CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Measurement

DIPLOMA THESIS

Flight Control System Unit for Small UAV Aircraft

Jaroslav Halgašík

Supervisor: Ing. Martin Hromčík, Ph.D.

Praha, 2014

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2013-2014**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	**Bc. Jaroslav Halgašík**
Studijní program:	**Kybernetika a robotika**
Obor:	**Letecké a kosmické systémy**
Název tématu česky:	**Návrh a realizace elektronického řídicího systému pro bezpilotní prostředek**
Název tématu anglicky:	**Flight Control System Unit for Small UAV Aircraft**

Pokyny pro vypracování:

Navrhněte a realizujte řídicí jednotku pro malý bezpilotní prostředek, využitelný ve výzkumných a výukových projektech na katedře řídicí techniky.

1. Proveďte rešerši dostupných řídicích systémů pro malé bezpilotní prostředky.
2. Seznamte se s možnostmi bezdrátové komunikace vhodné pro použití s bezpilotním prostředkem.
3. Na základě získaných poznatků navrhněte a realizujte hardwarovou platformu řídicího systému.
4. Seznamte se s klasickou hierarchickou strukturou řídicích algoritmů pro stabilizaci a řízení letadla.
5. Implementujte potřebné řídicí smyčky na vytvořeném hardwaru.
6. Implementujte uživatelské rozhraní pro obsluhu bezpilotního prostředku.
7. Proveďte experimenty, které ověří funkci a vlastnosti řídicích algoritmů v reálném prostředí.

Seznam odborné literatury:

[1] Nelson, R. C.: Aircraft stability and automatic control. McGraw Hill, 1998

Vedoucí diplomové práce:	Ing. Martin Hromčík, Ph.D. (K13135)
Datum zadání diplomové práce:	24. září 2013
Platnost zadání do[1]:	23. ledna 2015

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
4

[illegible]
Prof. Ing. Vladimír Haasz, CSc.
vedoucí katedry

v. z. [illegible]
Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 24. 9. 2013

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

Acknowledgements

First and foremost, I would like to thank to Doc. Ing. Martin Hromčík, Ph.D., supervisor of this project for the valuable guidance and advice.

Besides, I wish to thank my parents for their support and encouragement throughout my study.

Abstract

This thesis describes design, development, tuning and verification of a low-cost control unit for a small unmanned aerial vehicle. This control unit can be used for control of fixed wing aircrafts as well as helicopters, multi-rotor vehicles or airships or other type of vehicles. Such vehicles can be built using RC models kits and parts, or developed from scratch using available parts like drives, controllers, servos. Hardware configurations of the on-board unit as well as of the ground station are presented in detail, and related software components are described and elaborated. Flight experiments that were executed during the development stages are presented to show readiness of the presented solution for intended applications.

Abstrakt

Cílem této práce je navrhnout a vytvořit elektronický řídící systém pro malý bezpilotní prostředek. Vytvořený systém je nezávislý na použitém prostředku a jeho typu, lze ho využít pro autonomní řízení letadel, vrtulníků, vzducholodí nebo jiných prostředků vytvořených na základě běžně dostupných modelářských komponent. Náplní této práce je detailní popis vytvořené řídící jednotky zahrnující návrh hardwaru i softwaru. Letové experimenty provedené v rámci vývoje byly provedeny pro ověření celkové funkčnosti systému a jejich popis a výsledky jsou součástí této práce.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Development of unmanned aerial vehicles (UAVs) has started at the beginning of the 19th century when the most of these original designs was meant to be teleoperated weapon or flying bomb. These days, the purpose of the unmanned vehicles is much broader and it is not strictly connected to military purposes anymore. A technological progress in last decades brings new opportunities and makes these flying vehicles more available for civilian use. These days, UAVs are flying on daily basis, patrolling the borders, monitoring agriculture, taking movies or pictures for TV documents, or serves for children as toys with augmented reality. An example of commercial UAV used by an army is Raven RQ-11, see Fig. 1.1. All these possibilities open a huge spectrum of improvements that can be done in UAVs. Therefore, a goal of this diploma project is *to develop a hardware control unit for this kind of devices, in order to provide an platform for testing and development of advanced control algorithms for UAVs.*

Figure 1.1: RQ - 11 Raven - Example of UAV used in an army [7]

Motivation for this project comes from research and educational activities at the Department of Control Engineering, FEE CTU, focused on flight dynamics, flight control systems, a design of control laws for aerospace applications and formation flying control algorithms. It is expected that not only the theoretical research made at the department, but also related university's courses and diploma masters' projects can strongly benefit from experimental verification possibilities.

Therefore, it must be ensured that the selected solution offers broad flexibility, tweakability and openness, which is important for expected necessary modifications of the control system as new challenges might appear during all phases of a particular project. For example, it might happen that requirements on flight performance and flight control system (FCS) functionalities are updated, new or modified communication devices are incorporated, specification/refinement of communication protocols for high-level agent-based formation and mission controls is changed. These aspects naturally call for a complete custom-built solution of a flight control unit. Therefore, this diploma project is aimed at development, realization and testing of a low-cost lightweight on-board hardware platform for RC-size aircraft, equipped with standard sensors and actuator drivers sets, microchip computer with dedicated software routines, and related ground station.

The prototype has been tested during flight experiments with a delta-wing foam A/C model and other platforms, and it performed very well and flawlessly. One of the RC model used for these flight experiments can be seen on Fig. 1.2.

Figure 1.2: CESSNA 182 - RC aircraft, one of the platforms used for flight experiments

The intended applications also determine general specifications of the developed control unit: small size, low cost, modularity, standard output interface for RC components, defined input interface for connecting to master system, and fully open and tweakable software solutions.

Despite the fact that similar systems exist and are commercially available (see Section 2 for a review), a decision was made to develop our own flight control unit from scratch due to following reasons. First, the idea of fully open and self- developed system was appealing as indicated in the paragraphs above. Second, the whole project can be seen as natural way to master and develop the principles learnt during the studies of flight control systems fundamentals.

1.1 Overview

This thesis is organised as follows. First, Chapter 2 brings a review of commercially available flight control systems. Then, Chapter 3 describes the theoretical background for controllers, measuring and navigation and Chapter 4 specifies the overall requirements for the project. Then, the main chapters describing hardware specification, see Chapter 5, and onboard software and algorithms, see Chapter 6, follow. They describe the whole structure of the developed autopilot unit in detail and all information necessary for manufacturing the new samples of the system can be found here. Following Chapter 7 describes the developed ground segment of the whole system. The overall system was tested in different real flight conditions and Chapter 8 evaluates these experiments and results. Chapter 9 presents existing applications of the developed system in several ongoing projects, which are using some parts of this project. Finally, the overall summary and conclusion is given in Chapter 10.

1.2 Definitions, Acronyms and Abbreviations

Acronyms and abbreviations used in this document are listed in the table below (1.1).

Abbreviations	Definitions
AHRS	Attitude and Heading Reference System
DMP	Digital Motion Processor
FPV	First Person View
GCS	Ground Control Station
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LQ	Linear-quadratic regulator
MEMS	Micro-Electro-Mechanical Systems
PCB	Printed circuit board
PID	Proportional-integral-derivative controller
RC	Radio Control
UAV	Unmanned Aerial Vehicle

Table 1.1: Definition of abbreviations used in this paper

Chapter 2

State of the art

Some flight control units for small UAVs are commercially available, see Fig. 2.1. These units are typically used for civilian tasks as aerial photography, mapping, promotion, research or just as hobby. Mainly the last group of users are speeding up the development of UAV control platforms these days. The boom of RC model kits equipped with First Person View systems significantly increases demand for stabilization and navigation systems for small aerial vehicles. This chapter summarizes some of the opensource and commercially available systems.

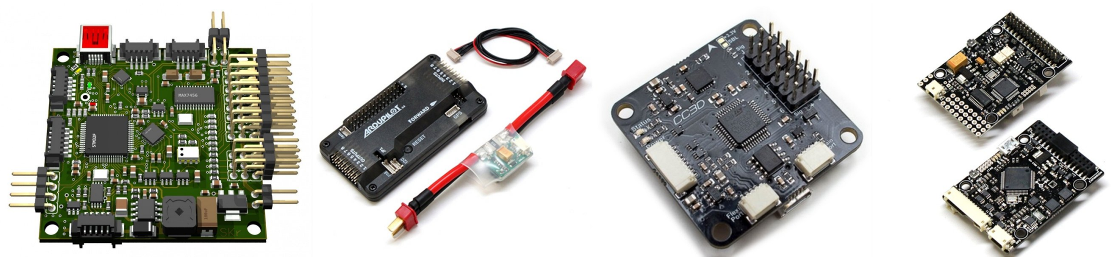


Figure 2.1: a) Openpilot, b) Ardupilot APM, c) Paparazzi, d) Pixhawk PX4FMU

2.1 Ardupilot APM 2.5+

The APM 2.5+ [10] is a complete open source autopilot system and the best-selling technology that won the prestigious 2012 Outback Challenge UAV competition. It allows the user to turn any fixed, rotary wing or multi-rotor vehicle (even cars and boats) into a fully autonomous vehicle; capable of performing programmed GPS missions with waypoints.

- Free open source firmware comes in different versions that support planes (”ArduPlane”), multicopters (quads, hex, oct, etc) and helicopters (”ArduCopter”), and ground rovers (”ArduRover”)!

- Simple setup process and firmware loading via a point-and-click utility. No programming required. (But if you do want to fiddle with the code, you can with the easiest embedded programming toolkit available: Arduino)
- Full mission scripting with point-and-click desktop utilities
- Can support hundreds of 3D waypoints
- Return-to-launch, loiter in place, "follow me" or just click on the map and tell it to "go to here" (with telemetry option)
- Auto takeoff and landing
- Two-way telemetry and in-flight command using the powerful MAVLink protocol
- Choice of free Ground Stations, including the APM Mission Planner, which includes mission planning, in-air parameter setting, on-board video display, voice synthesis, and full datalogging with replay.
- Cross-platform. Supports Windows, Mac and Linux. Use the graphical Mission Planner setup utility in Windows (works under Parallels on a Mac or Mono on Linux) or use a command-line interface on any other operating system. Ground stations are available for all three operating systems. Based on the Arduino programming environment, which is also fully cross-platform.
- Autonomous takeoff, landing and special action commands such as video and camera controls
- Supports full "hardware-in-the-loop" simulation with Xplane and Flight Gear
- Hardware includes the following:
  - 3-axis gyro
  - 3-axis accelerometer
  - 3-axis magnetometer
  - Barometric pressure sensor for altitude
  - 5Hz GPS module
  - Voltage sensors for battery status
  - 4Mb of onboard datalogging memory. Missions are automatically datalogged and can be exported to KML
  - Built-in hardware failsafe processor, can return-to-launch on radio loss.
  - (Optional) Airspeed sensor
  - (Optional) Current sensor

Other details are in Ref. [10].

2.2 Pixhawk PX4FMU

PX4FMU [8], [9] is a high-performance autopilot-on-module suitable for fixed wing, multi-rotors, helicopters, cars, boats and any other robotic vehicles. It is targeted towards high-end research, amateur and industry needs.

- 168 MHz / 252 MIPS Cortex-M4F
- Hardware floating point unit
- POSIX-compatible RTOS
- SIMD extensions
- 192KB SRAM / 1024 KB Flash
- USB Bootloader (software updates, Windows, Linux, Mac OS supported)

- 4x UART, 2x I2C, 1x SPI, 1x CAN
- External magnetometer port (I2C1 or I2C3, compatible with this board: 3DR magnetometer breakout board)
- microSD slot
- PPM / RC control input (sum signal format, many compatible receivers, all channels on one connector)
- Up to 8 GPIO, 2 25mA high power, up to 6 PWM (servo out)
- Battery sense (1-18V), Buzzer (up to 1.0 A, VBAT driven)
- Reverse polarity protection on all power inputs
- Buzzer (PWM) output
- JTAG / SWD (ARM-Mini 10 pos / 0.05" connector)

More detailed information can be found at official websites Ref. [8] or [9].

2.3 OpenPilot Revolution

OpenPilot [11] is an Open Source project aimed at providing the community with low-cost but powerful stabilization and autopilot platforms. OpenPilot's strength is the community that has developed, where individuals contribute their time, effort and skills so all can benefit. As a non-profit open source program, OpenPilot aims to bring this technology down in price to make it more affordable to deploy.

The OpenPilot Revolution board, also called 'Revo', is a new breed of Autopilot using the STM32F4 Micro-controller. This is important as it contains a hardware floating point unit (FPU), which is a huge advancement for hobby-class autopilots.

Of course, OpenPilot has been 32bit from start, and the FPU is another step up the performance ladder.

The Revolution is an autopilot for multi-rotors, helicopters and fixed wings. It is a full 10 degrees of freedom (DOF) with magnetometer, gyroscopic, accelerometer and pressure sensors.

More information can be found in Ref. [11].

2.4 Paparazzi

Paparazzi [12] is a free and open-source hardware and software project encompassing an exceptionally powerful and versatile autopilot system for fixedwing aircrafts as well as multicopters. Being open enables you to add more features and improve the system. Using and improving Paparazzi is wholeheartedly encouraged by the community, thus Paparazzi is quickly evolving into an even more powerful system.

The project includes not only the source code with great code like Kalman filtering code but even all airborne hardware information needed, from Autopilot boards to zesty designed IMU's. A powerful ever-expanding array of ground hardware and software including modems, antennas, and a highly evolved user-friendly ground control station is included as icing on the cake. All hardware and software is open-source and freely available to you under the GNU licencing agreement.

Using and improving Paparazzi is wholeheartedly encouraged by the community, thus Paparazzi is quickly evolving into an even more powerful system.

The description is taken from official website of the project - Ref. [12].

Chapter 3

Theoretical background

3.1 Equations of motion

For a description of a motion and a dynamic behavior of a flying system two coordinate frames and 6 states are required: an earth fixed frame and a mobile frame whose dynamic behavior can be described relatively to the fixed frame. The earth fixed axis system will be regarded as an inertial reference frame one in which the first law of Newton is valid. This reference frame is designated by O_{earth}. The mobile frame O_{frame} is settled on Aircraft-Body-Centered, and has its origin coincident with the plane's center of gravity. The attitude about all 3 axis of rotation is given with 6 states: the Euler angles, ϕ (roll), θ (pitch), ψ (yaw), see Fig. 3.1, and the angular velocities around each axis of the O_{frame} [P Q R].

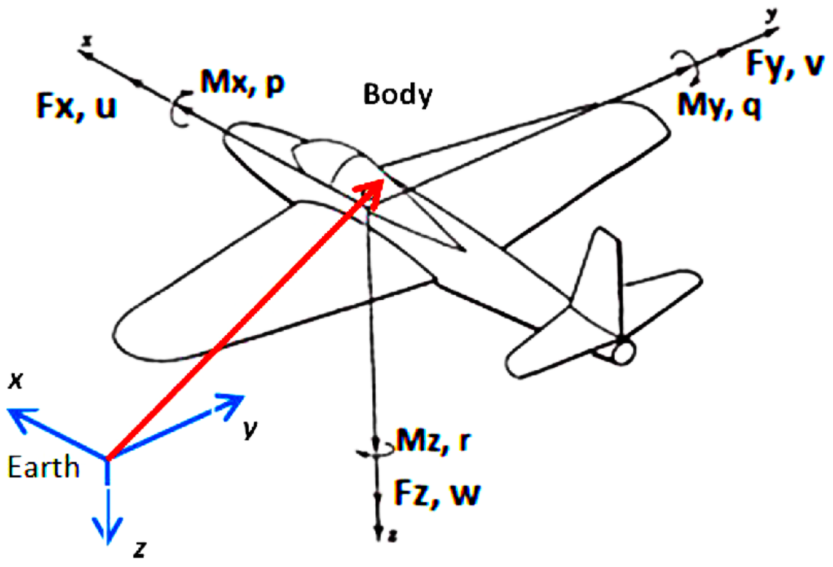


Figure 3.1: Coordinate Frames and Motion Parameters

Another 6 states are necessary, the position of the center of gravity (or COG) [X

Y Z] and respective linear velocity components [U V W] relative to the fixed frame. In sum, the motion of aircraft is defined with 12 states that describe 6 degrees of freedom.

The orientation of the mobile frame relative to the fixed one, can be achieved by using transformation matrix (S), which is obtained from the product of each angular rotation matrix. R(ϕ), R(θ), R(ψ) about body frame, as given in the following:

$$R'(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{pmatrix} \tag{3.1}$$

$$R'(\theta) = \begin{pmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{pmatrix} \tag{3.2}$$

$$R'(\psi) = \begin{pmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$S = R'(\phi)R'(\theta)R'(\psi) \tag{3.4}$$

For the modeling, the aircraft is assumed to be a rigid body, so the Newton-Euler equations can be used to describe its dynamics. The equations of the net force (F_{net}) and the moment (M_{net}) acting on the aircraft body are formulated by Eq. (3.5) and Eq. (3.6).

$$F_{net} = \frac{d}{dt}[mv]_B + \dot{\omega}x[mv]_B \tag{3.5}$$

$$M_{net} = \frac{d}{dt}[I\dot{\omega}]_B + \dot{\omega}x[I\dot{\omega}]_B \tag{3.6}$$

I is the inertia matrix of the plane, v is the vector of linear velocities and ω is the vector of angular velocities. The force equation can be further written as

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} + \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times m \begin{bmatrix} U \\ V \\ W \end{bmatrix}. \tag{3.7}$$

When equations (3.5), (3.6), (3.7) are combined in three-dimensional matrices, translational equations of motion can be formulated by Eq. (3.8).

$$\begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} - \begin{bmatrix} QW - RV \\ RU - PW \\ PV - QU \end{bmatrix} \tag{3.8}$$

The inertia matrix I and moment equation can be expressed as

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}, \tag{3.9}$$

$$M_{net} = \begin{bmatrix} I_{xx} & -I_{xy} & I{-}_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} + \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \begin{bmatrix} I_{xx} & -I_{xy} & I{-}_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix}. \tag{3.10}$$

Moreover, using the transformation matrix S, angular velocities can be computed by

$$\begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{3.11}$$

The presented equations describing the general motion of an aircraft are nonlinear and cause a great calculation effort. Taking the symmetrical structure of aircraft according to x and z- axes into account, the inertia matrix can be simplified as in Eq. (3.12).

$$I = \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{zx} & 0 & I_{zz} \end{bmatrix} \tag{3.12}$$

More details can be found in Ref. [4].

3.1.1 Linearized Motion Equations and Altitude Control of Aircraft

For an altitude control simulation, it can be assume that the aircraft has no ϕ (roll) and ψ (yaw) angle changes and makes only θ (pitch) motion in x-z plane.

