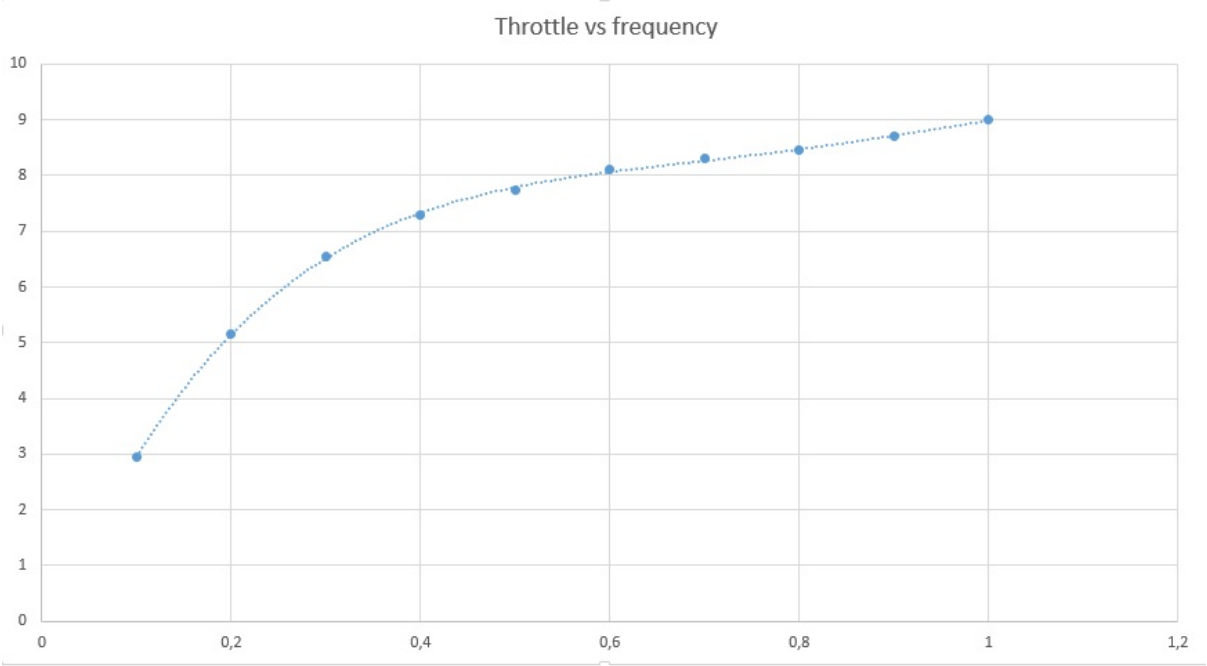

Figure 17: On the horizontal axis the throttle input and on the vertical axis the frequency output of the mechanism without wings

