COPENHAGEN SCHOOL OF
DESIGN AND TECHNOLOGY

Miroslav Lakota
4th semester IT tech
Final project
1.6.2018

Fixed wing drone stabilization and control system

Miroslav Lakota
4th semester IT tech
Final project
1.6.2018

Fixed wing drone stabilization and control system

Tro- og loveerklæring

Det erklæres herved på tro og love, at undertegnede egenhændigt og selvstændigt har udformet denne opgave. Alle citater i teksten er markeret som sådanne, og opgaven eller væsentlige dele af den har ikke tidligere været og er ikke aktuelt fremlagt i anden bedømmelsesammenhæng.

Undertegnede er gjort bekendt med, at overtrædelse af reglerne om videnskabelig redelighed behandles i henhold til §19 i Bekendtgørelse om prøver og eksamen i erhvervsrettede uddannelser nr. 1016 af 24/08/2010.

Solemn Declaration

I solemnly declare that I have personally and independently created this report. I have clearly marked any and all quotes in the text as such, and neither the report nor any essential parts of it are at present or have previously been submitted for any other examination.

I am aware that any violation of the rules on academic integrity shall be treated in accordance with Article 19 of the Danish Order No 1016 of 24 August 2010 on Tests and Examinations in vocational educations.

_____

Miroslav Lakota 1.6.2018

COPENHAGEN SCHOOL OF
DESIGN AND TECHNOLOGY

Miroslav Lakota
4th semester IT tech
Final project
1.6.2018

Synopsis

This report describes research, development and the process of making a fixed wing drone stabilisation system, which I made as a Final project in the 4th semester of IT technology programme. It is supposed to give a reader an overview of what it takes to write a drone control and stabilisation system, going through a basic physical theories involved in such design.

Miroslav Lakota
4th semester IT tech
Final project
1.6.2018

Table of Contents

Preface

Making a project combining hardware and software together is neither cheap, nor easy to make. It requires material, time, tools, and lots of patience. Therefore I would like to thank the company Danish Aviation Systems ApS which provided me with work space, all hardware parts, my colleagues Riccardo Miccini and Allan Hein for their advices, help with programming and patience while explaining, Steven Friberg for help with construction of the airplane frame and transportation to the test field.

Introduction

Unmanned Aviation has a relatively short history. First unmanned hydrogen filled airships started to appear in the 19th century. During the 2nd World War they found their use mainly as dummy targets for artillery. With improvements to electronics and a decrease in price in 2000s, the number of amateur pilots grew. Flying an airplane was relatively difficult and required hours of practice. A half decade later, manufacturers started to produce small MEMS (Microelectromechanical systems) gyroscopes and IMUs (Inertial measurement units) for first smartphones. Technology came to the point, when hardware equipment is light enough (few grams), cheap and at the same time powerful and accurate enough to estimate the orientation of a flying object. From this point, it was just a small step to first attempts to control airplanes autonomously.

Currently, companies producing commercially available drones e.g. DJI or SenseFly are trying to decrease the level of difficulty and skill needed to fly a drone and at the same time make them as autonomous as possible. Other companies are aiming for commercial usage of drones in transport, monitoring, farming, and many other fields. Electronics in all drones has to have one common function – it has to be able to control the orientation and position of the drone in the air.

And this is the moment when the purpose of this project comes in, and that is to create a system which can stabilise the airplane and allow the user to control its position, using still relatively cheap and accessible components. Of course, this system would not be the first one of its kind. There are more of them and they are using multiple algorithms. During my internship at the company Danish Aviation Systems I prepared a fixed wing drone – flying wing eBee and equipped it with electronics necessary for this system. Honestly, I thought that making one myself will be easy. I was terribly wrong.

Problem formulation and project scope

Making drones autopilot is a difficult and complex task and it cannot be accomplished by a single developer in a bearable amount of time, especially if the developer needs to first study the subject. Therefore, it was not possible to do all the work in the Final project's predefined period. It turned out to be important to define, what work is being done as a part of the Final project and what is excluded from it.