Therefore, the motion equations can be linearized and whole system variables: F_y, F_z, p, q, r, M_x, M_y, M_z, u, v, w are linearized about F_{x0}, F_{y0}, F_{z0}, p_0, q_0, r_0, M_{x0}, M_{y0}, M_{z0}, u_0, v_0, w_0 values. Except u_0, F_{x0}, F_{z0}, all other initial values are assumed to be zero. F_{x0} and F_{z0} initial values depend on the weight of aircraft as formulated in Eq. (3.13).

$$\begin{aligned} F_{x0} &= mg\sin\theta_0 \\ F_{z0} &= mg\cos\theta_0 \end{aligned} \tag{3.13}$$

For little angular deviations $\Delta\theta$, following expressions can be formalised.

$$\begin{aligned} \sin(\theta_0+\Delta\theta) &\cong \sin\theta_0 + \Delta\theta\cos\theta_0 \\ \cos(\theta_0+\Delta\theta) &\cong \cos\theta_0 - \Delta\theta\sin\theta_0 \end{aligned} \tag{3.14}$$

Then, the linearized equations are

$$\begin{aligned} m\Delta\dot{u} + \Delta\theta mg\cos\theta_0 &= \Delta F_x, \\ m\Delta\dot{w} + \Delta\theta mg\sin\theta_0 - mu_0\Delta q &= \Delta F_z, \\ I_{yy}\Delta\dot{q} &= \Delta M_y, \\ \Delta\dot{\theta} &= \Delta q. \end{aligned} \tag{3.15}$$

Moreover, when assuming stable flight conditions, the input forces and moments must be also linearized. It can be recognized that the forces F_x in the direction of x-axis depend on the linear velocities u, w, throttle (gas) $\delta + T$ and the elevator δ_e (pitch angle) values. Changes of the F_z forces and M_y moment are dependent on u, w linear velocities; $\dot{w}$ angular acceleration and again on throttle (gas) δ_T and elevator δ_e (pitch angle) values. Then, the linearized motion equations can be formalised as:

$$\begin{aligned} m\Delta\dot{u} &= X_u\Delta u + X_w\Delta w - mg\cos\theta_0\Delta\theta + X_{\delta_e}\delta_e + X_{\delta_T}\delta_T, \\ m(\Delta\dot{w} - \Delta q u_0) &= Z_u\Delta u + Z_w\Delta w + Z_{\dot{w}}\Delta\dot{w} + Z_q\Delta q - mg\sin\theta_0\Delta\theta + Z_{\delta_e}\delta_e + Z_{\delta_T}\delta_T, \\ I_{yy}\Delta\dot{q} &= M_u\Delta u + M_w\Delta w + M_{\dot{w}}\dot{w} + M_q\Delta q + M_{\delta_e}\delta_e + M_{\delta_T}\delta_T. \end{aligned} \tag{3.16}$$

The state space model representation of the system can be expressed by following linearized equations:

$$\begin{aligned} E\Delta\dot{X} &= \bar{A}\Delta X + \bar{B}u, \\ \Delta\dot{X} &= E^{-1}(\bar{A}\Delta X + \bar{B}u) = A\Delta X + Bu, \end{aligned} \tag{3.17}$$

$$\begin{bmatrix} m & 0 & 0 & 0 \\ 0 & m - Z_{\dot{w}} & 0 & 0 \\ 0 & -M_{\dot{w}} & I_{yy} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta\dot{u} \\ \Delta\dot{w} \\ \Delta\dot{q} \\ \Delta\dot{\theta} \end{bmatrix} =$$

$$\begin{bmatrix} X_u & X_w & 0 & -mg\cos\theta_0 \\ Z_u & Z_w & Z_q + mu_0 & -mg\sin\theta_0 \\ M_u & M_w & M_q & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta w \\ \Delta q \\ \Delta\theta \end{bmatrix} + \begin{bmatrix} X_{\delta_e} & X_{\delta_T} \\ Z_{\delta_e} & Z_{\delta_T} \\ M_{\delta_e} & M_{\delta_T} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_e \\ \delta_T \end{bmatrix}. \tag{3.18}$$

More details can be found in Ref. [4].

3.2 Dynamical stabilisation - autopilot

Systems responsible for the dynamic stabilisation of an airplane during a flight are called *autopilots*, when Euler angles, its derivations and angles of attack need to be stabilise. Autopilots represent the first a the second hierarchical layer of control. The first layer consists of angular rates dampers. The second layer is responsible for control and stabilisation of Euler angles in horizontal and vertical plane of aircraft.

More details can be found in Ref. [4].

3.2.1 Stabilisation of longitudinal motion of an aircraft

The first layer of longitudinal stabilisation is a pitch rate damper. The feedback signal is derivation of the pitch angle $\dot{\theta}$, which is pitch rate measured by MEMS angular rate sensor in this case. The damper changes the dynamical behavior on high frequencies. The input value for this damper is the reference pitch rate given by a pilot or by the second layer stabilization algorithm. The input for the stabilization loop is given by a pilot as desired pitch angle or by navigation loop, which controls the height of the flight. A block scheme of the longitudinal motion can be seen in Fig. 3.2. Details can be found in [4].

3.2.2 Stabilisation of lateral motion of aircraft

The controller for lateral motion has similar structure as the longitudinal control loop. The first layer of lateral stabilisation is roll rate damper. The feedback signal is roll rate, which is measured by MEMS angular rate sensor. The input value for this damper is the reference roll rate given by a pilot or by the second layer stabilization

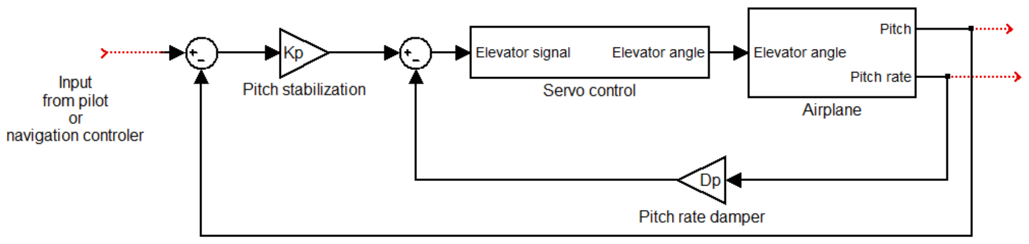


Figure 3.2: Block scheme of the longitudinal motion controller [4]

algorithm. The input for stabilization loop is given by a pilot as desired roll angle or by navigation loop, which controls the yaw motion and navigation. A block scheme of the lateral motion can be seen in Fig. 3.3. More details can be found in [4].

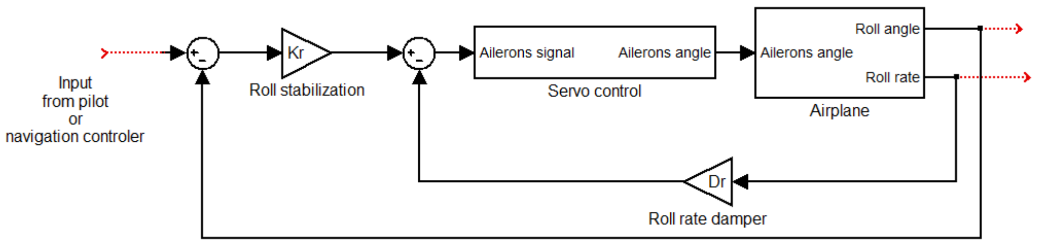


Figure 3.3: Block scheme of the lateral motion controller [4]

3.3 GPS and navigation

The navigation of a flying vehicle is based on an algorithm called *Cross track error*. This algorithm is based on calculating the perpendicular distance between actual position of a flying UAV and a line defined by two consequences waypoints. For purpose of this project, this algorithm is calculated on an spherical approximation of the Earth (so the uncertainties of the real geoid geometry are neglected). This approach has the precision good enough for this application and it is not so demanding for computational processing time as the algorithms calculated on ellipsoid or reference geoid.

The *Cross track error* algorithm consists of three major subroutines: distance between two waypoints calculation, heading to the next waypoint calculation and cross track error calculation and minimization.

For calculation of the distance between two waypoints, a method called *Equirectangular approximation* [34] is used. This method is based on *Pythagoras' theorem* and it is suitable for use in applications, where only small distances are calculated (less than 100 km) and where the overall performance is important, rather than accuracy. More details about these method can be found in [34], the equation follows:

$$\begin{aligned} x &= (\lambda_2 - \lambda_1) \cdot cos\left(\frac{\varphi_1 + \varphi_2}{2}\right) \\ y &= \varphi_2 - \varphi_1 \\ d &= R \cdot \sqrt{x^2 + y^2}, \end{aligned} \tag{3.19}$$

where φ is latitude, λ is longitude and R is earth's radius (mean radius = 6 371 km), d is the computed distance between two points.

The heading to the next waypoint is calculated according to a great circle path (orthodrome) azimuth. The azimuth of orthodrome is changing during the path over it, so it has to be recalculated in every loop of navigation algorithm to achieve sufficient accuracy. More details are in [34], the equation follows:

$$\theta = atan2(sin(\lambda_2 - \lambda_1) \cdot cos(\varphi_2), cos(\varphi_1) \cdot sin(\varphi_2) - sin(\varphi_1) \cdot cos(\varphi_2) \cdot cos(\lambda_2 - \lambda_1)), \tag{3.20}$$

where φ is latitude, λ is longitude and θ is computed azimuth.

The last step is to calculate a final cross track error and its minimization in control loop. The subroutines described above are used in final calculation of cross track error. The illustration of the trajectory can be seen in Fig. 3.4. The overall control algorithm can be divided to following steps:

- At first, the distance to the next waypoint (WP2) *distance_to_WP* is calculated - the function *get_distance_to_WP* described above is used.
- Check of the direction of the flight is done. The distance to next waypoint has to go down, which means that the aircraft is approaching the waypoint. The other case can occur, when the aircraft is behind the desire waypoint, in this case the next waypoint (WP3) is chosen next.
- The azimuth from the actual position to the next waypoint *angle_to_WP* is calculated.
- The azimuth of the defined trajectory is calculated, it means the azimuth from WP1 to WP2: *angle_of_line*. Both these azimuths are calculated in function *get_azimuth_to_WP* described above.

- The angle of the triangle corresponding to the actual situation (actual position, WP1, WP2) is calculated next:

```
angle_of_triangle = angle_to_WP - angle_of_line.
```

Then the cross track error from path can be calculated:

```
x_track_error = distance_to_WP * sin(angle_of_triangle).
```

The control loop based on PID regulation keeps the *x_track_error* minimal through changing the *yaw_action* of aircraft. The waypoint is marked as reached when the distance to it is inside a tolerance band. In that case, the next waypoint is chosen and the algorithm starts from the beginning on the path line defined by WP3 and WP2.

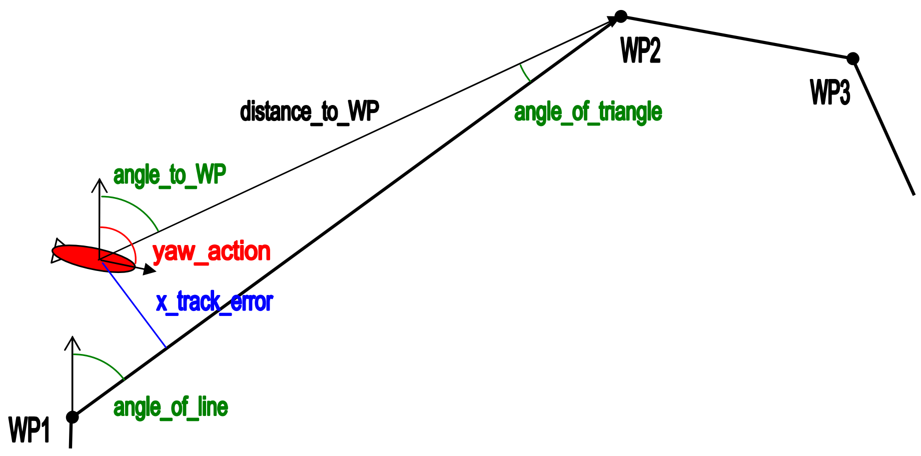


Figure 3.4: Ilustration of Cross track error algorithm and used variables.

Chapter 4

Design analysis

The main goal of the FCU system, which have been developed, is to build control unit, which can be used in any type of vehicle based on RC model or its parts. Thus, it is necessary to substitute an original RC transmitter and receiver system with more suitable communication channel. A minimal required range of this wireless communication system is 1 mile.

Other requirements follow from expected functions of autonomous control algorithms, primarily an attitude and heading reference system, a localization system, a measuring of airspeed, a barometric altitude, an angle of attack and other flight parameters. The system should be capable of implementation of complex control laws and sensor fusion algorithms such as Kalman Filter for AHRS and LQ or other MIMO feedback loops for stabilization, navigation or formation control.

Last group of requirements respects the potential higher-level algorithms, which are expected to be developed on top of the control unit – a wind estimation, an variometer, a distance measuring or an estimation of a relative position in a formation. An important feature in this category is also high data rate capability of communication channel, which can be used by multiple vehicles in an formation or for large datasets transferring.

Additional requirements are related to the Ground Control Station (GCS). Main purpose of GCS is real-time control of a vehicle in manual mode or command forwarding in one of the autonomous regimes. These commands can be set through the basic user interface on GCS, or can be forwarded from a personal computer or a mobile device with Android OS. The basic flight parameters should be shown on the GCS.

A solution based on one single communication channel – i.e. without use of original RC transmitter – was selected, unlike to other existing projects. Such a way, the system is independent on a specific RC system, and the range of the system will be always fully defined just by the used communication channel. The main node of Ground Station System is a hardware platform with basic user interface. This module has to be robust and reliable, but this approach makes the system independent on personal computer and its malfunctions. The solution used in this work is shown on the block diagram on figure (4.1).

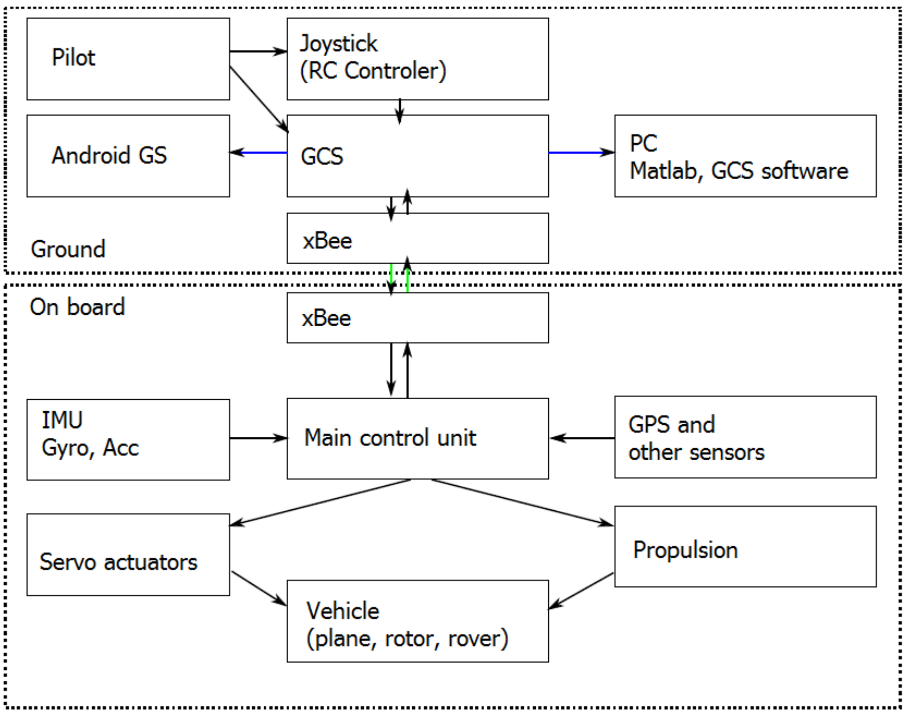


Figure 4.1: Block scheme representing the overall concept and hierarchy of used control system.

Chapter 5

Hardware specifications

Specifications of the whole system and selected subsystems respecting the requirements described in above section are detailed in this section.

5.1 Selection of Components

5.1.1 Main Processor

A microcontroller used in the flight control unit should be capable of reading all values from sensors with different interfaces including analog outputs, I2C, SPI buses, or serial communication. It should have capability of computing complex algorithms for AHRS or feedback loops. For this project, the 32 bit ARM controller STM32F100RB [13] was chosen. The advantage is that it has the same pinout as higher versions of controllers from STM, which means that more powerful processor can be used with the same PCB if necessary. Basic parameters of this microcontroller are in the table 5.1 and more details can be found in product datasheet [13].

5.1.2 Communication Channel

The first functionality, which has to be solved, is the communication. In this project, the commercially available x-bee modules have been selected. The main reason is that the development of new communication channel from scratch overlaps the magnitude of this project and it can be done in the future, if necessary. The advantage of x-bee (or the modules implements Zigbee protocol) is that there are more variants of a specific version, which differ in frequencies, output power, coverage, data rates, but all have the same interface for connecting to master device. Another significant advantage of communication modules based on Zigbee protocol is the native support of multi-point networking, which can be useful in formation control

Architecture	ARM Cortex-M3
Program Memory Size	128KB
RAM Memory Size	8KB
CPU Speed	24MHz
No. of I/O's	51
MCU Case Style	LQFP
No. of Pins	64
Embedded Interface Type	I2C, SPI, USART
Supply Voltage Min	2V
Supply Voltage Max	3.6V
No. of PWM Channels	6
No. of Timers	6
Peripherals	ADC, DAC, DMA, PWM, RTC, Timer
Program Memory Size	128KB
Price (Euro)	3,5

Table 5.1: STM32F100RB: Basic parameters of used CPU [13]

projects. The disadvantage of this solution is non zero latency of communication channel, but the latency is still small enough to provide real time manual control. The Zigbee modems used in this project are XBee Pro 2.4 GHz 50mW Series 2.5, one configured as Coordinator used in GCS and another in flier control unit configured as Routers. Another modems used in this project are XBee Pro 50mW GHz Series 1 and XBee Pro 868MHz. The evaluation of the parameters of these modules and its comparison can be found in the Experimental results. The picture of the xbee module can be seen in Fig. 5.1a, more information can be found in datasheet [14].

(a) XBee Pro 2.4 GHz [14]

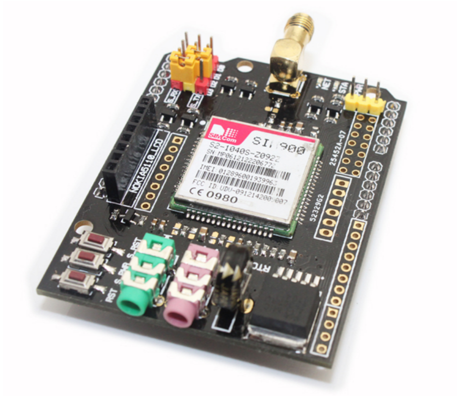

(b) GPRS/GSM Shield - EFCom [15]

Figure 5.1: Communication modules, which can be used in this project

Another option for the communication channel is a GSM/GPRS modem, which can provide connection for higher distances. The usability of this system is dependent on GSM signal coverage in particular area. The disadvantage of this option is latency, which is not determine. Thus, it is not possible to control the vehicle in manual mode. This communication channel can be used for transferring commands in the autonomous mode of the flight or to transfer larger datasets, which are not real-time dependent such as video stream. The example of GSM/GPRS module, which can be used optionally - PRS/GSM Shield - EFCom [15] - is shown in Fig. 5.1b.

5.1.3 AHRS Unit

For the attitude and heading reference system, a full inertial measurement unit is used, which includes 3 axes gyro, accelerometer and magnetometer. This IMU is based on MEMS sensors, which are cheap and small, and precise enough for this application. Two versions of the IMU unit was used in this work – one based on MPU6050 [16] chip integrates 3x gyro and 3x accelerometer with HMC5883L [18] magnetometer, and second version based on LSM330 [17] and the same magnetometer. These two versions are equivalent for this project, because DMP unit in MPU6050 is not used and all calculations for AHRS algorithm are processed in the main processor. The basic features of both IMU sensors are summarized in the table (5.2).

	MPU6050	**LSM330**
Supply Voltage Min	2.5V	2.4V
Supply Voltage Max	3.6V	3.6V
Sensor Case Style	QFN	LGA
No. of Pins	24	28
Gyroscope Range	±250, 500, ±1000, ±2000°/s	±250, ±500, ±2000°/s
Acceleration Range	±2g, ±4g, ±8g, ±16g	±2g, ±4g, ±8g, ±16g
Operating Temperature Range	-40°C to +85°C	-40°C to +85°C
Price (Euro)	24	5,6

Table 5.2: MPU6050 [16], LSM330 [17]: Basic parameters of used IMU sensors

(a) MPU6050 [16] (b) LSM330 [17] (c) HMC5883L [18]

Figure 5.2: MEMS IMU sensors used in this project

5.1.4 GPS

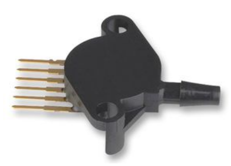
The main sensor used for localization and navigation is GPS receiver, in this project the GPS unit is based on MediaTek MT3329. This GPS unit provides 10 Hz sampling frequency, high sensitivity: up to -165dBm tracking, build-in patch antenna, position accuracy 3m and low power consumption 48 mA. More information can be found in datasheet [19].