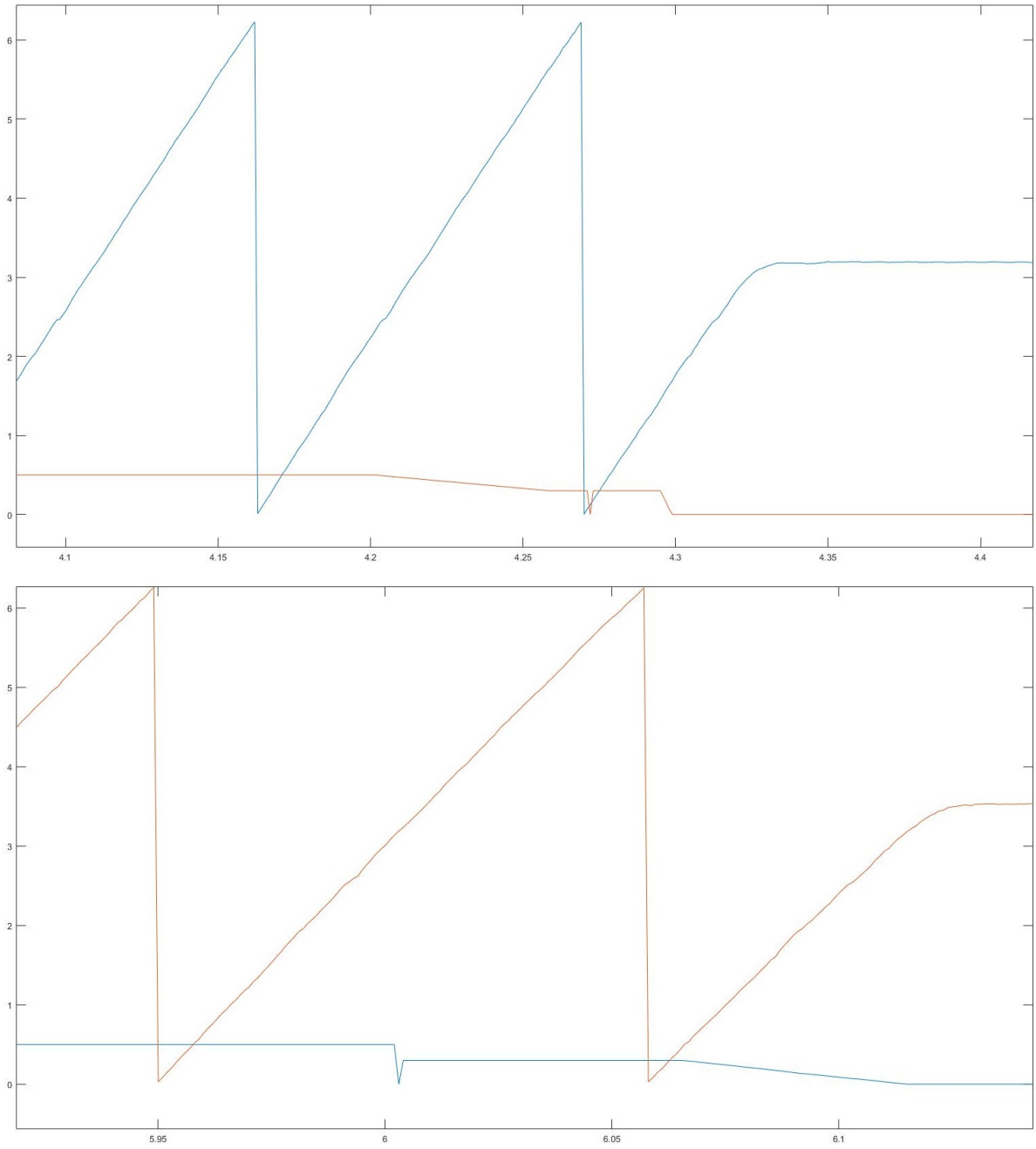

Figure 18: On the horizontal axis the throttle input and on the vertical axis the frequency output of the mechanism without wings. The upper graph is with the smooth braking based on frequency and the lower based on simple P control

