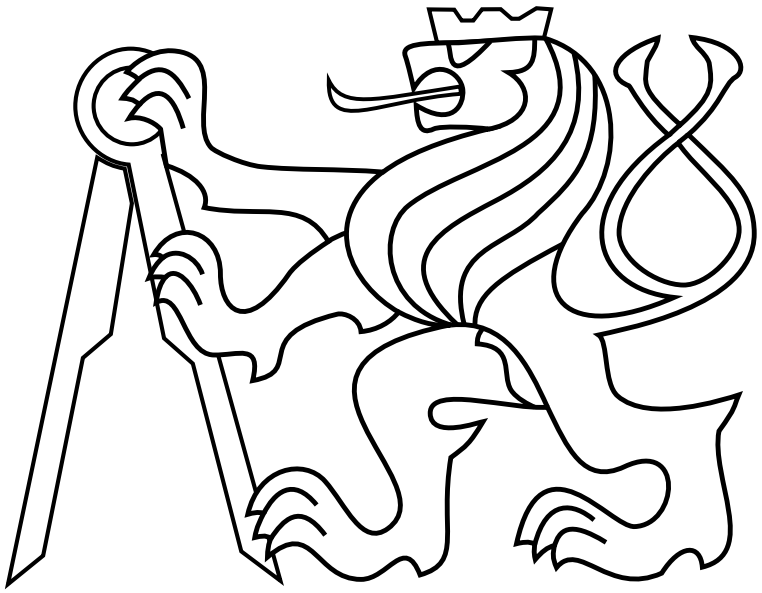
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Diploma Thesis



Václav Endrych

Control and stabilization of an Unmanned Helicopter Following a Dynamic Trajectory

Department of Cybernetics

Thesis supervisor: **Ing. Martin Saska, Dr. rer. nat.**

Prague 2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Václav E n d r y c h

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Control and Stabilization of an Unmanned Helicopter Following a Dynamic Trajectory

Guidelines:

The aim of the work is to design and implement an algorithm for stabilization of a Micro Aerial Vehicle (MAV) and for following a given dynamic trajectory.
Work plan:

- To integrate and implement an altitude controller using data from a sonar in its feedback and to verify the developed method in the task of autonomous landing and take off.
- To design and implement a controller for stabilization of MAV hovering above a fixed location using data on relative velocity from PX4FLOW sensor.
- To extend the controller with the possibility of following a trajectory, which may be changed during the flight.
- To verify the system performance in a set of real experiments; to analyze limits of the system usability (required speed, shape of the trajectory etc.) and in case of available hardware to verify the system in a task of two MAVs formation flying.

Bibliography/Sources:

[1] J. Eckert, R. German, F. Dressler: On autonomous indoor flights: High-quality real-time localization using low-cost sensors, IEEE International Conference on Communications (ICC), 2012.
[2] T. Lee, M. Leok, N.H. McClamroch: Geometric tracking control of a quadrotor UAV on E(3), IEEE Conference on Decision and Control (CDC), 2010.
[3] M. Saska, Z. Kasl., L. Preucil: Motion planning and control of formations of micro aerial vehicles. Accepted for IFAC World Congress 2014.

Diploma Thesis Supervisor: Ing. Martin Saska, Dr. rer. nat.

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Václav E n d r y c h

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Řízení a stabilizace bezpilotní helikoptéry sledující dynamicky se měnící trajektorii

Pokyny pro vypracování:

Cílem práce je navrhnout a implementovat algoritmus pro stabilizaci bezpilotní helikoptéry a sledování zadané trajektorie, která se může dynamicky měnit.
Plán prací:

- Navrhnout a implementovat výškový regulátor využívající údaje z ultrazvukového senzoru ve zpětné vazbě a otestovat jej v úloze autonomního přistávání a vzletu.
- Navrhnout a implementovat regulátor pro stabilizaci helikoptéry vznášející se nad pevným bodem na základě údajů o relativní rychlosti poskytnutých senzorem PX4FLOW.
- Rozšířit regulátor o možnost sledování trajektorie, která se během letu může měnit
- Ověřit funkčnost systému sérií reálných experimentů; analyzovat omezení použitelnosti systému (požadovaná rychlost, tvar trajektorie apod.) a v případě dostupnosti hardwaru ověřit systém v úloze stabilizace dvou helikoptér letící ve formaci.

Seznam odborné literatury:

[1] J. Eckert, R. German, F. Dressler: On autonomous indoor flights: High-quality real-time localization using low-cost sensors, IEEE International Conference on Communications (ICC), 2012.
[2] T. Lee, M. Leok, N.H. McClamroch: Geometric tracking control of a quadrotor UAV on E(3), IEEE Conference on Decision and Control (CDC), 2010.
[3] M. Saska, Z. Kasl., L. Preucil: Motion planning and control of formations of micro aerial vehicles. Accepted for IFAC World Congress 2014.

Vedoucí diplomové práce: Ing. Martin Saska, Dr. rer. nat.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne..... .....

Acknowledgements

Firstly, I would like to thank Dr. Martin Saska for a professional supervision of my thesis and for a kindly approach while consulting problems. Further, I want to thank to Bc. Tomáš Báča for a lot of technical advice and for his dedicated help during all the experiments and to Ing. Zdeněk Kasl for providing the precomputed trajectories.

To all who read this thesis I thank for thorough correction of the text. At last, I want to thank to my girlfriend, family and friends for the encouragement during my studies.