5.1.5 Pressure sensor

Another important flight parameters, which should be measured during flight, are barometric height and airspeed. These values are measured with pressure sensors MPX4115A and MPXV7002DP. Parameters of these sensors are in the table (5.3).

	MPX4115A	**MPXV7002DP**
Pressure Type	Absolute	Differential
Operating Pressure Min	15kPa	-2kPa
Operating Pressure Max	115kPa	2kPa
Supply Voltage Min	4.85V	4.75V
Supply Voltage Max	5.35V	5.25V
Sensor Case Style	SIP	SOIC
Sensitivity, V/P	45.9mV/kPa	1V/kPa
Price (Euro)	12,3	14

Table 5.3: MPX4115A [20], MPXV7002DP [21]: Basic parameters of used pressure sensors


(a) MPX4115A[20]

(b) MPXV7002DP [21]

Figure 5.3: MEMS Pressure sensors used in this project

5.1.6 Voltage regulator for power supply

It is based on fixed low drop positive voltage regulator LD1117S33TR [22], basic parameters are listed in the table (5.4). This regulator has been chosen with respect to required power consumption, price, and compactness of the whole power source layout design.

Input Voltage Max	15V
Output Voltage Nom.	3.3V
Output Current	800mA
Dropout Voltage Vdo	1.1V
Voltage Regulator Case Style	SOT-223
No. of Pins	3
Operating Temperature Min	0°C
Operating Temperature Max	125°C
Price (Euro)	0,5

Table 5.4: LD1117S33TR [22], low drop positive voltage regulator

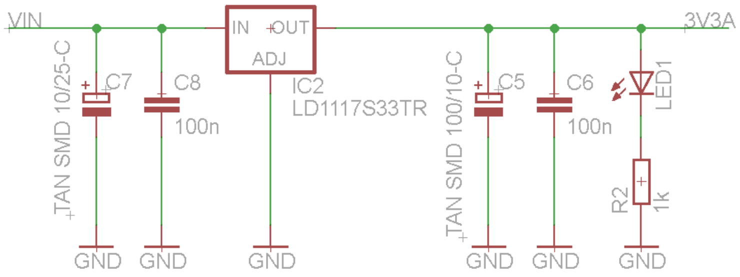
5.2 Onboard Hardware - Version 1

The first version of autopilot controller unit has been made with emphasis on modularity, extensibility and scalability. Therefore, the electronic system is divided into smaller modular pcbs, which ensures one and only concrete functionality. Individual modules can be substituted with new one, for example when new MEMS sensor is available on the market. This approach also simplifies testing and developing procedures, because it enables to connect individual parts to development board and independent testing.

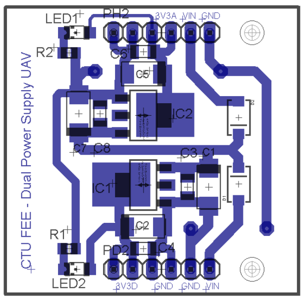
5.2.1 Power supply module

The first independent part of the whole system is power supply. Power supply is the critical and fundamental part of the system, its functionality can not be replaced, thus the design includes backup identical power supply in hot redundancy configuration. The main power supply is switched to backup immediately after the voltage is getting lower.

The schematic design can be seen in Fig. 5.4a. It shows only one of the power supplies, the backup one is identical – it is created according to the datasheet, with emphasis on power blocking capacitors. The design can be used in the main – backup configuration, or as two individual power supplies for analog and digital circuits. The PCB layout is in Fig. 5.4b. The layout is designed as single layer board, with low price on mind. Pads for voltage regulators are scaled up, for better thermal dissipation. The input and output connectors are mirrored on the main controller board and this module is connected directly on top of the main PCB.



(a) Schematics



(b) PCB layout

Figure 5.4: Power supply module based on LD1117S33

5.2.2 Main controller module

The purpose of this main module is to connect all auxiliary modules to a single unit. The main microcontroller for whole system is ARM Cortex M3 - STM32F100RB. There are slots for communication module (xBee), inertial measurement unit, and input/output ports located on this board. The input / output ports includes 8 PWM outputs for controlling servos, 3 USART lines for x-bee, logger and GPS, I2C for inertial measurement unit, 6 analog inputs, and GPIO pins for another use. The PCB is double-sided and can be seen in Fig. 5.5. The dimensions of PCB is 80 × 40 × 15 mm with all modules connected. The schematics of this module is part of Appendix.

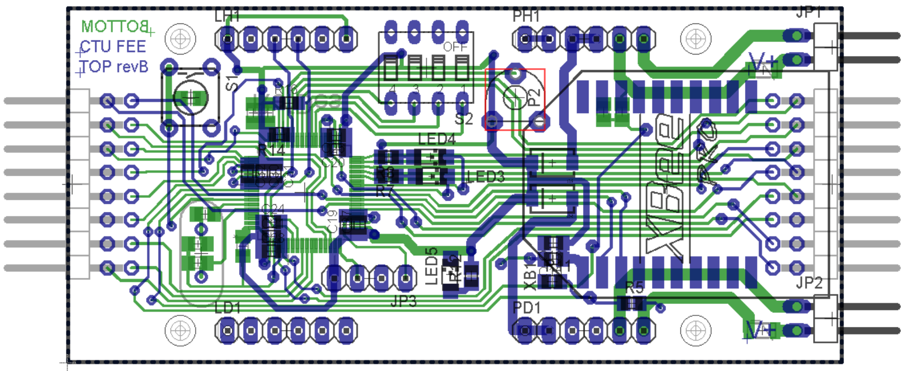
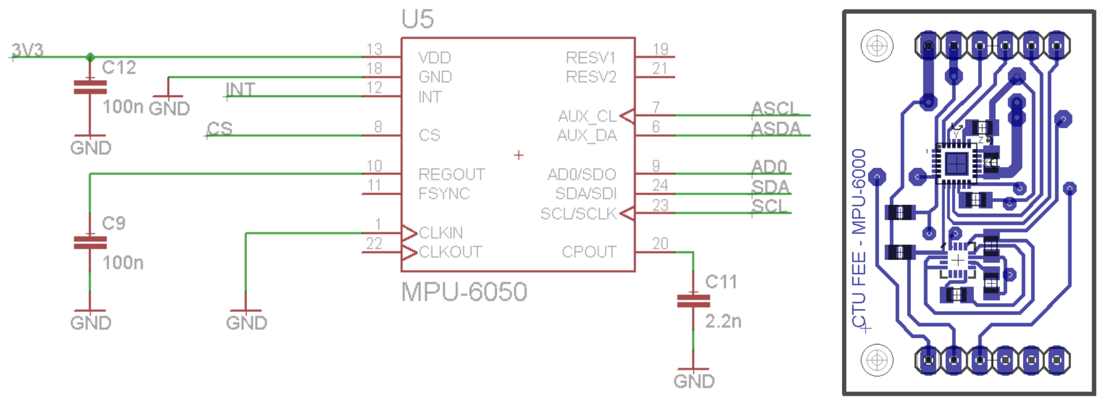


Figure 5.5: Main controller module - PCB layout

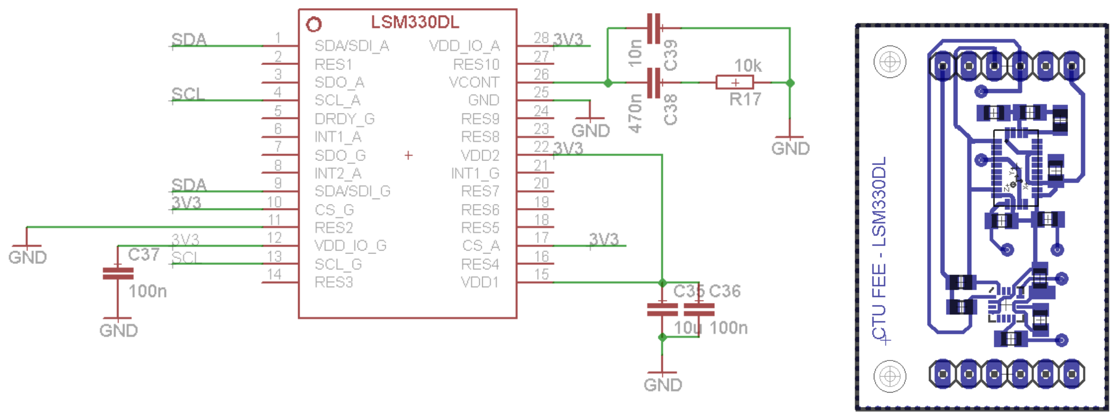
5.2.3 Inertial measurement board

This module merges the funcionalities of MEMS accelerometers, angular rate sensors and magnetometer. These sensors are connected through I2C bus to the main controller, so there is no additional computing device on this module. In this project two different inertial measurement boards has been developed. The first one is based on MPU-6050 MEMS sensor. For the second board is used cheaper IMU sensing device - LSM330DL. The comparison of these two sensors is in table (5.2). Schematics and PCB layout for the first version fitted with MPU-6000 is on fig (5.6a) and (5.6b). Schematics and layout for LSM330DL is on fig (5.7a), (5.7b). Both of these modules can be optionally fitted with magnetometer HMC5883, this sensor is connected to the same I2C bus. The schematics for magnetometer is on figure (5.8).



(a) Schematics (b) PCB layout

Figure 5.6: Inertial measurement module based on MPU-6050



(a) Schematics (b) PCB layout

Figure 5.7: Inertial measurement module based on LSM330DL

5.2.4 Data logger board

Data logger is also designed as independent module, connected to the main board via one of USARTs. This design allows to have SD card with FAT system operated by independent microprcessor with no extraordinary load for main computational unit. The schematics and layout for this board can be found in Appendix A. The

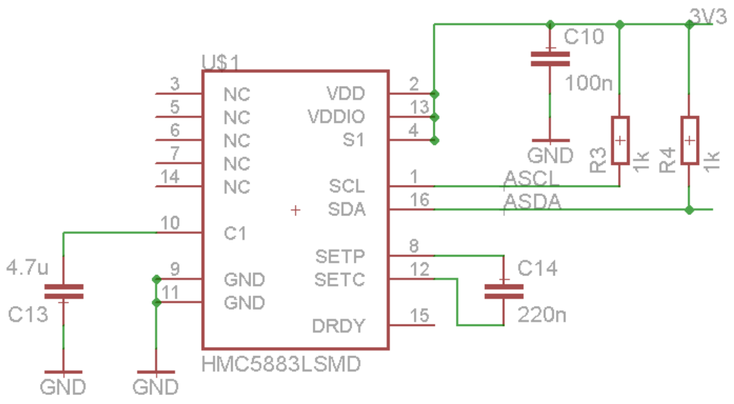


Figure 5.8: Magnetometer connection schematics

board is equipped with AVR processor ATMEGA328P directly connected to SD card via SPI.

5.2.5 Communication module

For the communication with ground station and computer the high range wireless modems are used. The board is fitted with slot for XBee Pro module. The design is independent on used module, the 2.4GHz, 868 MHz or 912MHz modems can be used. These modules are connected via UART 1 to main microcontroller. Different communication module can be connected through side connectors, if necessary. The connection with xBee module is using only 4 wires, it is the basic supported connection, with no more functionalities than wireless UART transmission. The schematics is shown on fig (5.9).

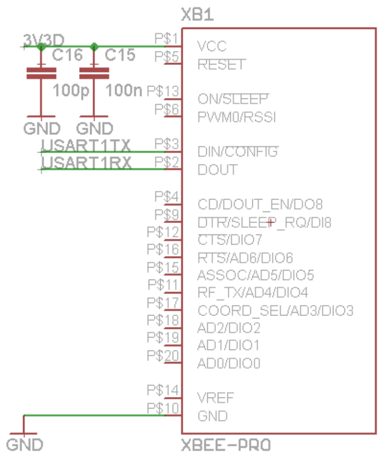


Figure 5.9: XBee connection schematics

5.2.6 Standalone sensors

For the proper functionality of the whole autopilot system additional sensors are necessary. This sensors are mostly connected to the main controller board via side connectors.

GPS

GPS module is connected to the UART 2 of the main microcontroller.

Analog pressure sensors and Angle of Attack

Pressure sensors are connected directly to the AD converters of the main controller. Two types of pressure sensors are used (differential a absolute). The Pitot-Static probe used form airspeed measurement with Angle of Attack probe can be seen on Fig. 5.10 .The calibration and verification experiments with the Airspeed and angle of Attack probe has been performed and can be found in Chapter .

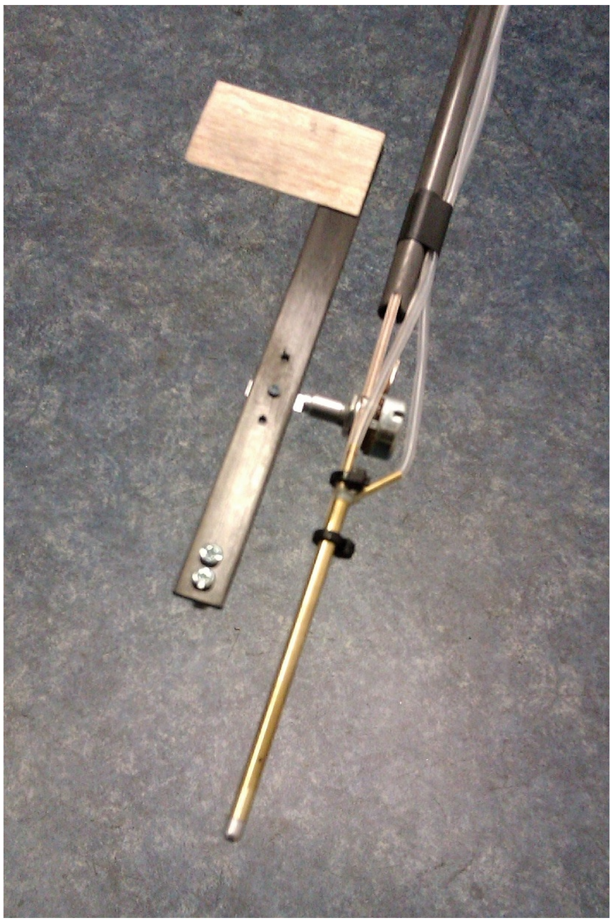


Figure 5.10: Pitot-Static probe and Angle of Attack probe

Ultrasonic distance sensor

For the precise landing the ultrasonic distance sensor can be used.

5.2.7 Hardware debugger

For flashing firmware to main microcontroller and for debugging the original STM SWD debugger is used. The HW debugger used is part of the STM32 VL Discovery kit. The connection of the debugger is in table (5.5), the placement of the connector on the Discovery kit can be seen on figure (5.11). More details can be found in the datasheet [23].

Pin	CN2	Designation
1	VDD_Target	VDD from application
2	SWCLK	SWD clock
3	GND	Ground
4	SWDIO	SWD Data

Table 5.5: STM32VLDISCOVERY - Debug connector CN2 [23]

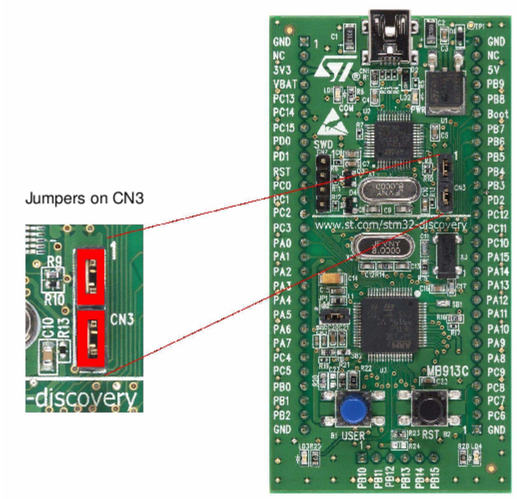
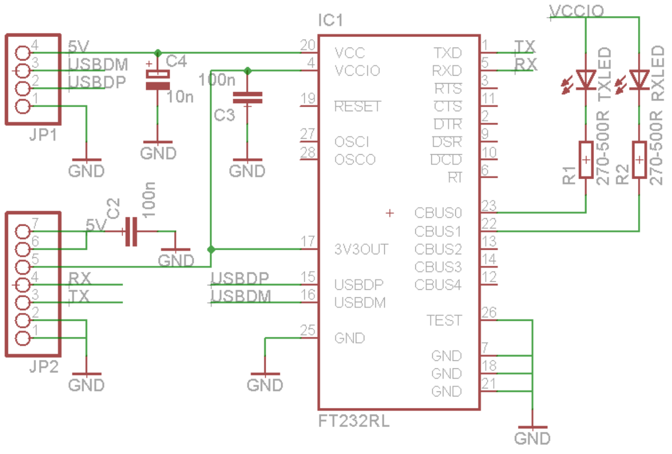


Figure 5.11: STM32VLDISCOVERY connection to external application [23]

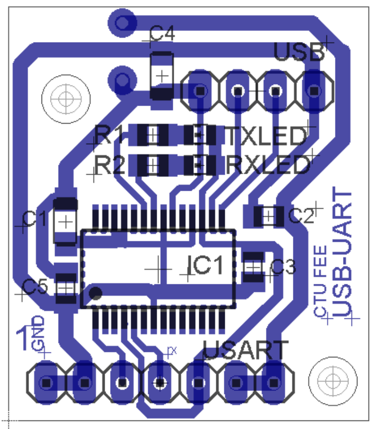
5.2.8 USART-USB Convertor

The connection between computer and ground station (or onboard unit - for debugging only) is done via USB to USART converter. This converter is based on FTDI integrated circuit FT232RL. The schematic can be seen on fig (5.12a), the pcb is on fig (5.12b). This converter can be also used for debugging - the connection of output ports is in table (5.6).

The first version of autopilot can be seen on pictures (5.13), (5.14). The picture (5.14) shows the whole system on board of the delta-wing foam A/C model.