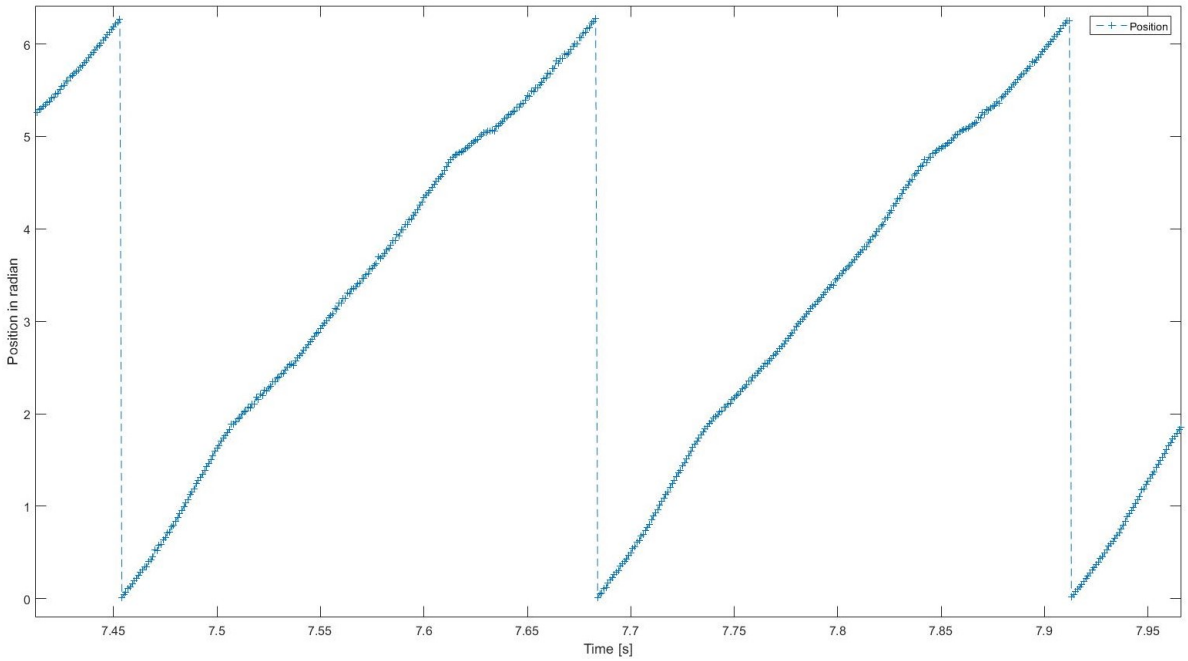

Figure 19: Zoomed in measurement data of the set-up with wings. At π the locking position. The most important reason for the non-linearity seems to be the inertia of the wings at the turning/dead-point. Pi is in the middle between these two points.