Main goals of this project are:

- Develop the logic hidden behind a control system, find out how the drone shall be controlled
- Write and application (single or multiple threads) controlling the drone in ChibiOS RTOS on STM32F745 microprocessor, including communication in between individual threads
- Test and debug the code in lab and on the field (in the air) on a fixed wing drone
- Make the control software open source and publish it on GitHub

Parts which are excluded from the final project:

- Building the drone, making and installing servos, making and installing a speed controller, preparing electronics with the microprocessor, gyroscope and accelerometer
- Setting up ChibiOS Kernel functions for STM32F745 (basically getting the clean ChibiOS running)
- Writing orientation sensor fusion (mixing data from gyroscopes, accelerometers, magnetometer and outputting the estimated orientation of the object in space)

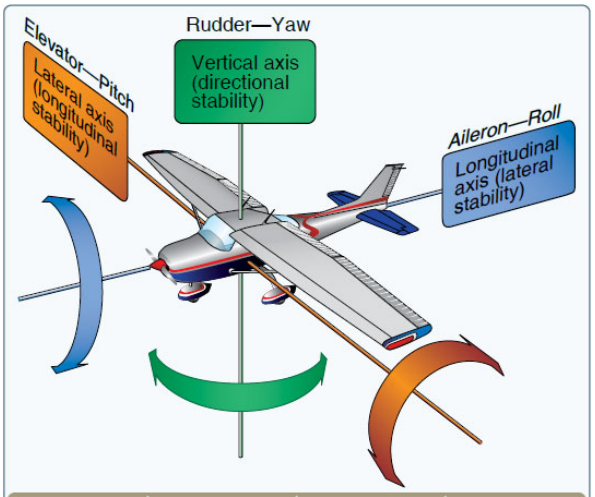
Despite the fact that these parts were excluded from the project, it was necessary for me to understand them and learn how they works. Therefore they are to some extend an inevitable part of this report.

The Project

The theory of airplane control

In order to design a control stabilisation system, it is important to understand how airplanes are controlled. Airplanes these days have many control surfaces . For a purpose of this project, we can take into account only main control surfaces which are:

- Ailerons (controlling Roll)
- Elevator (controlling Pitch)
- Rudder (controlling Yaw)



Primary Control Surface	Airplane Movement	Axes of Rotation	Type of Stability
Aileron	Roll	Longitudinal	Lateral
Elevator/ Stabilator	Pitch	Lateral	Longitudinal
Rudder	Yaw	Vertical	Directional

Figure 1: Airplane's axis of movement.

These surfaces allow us to control the airplane‘s rotation around 3 axes of rotation. As an example – increasing the angle of elevator by 5° can give us pitch angular velocity of 20° per second. In an ideal airplane, in still air at constant speed, the pitch angular velocity would be linearly depend on the elevators position. Unfortunately, this is not the case in real life. The airplane’s reactions to the input depend on many factors including airspeed, orientation, weight, aerodynamic design, pressure and temperature of the air, current position of the control surface and movement of the air around the airplane. In the end, under some circumstances, ailerons or rudder can have completely reversed control. The fact that every airplane has a different behavior makes this even more complicated.

Provided that we have an accurate numerical model of the airplane and sufficient computing power, we can calculate and estimate majority of these effects. Unfortunately, a microcontroller with a sufficient computing power has not been developed yet. Luckily, in order to successfully control the airplane, we don‘t necessarily need to take into account all non-linearities and we can still make the drone fly well enough.

In industrial applications, engineers have been using PID controllers since 1920s for controlling various controlling various processes – controlling speed, position, rotation. Now, almost 100 years later, we are still using these controllers in our microprocessors because of their simplicity. I would like to use a slightly modified version of the PID controller also in this project.