Poděkování

V první řadě bych chtěl poděkovat Dr. Martinu Saskovi za odborné vedení mé práce a za vlídný přístup při řešení problémů. Dále bych chtěl poděkovat Bc. Tomáši Báčovi za mnoho technických rad a za jeho ochotnou pomoc při všech experimentech a také Ing. Zdenku Kaslovi za poskytnuté trajektorie.

Děkuji všem, kteří práci přečetli, za důkladnou korekturu textu. Nakonec chci poděkovat své přítelkyni, rodině a přátelům za podporu při studiích.

Abstract

This thesis is concerned with design of a position control system for Micro Aerial Vehicles (MAVs). In the introduction, state of the art is briefly summarized and the architecture of MAV platform used in this thesis is described. The result of this thesis is a functional position control system that improves properties of stabilization compared to the system that was previously used for MAV stabilization. The design process of the new system is described as well as the implementation of more advanced features as autonomous takeoff and landing and autonomous trajectory following. The system was validated both by simulations using the identified model of MAV's dynamics and by a series of experiments with a real MAV swarm.

Keywords

control system, stabilization, trajectory following, MAV, formation

Abstrakt

Tato práce se zabývá návrhem systému pro řízení pozice malých bezpilotních helikoptér (MAV). V úvodu práce je stručně shrnut současný stav tohoto vědního oboru a je popsána architektura helkoptéry použité v rámci této práce. Výsledkem práce je funkční systém pro řízení pozice MAV zlepšující vlastnosti stabilizace oproti systému dříve používanému ke stabilizaci MAV. V práci je popsán postup návrhu systému a dále implementace pokročilejších funkcí jako je automatický vzlet a přistání či samočinné sledování trajektorie. Funkčnost výsledného systému byla ověřena jednak simulacemi s pomocí vytvořeného modelu dynamiky MAV a jednak sérií experimentů se skutečným rojem helikoptér.

Klíčová slova

řídicí systém, stabilizace, sledování trajektorie, bezpilotní helikoptéra, formace

Contents

List of Figures

List of Tables

List of Abbreviations

DOF Degrees of Freedom

FPU Floating Point Unit

GPS Global Positioning System

GRASP General Robotics Automation Sensing and Perception Laboratory

IDSC Institute for Dynamic Systems and Control

IMR Intelligent and Mobile Robotics Group

LSM Least Squares Method

MAV Micro Aerial Vehicle

MCU Microcontroller Unit

MEMS Mircro Electro Mechanical Systems

MIMO Multiple Input and Multiple Output

PCB Printed Circuit Board

PID Proportional Integral Derivative Controller

PPM Pulse Position Modulation

PWM Pulse Width Modulation

RC Radio Control

RMS Root Mean Square

UART Universal Asynchronous Receiver and Transmitter

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

1 Introduction

One of the core research topics of Intelligent and Mobile Robotics Group (IMR) at Czech Technical University in Prague is Swarm Robotics which is aimed to integrate principles and theoretical background of bio-inspired swarm behaviors with methodology describing cooperative localization of autonomous robots and with methods of artificial intelligence to enable applicability of swarm robotics in realistic outdoor scenarios of surveillance and reconnaissance. [1]

The main platform used for the swarm research are swarms of Micro Unamended Aerial Vehicles (MAVs) also referred to as drones. For the purposes of research, each MAV must be able to localize itself relatively to at least one of its neighbors. In a formation of MAVs it is beneficial if at least one of the drones (the leading MAV) is able localize itself relatively to the surrounding environment. Each drone must be able to control its position and autonomous take-off and landing is also desired.

The research effort at IMR aims at development of distributed autonomous system, which shall be able to control MAVs in tight formations with high positioning precision. Their behavior should be independent on the environment conditions. In other words they should be able to fly both indoors and outdoors. This means that a localization using GPS and barometers which is nowadays widely used by the commercial multi-rotor platforms is not usable. Also the swarms should operate without any major modifications of the environment as for example external high-speed cameras or without direct assistance of ground computers which is an approach usually used in laboratory conditions.

So our aim is to design a platform of autonomous MAVs that would rely purely on the on-board sensors and cameras and on the on-board computation power. The goal of this thesis is to design and implement a position control system for the Unmanned Aerial Vehicle platform used at IMR enabling to stabilize it in a fixed spot or to follow a dynamic trajectory. The designed system will be verified both by simulations and by experiments with a real-world MAV swarm.

The chapter 1 of this thesis gives an overview of the current state of the art and describes the architecture of the drones that were used in this thesis. It also briefly describes related safety and control issues. Chapters 2 and 2 describe the design and implementation of the systems controlling altitude and horizontal position of a MAV respectively. Both chapters include comparison to the previous control system. The last chapter 2 describes implementation of the other features such as autonomous takeoff, landing and trajectory following. The experiments performed to verify the implemented control system are captured in videos at the attached DVD.