(a) Schematics



(b) PCB layout

Figure 5.12: USART-USB Convertor based on FT232RL

Pin	Description
1	Ground
2	Ground
3	USART TX
4	USART RX
5	3,3 V (100mA)
6	5 V (250mA)
7	5 V (250mA)

Table 5.6: USART-USB Converter connection

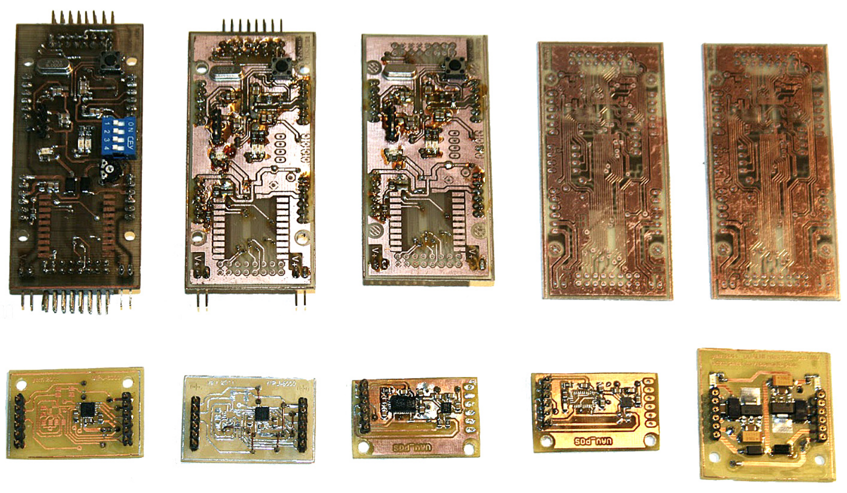


Figure 5.13: Modules of Version 1 - onboard flight control unit

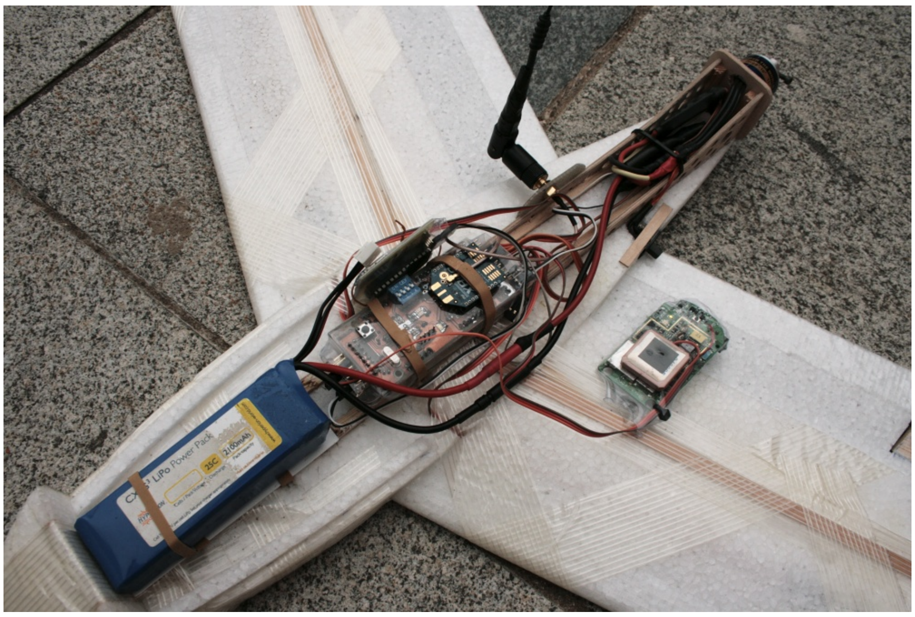


Figure 5.14: Onboard flight control unit

5.3 Onboard Hardware - Version 2

The second version of the onboard autopilot hardware is created as a small compact device which combines all the sensors and functionalities verified in the first version. The purpose of this device is to create simple design, which can be manufactured in bigger series and can be used as flying control unit in more cooperating vehicles. The low cost, compactness, light weight and robustness has been considered to create this unit.

The whole unit is created as single board device, designed on two layer PCB. Components for this unit has been chosen the same as for version 1, very similar is also schematics of individual parts, it can be seen in Appendix A. The microcontroller used is STM32F100RB, the inertial measurement chip is LSM330DL, power supply regulator is LD1117S33TR. The additional device on this board is flash memory connected through SPI to the main controller - thus it is not necessary to use external data logger, and the whole system is smaller. Flash memory used in this design is Micron N25Q064A [24] – the schematics can be seen on fig (5.15). The PCB layout of the whole board is on fig (5.16a), (5.16b). The assembled modules can be seen on figure (5.18), the main module with standalone sensors connected is on picture (5.19).

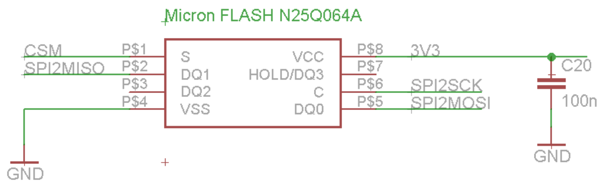
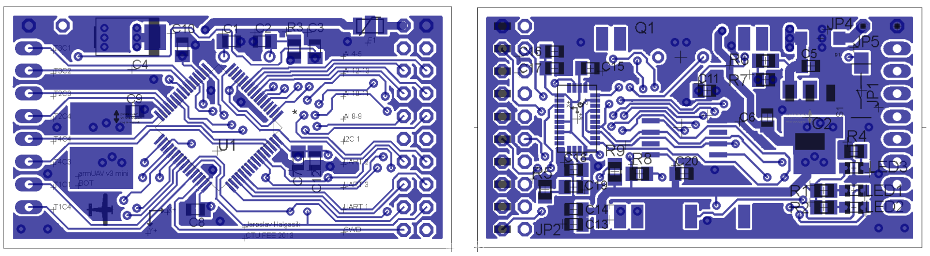
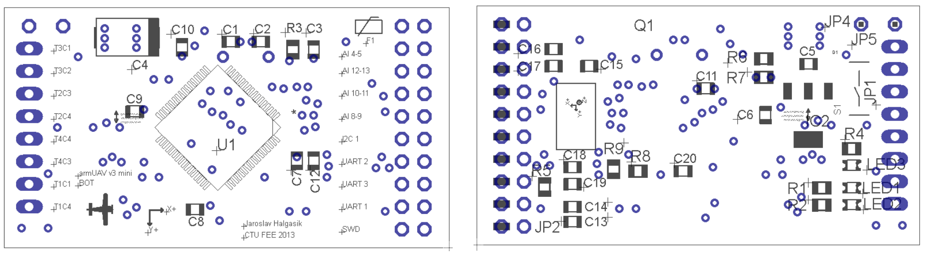


Figure 5.15: Micron N25Q064A - connection schematics [24]



(a) Bottom layer (b) Top layer

Figure 5.16: Design of Onboard Module - Version 2



(a) Bottom layer (b) Top layer

Figure 5.17: Assembly layout of Onboard Module - Version 2

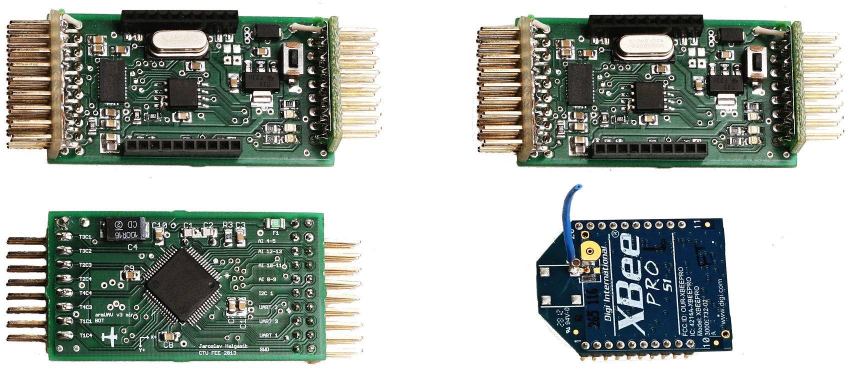


Figure 5.18: Assembled Modules of Version 2 - onboard flight control unit

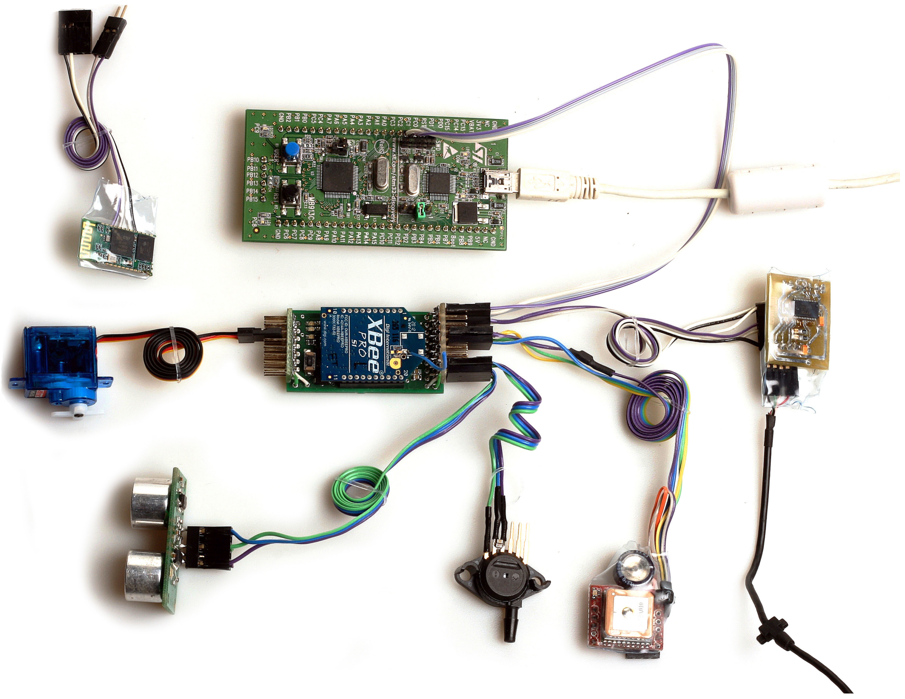


Figure 5.19: Modules of Version 2 - main module with standalone sensors

Chapter 6

Onboard software and algorithms

The onboard software is fundamental part of whole unmanned system. It has to be capable of autonomous operation with no connection to ground station or to another computational unit. There are many independent tasks cooperating together to ensure the whole functionality of the system.

These tasks can be divided to groups which are responsible for similar functionalities. The basic set of these groups is: initialization of microcontroller and peripherals, low level communication with sensors, communication with ground segment and higher control algorithms.

This section describes these groups of functionalities in detail. The whole code used in this project is shown in appendix and it can be found in digital attachment to this thesis as a project for Keil IDE. The basic installation procedure description of Keil IDE and necessary toolchains follows.

6.1 Initialization of microcontroller and peripherals

The purpose of this module is to set up low level functionalities of ARM core of microcontroller and its peripherals. Most of these functions are part of the file *init.c*.

6.1.1 System timer

At first, the system timer of the core is initialized to generate interrupts every 1 millisecond. This interrupt is used for measuring of the on board time:

```
SysTick_Config(SystemCoreClock / 1000);
```

6.1.2 Universal Asynchronous Receiver and Transmitter - UART

The initialization procedure continues with initialization of USART. The explanation of USART initialization process follows, more information can be found in examples [25] or stm Reference Manual [26]. The code is divided in sections for the readability: Section 1 is turning on the clock pulses for peripheral – in other words it turns on the USART device – otherwise it stays in low power mode. Section 2 configures the Input and Output pins for USART. Section 3 configures the concrete properties for USART device: baud rate, number of stop bits, parity, hardware control and other features. Section 4 is responsible for connecting the interrupts from USART to NVIC table. Interrupts are then handled separately in another software module and they are described later in this document. These sections are similar in the most of the initialization procedures for ARM microcontroller, in general it includes turning on the clock for concrete peripheral device, setting up the concrete IO pins, setting up the properties of device and connecting interrupt vectors, if necessary.

```
void USART_Init(void)
{
  /* Structures declaration */
  GPIO_InitTypeDef GPIO_InitStructure;
  USART_InitTypeDef USART_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Section 1 - Power on peripheral clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
  RCC_APB2Periph_AFIO, ENABLE);

  /* Section 2 - Setting up IO pins (PA9 as TX, PA10 as RX) */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);


  /* Section 3 - Properties of USART */
  USART_InitStructure.USART_BaudRate = 57600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl =
      USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);

  USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
  USART_Cmd(USART1, ENABLE);
  /* Section 4 - Configuration of interrupts */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

  NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

6.1.3 Timers

The next peripheral device of microcontroller which has to be initialized and configured before the main software starts are timers. The example of initialization of timer 3 follows. Timer 3 is one of the timers used for driving the servo motors. The explanation of controlling a RC servo motor follows later in this document. This timer is configured to have the frequency of 50Hz, which is the carrier frequency of the servo signal. The resolution given by this timer is set to microseconds which gives enough of precision for continuous driving of servo. The output value for the servo is then set in the range 1000 – 2000 micro seconds.

To achieve this settings the prescaler value of the timer is set to 24 (the main clock of controller is set to 24MHz, thus this prescaler value gives micro second resolution), the period of the timer is set to 20 000 micro seconds, the timer is set to counting up mode, the autoreload function of the timer is enabled. With this setting the signal for servo is generated with no need of computational time of the main core of microcontroller. Lastly the generated signal on particular channel of the timer is connected to one of the output pins. These settings are done in Section 3 of the initialization process, the other sections are similar to the initialization of the USART device.

```
void TIM3_Init(void)
{
  PrescalerValue = 24;

  /* Section 1 - Power on peripheral clock */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3 , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
  RCC_APB2Periph_GPIOC |RCC_APB2Periph_AFIO , ENABLE);

  /* Section 2 - Setting up Output pins */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA , &GPIO_InitStructure);

  /* Section 3 - Time base configuration */
  TIM_TimeBaseStructure.TIM_Period = 20000;
  TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

  /* PWM1 Mode configuration: Channel1 */
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0;
  TIM_OC1Init(TIM3, &TIM_OCInitStructure);

  TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);

  /* PWM1 Mode configuration: Channel2 */
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0;
  TIM_OC2Init(TIM3, &TIM_OCInitStructure);

  TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);
  TIM_ARRPreloadConfig(TIM3, ENABLE);

  /* TIM3 enable counter */
  TIM_Cmd(TIM3, ENABLE);
}
```

6.1.4 Analog to digital converter

The next important peripheral device which has to be initialized before it is used is Analog to digital converter. ADC converter is used for monitoring onboard voltage and current but also pressure sensors, and other fundamental sensors are connected through it.

The initialization procedure of ADC is explained here. The procedure is different than the previous for USART and Timer. The difference is in Section 3 of initialization process, because here has to be selected the channel for measuring. This selection has to be done before every single measurement with different channel. In section 4 the calibration process of the ADC is performed.

```
void ADC_Init(void)
{
  /* Structures declaration */
  ADC_InitTypeDef  ADC_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Section 1 - Power on peripheral clock */
  RCC_ADCCLKConfig(RCC_PCLK2_Div2);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
  /* Section 2 - Configuring properties of ADC */
  ADC_InitStructure.ADC_Mode  = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode  = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode  = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConv  = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign  = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init( ADC1, &ADC_InitStructure );

  /* Section 3 - Selection of the ADC channel */
  switch(ADC_channel)
  {
    case 11:
      ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1,
        ADC_SampleTime_13Cycles5);
      break;
    case 4:
      ...
  }

  /* Section 4 - Selection of the ADC channel */
  ADC_ExternalTrigConvCmd( ADC1, ENABLE );
  ADC_Cmd(ADC1, ENABLE);

  ADC_ResetCalibration(ADC1);
  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
}
```

6.1.5 SPI / I2C bus

Next important peripheral is SPI / I2C communication bus. Both of them are used in this project to communicate with inertial measurement sensors. The SPI initialization is explained here. The important difference in compare to USART init is that the chip select lines has to be configured before the SPI driver is configured. This lines are set to logical 1, thus the sensors and other devices connected to bus are not affected before the bus driver is configured properly.

```
void SPI2_Init(void){
  /* Structures declaration */
  SPI_InitTypeDef  SPI_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Section 1 - Power on peripheral clock */
  RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOB |
    RCC_APB2Periph_GPIOC, ENABLE);
  RCC_APB1PeriphClockCmd( RCC_APB1Periph_SPI2, ENABLE );

  /* Section 2 - Setting up IO pins: SCK, MISO and MOSI */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
  /* Configure I/O for Chip select */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_12 | GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* Deselect the SPI devices: Chip Select high */
  GPIO_SetBits(GPIOC, GPIO_Pin_12);
  GPIO_SetBits(GPIOC, GPIO_Pin_13);
  GPIO_SetBits(GPIOC, GPIO_Pin_7);

  /* Section 3 - Selection of the ADC channel */
  SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
  SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
  SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
  SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
  SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4;
  SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
  SPI_InitStructure.SPI_CRCPolynomial = 7;
  SPI_Init(SPI2, &SPI_InitStructure);

  /* Enable SPI1  */
  SPI_Cmd(SPI2, ENABLE);
}
```

6.2 Low level communication with sensors

There are four types of communication with sensor used in this project: USART, SPI, I2C and ADC. Usage of all these types is described here in detail with examples. For USART the typical sensor is GPS, for SPI it is the FLASH memory and inertial measurement sensor, for I2C another inertial measurement sensor and through ADC pressure sensors are connected.

6.2.1 GPS over USART

The communication with GPS over USART runs on three different levels, through different software modules. The first and most low level is handling of interrupts. The interrupt is created every time the single byte is received on RX line. The interrupt handler takes this single byte and store it in buffer of received bytes. Similar is the procedure of sending data. The buffer of bytes which are ready to be send is handed to the IRQ handler functions which sends one byte every time the USART hardware peripheral is ready. The IRQ function - part of file *stm32f00x_it.c*, which is responsible for this low level functionality is shown here:

```
void USART2_IRQHandler(void)
{
  /* Interupt handler for recieving single byte */
  if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
  {
    UART2_rx_buff[UART2_rx_index] = USART_ReceiveData(USART2);
    UART2_rx_index ++;
    if(UART2_rx_index == BUFF_2_SIZE)
    {
      UART2_rx_index = 0;
    }
  }

  /* Interupt handler for sending single byte */
  if(USART_GetITStatus(USART2, USART_IT_TXE) != RESET)
  {
    USART_SendData(USART2, UART2_tx_buff[UART2_tx_index++]);
    if(UART2_tx_buff[UART2_tx_index-1] == 0x00 || UART2_tx_index ==
        BUFF_2_SIZE)
    {
      USART_ITConfig(USART2, USART_IT_TXE, DISABLE);
      UART2_tx_index = 0;
    }
  }
}
```

The second follow-up step of GPS data flow is the polling of the received data buffer. If this buffer has the completed message inside, then the message is handed to the parsing function. The polling approach has been chosen instead of asynchronous interrupt to ensure that the decoding of the message will be processed when there is enough processing time and no other task will be corrupted by the parsing procedure – because the parsing procedures depends on the type of GPS message and the consumed time is not deterministic.

```
while(1)
{
  gps_tmp = UART2_getByte();
  if(gps_tmp == -1){
    break;
  }
  else
  {
    if (gps_tmp != 0x0a)
    {
      GPS_msg[GPS_msg_buf_index] = (char)gps_tmp;
      GPS_msg_buf_index ++;
      if(GPS_msg[GPS_msg_buf_index-1]== 0x0d || GPS_msg_buf_index ==
          BUFF_GPS_SIZE)
      {
        GPS_msg[GPS_msg_buf_index] = 0x00;
        GPS_msg_buf_index = 0;
        readData_GPS(&GPS_msg[0],&gps_data);
      }
    }
  }
}
```

The top level of GPS data processing is parsing of GPS messages according to NMEA standard. The detailed description of this standard can be found in [27]. From the whole set of messages defined by this standard and generated by GPS receiver, only a few are important for this project. These messages are explained here.

NMEA consists of sentences, the first word of which, called a data type, defines the interpretation of the rest of the sentence. Each Data type would have its own unique interpretation and is defined in the NMEA standard. The most important NMEA sentences include the GGA which provides the current Fix data, the RMC which provides the minimum GPS sentences information, and the GSA which provides the Satellite status data.

GGA - essential fix data which provide 3D location and accuracy data:

```
Example:
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Explanation:
     GGA          Global Positioning System Fix Data
     123519       Fix taken at 12:35:19 UTC
     4807.038,N   Latitude 48 deg 07.038'␣N
␣␣␣␣␣01131.000,E␣␣Longitude␣11␣deg␣31.000' E
     1            Fix quality: 0 = invalid
                               1 = GPS fix (SPS)
                               2 = DGPS fix
                               3 = PPS fix
                               4 = Real Time Kinematic
                               5 = Float RTK
                               6 = estimated (dead reckoning) (2.3 feature)
                               7 = Manual input mode
                               8 = Simulation mode
     08           Number of satellites being tracked
     0.9          Horizontal dilution of position
     545.4,M      Altitude, Meters, above mean sea level
     46.9,M       Height of geoid (mean sea level) above WGS84
                      ellipsoid
     (empty field) time in seconds since last DGPS update
     (empty field) DGPS station ID number
     *47          the checksum data, always begins with *
```

GSA - GPS DOP and active satellites. This sentence provides details on the nature of the fix. It includes the numbers of the satellites being used in the current solution and the dilution of precision (DOP). DOP is an indication of the effect of satellite geometry on the accuracy of the fix. It is a dimensionless number where smaller is better. For 3D fixes using 4 satellites a 1.0 would be considered to be a perfect number, however for overdetermined solutions it is possible to see numbers below 1.0:

```
Example:
$GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39
```

```
Explanation:
     GSA       Satellite status
     A         Auto selection of 2D or 3D fix (M = manual)
     3         3D fix - values include: 1 = no fix
                                        2 = 2D fix
                                        3 = 3D fix
     04,05...  PRNs of satellites used for fix (space for 12)
     2.5       PDOP (dilution of precision)
     1.3       Horizontal dilution of precision (HDOP)
     2.1       Vertical dilution of precision (VDOP)
     *39       the checksum data, always begins with *
```

RMC - NMEA has its own version of essential gps position, velocity and time data:

```
Example:
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Explanation:
     RMC           Recommended Minimum sentence C
     123519        Fix taken at 12:35:19 UTC
     A             Status A=active or V=Void.
     4807.038,N    Latitude 48 deg 07.038'␣N
␣␣␣␣␣01131.000,E␣␣Longitude␣11␣deg␣31.000' E
     022.4         Speed over the ground in knots
     084.4         Track angle in degrees True
     230394        Date - 23rd of March 1994
     003.1,W       Magnetic Variation
     *6A           The checksum data, always begins with *
```

The code for parsing of GPS NMEA sentences is part of appendix, the structure is outlined here. The sentence is split to array of string tokens at first. Then the meaning of the first value is checked, and the data type of RMC, GSA or GGA is taken for next processing. If one of these type of sentences is detected the tokens are parsed according to standard described above.