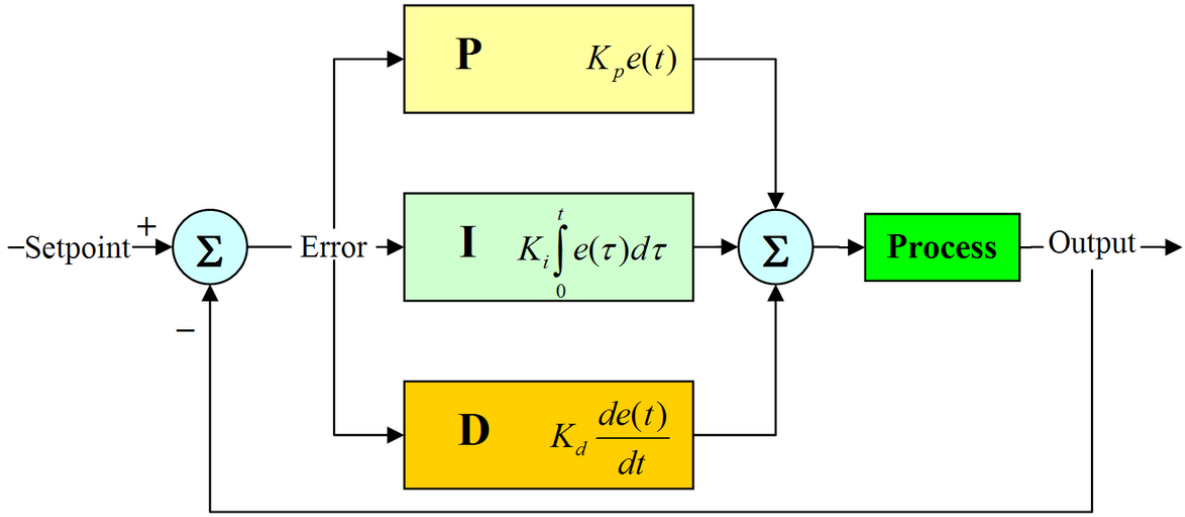


Figure 2: A typical PID controller.

Hardware build

The airplane used for this project is a modified eBee from a company senseFly. This dron – flying wing has been chosen because of it's great flying characteristics. Ebee has been designed from the first step to be used easy to fly and to be controlled by an autopilot.

Flying wings are slightly different from normal airplanes, they do not have a tail, where the elevator and rudder are usually placed on, therefore their ailerons have a mixed function of an Elevator and ailerons. Rudder can be partially replaced by air brakes located on each side of the wing, however that is not the case here and the flying wing is lacking yaw control.

Figure 3: Modified eBee from senseFly.

Majority of the electronics components supplied by the original manufacturer has been removed and replaced. Motor stayed as the only original part. The ESC (Electronic Speed Controller) was replaced by an ESC developed in the company (HW design made by me in a different project). Both servos were replaced by ultra fast UART controlled servos with an update rate over 1khz (HW and SW made by me).

Instead of the senseFly autopilot, I used a board which has among other sensors:

- an MCU STM32F745 (the main computing unit)
- a gyroscope and an accelerometer BMI160 (orientation fusion)

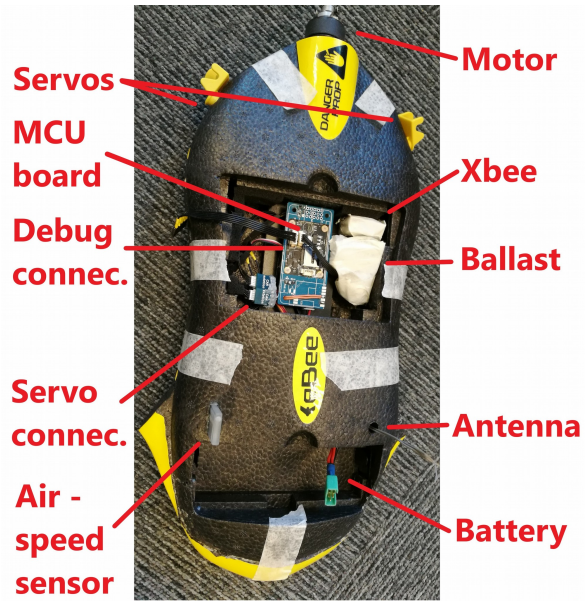


Figure 4: Components from the top side.