1.1 State of the art

Nowadays, MAV systems are capable of performing high-precision autonomous flights and of mutual cooperation in swarms. For example, with the system being developed at the General Robotics Automation Sensing and Perception Laboratory (GRASP) at the University of Pennsylvania, MAVs are capable to perform fast and agile maneuvers and to perform motion with complex trajectories. They are also able to fly in formations that may dynamically change according to the current situation. The MAVs are localized using an external system based on motion-capture cameras that is able to detect the position of MAVs and possible obstacles 100 times per second. Each MAV is then controlled by an onboard processor that sends commands to the motor 600 times per second. [2]

With the control system developed at Institute for Dynamic Systems and Control (IDSC) from the Swiss Federal Institute of Technology, MAVs are able to adapt to changing conditions due to iterative learning algorithms. They are able to interact with objects and to cooperate to realize complex tasks. This is again enabled by a high-precision external localization system and high-performance radio links. [3]

To evade the requirement of a sophisticated localization system and high computation power demands the authors of [4] use a localization framework consisted of a swarm of low-cost mobile sensor nodes that autonomously spread in a given area to form a reference grid for MAV's localization based on the ultrasound distance measurement.

On the approach of controlling the attitude of the body of a drone and therefore also its position, there are many methods used varying from quite simple solutions to very sophisticated ones. Method used in [4] is a plain PID control based on a simplified system model restricted to small attitude angles, which is a method most suitable for implementation onto an embedded on-board control unit. More complex methods use quaternions [5] or special orthogonal groups [6] for body attitude representation to evade kinematic singularities occurring in minimal representations such as the Euler angles. Controllers working with these attitude representation demand more computational power and further CPU time has to be spent to convert the measured values from sensors and outputs for motor controllers from and to the given representation.

Even more sophisticated methods include for example singular perturbation control [7], backstepping nonlinear control [8], iterative learning of feed-forward corrections [9] or usage of neural networks [10] or multi-agent systems [11].

1.2 Target platform

The multi-MAV autonomous system currently being developed at IMR uses drones based on *L4-ME* kit fabricated by the MikroKopter company. It includes four brushless motor controllers *BL-Ctrl V1.2* with integrated current measurement units and the stabilization module *Flight-Ctrl V2.5 ME* with integrated air pressure sensor, MEMS 3-axis gyroscope and accelerometer. [12]

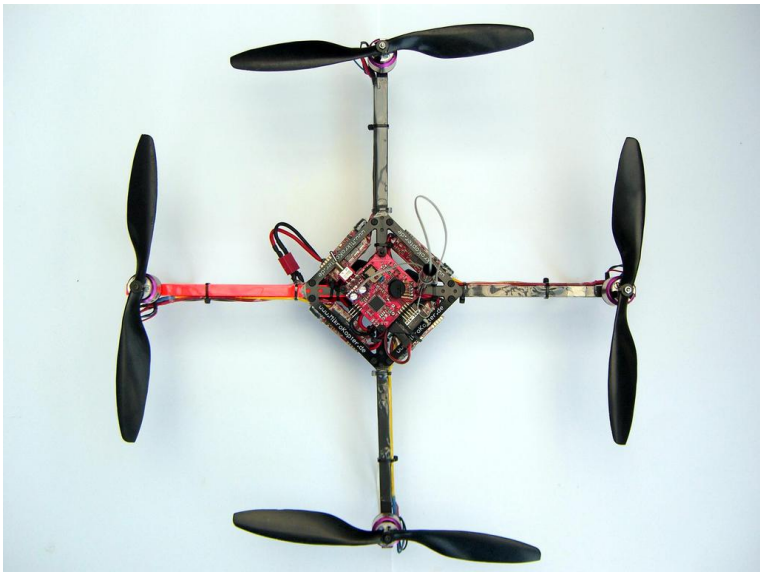


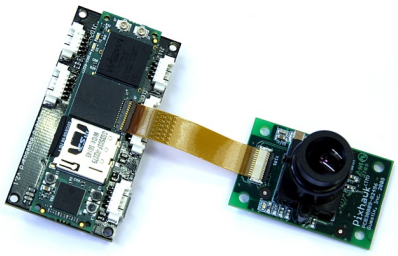
Figure 1: Assembled MK *L4-ME* Kit [12]

The MAV is equipped with a Camera Module for relative visual localization that is based on the *Caspa* camera module and *Overo* board made by Gumstix. The module is designed to localize a special circular mark with fixed dimensions (further referred to as the *blob*) which can be seen in figure 2. The Camera Module is capable to compute the full 3D coordinates of the blob from its position in the camera frames and from the knowledge of its absolute dimensions. The module is able to generate the position at the rate varying from 60 to 33 Hz depending on the current conditions. [13]

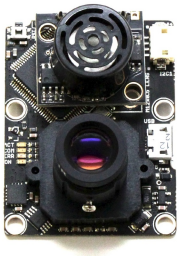
Another component added to the drone platform is a *PX4Flow* Smart Camera module extended by Maxbotix *HRLV-EZ4* ultrasonic range finder. The *PX4Flow* module is an optical flow camera pointed to the ground working in a similar way as an optical mouse. It provides information about absolute velocity of the MAV relative to the world along the two horizontal axes parallel to the ground whereas the sonar sensor provides the absolute altitude relative to the ground surface. The module includes an integrated 3-axis gyroscope that is used to compensate rotational motion of the sensor in the optical flow signal. The *PX4Flow* module is able to convert the velocity values from pixels per frame to meter per second using the information about absolute ground distance. The data output of the module is provided in the format of MAVLink messages. [15]

Figure 2: Relative visual localization marker (the blob) [14]