```
void readData_GPS(char *GPS_buffer, STRUCT_GPS_DATA *gps_dat)
{
  if(GPS_buffer[0] == '$')
  {
    if(GPS_buffer[3] == 'R' && GPS_buffer[4] == 'M' && GPS_buffer[5] == 'C'
       )
    {
      endDecoding = 0;
      while (!endDecoding)
      {
        iBuf = 0;
        while(1)
        {
          ...
          /* Split the sentence to tokens divided by comma */
        }

        switch(Ipch)
        {
          case 1: // UTC
            (*gps_dat).UTC_GPS = atof(buffer);
          break;

          case 2: // Data valid?
            if(buffer[0] == 'A')
            {
              (*gps_dat).validData_GPS = 1;
            }
            else
            {
              (*gps_dat).validData_GPS = 0;
            }
          break;

          case 3: // Latitude
            tmp = atof(buffer);
            tmpDeg = floor (tmp/100);
            tmpMin = tmp - (tmpDeg*100);
            (*gps_dat).latitude_GPS = tmpDeg + (tmpMin/60);
          break;
          ...
          ...
        }
        Ipch++;
      }
    }
  }
}
```

After all the data processing the GPS packet is stored in structure called *gps_data*. From this structure the position data can be accessed for next processing.

```
typedef struct
{
  // GPRMC
  float UTC_GPS;
  int validData_GPS;
  double latitude_GPS;
  int north_GPS;
  double longitude_GPS;
  int east_GPS;
  float groundSpeed_GPS;
  float azimuth_GPS;
  int date_GPS;

  // GPGSA
  int dimensions_GPS;
  float PDOP_GPS;
  float HDOP_GPS;
  float VDOP_GPS;

  // GPGGA
  float timeOfLastFix_GPS;
  int SatsInView_GPS;
  float height_GPS;
} STRUCT_GPS_DATA;
```

6.2.2 Inertial measurement sensor through SPI

For the inertial measurement the LSM330 and MPU6000 is used in this project. Both of these chips integrates three axis accelerometer and three axes angular rate sensor. Both of these chips has digital output and are capable of communication through SPI or I2C. The communication over SPI with LSM330DL is described here.

The timing of the low level SPI communication is on figure (6.1).The LSM330DL SPI is a bus slave. The SPI allows to write and read the registers of the device. The Serial Interface interacts with the outside world with 4 wires: CS, SPC, SDI and SDO (SPC, SDI, SD0 are common). CS is the serial port enable and it is controlled by the SPI master. It goes low at the start of the transmission and goes back high at the end. SPC is the serial port clock and it is controlled by the SPI master. It is stopped high when CS is high (no transmission). SDI and SDO are, respectively, the serial port data input and output. These lines are driven at the falling edge of SPC and should be captured at the rising edge of SPC. These procedures are handled by SPI driver in the main microcontroller.

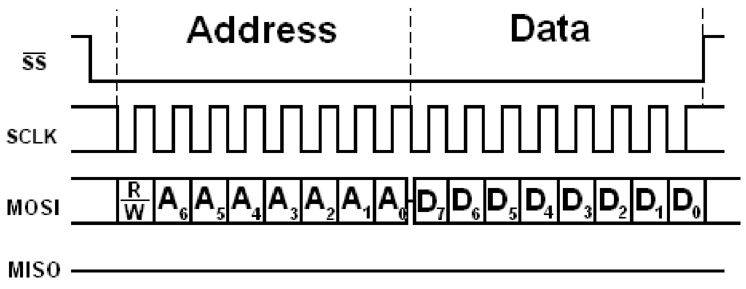


Figure 6.1: Timing of standard SPI Write cycle [28]

bit 0: RW bit. When 0, the data DI(7:0) is written into the device. When 1, the data DO(7:0) from the device is read. In the latter case, the chip will drive SDO at the start of bit 8.

bit 1: MS bit. When 0, the address will remain unchanged in multiple read/write commands. When 1, the address will be auto-incremented in multiple read/write commands.

bit 2-7: address AD(5:0). This is the address field of the indexed register.

bit 8-15: data DI(7:0) (write mode). This is the data that will be written into the device (MSb first).

bit 8-15: data DO(7:0) (read mode). This is the data that will be read from the device (MSb first).

The code which handles this functionalities in microcontroler side:

```
uint8_t SPI_SendByte(uint8_t byte)
{
  /* Loop while DR register in not empty */
  while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);

  /* Send byte through the SPI1 peripheral */
  SPI_I2S_SendData(SPI2, byte);

  /* Wait to receive a byte */
  while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == RESET);

  /* Return the byte read from the SPI bus */
  return SPI_I2S_ReceiveData(SPI2);
}
```

This code is using the standard peripheral libraries provided by STM, More information can be found in [25] . The LSM330 chip has to be initialized before it can be used for inertial measurements. This process consists of set of commands sent through the SPI bus. The whole set of supported commands to set up the chip can be found in the datasheet [17]. The procedure used in this project is:

```
void lsm330spi_init( void)
{
  GPIO_ResetBits(GPIOC, GPIO_Pin_12);
  SPI_SendByte(0x20);
  SPI_SendByte(0x57);
  GPIO_SetBits(GPIOC, GPIO_Pin_12);
  ...
  ...
}
```

At first the chip select trigger is selected for this particular chip. Then the address byte and data command is send to the chip – to configure inside registers. The table (6.1) with register which are configured follows.

Address	**Data**	**Comment**
0x20	0x57	
0x23	0x08	
0x20	0x6F	
0x23	0x10	

Table 6.1: LSM330 - Registers to be configured during initialization

Then the chip can be accessed with reading commands. With this commands, the accelerometer and angular rate measurements can be provided. The measurements are provided with 16 bits resolution, but the SPI can handle only one byte, thus the measurements has to be connected together. The process is explained in the following code:

```c
void lsm330spi_update( ins_meas * measurement){
  uint8_t buf;
  int16_t tmp;

  //accs
  GPIO_ResetBits(GPIOC, GPIO_Pin_12);   //select onchip accelerometer
  SPI_SendByte(0xA8);                   //select data register
  buf = SPI_SendByte(0x0);              //read value
  GPIO_SetBits(GPIOC, GPIO_Pin_12);
  tmp = buf;

  GPIO_ResetBits(GPIOC, GPIO_Pin_12);   //select onchip accelerometer
  SPI_SendByte(0xA9);                   //select data register
  buf = SPI_SendByte(0x0);              //read value
  GPIO_SetBits(GPIOC, GPIO_Pin_12);
  tmp |= buf << 8;
  measurement->accs.x = tmp * ACC_LSB;
  ...
  ...
}
```

Individual commands used for reading of measurements from the LSM330 are listed in table:

After the reading process is done the measurements are stored in data structure *ins_meas*. From this structure it can be accessed in higher level algorithms.

```c
typedef struct
{
  vec3 accs;  // [rads/s]
  vec3 gyrs;  // [m/s^2]

} ins_meas;

typedef struct
{
  float x;
  float y;
  float z;
} vec3;
```

6.2.3 FLASH memory through SPI

Flash memory is operated in similar way as inertial measurement sensor – through SPI bus. The memory is used mostly for data logging, thus only sequence write and read is implemented, no file system is used. The functions for cooperation with FLASH memory are implemented in module *micronFlash.c*. The physical layer of communication is using the same SPI driver as inertial measurement sensor, so the low level routines are in common for these chips. The commands which are used in this project are listed in the table (6.2), all other details can be found in datasheet [24].

Command	Code	Data bytes	Address bytes
Read	0x03	1 – inf	3
Write enable	0x06	0	0
Write disable	0x04	0	0
Page program	0x02	1-256	3
Bulk erase	0xC7	0	0

Table 6.2: FLASH memory N25Q064A - Registers used for manipulation with memory

Memory is connected in SPI extended mode, the connection can be seen on figure (6.2). The extended mode represents the basic SPI connection with only two data lines. Another option is Dual or Quad SPI connection, where the datalines ar multiplied to achieve higher data rate.

The example of code which is used to write single byte to memory follows.

```
void mem_write_en(int enable)
{
  GPIO_ResetBits(CS_port, CS_pin);
  if(enable)
  {
    SPI_SendByte(0x06); // write enable
  }
  else
  {
    SPI_SendByte(0x04); // write disable
  }
  GPIO_SetBits(CS_port, CS_pin);
}

void mem_write_byte(uint8_t value, uint32_t address)
{
  mem_write_en(1);
  GPIO_ResetBits(CS_port, CS_pin);
  SPI_SendByte(0x02);

  memcpy ( &address_bytes, &address, 4 ); // address
  SPI_SendByte(address_bytes [2]);
  SPI_SendByte(address_bytes [1]);
  SPI_SendByte(address_bytes [0]);

  SPI_SendByte(value);                    // data
  GPIO_SetBits(CS_port, CS_pin);
}
```

6.2.4 Analog to digital converters

AD converters are used for measuring the voltages, temperatures, currents an other information describing the state of the overall system, but also pressure measurements or other values necessary for autopilot control algorithms. The usage of sensors connected to one of the AD converters is simpler than using one of the buses

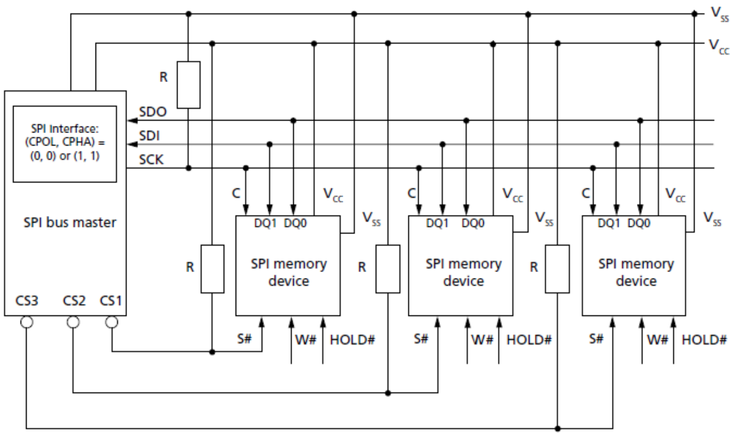


Figure 6.2: Micron Flash memory - connection to SPI bus [24]

described above.

Once the AD converter is properly initialized, the read function can be called directly. The output of this function is the analog value of voltage level with 12 bits of resolution. The switching of input channel is done by re-initialization of the converter with another settings. This approach is not the fastest, but it is deterministic and the consumed processing time of the whole conversion process can be estimated. The example below shows the usage of the ADC with basic conversion and filtration of analog value. In the end of the process the variable *voltmeter* contains the actual voltage of the onboard battery.

```
...
ADC_DeInit(ADC1);
ADC_channel = 4;
ADC_Inicializace();
ad_in3 = ADC1_Read();
ad_in3_filt = 0.85*ad_in3_filt + 0.15*(float)ad_in3;
voltmeter = ((float)ad_in3_filt/4096)*16.2;
...
```

6.2.5 Servo motor control

For manipulation of the control surfaces of the aeroplane (ailerons, elevatro, rudder) the RC model servo motors are used. The input signal for the servo motor is called *Pulse-position modulation signal (PPM)*. This signal has 50 Hz frequency and the driving pulse has typically duration of 1000 micro seconds to 2000 micro seconds. The time plot of the signal can be seen on fig (6.3).

The generation of the signal is handled by peripheral timer of the main micronotroller, so it consumes no additional computing time of main controller. The

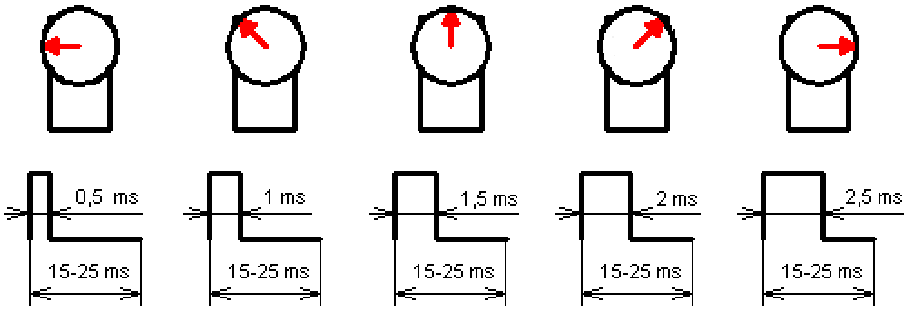


Figure 6.3: RC servo PPM signal [29]

timer is set up in the initialization phase, described above. The length of the driving pulse is changed through the control register of the timer. The code follows:

```
void set_servo(int whichOne, float angle)
{
  if(angle < SERVO_MIN)
  {
    angle = SERVO_MIN;
  }
  if(angle > SERVO_MAX)
  {
    angle = SERVO_MAX;
  }

  switch(whichOne)
  {
    case 1:
      TIM3->CCR1 = (int)((angle*9.44444)+650);
    break;
    case 2:
      TIM3->CCR2 = (int)((angle*9.44444)+650);
    break;
    ...
  }
}
```

6.3 Control Algorithms

6.3.1 AHRS Software

Precise and reliable estimation of orientation plays crucial role for unmanned vehicle. The most common solution to determination of the three orientation angles: pitch, roll, and yaw, relies on the AHRS that exploits inertial data fusion (accelerations and angular rates) with magnetic measurements. However, in real world applications strong vibrations and disturbances in magnetic field usually cause this approach to provide poor results. Therefore, we have devised a new approach to orientation estimation using inertial sensors only. It is based on modified complementary filtering and was proved by precise laboratory testing using rotational tilt platform as well as by robot field-testing. As it appears, the algorithm outperformes the commercial magnetometer-aided AHRS solutions [1].

The data fusion process can be described as follows (for the block scheme, see figure (6.4)):

First, the calibrated accelerations and angular rates are pre-filtered – various filter types were investigated to deal best with the vibrations.

Second, the coarse alignment algorithm is applied on the inertial data to obtain pitch and roll angles. Then, the angular rate data are numerically integrated; quaternions are used for attitude representation to ensure smooth rotations and save computations when chaining the rotations (compared to rotation matrices approach).

Third, angles obtained from the coarse alignment and by the integration are fused together using specially designed complementary filter [2], [3]. Fourth, results of the data fusion are fed back to the angular rates channel to ensure stable solution and to minimize the error due to noise integration and drift.

For more details of this method see Ref. [1].

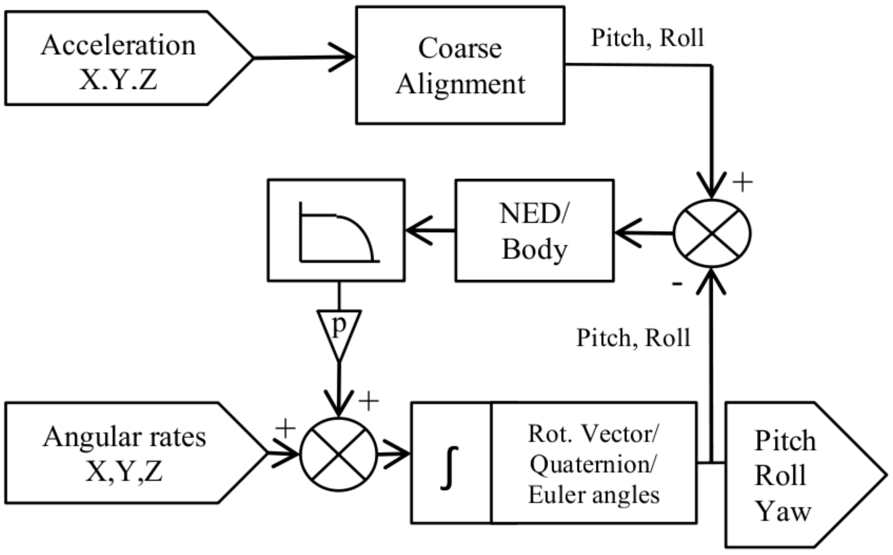


Figure 6.4: Block scheme representing the principle of the orientation determination algorithm: a complementary filtering approach that uses inertial data only. [1]

6.3.2 Stabilization Algorithms

The basic stabilization algorithms have been implemented in this project. Block scheme of used stabilization feedback loops for lateral and longitudinal motion is on figure (6.5). The first loops are angular-rate dampers, the subsequent feedback loop implement simple P control of attitude angles stabilization. The system is capable of tuning the controllers gains independently from ground control station.

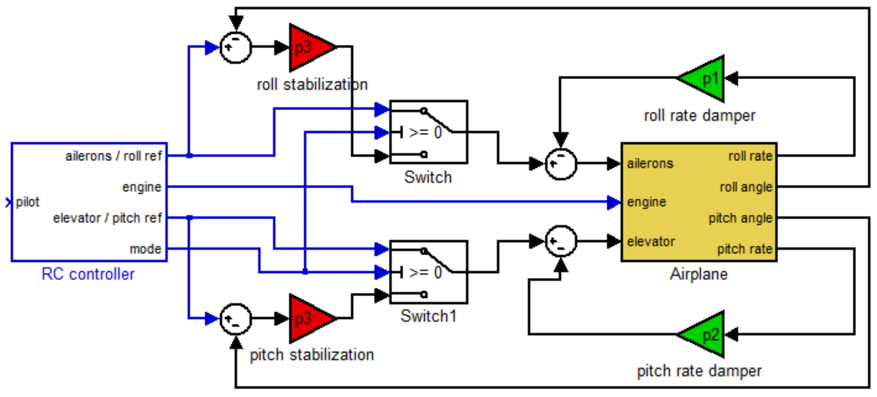


Figure 6.5: Block scheme representing stabilization algorithms.