- a magnetometer BMI150 (orientation fusion)
- a barometer BME280 (for measuring altitude)
- an SDcard slot (for logging data)
- a voltage divider (for measuring battery voltage)

There are 3 more devices inside:

- Xbee with a protocol 802.15.4 (for debugging in the air)
- an analogue differential pressure sensor with a pitot tube (for measuring the air speed)
- Futaba receiver (for receiving from the pilot)

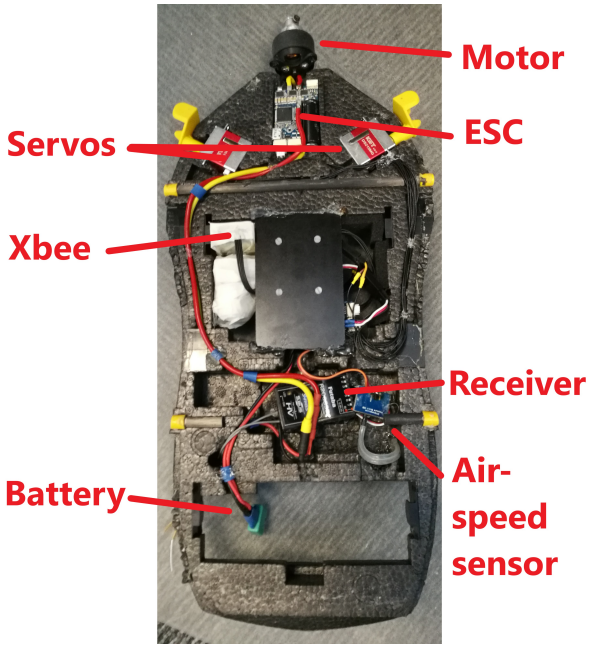


Figure 5: Components inside of eBee.

All the internal parts of the drone were modified in order to get an easy access to electronics. I have not experienced any major issues or problems with the hardware equipment. Few issues which showed up include an unstable offset on the output of the analogue air speed pressure sensor and servos getting hot because of a bad cooling inside of the airplane and a high speed PID loop. In general, very satisfied with the hardware.

ChibiOS RTOS

ChibiOS RTOS is an open source Real Time Operating System made by Giovanni Di Sirio. ChibiOS is well documented system and one of it's biggest advantages is that it allows the user to run multiple threads on a microprocessor with a predictable timing and a predictable result. This a quite important feature when you are designing an autopilot or any other control system which has to have a low latency. In a drone, there is a need to run many control loops (almost) at the same time – estimate orientation, read data from the receiver, the speed sensor, gyroscope, accelerometer, safely process the data and output them to the servos or a motor. An RTOS with a high power MCU can take care of this task.

My MCU board already had a premade project template with ChibiOS 17.6. It was also already running multiple threads and therefore my first step was to clean them up. I had to learn how ChibiOS works, how the scheduler switches in between threads, how to set up peripherals and of course how to create new threads.

A typical RTOS can run multiple threads „virtually" at the same time. In case of our application, it's more than 5 threads. The MCU is switching in between these threads at a high frequency, creating an illusion that they are running simultaneously. Threads need to be able to communicate together in a safe way. Unfortunately, it's not possible to do it in an easy way – just reading them from a piece of memory, because they can get corrupted. A ChibiOS introduces multiple ways of communicating in between threads. I spent some time studying these options, in order to determine which one is the best for my application.

- RT Synchronous Messages
- RT Mailboxes
- Mutexes
- Semaphores

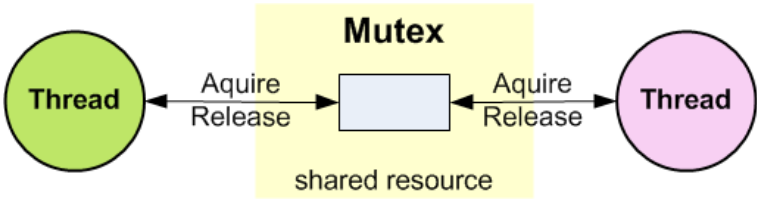


Figure 6: Mutex can be aquired only by a single thread at a time

All of them work in a slightly different way. I decided to use Mutexes in my project, mainly because there was already a very nice implementation of them in the template made by my former colleagues Erik Bærentsen and Allan Hein. This saved me some work, but on the other hand I had to understand the way their code works.

Control algorithm

One of the most interesting part of this project has definitely been studying and designing the control algorithm. On the other, it has also been the most difficult one. The goal from the beginning was to make a software which could in the future support an autopilot. A huge help was to study code from iNav stabilisation board. This was definitely a huge inspiration.