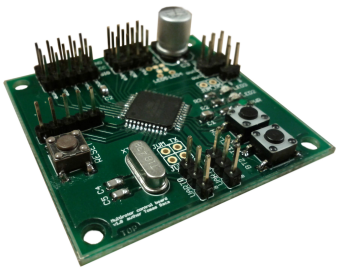
Finally, there is a custom-made Control Board based on the AVR *ATmega164P* 8-bit microcontroller running at 18.432 MHz. It serves as the central point of the system interconnecting all the other electronic components of the drone. The module is designed to handle all the communication tasks. It merges the PWM signals from the RC receiver into a single PPM signal for the Stabilization Module. It receives and decodes the MAVLink messages from the *PX4Flow* unit and if available, it also receives the position information from the Localization Module. Rest of the computing power of the Control Board is reserved for control purposes such as stabilization of the MAV on a fixed spot or following a moving target. [14]



(a) Gumstix modules [16]



(b) PX4Flow module [15]



(c) The Control Board [14]

Figure 3: Additional UAV components

Furthermore, the module is capable of communication with a ground computer station via the Gumstix microcomputer and its WiFi link [1]. However, the primary commands for the Control Board that control its behavior and therefore the behavior of the whole system are channeled through the RC receiver. This way of controlling the MAV is convenient for a trained pilot and it enables fast response of the pilot to the actual situation.

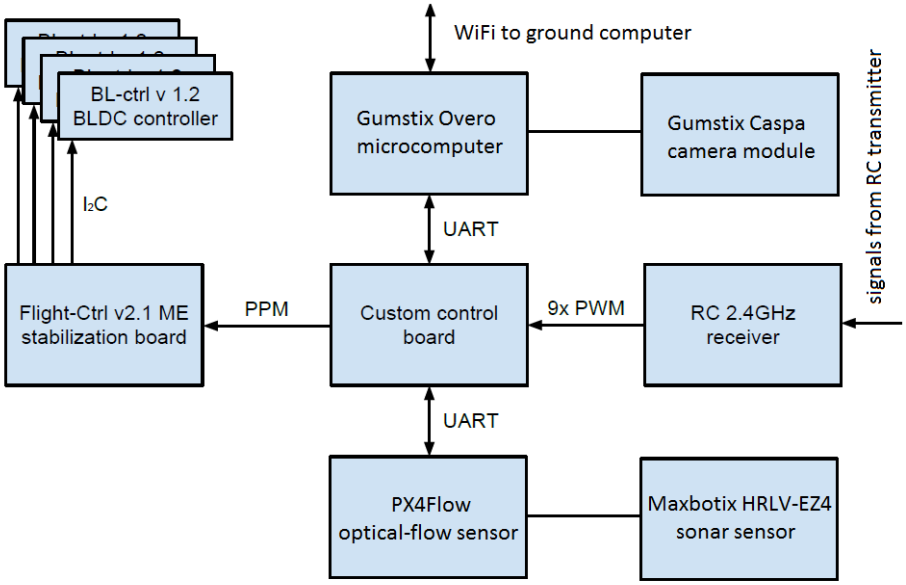


Figure 4: Architecture of MAV's electronic components [2]

The block diagram in figure 4 shows the overall architecture of the system. It is obvious that this architecture makes the Control Board able to completely take over control of the drone. Nevertheless, this is currently not desired due to the reasons specified in the next section. The structure includes one significant deficiency that is caused by the fact that the Control Board has only two UART communication lines [3]. The problem is that the Control Module does not have the information about the angles of the drone's body. The Stabilization Unit has an integrated 3-axis gyroscope and accelerometers to be able to stabilize the MAV's attitude and the PX4Flow board also has a gyroscope to compensate the optical flow data. But there are no sensors on the Control Board and it is not possible to transfer the information to it from other modules on the current hardware.

[1] WiFi communication is currently used for debugging purposes only because the WiFi link in combination with the 2.4 GHz RC signal is not sufficiently reliable.

[2] This figure is based on Figure 8 from [14].

[3] Originally, the architecture did not contain the PX4Flow sensor and the second UART line was connected to the Stabilization Board which was modified to send the information about attitude angle periodically. [14]

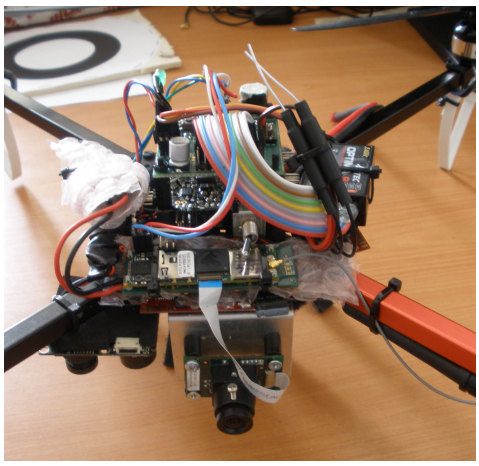


Figure 5: Complete drone electronics

Figure 6: RC transmitter

1.3 Safety

The MAV platform used in this thesis is capable of motion with great speed and acceleration. Its propellers are uncovered, rotate at high rate and have quite sharp edges. Therefore, the drone may be dangerous if uncontrolled and it can cause damage to property or serious injuries. For this reason, the MAV control system designed in this thesis must respect the rules specified below.

1. The drone must be able to take-off only when armed by a human.
2. The speed of its motion must be limited in the autonomous mode.
3. It must be possible to switch from the autonomous to fully manual control at any instant.
4. In case that anything goes wrong, it must be possible to manually override the control signals of the controller.
5. If a fault is detected, the drone should land safely on the ground and shutdown the motors.
6. If a failure is detected, the whole system must shut down immediately, i.e. stop the motors and drop to the ground.