```
float damper_P(float measure, float const_P)
{
  return - (measure * const_P);
}

float filt(float actual, float last, float filt)
{
  return actual * filt + last * (1 - filt);
}

float regulator_P(float reference, float measure, float const_P)
{
  return (reference - measure)*const_P;
}

float regulator_PI(float reference, float measure, float const_P,  float
    const_I, float *sum, float windup)
{
  float ret =0;
  ret = (reference - measure) * const_P + *sum * const_I;
  (*sum) += (reference - measure);
  if(*sum > windup)
  {
    (*sum) = windup;
  }
  if(*sum < -windup)
  {
    (*sum) = -windup;
  }
  return ret;
}
```

6.3.3 Navigation algorithms

The implementation of "cross track error" trajectory following algorithm is described in this section. The algorithm consists of three major subroutines: distance between two waypoints calculation, heading to the next waypoint calculation and cross track error calculation and minimization. The mathematical background for this algorithm is described in section 3.3.

Code of concrete implementation of distance between two waypoints algorithm follows. Inputs of the function *get_distance_to_WP* are GPS coordinates of two waypoints and the return value is the distance between them:

```
double get_distance_to_WP(double lat1, double long1, double lat2, double
    long2)
{
  ...
  lat1r = lat1*D2R;
  lat2r = lat2*D2R;
  long1r = long1*D2R;
  long2r = long2*D2R;

  x = (long2r-long1r) * cos((lat1r+lat2r)/2);
  y = (lat2r-lat1r);
  return sqrt(x*x + y*y) * (6378000);
}
```

Implementation of heading to the next waypoint calculation is in function *get_azimuth_to_WP* which returns the azimuth to the next waypoint:

```
double get_azimuth_to_WP(double lat1, double long1, double lat2, double
    long2)
{
  ...
  lat1r = lat1*D2R;
  lat2r = lat2*D2R;
  long1r = long1*D2R;
  long2r = long2*D2R;

  yParam = sin(long2r-long1r) * cos(lat2r);
  xParam = cos(lat1r)*sin(lat2r) - sin(lat1r)* cos(lat2r)* cos(long2r-
      long1r);
  ang = atan2(yParam, xParam)/D2R;

  if(ang < 0)
  {
    ang = ang + 360;
  }
  return ang;
}
```

6.4 Communication with Ground Segment

This section is describing the communication with the ground station, the used protocol and the functionalities that are responsible for executing of commands from ground and sending the telemetry.

The protocol used in this project is MAVlink protocol [30], which is an unofficial standard for the use in similar low cost unmanned vehicles project. The advantage of using standardized protocol is to have the option of using the open source Ground Station software for PC or mobile device, the disadvantage is higher percentage of overhead connected to this protocol.

MAVLink is a very lightweight, header-only message marshaling library for micro air vehicles. It can pack C-structs over serial channels with high efficiency and send these packets to the ground control station. The anatomy of one packet is inspired by the CAN and SAE AS-4 standards. The packet composition can be seen on Figure (6.6) and the content is explained in Table (6.3). More information and details about MAVlink communication protocol can be found in Ref. [30].

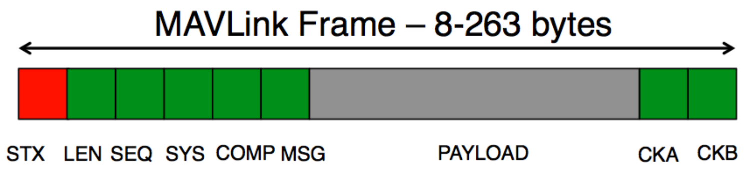


Figure 6.6: MAVlink - packet anatomy [30]

Byte	Content	Explanation
STX	Packet start sign	Indicates the start of a new packet.
LEN	Payload length	Indicates length of the following payload.
SEQ	Packet sequence	Each component counts up his send sequence. Allows to detect packet loss
SYS	System ID	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
COMP	Component ID	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
MSG	Message ID	ID of the message - the id defines what the payload “means” and how it should be correctly decoded.
Payload	Data	Data of the message, depends on the message id.
Checksum	Low byte, high byte	ITU X.25/SAE AS-4 hash

Table 6.3: MAVlink - Packet composition [30]

The set of telemetry packets used in this project is listed in the table (6.4). The whole set of commands which can be used as expansion, or if more functionalities from GS software are needed can be found in [30].

Packet type	**Comment**
HEART beat	The heartbeat message shows that a system is present and responding. The type of the MAV and Autopilot hardware allow the receiving system to treat further messages from this system appropriate (e.g. by laying out the user interface based on the autopilot).
ATTITUDE	The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-right).
NAV	Outputs of the APM navigation controller. The primary use of this message is to check the response and signs of the controller before actual flight and to assist with tuning controller parameters.
POSITION	The position is in GPS-frame (right-handed, Z-up). It is designed as scaled integer message since the resolution of float is not sufficient.
VFR HUD	Metrics typically displayed on a HUD for fixed wing aircraft.
SYSTEM STATUS	The general system state.

Table 6.4: MAVlink - Packet types used in this project [30]

The sw module which is responsible for handling telemetry packets is called *mavlink_telemetry.c*. This module covers the user functionalities of the MAVlink protocol, and it is using library for encapsulating the packets. More information about the library can be found in [30]. The example of creating the packet follows:

```
...
// MAVLINK NAV
nav.nav_roll = (&ins_dataset)->EUL.x*D2R;
nav.nav_pitch = (&ins_dataset)->EUL.y*D2R;
nav.nav_bearing = (&ins_dataset)->EUL.z*D2R;
nav.target_bearing = angle_to_WP;
nav.wp_dist = distance_to_WP;
nav.alt_error = altitude_err;
nav.xtrack_error = 0;
nav.aspd_error = 0;

mavlink_msg_nav_controller_output_encode(system_id, MAV_COMP_ID_IMU, &msg
    , &nav);
mav_len = mavlink_msg_to_send_buffer(buf, &msg);

sendUART1((char*)&buf[0],mav_len);
...
```

Chapter 7

Ground Control Station

7.1 GCS - Hardware description

The second essential part of the whole system for UAV is Ground station. In this project ground station consists of three individual parts: communication hardware, computer with visualisation software and RC transmitter used as joystick. This section describes the developed hardware. This ground station is designed to be the main node for interaction between the user, onboard control unit and software ground system running on computer or Android mobile device.

One of the main tasks of the GCS is to replace the original transmitter of RC system and provide duplex communication channel with onboard control unit. The GCS must be robust and reliable, because it is the only way how to communicate with flying UAV. This is the reason why complex hardware ground station is used instead of only xBee-USB adaptor used in similar projects.

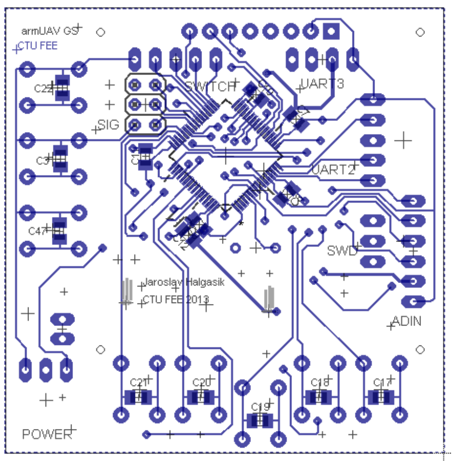
GCS is based on STM32F100RB processor, it is equipped with graphical display, set of trimmers and switches. GCS is equipped with 3x USART lines for x-bee module, bluetooth module and PC connection.

The schematic for the GCS can be seen in Appendix (D), the PCB layout of GCS is on Fig. 7.1a and 7.1b and the picture of GCS connected to the RC transmitter is shown on Fig. 7.2a, 7.2b, 7.3.

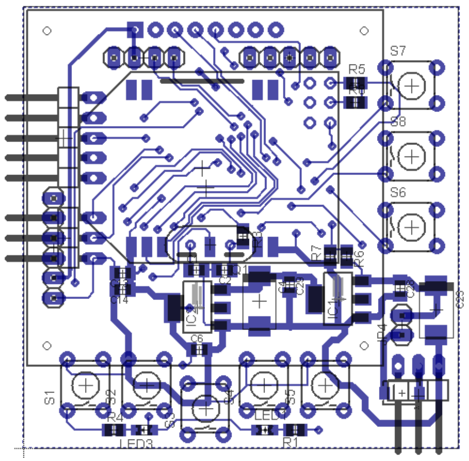
7.2 GCS - Software and algorithms

The ground part of the software is largely based on the flight onboard software. The peripherals initialization and other low level functionalities are similar.

The main function of the ground station is to provide central node between flying



(a) Bottom layer



(b) Top layer

Figure 7.1: Ground Control Station PCB layout

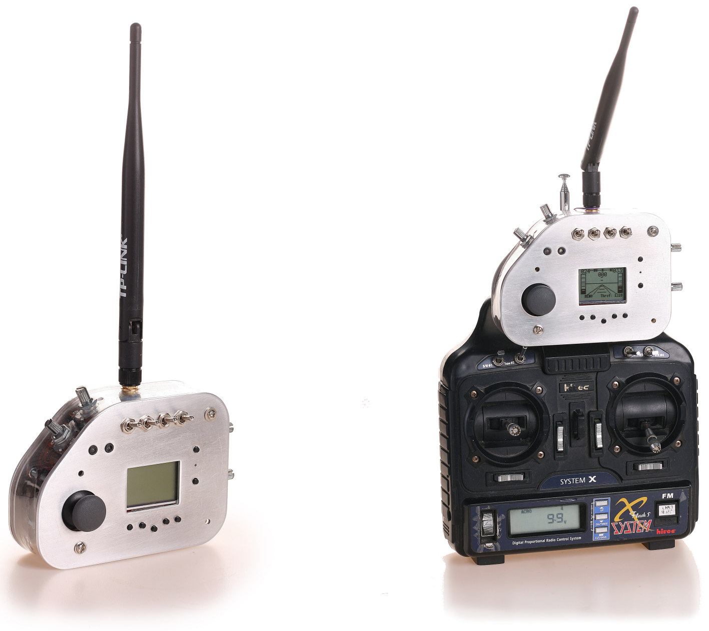


Figure 7.2: Ground Control Station Prototype

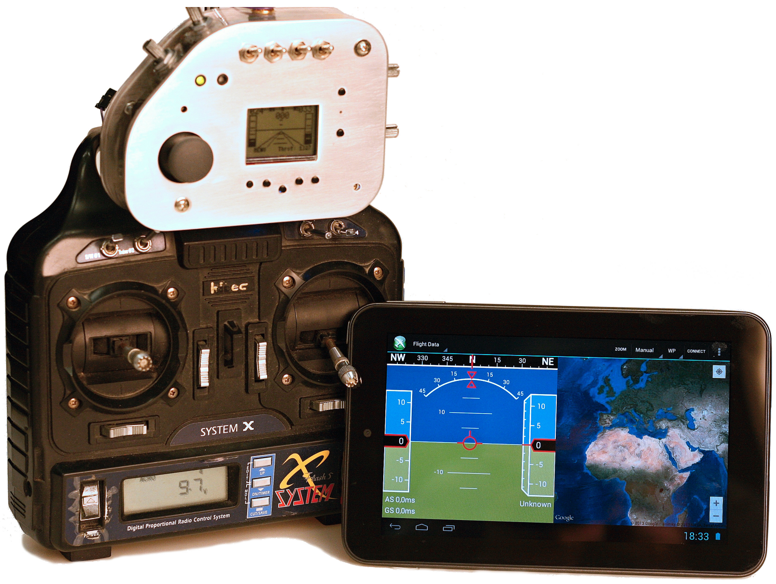


Figure 7.3: Ground Control Station connected to RC transmittter and Android tablet

vehicle, personal computer with visualisation software on the ground and the user / pilot. Thus the ground station has user interface with LCD display, set of trimmers and switches and can be connected to RC transmitter. The communication with the display and the communication with the RC transmitter is explained in this section. USART, analog to digital converter, switches and other functionalities are similar to the flight software and are explained above.

7.2.1 Graphical display

The graphical display used in this project is PCD8544, it is 48 x 84 pixels matrix LCD with driver. This display has been used in previous generation of mobile phones. The physical layer of communication between display driver and main microcontroller is similar to SPI, but it has some special differences. The pins which has to be connected and operated to ensure the communication are listed in the table 7.1. More information about connection can be found in datasheet [31].

For the low level functionalities of the display, the open source library has been used. The library has been created for Arduino, thus some changes has to be done, to proper function on microcontroller used in this project. The pin definition and initialization has been modified, the result can be seen below. Also the function for the lowest level of sending a byte to the display driver had to be changed and is listed below. More information about the library can be found in Ref. [32].

Pin function	Connection to mpc	Comment
Enable	Port C, pin 12	The enable pin allows data to be clocked in. The signal is active LOW.
Reset	Port C, pin 11	This signal will reset the device and must be applied to properly initialize the chip. The signal is active LOW.
Command	Port C, pin 6	Input to select either command/address or data input.
Data	Port A, pin 7	Input for the data line.
Clock	Port A, pin 5	Input for the clock signal: 0.0 to 4.0 Mbits/s.

Table 7.1: PCD8544 LCD - Connection [31]

```
void InitLCD(int contrast)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  if (contrast>0x7F)
    contrast=0x7F;
  if (contrast<0)
    contrast=0;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
  GPIO_InitStructure.GPIO_Pin = LCD_PIN_DATA | LCD_PIN_CLOCK;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  GPIO_InitStructure.GPIO_Pin = LCD_PIN_ENABLE | LCD_PIN_RESET |
      LCD_PIN_COMMAND;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  //resetLCD;
  GPIO_SetBits(GPIOC, LCD_PIN_RESET);
  Delay(10);
  GPIO_ResetBits(GPIOC, LCD_PIN_RESET);
  Delay(10);
  GPIO_SetBits(GPIOC, LCD_PIN_RESET);

  _LCD_Write(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION, LCD_COMMAND
      );
  _LCD_Write(PCD8544_SETVOP | contrast, LCD_COMMAND);
  _LCD_Write(PCD8544_SETTEMP | LCD_TEMP, LCD_COMMAND);
  _LCD_Write(PCD8544_SETBIAS | LCD_BIAS, LCD_COMMAND);
  _LCD_Write(PCD8544_FUNCTIONSET, LCD_COMMAND);
  _LCD_Write(PCD8544_SETYADDR, LCD_COMMAND);
  _LCD_Write(PCD8544_SETXADDR, LCD_COMMAND);
  _LCD_Write(PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL, LCD_COMMAND);
  clrScr();
  update();
  cfont.font=0;
}
```

```
void _LCD_Write(unsigned char value, unsigned char dataOrCommand)
{
  int i;
  if (dataOrCommand)
    GPIO_SetBits(GPIOC, LCD_PIN_COMMAND);
  else
    GPIO_ResetBits(GPIOC, LCD_PIN_COMMAND);

  GPIO_ResetBits(GPIOC, LCD_PIN_ENABLE);

  for (i = 0; i < 8; i++)
  {
    GPIO_ResetBits(GPIOA, LCD_PIN_CLOCK);
    if (value & 0x80)
      GPIO_SetBits(GPIOA, LCD_PIN_DATA);
    else
      GPIO_ResetBits(GPIOA, LCD_PIN_DATA);
    value <<= 1;
    GPIO_SetBits(GPIOA, LCD_PIN_CLOCK);