11.2 20-sim submodels

11.2.1 Model

Model without wings

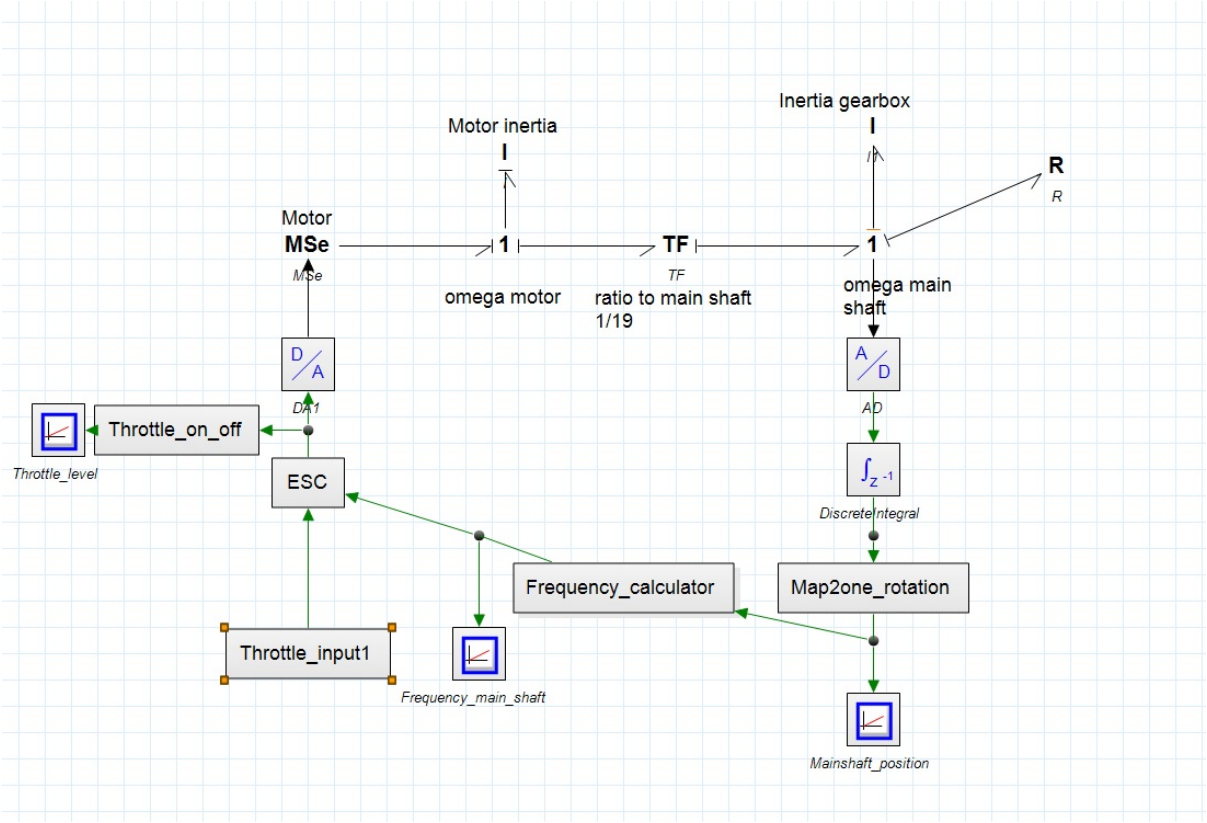

Figure 20: 20-sim model of the set-up without the wings

Model with wings

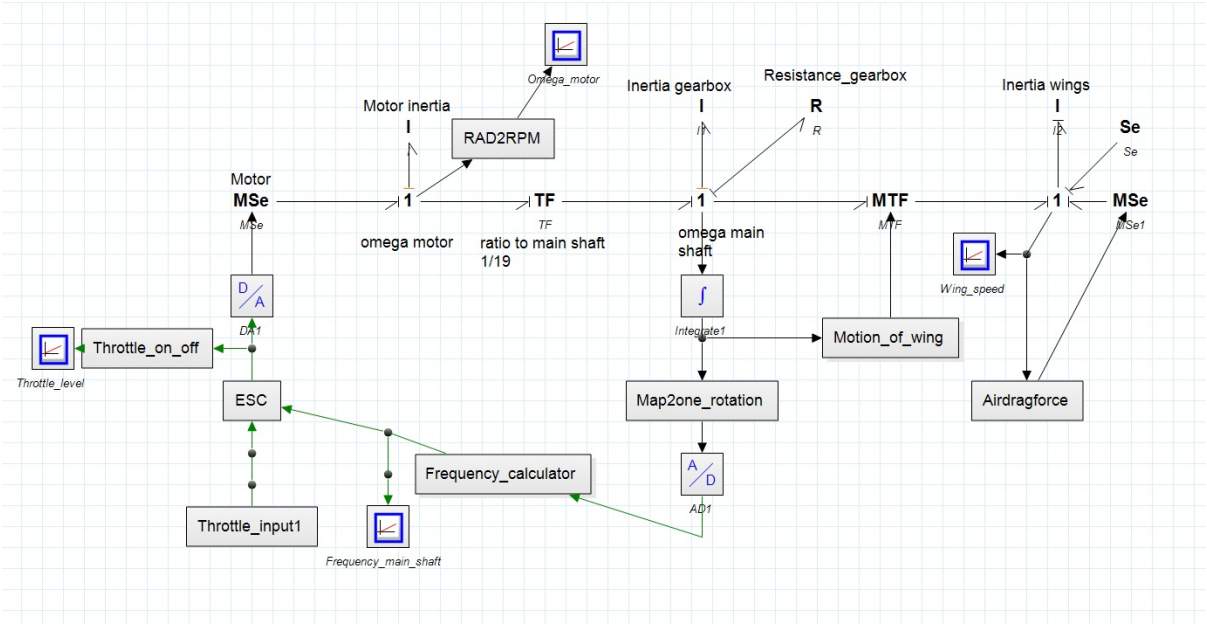

Figure 21: 20-sim model of the set-up with the wings

Motion_of_wing

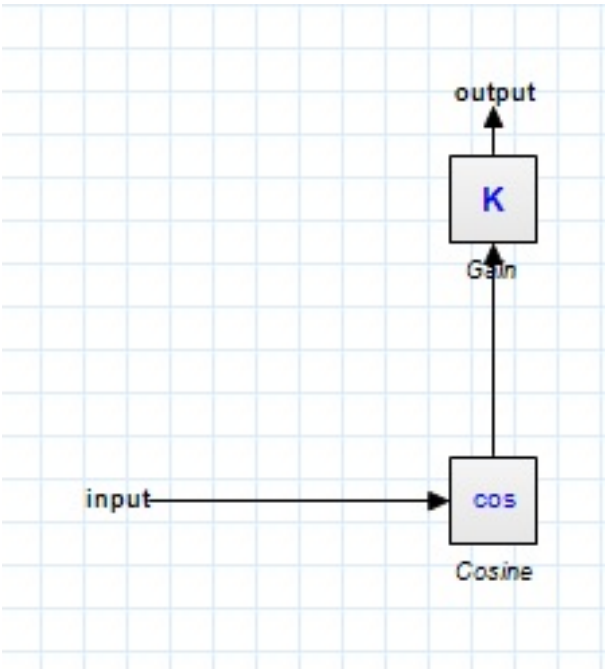

Figure 22: Internals of the `Motion_of_wing` block. A cosine for the motion itself and the gain for the length of the wings.