1.4 Levels of control

1.4.1 Base level

A drone flying in the air is an inherently unstable dynamic system. From the control theory point of view, it is an astatic non-linear MIMO system. As an object moving in a 3-dimensional space it has 6 degrees of freedom (DOFs). However, only 4 are independently controllable because there are only 4 control inputs. The inputs of the dynamic system are the angular rates of the four propellers of the MAV. The outputs to be controlled are the 3 coordinates of position of the drone's body in the world frame and the angle of rotation around the **z** axis (further referred to as the *yaw* angle) of the frame fixed to the body of the drone.

The orientation of the body-fixed frame is displayed in figure 7 where the positive direction along the **x** axis is considered to be direction of a forward motion. The angle of rotation of the body around the **x** axis is referred to as the *roll* angle and the angle of rotation around the **y** axis is referred to as the *pitch* angle. Both of these angles affect the direction of the linear acceleration of the drone's body and therefore are considered internal state variables of the system along with the linear velocity vector.

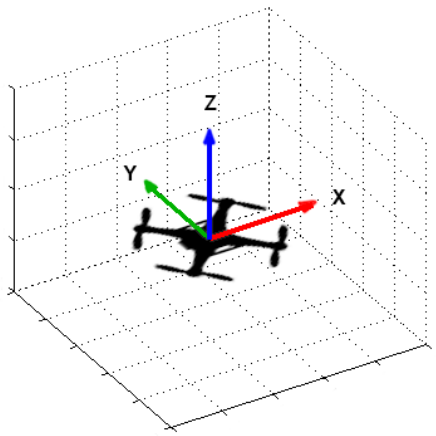


Figure 7: Body-fixed frame orientation

Each of the spinning propellers of the MAV generates two forces - thrust and torque. The magnitude of both forces is proportional to the square of angular rate of the propeller. The opposite propellers rotate in the same direction, two of them rotate clockwise and two counterclockwise. Therefore, it is possible to control the three torques acting on the body (related to *roll*, *pitch* and *yaw* angles) by controlling the ratios of propeller spinning rates.

And the linear acceleration of the body of the MAV can be computed according to the equation

$$\vec{a} = \frac{1}{m}(\vec{F_c} + \vec{F_g}) \,, \tag{1}$$

where $\vec{a}$ is the acceleration of the body, m is its total weight, $\vec{F_g}$ is the gravitational force and $\vec{F_c}$ is the *total thrust* which is the sum of thrust forces of all the propellers acting along the **z** axis of the body-fixed frame.

1.4.2 Manual level

Each of the motor controllers controls an angular rate of one of the propellers based on the request from the Stabilization Board. And the Stabilization Board stabilizes the attitude of the body of the MAV using the information from its gyroscope in a feedback. This makes the Stabilization Board along with the four motor controllers the first level of control of the drone. The input signals of the Stabilization Board are:

- **Elevator** - corresponds to desired *pitch* angle,
- **Aileron** - corresponds to desired *roll* angle,
- **Rudder** - corresponds to desired *way* angular rate,
- **Throttle** - corresponds to desired *collective thrust*.

The control signals described above are coded in the PPM modulation so their value technically correspond to a count of cycles of a timer that generates the signal pulses. The possible values range 2304 to 4608. For the purposes of this thesis, the values of the control signals will be considered to have no unit and will be used as is.

The stabilization performed by the Stabilization Board is essential to make MAV manually controllable. However, it requires a skilled pilot to control the drone at this level, e.g. the *pitch* angle corresponds non-linearly to the forward acceleration of the drone and simultaneously affects the vertical acceleration. Also there is a drift in the attitude of the drone's body and in its linear acceleration causing further drift in its velocity and position. So it requires a constant concentration of the pilot to stabilize the drone in a fixed spot.

Safety: The Stabilization Board partially implements the safety rule 6 (see 1.3), because if it cannot receive the control signals, it shuts down the motors.

1.4.3 Autonomous level

The second level of control is implemented by the Control Board and it performs autonomous stabilization of the position of the MAV. Originally, the position stabilization controller was designed only to stabilize the drone in a given position relative to the blob, e.g. 2 meters in front of the object marked with the blob. But this controller provided poor stabilization performances when the reference blob was moving. [14]

Prior to the beginning of work on this thesis, the MAV platform was supplemented by the *PX4Flow* sensor and the position controller has been extended so that if no blob is detected in the *Caspa* camera frames, it would regulate the velocities given by the optical flow to zeros and the altitude given by the sonar to a constant value.

The output of the position controller in the Control Board is implemented in such way that the output signals of the controller are securely limited, added to the control signals from the RC receiver and the merged signals are channeled to the Stabilization Board. Furthermore, one of the auxiliary signals from the RC receiver can be used to disable adding of the controller signals to the output.

Safety: The concept described in the previous paragraph ensures the safety rules 1, 3 and 4. The position controller can be alway turned of by the pilot using the RC transmitter. If it wold be impossible to turn off the controller for some reason, the pilot is still able to control the MAV and the manual control signals can always overpower the controller signals because of the safety limit of controller outputs.

Concerning the safety rune 1, the Stabilization Board requires a special sequence of highest and lowest values of the control signals (the *arming gesture*) to start the motors. So the position controller is unable to arm the motors by itself due to the safety limit. In addition to that, the controller is unable to set the value of the *Throttle* control signal high enough for takeoff by itself. The pilot must set a manual *Throttle* offset of a corresponding value to enable the controller to takeoff.