Each block has it‘s own function:

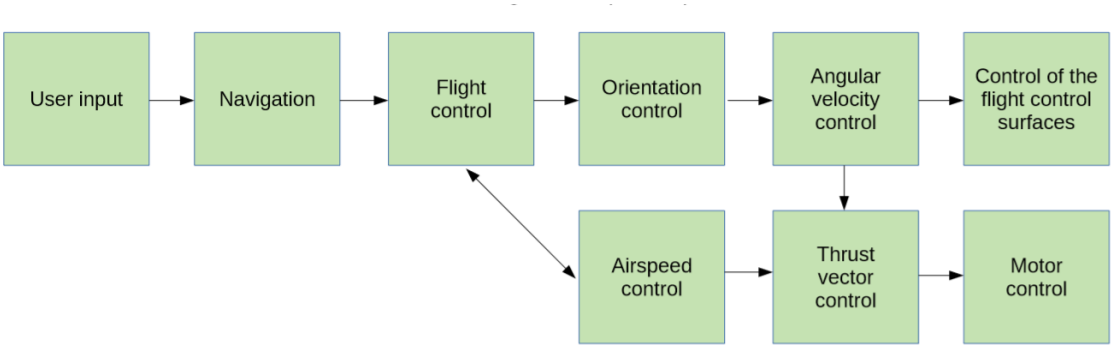


Figure 7: Design of a simple Autopilot

- User input: user decides where and how to flight, e.g. he sets up waypoints in the system
- Navigation: navigation checks the position of the airplane and decides where and how to fly
- Flight control: determines what speed and orientation shall the airplane be flying at, takes care of error conditions e.g. in case the engine dies and the Air Speed control fails, prevents the airplane from stalling, and also limits the airplane from going over its construction limits – flying at too high load or upside down
- Orientation control: runs a quaternion based PIV loop which compares current orientation of the airplane determined by sensor fusion with desired orientation given by a Flight control loop
- Angular velocity control: runs a PIFF (ff stands for feed – forward) loop for keeping the angular velocity at the value given by orientation control

- Airspeed control: runs a PI loop for keeping velocity at the value given by the Flight controller and outputs the value into a motor, in case it hits it's limits, informs the Flight control loop about the error condition
- Thrust vector control: if the airplane has 2 or more motors, it might be helpful to change it's speed on the fly to help the airplane turn (or change direction)
- Control of the flight control surfaces: contains all systems necessary for controlling the control surfaces (ailerons, elevator, rudder), e.g. servo motor driver
- Motor control: contains all systems necessary for controlling motors (e.g. sending data to an ESC)

Implementing this number of features would be too much for a single person. Therefore, for the purpose of this project, I simplified the design, removed the navigation part (the pilot determines what angle the airplane shall be flying at) and merged Servo and Motor control into single block.

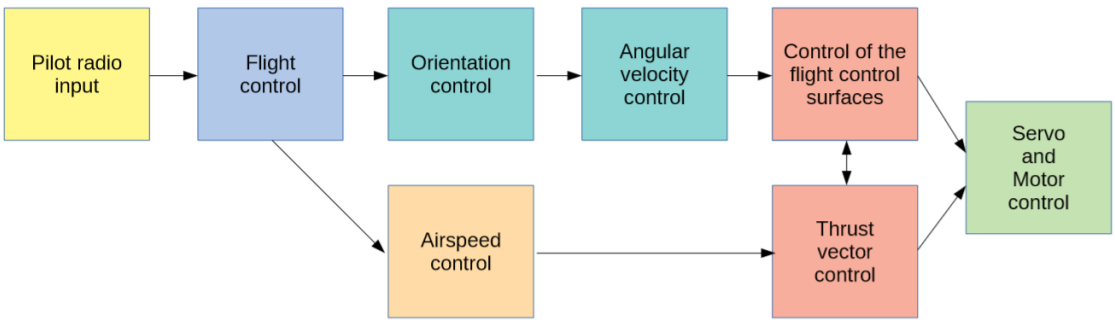


Figure 8: Fixed wing drone control system. Different colours represent different threads.

As you can see from the description, the design is using 3 types of control loops:

- a PIV loop for controlling the orientation
- a PIFF loop for controlling the angular velocity
- a PI loop for controlling the Air Speed

Writing software

Writing the software for autopilot has been a challenge due to multiple reasons.