Airdragforce

This submodel determine the airdragforce based on the input v. $F_d = 1/2\rho v^2 C_d A_{wing}$ is used. Which is the formula for the for a moving plate. Since the wings are not really a moving plate some correcting factor is introduced.

```
constants
```

```
        real RHOair = 1.225;
        real Cd = 1.19;
        real Awing = 0.089999726371063;
equations
  output = 1.5*(-2*0.5*RHOair*v*abs(v)*Cd*Awing);
  //with factor 1.5 to compensate for the fact that it is not a moving plate
```

Map2one_rotation This maps the output from the integrated velocity of the main shaft into one rotation.

```
equations
  output = input-(((input)div(2*pi))*2*pi);
```

ESC

Here `input1` is the measured frequency. `Input` is the throttle input and the formula which contains `input` is equation (1) on page 15.

```
constants
        real P=40; //estimated Power in watts
        real hz=9; //estimated hz for torque calculation

equations
if input1 < -18.221*input^4+60.014*input^3-72.009*input^2+39.529*input then
output = P/(hz*2*pi)/6;// the divide by 6 is just by trail and error
else
if input1> -18.221*input^4+60.014*input^3-72.009*input^2+39.529*input then
        output=-0.08;
        else
                output=0;
        end;

end;
```

Rubberbands

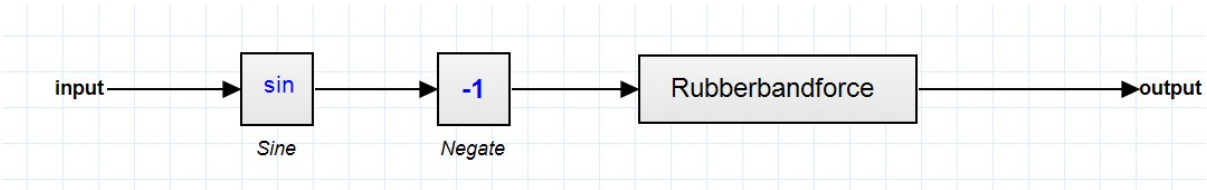

Figure 23: `Rubberbands` block, with a sine, negate block and the `Rubberbandsforce`. The sine and negate block are there to translate the position of the main shaft into some displacement of the wings and thus some extension of the rubber bands.

Rubberbandforce

```
constants
real df=2; // Delta force in newton, between highest and lowest position
real cf=6; // Constant force in newton
```

```
equations
  output = cf+input*df;
```