2 Altitude controller design

2.1 Original controller

At the beginning of the work on this thesis, a short experimental flight was performed with one MAV and the flight data was captured so that we were able to investigate the behavior of the MAV when controller manually and by the controller that was in use previously to this thesis. The aim was to obtain data to be used for identification of the dynamics of a flying MAV and to use this information to improve the quality of position stabilization by improving the implementation of the former controller.

The captured data is displayed in figure 8. The controller was set to stabilize the drone at altitude of 1 meter. The top chart shows the altitude of the drone measured by the ultrasonic rangefinder of the PX4Flow module and the bottom chart shows the *Throttle* control signal. The manual offset of the signal is approximately 3400.

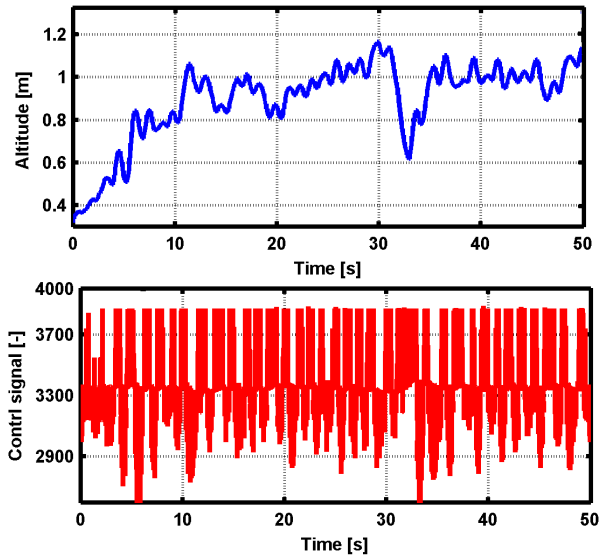


Figure 8: Original altitude controller

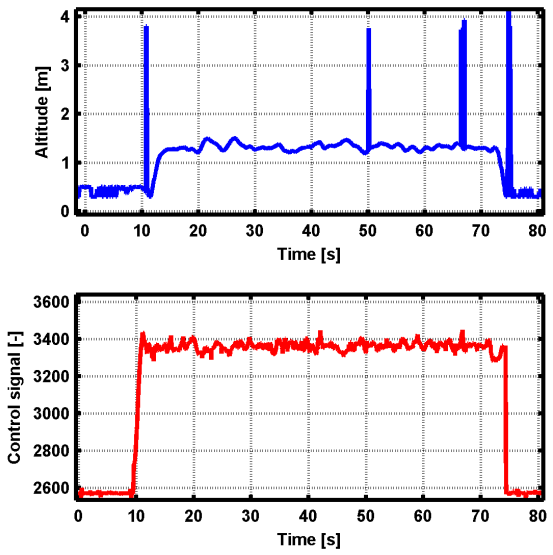


Figure 9: Manual control of altitude

It can be seen from the figure that the controller does not provide a sufficient performance. The actual altitude of the MAV is significantly oscillating and the maximal deviation from the required altitude is approximately 0.39 meters which is unacceptable for a multi-MAV application. Also the controller generates undesirable peaks in the control signal which cause overly aggressive and erratic response of the controlled system and make the data analysis very difficult. The magnitude of spikes is large enough to reach the upper safety limit of the controller output that can be noticed in the lower chart of figure 8.

The performance of the original controller can be compared with performance of an experienced pilot (figure 9). The pilot performs a takeoff, stabilizes the drone in a constant altitude for one minute and lands on the ground. The maximal deviation of altitude from its mean value over the time of stabilization is proximately ±0.15 m even though the magnitude of changes of the control signal is much smaller compared to the autonomous controller. Our goal is to adjust the controller implementation so to make its performance comparable to the one of the pilot.

Note that the ultrasonic sensor of the PX4Flow module is not always able to measure the distance from the ground properly. It figure 9, it can be seen that it is not able to detect distances smaller than approximately 0.3 m. Also if the sensor doesn't capture the reflected wave properly it returns an invalid value which generates a peak in the distance signal (figure 9).

2.2 System identification

The theoretical model of the dynamics of a flying quad-rotor MAV has been described in section 1.4.1. For the purposes of altitude controller design, we will try to identify the dynamics along the vertical axis of the real MAV used at IMR from the Control Board's point of view. So the value of the *Throttle* signal shall be considered the input of the dynamic system and the value measured by the ultrasonic sensor shall be considered the output of the system.

If we assume that the *roll* and *pitch* angles are kept small either by a pilot or by a controller, so based on (1) we can write that

$$\ddot{p}_{(t)} \approx \frac{F_{c(t)}}{m} - g \,, \tag{2}$$

where $\ddot{p}$ is the acceleration of the body of the drone along the vertical axis, F_c is the magnitude of the total thrust, m is the total mass of the drone and g is the acceleration of gravity.

For now, we will consider the dynamics of MAV motors and spinning propellers negligible compared to the dynamics of the MAV's body. Considering the purpose of the motor controllers and the Stabilization Board we can assume that

$$F_{c(t)} = k_F(c_{(t)} + c_{o(t)} - c_{min}) + F_{min} \,, \tag{3}$$

where F_c is the collective thrust magnitude, c is the *Throttle* signal from the controller, c_o is the *Throttle* offset from the RC receiver, c_{min} is the minimal possible value of the control

signal, F_{min} is the thrust generated by motors at their idle speed and k_F is an unknown constant.