  }
  GPIO_SetBits(GPIOC, LCD_PIN_ENABLE);
}
```

7.2.2 Communication wit RC transmitter

The RC transmitter is connected to the ground station through the standard trainer cable. This port is used by default to connect two transmitters together to have set for trainer and trainee. The signals for servos are coded in this signal and only one of the transmitter is broadcasting these commands. In this project this signal is read by the ground station electronic, parsed and used as input values for the autopilot onboard. The signal is PPM (Pulse-position modulation), it is similar to the signal for servo control, but more channels are multiplexed to the one signal. The communication is one way – from transmitter to the ground station, no signal is going in another direction. The time plot of this signal can be seen on the Fig. 7.4. The signal is connected to the input of ground station PPMIN (Port B, Pin 6).

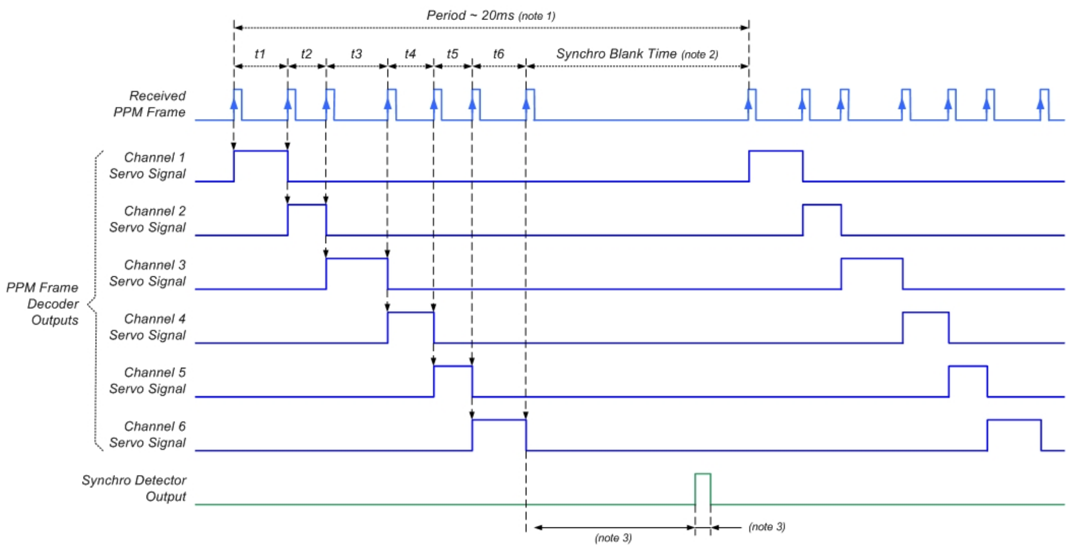


Figure 7.4: Pulse-position modulated signal from RC transmitter [33]

Chapter 8

Experimental results

Experiments and test flights has been made to proof the overall functionality of the electronic system and sensors subsystem, the capability of controlling a flying vehicle in real environment. Since the goal of this project is not to develop a concrete single autonomous vehicle, but the controller unit itself, the tests has been made with different platforms – fixed wing aircraft with classical configuration, flying wing, rover and blimp. The pictures and short description of used platforms follows, the experiments are listed next in this chapter.

8.1 Communication subsystem - xBee modules

As the main communication module for this project the xBee modules has been chosen. The advantages of this system are low price, high data rate and most of all the compatibility through the product family. This means that the modules can be changed and different frequency, data rates or ranges can be used. In this project three independent pairs of modules has been considered – xBee Pro 2.4 GHz 50mW Series 2.5, xBee Pro 50mW GHz Series 1 and xBee Pro 868MHz. More information about these modules can be found in Ref. [14].

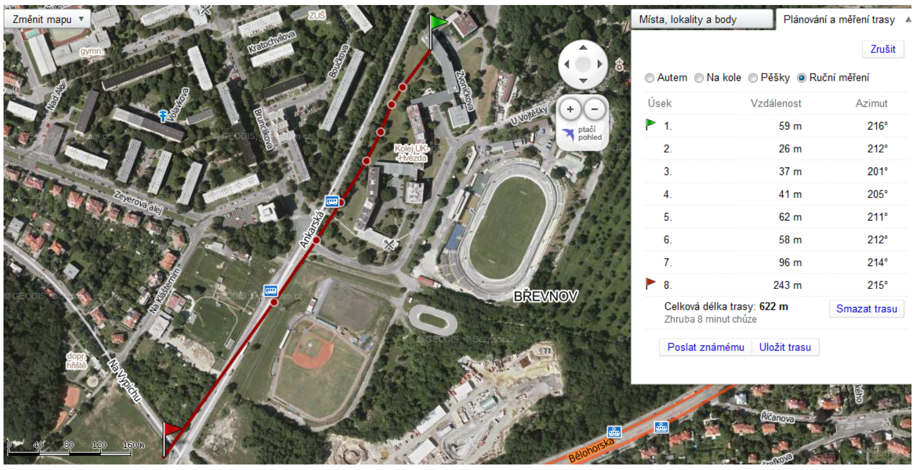
The range of these modules has been tested. Tests has been made with three different position of dipole antennas – parallel, cross polarization and perpendicular. Used antennas are listed in table (8.1). The RSSI has been measured by modules itself, so it is not the precise value of the attenuation in dB, but it can be used as criteria in tests and decision proces. The measurements has been made in the real environment of city, with the affection of the transmitters in th surrounding like WiFi spots, GSM cells and other. The location situation can be seen on the Fig. 8.1.

The result of this test shows, that the most suitable module for UAV project is module xBee PRO 868 MHz. This module is able to achieve solid data link in the real environment with interferences and noise to range more than 600m. The datasheet value of the range is up to 10km, but it is possible only in ideal conditions with high gain antennas. The plots of these measurements are on Fig. 8.2.

The xBee PRO modules for 2.4 GHz has similar range characteristic in compare to each other. The range achieved in configuration with 5dB and 8dB antennas is over 200m. The datasheet value of range for ideal condition is up to 1600 m. The module xBee S2B is more suitable for application where more then two modules are communicating together, like mesh nets or routed networks. For the point to point application needed in the UAV project the protocol overhead and meta data is disadvantaging. When the connection is lost between these modules, the time needed for the recovery is more than 200 ms, which is too high for manual aircraft control. The plots which compares all modules can be seen on Fig. 8.3.

xBee	868 MHz Ground	868 MHz Onboard	2.4G S1 Ground	2.4G S1 Onboard	2.4G S2 Ground	2.4G S2 Onboard
Antenna	5 dB	3 dB	8 dB	3 or 5 dB	8 dB	3 or 5 dB

Table 8.1: XBee - Dipole antennas used for range testing



Úsek	Vzdálenost	Azimut
1.	59 m	216°
2.	26 m	212°
3.	37 m	201°
4.	41 m	205°
5.	62 m	211°
6.	58 m	212°
7.	96 m	214°
8.	243 m	215°

Celková délka trasy: **622 m**
Zhruba 8 minut chůze

Figure 8.1: XBee range testing - situation map

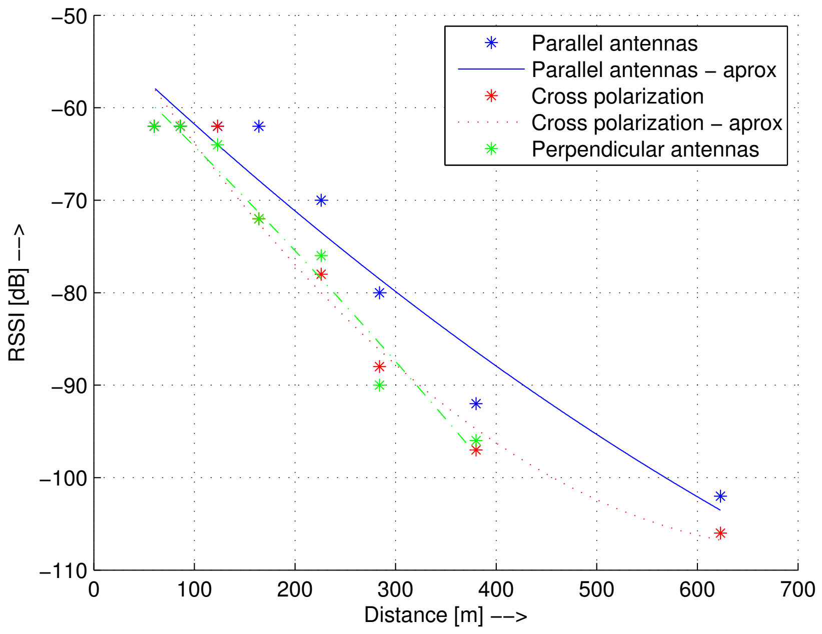


Figure 8.2: xBee PRO 868 MHz range testing - attenuation for different antenna positions

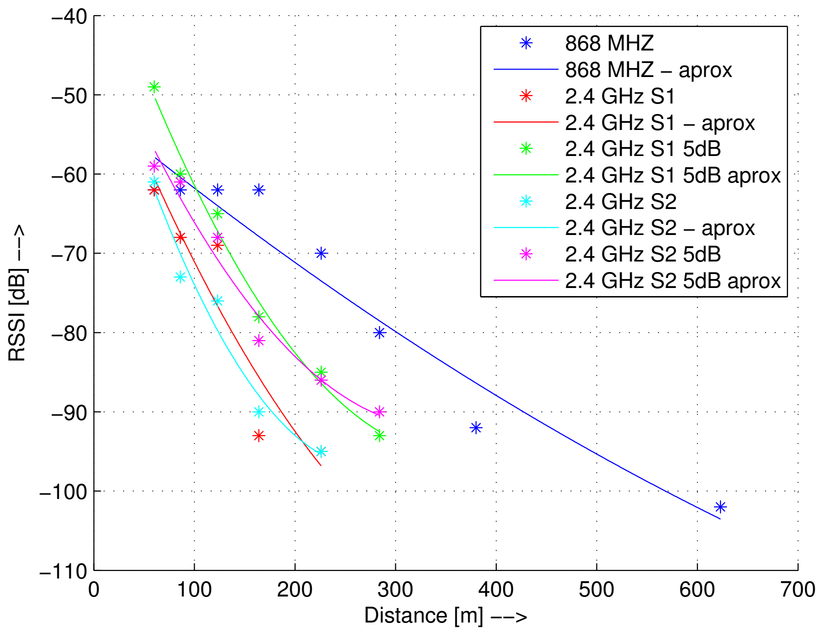


Figure 8.3: xBee PRO range testing - attenuation for different modules and antenna positions

8.2 Attitude and heading reference system - verification

The algorithm for Attitude and heading reference system (AHRS) calculation is described in section 6.3.1, more details can be found in Ref. [1], [2], [3]. The AHRS algorithm is one of the fundamental subsystem for the successful control of UAV. The testing of this subsystem has been performed in the first place. The airplane has been driven manually, the onboard FCU calculated the Euler angles and stored it to the SD card. The onboard video of the flight has been taken, and it has been used as reference for Euler angles in the post-processing comparison. The plot from this experiment can be seen on Fig. 8.4. The blue line on this plot is Roll angle computed with the algorithm of data fusion from angular sensors and accelerometers from section 6.3.1, the red line also the Roll angle, but computed only from accelerometers measurement. The differences can be seen after the start of the airplane, method using only accelerometers is not able to calculate proper Roll angle - because of the centrifugal force in steady turn. The video from this experiment can bee seen in Ref. [35].

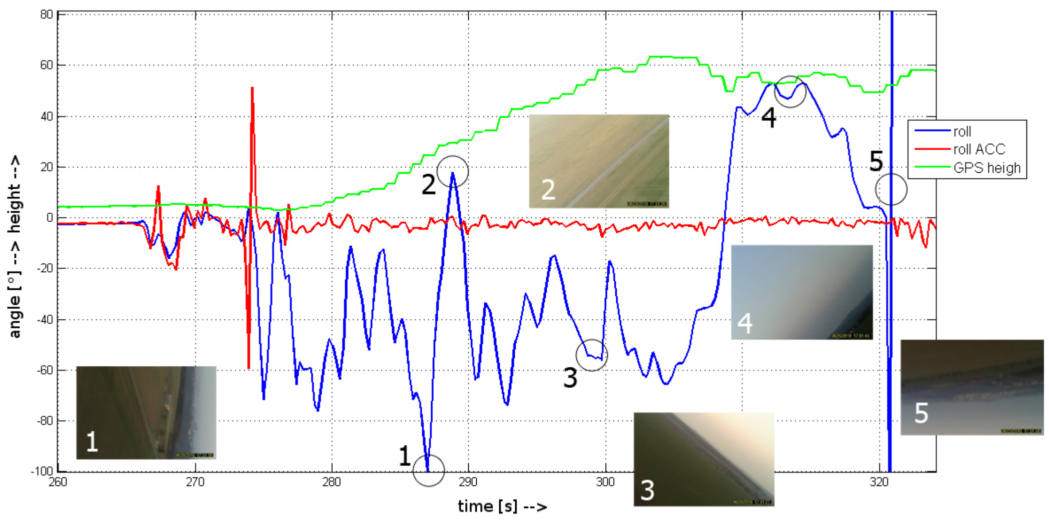


Figure 8.4: Attitude and heading reference system - verification flight in real environment

8.3 Airspeed measurement and Angle of Attack probe

The next sensors which has to be calibrated and verified before they can be used for control loop are Pressure airspeed sensor and Angle of Attack probe. The description of these sensors is in section 5.2.6.

The plot of airspeed measured with Pitot-static probe can be seen on Fig. 8.5. The measurement has been done in the wind tunnel, constant airspeed has been sustained. It can be seen, that the noise coming from the probe is more significant in higher airspeed. The processing of the airspeed has to contain low pass filtering.

On the Fig. 8.6, 8.7 are plots from Angle of Attack probe (photo can be seen on Fig. 5.10. The angle of attack has been changed continuously during the experiment from 0° to 5° and then from 0° to -5°. The noise from this probe is very high because of the oscillations of the developed construction of the probe.

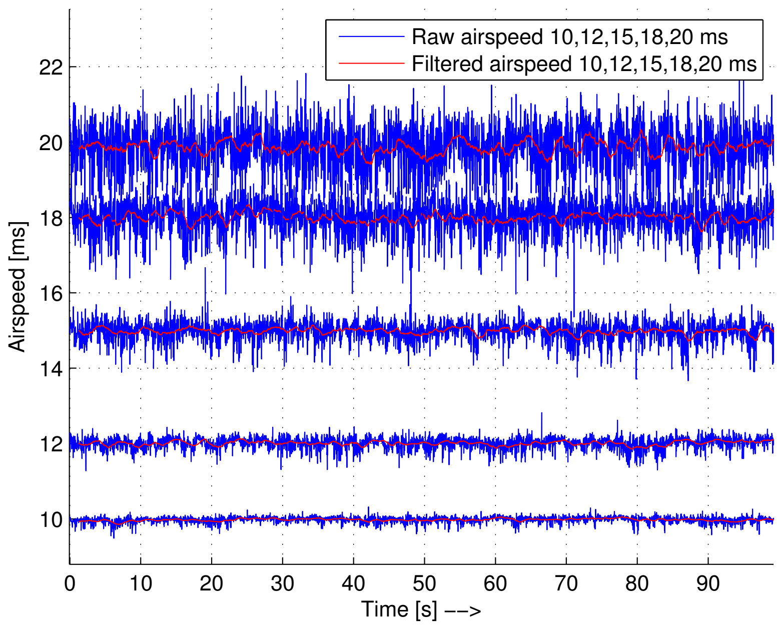


Figure 8.5: Pitot-static probe calibration

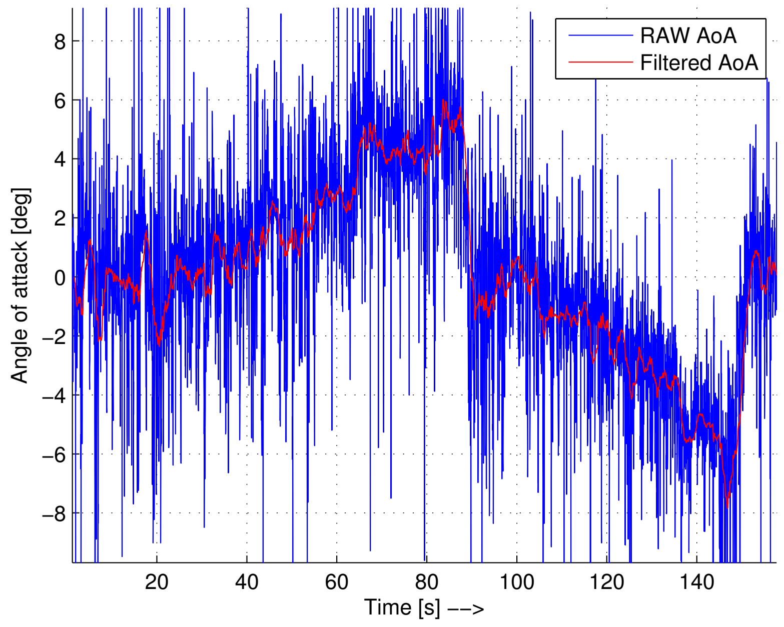


Figure 8.6: Angle of Attack probe calibration - 10 m/s

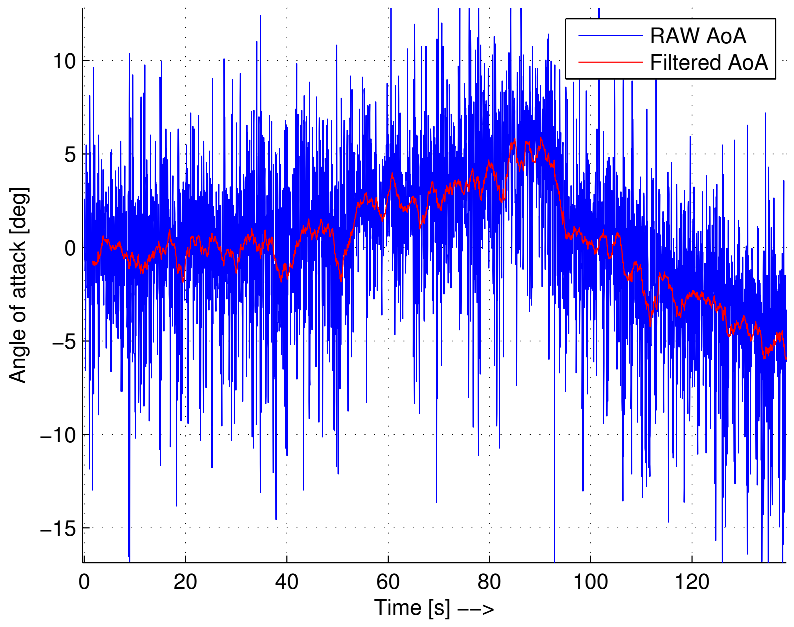


Figure 8.7: Angle of Attack probe calibration - 20 m/s

8.4 Control loop verification and PID tuning

The control loop used in this project is described in section 6.3.2. The experimental flight has been performed to verify the functionality of this loop and to tune the PID regulators. At first the dampers has been tuned and tested. The behavior of airplane flying on manual mode and with roll damper turned on (in time 501 s) can be seen on Fig. 8.8. The high frequency oscillations of the airplane are filtered out by the roll damper and the reactions are smoother and the airplane is more stable.

The second part of this experiment is to tune and verify the stabilisation loops. In this flight mode, the set point for the Euler angles are set from ground and the regulator has to hold the airplane in the desired position. Thus there is no direct connection between pilot and ailerons or elevator. The Fig. 8.9, 8.10 shows the tuning of the roll stabilization, Fig. 8.11 shows the response of the airplane to some casual input. Fig. 8.12 shows the autonomous start of the airplane - fixed pitch angle has been set up.

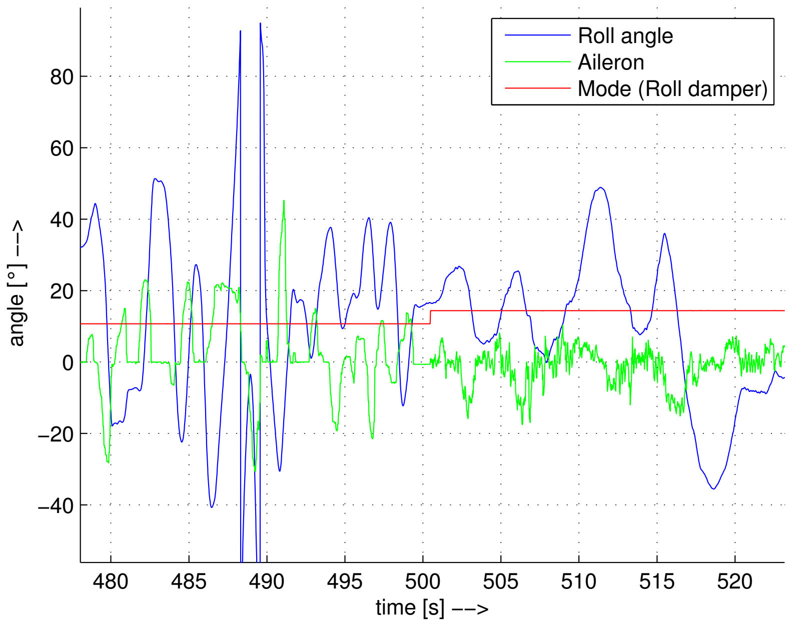


Figure 8.8: Roll damper - turned on in time 501 s

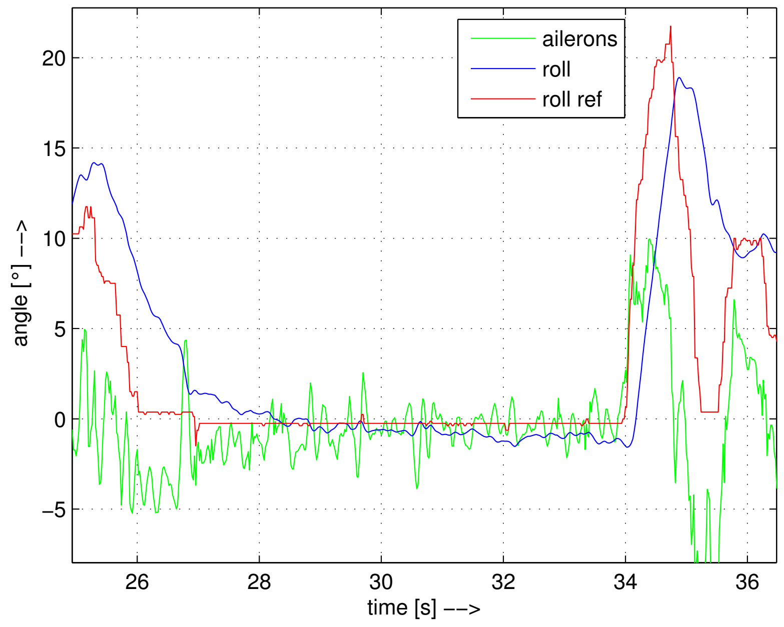


Figure 8.9: Tuning of the roll stabilization - well tuned regulator

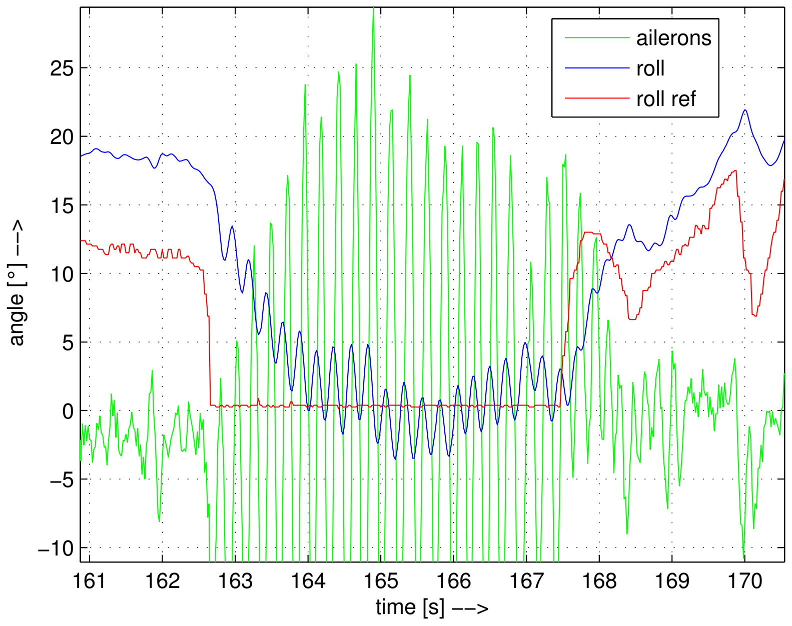


Figure 8.10: Tuning of the roll stabilization - oscillations caused by the high gain

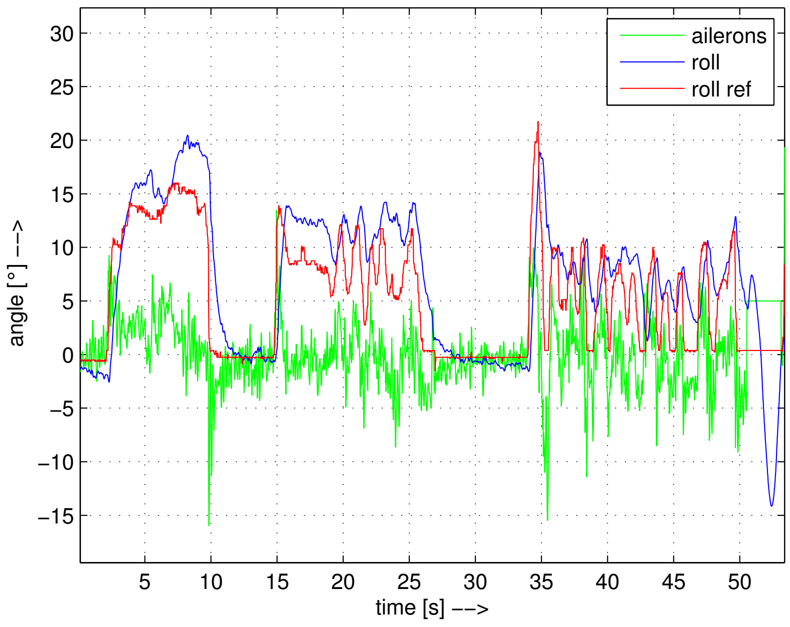


Figure 8.11: Response of roll stabilization to input from pilot

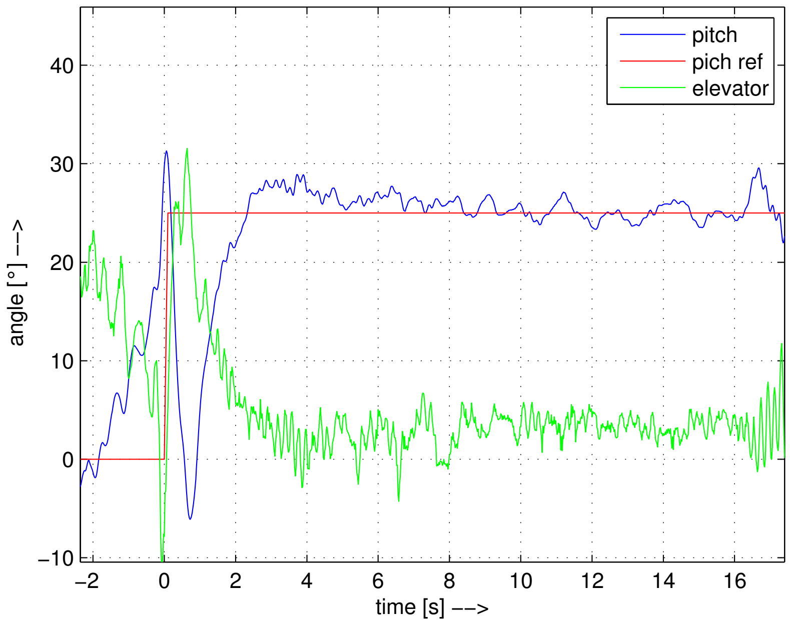


Figure 8.12: Autonomous start of the airplane

8.5 GPS Waypoint navigation

The next experiment flight has been performed with the airship. The navigation algorithm described in section 6.3.3 has been tested and verified here. The plot on Fig. 8.13 shows the trajectory defined by waypoints and the flight of the airship.

Some intermediate results used inside of the algorithm can be seen on Fig. 8.14. This plot corresponds to one round of the overall flight. Blue line shows the distance to the next waypoint, it can be seen that it is getting smaller when the airship is approaching the next waypoint,

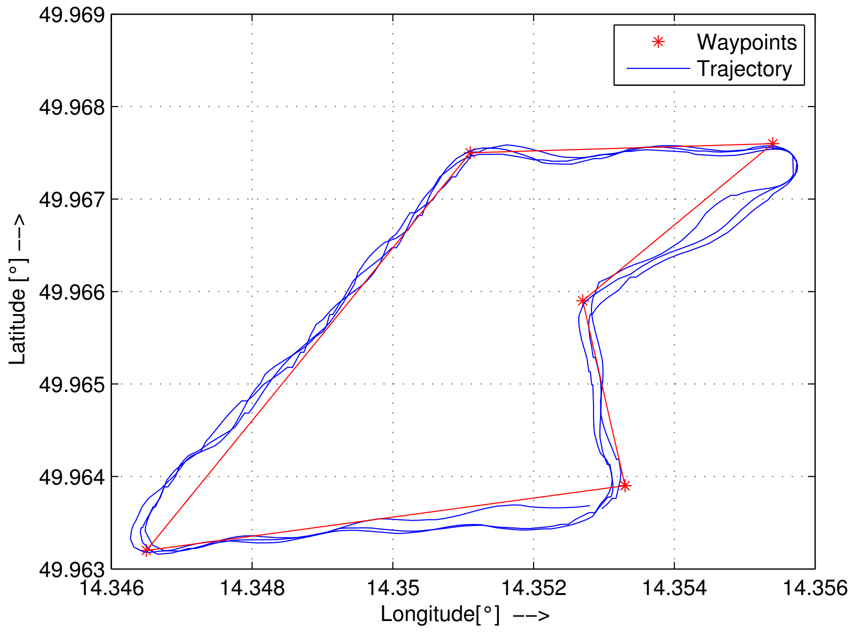


Figure 8.13: Autonomous flight of the airship

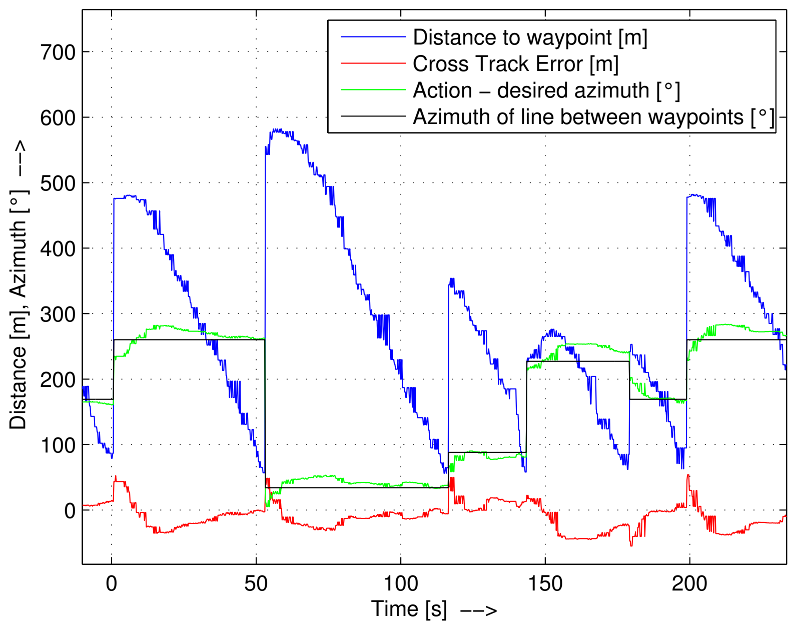


Figure 8.14: Autonomous flight of the airship - intermediate results

Chapter 9

Following projects

This section describes the projects which are connected to the developed Flight control unit. First project is students UAV, created by the students of mechanical and electrical engineering at Czech Technical University in Prague (CTU) in cooperation with the Aerospace Research and Test Establishment (VZLU) in Prague. This project uses both the hardware and software part of the developed FCU. Second following project is the Airship created by CTU in cooperation with private company AirshipClub [36]. The AirshipClub has developed its own flight control system, it uses only SW and algorithms developed in this project.

9.1 Modular UAV

The purpose of the project is to design, develop and produce a small autonomous flying vehicle. On the project are involved Josef Novák (CTU FEE), Petr Pahorecký (CTU FEE), Petr Adámek (CTU FME) and Pavel Hospodář (VZLU). More informations about the project can be found in Ref [38].

UAS developed and targeted by this project is meant to be used primarily as a flying platform for subsequent research and educational activities by CTU and VZLU staff. The project involves construction of a flight mechanics mathematical model, simulation of dynamical behavior, development of a control hardware unit, design of control laws, mechanical construction of the aircraft and the flight tests and measurements at the final foreseen stage. Modularity of the mechanical design, hardware components as well as the software components is the main theme throughout the project.