11.2.2 Final controller

Complete controller

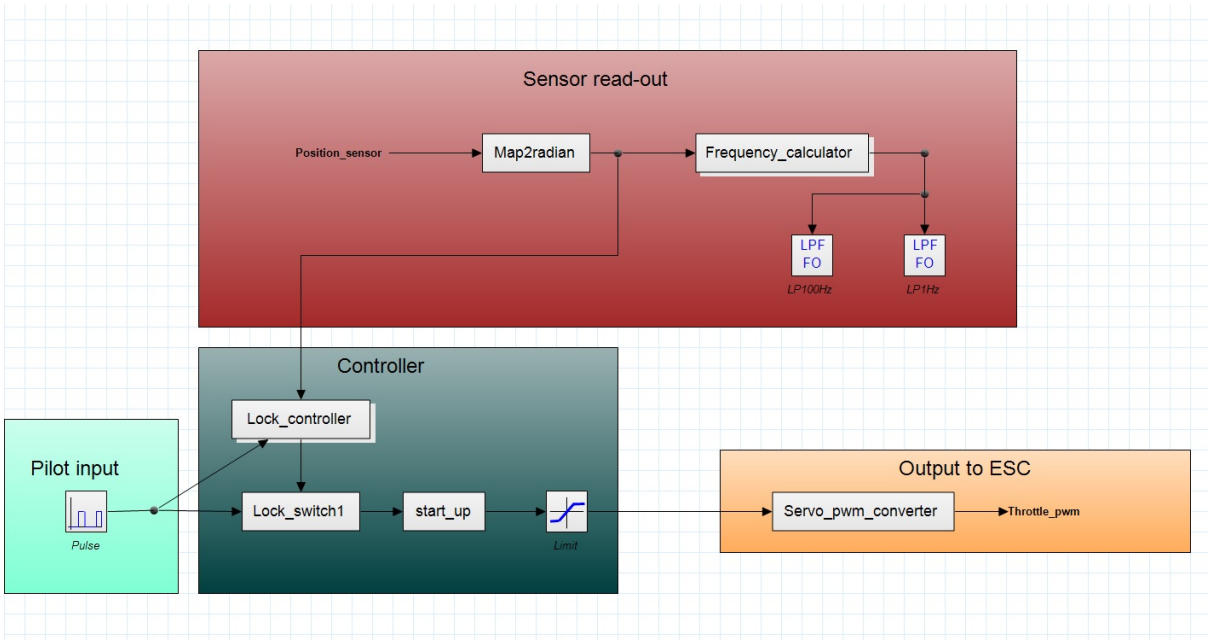

Figure 24: An overview of the complete controller.

Map2Radian

```
constants
        real L = 0.3 ; //output from sensor in locking position
        real Z = 0.0104; //Lowest output of sensor
        real H = 3.26; //Highest output sensor
variables
        real D; //translated to get the locking in pi
equations
  // start typing here
  D = ((input-Z)/(H-Z)*2*pi)+(1/2*(H-Z)-L)/(H-Z)*2*pi;
        if D<0 then
                output=D+2*pi;
                else
                        if D>2*pi then
                                output=D-2*pi;
                        else
                        output=D;
                end;
        end;
```

Frequency_calculator

The sub-model that is used to calculate the frequency of the wingflaps. It is based on a standard rotation matrix.

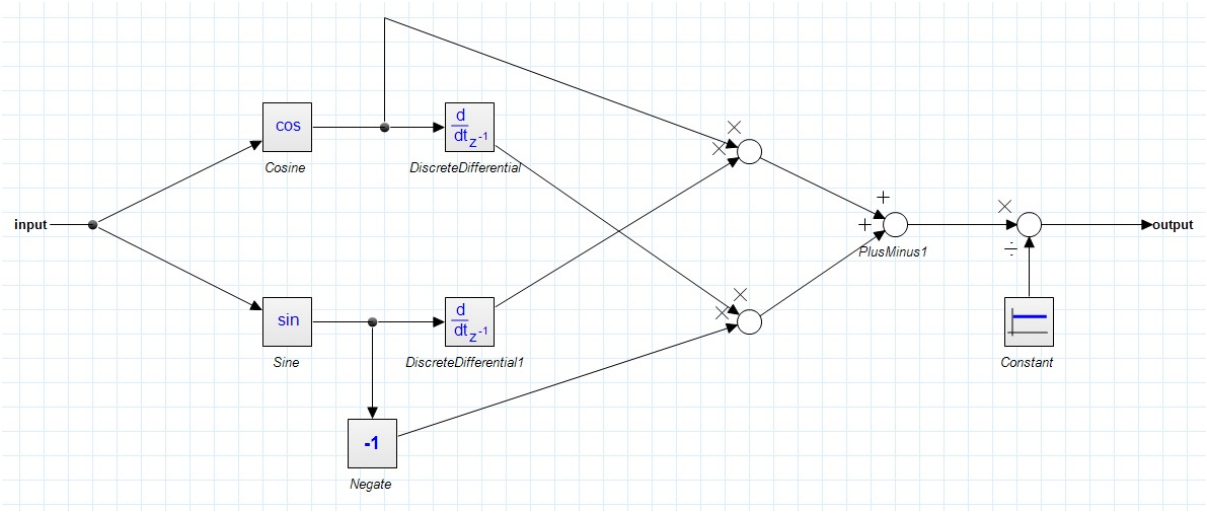

Figure 25: Here the internals of the `Frequency_calculator` can be seen

.