Using equations (2) and (3) we get

$$\ddot{p}_{(t)} = \frac{k_F}{m}(c_{(t)} + c_{o(t)} - c_{min}) + \frac{F_{min}}{m} - g\,. \tag{4}$$

And if we consider c_o to be constant over the time the controller is turned on, we can simplify (4) to

$$\ddot{p}_{(t)} = k_c \cdot c_{(t)} + k_o\,, \tag{5}$$

where p is the position of the drone along vertical axis, c is the controller output signal and k_c, k_o are unknown constants.

Due to numerical precision, it is more convenient to denote the constants according to the inverse formula

$$c_{(t)} = k'_c \cdot \ddot{p}_{(t)} + k'_o\,. \tag{6}$$

Than we can write that

$$\ddot{p}_{(t)} = \frac{1}{k'_c}(c_{(t)} - k'_o)\,. \tag{7}$$

This way we obtain a second order model of the dynamic system

$$\frac{d}{dt}\begin{bmatrix} p \\ \dot{p} \end{bmatrix} = \begin{bmatrix} \dot{p} \\ \frac{1}{k'_c}(c_{(t)} - k'_o) \end{bmatrix}\,. \tag{8}$$

To obtain the values of constants k'_c and k'_o, we use the data measured during a real flight. It is not possible to integrate the control signal, because integrating noise and using imprecise initial conditions lead to a curve drifting away from the actually measured altitude - see figure 10). However, derivation of the raw position signal is also not a good way as the noise of the signal including peaks from faulty sensor readings is magnified by the derivation. This effect can be well seen on the velocity signal obtained from the unfiltered data in Appendix B.

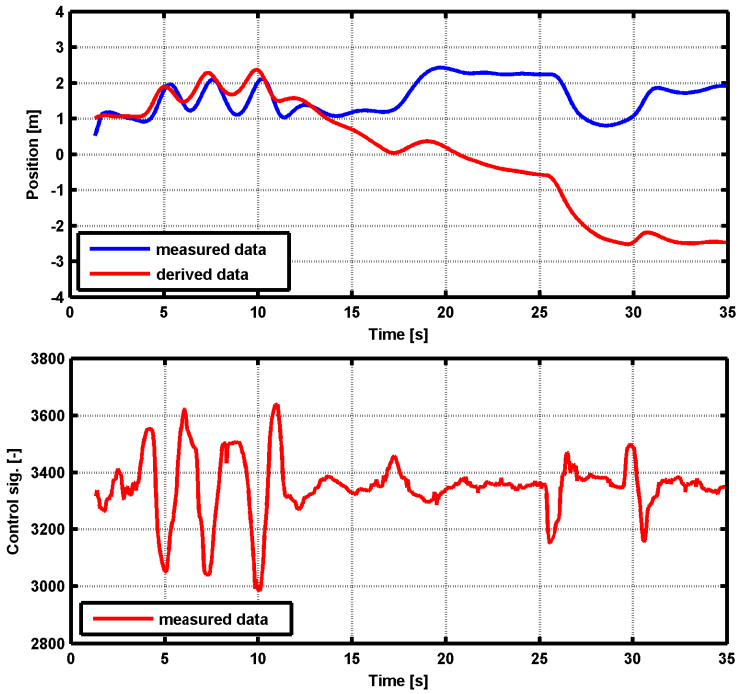


Figure 10: System identification - altitude

At first, we tried to solve this problem by filtering out the extreme values (in this case altitudes greater than 3 m) and using a sliding average filter on the data. The value of the filtered signal at each instant was computed as an average of surrounding 32 samples of the original signal, that is a window of approximately ±0.23 s. Nevertheless, the acceleration signal obtained by double differentiation of the filtered signal was still not usable (the gray signal in the bottom chart of B).

Eventually, approximation by splines (using the Matlab function `spline` for cubic spline interpolation and subsampling of data) has shown to be an efficient approach to filter the data. The results are depicted as the red curves in appendix B. To find the constants k_c' and k_o', the least squares method has been used. The resulting constant values are

$$k_c' = 116.8 \text{ m}^{-1}\text{s}^2 \,, \quad k_o' = 3353.8 \,. \tag{9}$$

Acceleration signal produced from the measured control signal using the model according to (7) and constant values from (9) is shown as the black line in the bottom figure of appendix B.

2.3 First controller test

For the first attempt of the new controller design, we chose a simple P-D controller with exponential filtration of the input altitude signal. The filter was added to suppress the peaks in controllers output generated by the differential action component. The differential component of the new controller differentiates the filtered position signal only (not the regulation error) which enables rapid changes (e.g. step-changes) of the setpoint. The designed filter and controller are defined by the discrete equations

$$p_{f(t)} = p_{f(t-T)} + \frac{T}{\tau_f}(p_{(t)} - p_{f(t-T)}) \,, \tag{10}$$

$$\begin{aligned} c_{(t)} &= k_P \cdot P_{(t)} + k_I \cdot I_{(t)} - k_D \cdot D_{(t)} \,, \\ P_{(t)} &= (sp_{(t)} - p_{f(t)}) \,, \\ I_{(t)} &= I_{(t-T)} + T(sp_{(t)} - p_{f(t)}) \,, \\ D_{(t)} &= \tfrac{1}{T}(p_{f(t)} - p_{f(t-T)}) \,, \end{aligned} \tag{11}$$

$$k_P = 612.5 \,, \quad k_I = 0 \,, \quad k_D = 857.5 \quad \tau_f = 0.1 \text{ s} \,, \tag{12}$$

where T is the sampling period of the controller, p_f is the filtered altitude signal, sp is the altitude setpoint and c is the overall action command of the controller. The values of constants k_p and k_d were obtained from a Matlab simulation depicted in figure 11.