The first one was that my knowledge of ChibiOS has been pretty limited in the beginning and I had to learn to use all the new functions. The second one was, that I was writing the whole project in C++, which I did not have enough experience with before. Before I could start writing new code I had to understand how the already written code works, which was difficult without proper documentation. In the beginning I was writing the code without being able to properly debug it, because I did not use the official ChibiOS studio (based on Eclipse) and instead worked with makefiles as my other colleagues. Since I started using the studio, debugging became much more efficient.

Each thread is basically a class which inherits from the „BaseStaticThread“ class. Threads have usually only one object, which is initialized in a separate class called „App wrapper“.

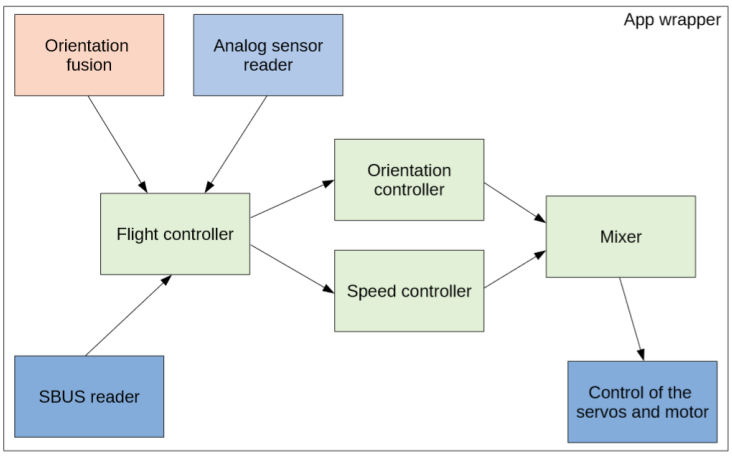


Figure 9: Threads and data flows among them.

The „App wrapper“ contains 2 namespaces. The namespace called „threads“ contains initialization of the threads‘s objects. The second namespace called „shared data“ contains initialization of data types‘s objects which are shared among threads.

Figure 9 shows data flows among different threads. The colour of the thread determines whether it has been made as a part of this project. Green colour means made (by me) as a part of this project. The blue colour means made (by me) before this project, however finished during the project period. The red colour means made by somebody else. Compared to Figure 8, few functions have been fused together into a single thread.

- The orientation controller takes care of both, orientation and angular velocity control.
- The mixer determines how shall each control surface and motor be controlled.

- The flight controller reads data from SBUS reader from the receiver and directly translates them into desired position.

For the purposes of this project I wrote my own C++ based PID library, which should in reality be called „PIVFF“ library, since it uses 4 types on controllers inside:

- P – proportional controller
- I – integral controllers
- V – velocity based controller
- FF – feed forward controller

One of the specialties of the library is that it has an input for attenuation which allows to change the sensitivity of the loop based on the speed of the airplane. It also allows the user to change parameters on the fly, and also to reset internal parameters or completely disable the whole loop without having any issues after reinitialisation. The library uses floating point numbers. The main reason for choosing floats over other data types is the comfortability of usage – it is very easy to rescale data without being worried about resolution. STM32F745 has a hardware floating point unit which can multiply floating point numbers in 3 clock cycles, and divide them/find the square root in 14 clock cycles, Therefore the library is running very fast on this hardware.

While writing, I was struggling mainly with writing code for a Quaternion based PIV controller for the Orientation control. There are very few sources and almost no examples online for this kind of a controller. I spent over 2 weeks working on this controller, spending most of the time by studying how to use quaternions, how to multiply and rotate them. In the end, I did understand how it can be done and even found a usable example. However, I would still need more time for testing and implementation of this feature.

The software has been published on my Github and can be found in a repository:

https://github.com/lakotamm/FixedWingStabilization/

Testing and debugging on the airfield

The airplane has been flying 4 times until now. The first time without any stabilisation software, purely in „pass through mode“ when I was controlling the airplane completely manually. This test was important for determining the right center of gravity, finding out the correct zero positions of the flight control surfaces setting up their maximum position. It also gave me a good idea of how the airplane behaves and how to „save it“ in case something goes wrong and I have take over the control.

Figure 10: Flying eBee. Looks like a normal airplane.