Lock_controller

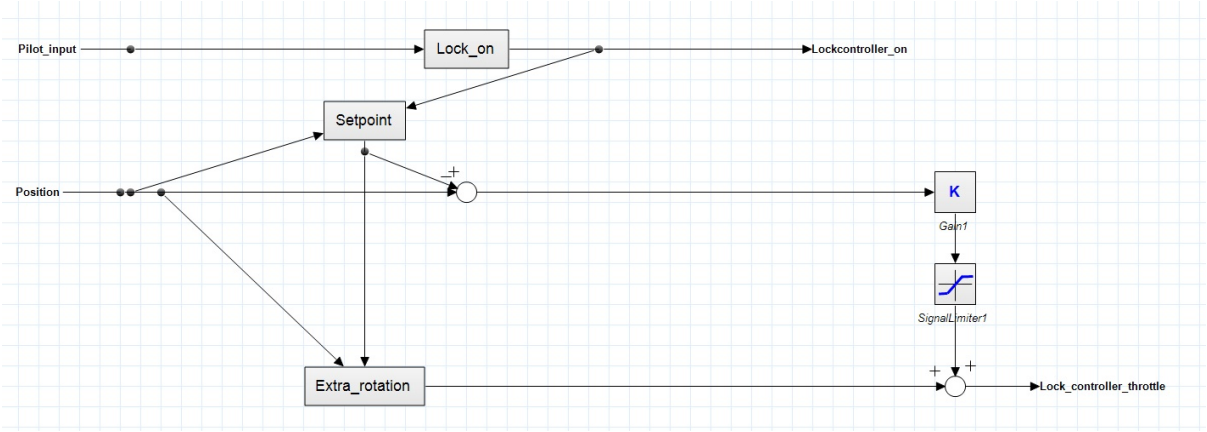

Figure 26: The final `Lock_controller` with simple P control. The gain can be altered to suit every set-up. And the signal limiter limits the throughput from 0 till 0.3.

Lock_on

The part of the control that turns on the rest of the `lock_controller`.

```
equations
if input<0.3 then  //threshold throttle level for the lock mode to be enabled
output = 1;
else
output=0;
end;
```

Setpoint

The `Setpoint` block sets the set-point such that it starts trying to reach 3π which is at most 1.5 rota-

tion away and when 2π is reached the setpoint goes to π which is at that point half a rotation/wingflap away.

```
variables
        real count;
code
if lock==0 then
output=position;
count=0;
else
        if position>1.95*pi then
                count=1;
        else
        count=count;
        end;

        if count==0 then
                output=3*pi;
                else
                        if count==1 then
                                output=pi;
                        else
                                output=pi;
                        end;
                end;
end;
```

Extra_rotation

In case the wings are stopped too late, such that they can not be pulled back into locking, the `Extra_rotation` block makes sure that it rotates further to try to lock again after one rotation. This critical point between being still able to lock and not being able to lock is around a position of 3.6, hence the 3.6 in the code below.

```
equations
if input>3.6 and Setpoint==pi then
        output=0.3;
        else
        output=0;
end;
```

Lock_switch

This block puts through the pilots throttle or the `Lock_controller` throttle.

```
equations
if Lock_controller_on == 1 then
output=Lock_controller;
else
        output=Pilot_input;
end;
```

start_up

This blocks keeps the throttle level at 0 at the start up of the system.

```
equations
 if time<3 then
 output = 0;
else
        output=input;
end;
```

Servo_pwm_converter

This block converts the throttle signal into a signal that can be received by the ESC. The refresh rate of this signal can be altered. In order for this to work the parameter in this code must be altered and the code in the RaMstix.tcf file in the 20-sim 4c, target, RaMstix, etc folder must also be changed.

```
parameters
        real frequency=300; // Refreshrate of the pwm signal
equations
output = input/(1000/frequency)+(1/(1000/frequency));
// pwm 0.02 is equal to 0%, pwm of 0.04 is equal to 100%
```

Ramstix.tcf

The Ramstix.tcf code that needs to be changed can be seen below. Here the 50 in the `ramstixSetPWMFrequency` function must be changed to the desired frequency, which must be the same as the one in the `Servo_pwm_converter`. So in this case it should be changed to 300.

```
<![CDATA[
// Set servo to 50 Hz.
ramstixSetPWMFrequency(ramstix_gpmc_fd, %PORT_CHANNEL%, 50);
ramstixInitPWMOut(ramstix_gpmc_fd, %PORT_CHANNEL%, NONE, NO_BRAKING, NON_INTERLACED);
]]>
```