For the aircraft design, number of preliminary specifications have been set:

- Ability of autonomous flight

- Forward looking camera placed on remotely controlled tilt platform
- Minimal endurance 30 minutes
- Maximum weight 10 kg
- Unconventional construction
- Maximum wing span of 2 m and space to install 6-component internal balance, so that the flying prototype can be tested in 3 m Low Speed Wind Tunnel at VZLU

The design of the aircraft can be seen on Fig. 9.1.

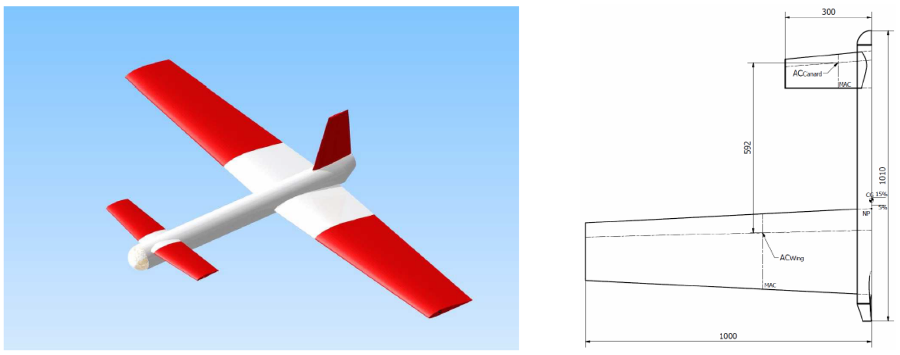


Figure 9.1: Design of aircraft - following project [38]

9.2 Airship

Second following project is the Airship created by CTU in cooperation with private company AirshipClub [36]. The AirshipClub has developed its own flight control system, it uses only SW and algorithms developed in this project. The experimental flight which has been realized with this airship is described in the section 8.5. The photo of airship can be seen on Fig. 9.2.

9.3 Documentation and online resources

Described project has been created as open source. All the source codes, hardware schematics and layout design, mechanical parts and description can be found online - Ref. [37]. This decision has been made to allow the usage of our flight control system to a broad spectrum of users, primarily it will be used for educational and research purposes on Czech Technical University, but there is no need to restrict

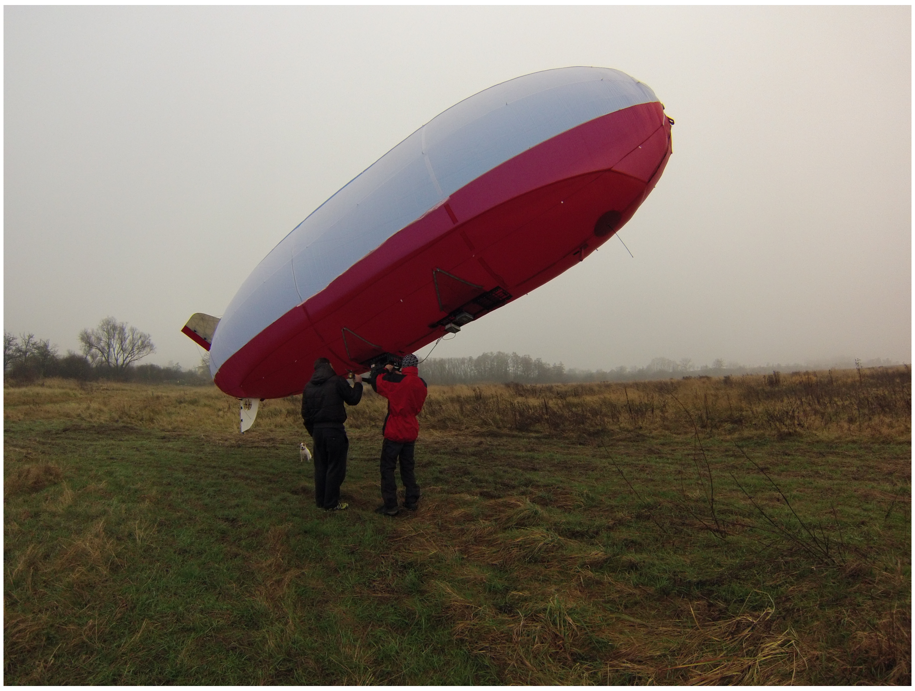


Figure 9.2: Airship - following project

the access for another users. On the other hand this approach opens the possibilities of some kind of community development or at least testing and bug tracking by external users.

The intermediate stages project has been presented on Pegasus-AIAA Student Conference, the conference papers can be found in Ref. [38] and are also attached to this work in digital form on CD.

Chapter 10

Conclusion

The goal of this thesis is to design and develop an electronic flight control unit for a small unmanned aerial vehicle, which can be used for research and educational activities at the Department of Control Engineering. The category of small UAVs includes fixed wing aircraft as well as helicopters, multi-rotor vehicles or airships or other types of vehicles with take-off weight lower than 5kg. The control unit has been designed, tuned and verified on multiple different autonomous vehicles. The final design of control unit, assembly layout, source codes and other technical details can be found in this thesis as well as the results of testing and experimental flights performed on multiple different airframes. The detailed technical materials needed for manufacturing of new flight control units can be found on attached CD-ROM and are available online [37].

Achieving of the defined goal - design of flight control unit (FCU) - was preceded by set of necessary subsequent steps. The overview and summary of state-of-the-art of existing FCU solutions has been done. The advantages of these units has been considered and the solution best fitting to the needs of research activities on CTU has been chosen and developed. The hardware and the software for the whole autonomous system including FCU, the Ground station and visualisation software for PC has been developed in this work, the requirements defined by CTU has been considered as well as the applicable features of available solutions.

One of the major issue of developing hardware FCU is to manage the reliable connection between a flying UAV and a ground station and a pilot. The solution based on xBee modules has been chosen for this purpose, functionalities of this wireless modem has been tested and the results are part of Chapter 8. The communication based on the digital communication channel only, with no need of RC transmitter, has been chosen for this project. This approach enables higher possibilities of usage the finished FCU for activities like formation flight testing, where multiple RC transmitters can be inconveniencing.

Two versions of hardware of the control unit has been designed and their description can be found in this thesis. Both versions are based on similar components, the version 1 is designed with modularity, tweak-ability and extensibility on mind. This approach enables the educational usage of the control unit, the sensors, computational unit and other subsystems can be developed and upgraded separately within the semester works or other thesis. The version 2 has been designed with emphasis on high reliability, small dimension and high level of integration. This approach fits better for flight tests in real environment, because the probability of malfunction caused by vibrations or other external influence is smaller.

The hardware for ground segment has also been developed in this work. The purpose of this subsystem is to provide reliable connection node between a flying air vehicle, a pilot and a computer. It is independent on used computer or operation system malfunctions and it provides basic control functions so the aircraft can be operated safely even if the PC is disconnected or halted. The hardware and mechanical design of the ground station is created with mechanical robustness, small size and low weight on mind.

The software consisting of multiple modules has been created in this work, the functionalities, which are necessary for an autonomous flight, has been implemented. This includes the Attitude and heading reference system based on MEMS accelerometer and angular rate sensor, the hierarchical control loops of autopilot, the communication protocol based on MAVlink, the navigation control loop using GPS and other functions necessary for autonomous flight has been implemented and are described in detail in this thesis. All these functions has been tested in the real environment during test flights and the results can be found in section 8.

Additional work can be done in the future or some other projects can be build with flight control unit developed in this thesis. Implementation of higher algorithms as LQ controller, Kalman filter data fusion or complex trajectory planning can be implemented in the future as semestral work or follow-up thesis. Another option of improvement is to use real time operating system on the onboard flight unit. More sensors can be connected to the system in the future for example ultrasonic rangefinder can be used for autonomous landing procedure.

All the objectives and the requirements defined in the assignment of this thesis has been accomplished. The final system designed and created in this work has been tested in real flight experiments and is ready now for implementation in another educational or research projects.

Bibliography

[1] Kubelka, V.; Reinstein, M.; , "Complementary filtering approach to orientation estimation using inertial sensors only," Robotics and Automation (ICRA), 2012 IEEE International Conference on , vol., no., pp.599-605, 14-18 May 2012

[2] M. Sotak, "Coarse alignment algorithm for ADIS16405," Przeglad Elektrotechniczny, vol. 86, no. 9, pp. 247-251, 9 2010.

[3] E.-H. Shin, Accuracy Improvement of Low Cost INS/GPS for Land Applications, M.S. thesis, Department of Geomatics Engineering, University of Calgary, December 2001.

[4] Doc. Ing. Zdislav Pech, CSc., Ing. Vratislav Věk, CSc. *Systémy řízení letu* 2006, Česká technika - nakladatelství ČVUT, Praha

[5] R.C.Nelson, *Aircraft stability and automatic control,* McGraw Hill, 1998

[6] Matlab *Software* MathWorks,
http://www.mathworks.com/products/matlab/

[7] RQ11-Raven
http://www.army-technology.com/projects/rq11-raven/

[8] Pixhawk PX4
https://pixhawk.ethz.ch/px4/modules/px4fmu

[9] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. Autonomous Robots (AURO), 2012.

[10] Ardupilot APM
http://code.google.com/p/ardupilot-mega/wiki/Introduction

[11] Openpilot
http://www.openpilot.org/products/openpilot-Revolution-platform

[12] Paparazzi
http://wiki.paparazziuav.org/wiki/MainPage

[13] Datasheet: STM32F100x4 STM32F100x6 STM32F100x8 STM32F100xB
http://www.farnell.com/datasheets/1690143.pdf

[14] Product Manual v1.xEx - 802.15.4 Protocol
http://www.farnell.com/datasheets/1681367.pdf

[15] GPRS/GSM Shield - EFCom
http://www.hwkitchen.com/products/gprs-gsm-shield-efcom/

[16] Invensense MPU6050 Datasheet
http://www.farnell.com/datasheets/1788002.pdf

[17] STM LSM330 Datasheet
http://www.farnell.com/datasheets/1458691.pdf

[18] HONEYWELL HMC5883L
http://www.farnell.com/datasheets/1683374.pdf

[19] MediaTek MT3329
https://docs.google.com/fileview?id=0BdHj7E2weiiNmU

[20] MPX4115A
http://www.farnell.com/datasheets/193192.pdf

[21] MPXV7002DP
http://www.farnell.com/datasheets/1718766.pdf

[22] LD1117S33TR
http://www.farnell.com/datasheets/1776449.pdf

[23] STM32VLDISCOVERY - STM32F100
http://www.st.com/st-web-ui/static/active/en/resource/ technical/document/user_manual/CD00267113.pdf

[24] Micron N25Q064A
http://www.farnell.com/datasheets/1674440.pdf

[25] STM32VLDISCOVERY firmware and examples
http://www.st.com/web/en/catalog/tools/PF257914

[26] STM32F100xx Reference manual
http://www.st.com/web/en/resource/technical/document/ reference_manual/CD00246267.pdf

[27] NMEA data
http://www.gpsinformation.org/dale/nmea.htm

[28] SPI Master / SPI Slave
http://www.pyroelectro.com/tutorials/spi_master_slave/spi_theory.html

[29] Servo control interface in detail
http://www.pololu.com/blog/17/servo-control-interface-in-detail

[30] Mavlink Communication Protocol
http://qgroundcontrol.org/mavlink/start

[31] Graphic LCD 84x48
http://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf

[32] Library for LCD
http://www.henningkarlsen.com/electronics/library.php?id=47

[33] PPM Signal Description
http://diydrones.com/profiles/blogs/705844:BlogPost:38393

[34] Calculate distance, bearing and more between Latitude/Longitude points
http://www.movable-type.co.uk/scripts/latlong.html

[35] Onboard video from AHRS testing
http://youtu.be/ddgEz6P8fnE

[36] AirshipClub
http://www.airshipclub.com

[37] Czech Technical University UAV project blog
http://ctuuav.blogspot.cz/

[38] Modular Unmanned Aerial Vehicle, *Petr Adamek, Jaroslav Halgasik, Josef Novak, Petr Pahorecky*, American Institute of Aeronautics and Astronautics, 2014

10.1 Appendix A: Description of attached CD-ROM

The attached CD-ROM contains:

- Hardware design (schematics and layout) of flight control unit
- Hardware design (schematics and layout) of ground station
- Software for flight control unit
- Software for ground station
- Datasheets used in this work
- Digital version of this thesis in .pdf