The next 3 test flights were made with a basic implementation of an angular velocity control with a PID loop. The test results were improving with each flight from „uncontrollable“ through „oscillating“ to „flyable“. During these tests I was regularly fixing the code in between flights. After these test flights, the structure of the whole code has been redesigned and brought to the current form. The PID controller has been replaced by – hopefully – a much better PIFF controller. The airplane has received a function which attenuates or increases the gain of the PIFF controller based on the air speed/pressure from the pitot tube. Future tests will show whether the current controller is better.

Next tests will be carried out with debugging on the fly. I plan to send data from the airplane through Xbee transciever and plot them using a Python plotter made by my former colleague Erik Bærentsen this should allow me to properly tune all the controller loops.

Despite the fact that the airplane has been flying many times with code which was not been tuned, finished and it had bugs, the airplane has never crashed. Here I have to say that it‘s most probably because of luck – once it was very close to falling down and my flying experience (being able to „save the airplane“ helps).

A video from one of the test flights can be seen on my Github, in project documentation folder. You can see there oscilations from the overtuned PID controller:
https://github.com/lakotamm/FixedWingStabilization/blob/master/project%20documentation/Video-%20low%20pass%20with%20oscilations.mp4

Conclusion

It was definitely interesting to study how to make a controller, how to design an autopilot and to compare different implementations made by different people. I spent tens of hour reading iNav Flight code, trying to understand what were those „highly skilled“ programmers trying to implement. In the end, after days of studying I also understood why Quaternion based controllers aren‘t implemented in all aplications. I also do understand that I was aiming high with this goal. It is definitely possible to implement Quaternion based controller on STM32F7. And it is also possible to learn to use quaternions to such extend, however, it requires a lot of time, which was more than we had for this project. I would like to continue implementing the controller, however, the next step will be to go, test and tune the airplane before the exam, and finally enjoy flying with it.

Despite the fact that the orientation control is not functional, I think I managed to fullfill majority of the goals. I designed the structure of an autopilot and also a fixed wing drone stabilisation system. I made a ChibiOS application for and STM32F745 consisting of multiple threads which are communicating together. I tested the written code on the airplane and flew it. The code was published on GitHub and I hope it might help somebody with writing his own – Fixed wing drone stabilization system.

I have to say that I really liked working on this project. It was a great choice. If I could, I would go into it again.

List of references

[1] Airplane‘s axis of movement.

https://www.flightliteracy.com/wp-content/uploads/2017/11/6-4.jpg (visited 1.6. 2018)

[2] A typical PID controller.

https://upload.wikimedia.org/wikipedia/commons/thumb/4/40/Pid-feedback-nct-int-correct.png/ (visited 1.6. 2018)

[6] Mutex.

https://www.keil.com/pack/doc/CMSIS/RTOS2/html/Mutex.png (visited 1.6. 2018)

Bibliography

Miroslav Lakota (1.6.2018) Fixed Wing Stabilization. Retrieved from https://github.com/lakotamm/FixedWingStabilization (visited 1.6.2018)

Erik Bærentsen (16.3.2017) Acquiring Accurate State Estimates For Use In Autonomous Flight. Retrieved from https://discoverycenter.nbi.ku.dk/teaching/thesis_page/Erik-MasterThesis_FinalAsHandedIn.pdf/ (visited 31.5.2018)

Giovanni Di Sirio (2006 - 2017) ChibiOS documentation. Retrieved from http://www.chibios.org/dokuwiki/doku.php (visited 31.5. 2018)

iNavFlight. INAV - navigation capable flight controller. Retrieved from https://github.com/iNavFlight/inav (visited 31.5. 2018)

John W. Eaton (1998-2018). GNU Octave. Retrieved from https://www.gnu.org/software/octave/ (visited 31.5. 2018)

Paweł Spychalski (2.2.2017). INAV 1.6: Fixed Wing PIFF Controller Retrieved from https://quadmeup.com/inav-1-6-fixed-wing-piff-controller/ (visited 31.5. 2018)

Quanser Inc. (8.29.2017) Controller. Retrieved from http://quanser-update.azurewebsites.net/quarc/documentation/controller_block.html (visited 31.5. 2018)

flightliteracy.com (2018-2018) Flight Control Systems – Primary Flight Controls (Part One). Retrieved from https://www.flightliteracy.com/flight-control-systems-primary-flight-controls-part-one/ (visited 1.6. 2018)