11.2.3 Controller based on braking curve

Lock_controller with braking curve

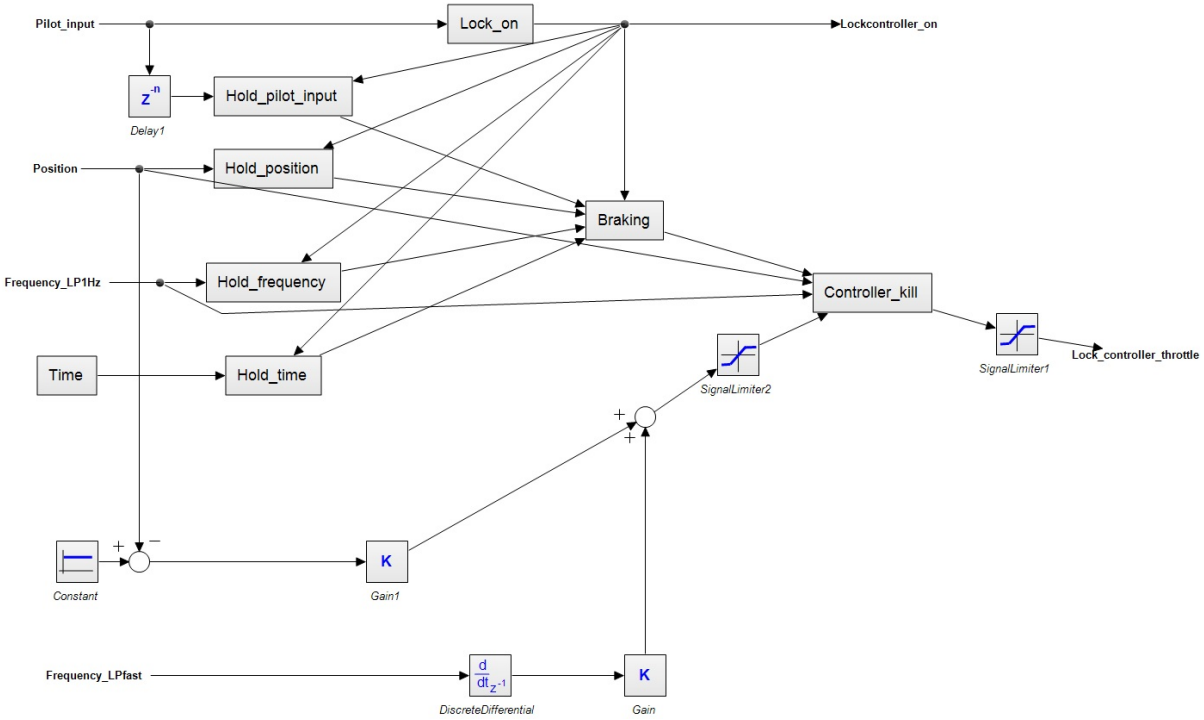

Figure 27: Here the internals of the `Lock_controller` with the braking curve can be seen.

Braking The `Position`, `Frequency` and `Time` are all inputs of this block coming from a `Hold` block. So while the `Lock_on==1` these are constants.

```
variables
real Brake_time;
//Time needed to break assuming linear frequency decrease,
//thus average frequency being 1/2 of the start frequency
equations
if Lock_on==1 then
Brake_time=(3*pi+(Position-pi))/((Frequency+0.001)*pi);
Controller_throttle=Pilot_input-(Pilot_input*((time-Time)/Brake_time));
else
Controller_throttle=Pilot_input;
end;
```

Hold

All the hold blocks have the same code inside. If the controller is switched on the output of this block stays the same.

```
variables
        real old;
equations
  if Lock_on==0 then
        output=input;
        old=input;
        else
                output=old;
                end;
```

Controller_kill This block makes sure that the last part of the braking is not done by the `Braking`, but by the PD controller.

```
equations
if Position>0 and input<0.3 then
//The last half rotation of the notch wheel is reached and the rest will be
//done with the PD controller with setpoint pi, the lock-position.
        if Position<1.2*pi then
        output=PD;
        else
        output=0.3;
        end;
else
        if input<0.3 then
                        output=0.3;
                        else
                        output=input;
                        end;
end;
```

11.2.4 20-sim parameters

The values of the 20-sim parameters as they are used for most simulations.

1. I motor: 5×10^{-6} kg
2. I gearbox: 7.68×10^{-6} kg
3. I rods: 1.875×10^{-4} kg
4. R gearbox: 1×10^{-6} N/m/s
5. I extra inertia: 0 N
6. Gearbox ratio: 1/19
7. Se wings: 0.2 N