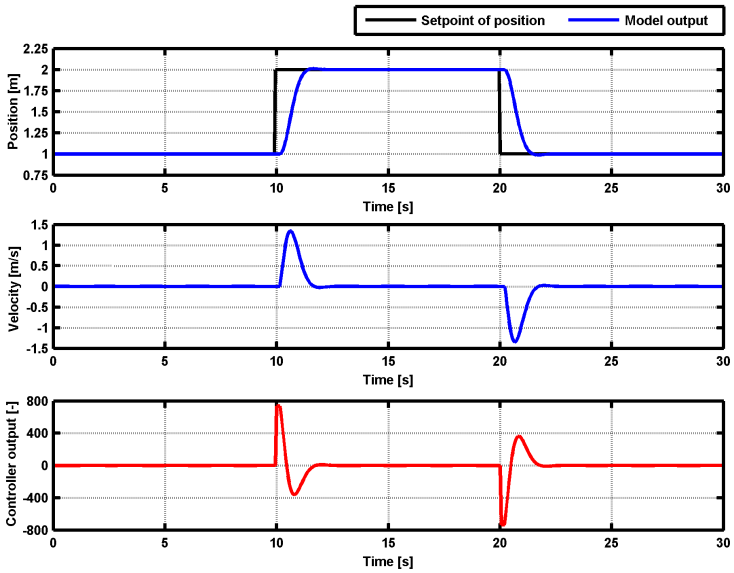


Figure 11: First altitude model

The described controller with the exponential filter and the model of the drone defined by equation (7) has been used for the simulation. The value of constant k'_c from (9) has been used and it has been assumed that the pilot sets c_o such that k'_o is zero. In other words, that he sets the value of the manual Throttle offset such that the drone hovers with minimal vertical acceleration when the controller is turned off.

To get a response of desired shape and find the corresponding values of constants k_P and k_D, the Matlab function `fminsearch` has been used. The result of the simulation can be seen in figure 11. The regulator manages to stabilize the model of MAV's dynamics in a new position 1 m above/below the previous position without oscillations in approximately 1.5 s which is the desired behavior.

Figure 12: MAV prepared for a flight

To verify the performance of the proposed controller with a real-world MAV, the controller was implemented into the Control Board of the MAV and a testing flight has been performed. The controller showed visibly worse performance in the real experiment compared to the simulated response.

The captured data from the testing is shown in figure 13. At the time from 0 to 5 s, the take-off phase is captured. The controller was turned on already on the ground (notice the wrong altitude sensor readings under 0.3 m) and then the pilot set the value of the manual command (c_o) high enough so that the controller was able to lift the drone in the air. The altitude of the drone controlled by the tested controller oscillated with amplitude approximately 0.2 m. In combination with the safety limits of the controller output, the controller basically performs a bang-bang regulation.

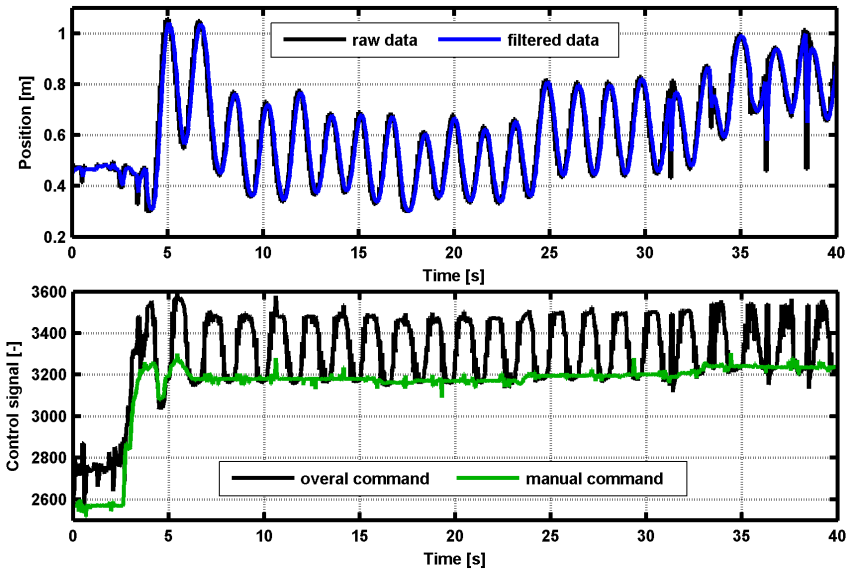


Figure 13: First altitude controller verification

We also found out that in reality, the manual offset of the control command cannot be easily set so that k'_o is zero, because the required value is actually not constants [4]. Furthermore, if the controller is turned on and controlling the altitude of the drone, the pilot sees the response of the MAV to the overall command signal and thus he looses the visual feedback of the influence on the manual command only.

Because of this issue, there was a nonzero vertical acceleration when the output of the controller was zero. And due to the absence of integration action component, the controller was unable to regulate to a zero error (the setpoint of altitude was 0.8 m). The conclusion of the experiment was that the simulation model is insufficient and so is the controller design based on the model.

[4] Due to external influences as battery charge state, current aerodynamic conditions, etc.

2.4 Improved controller design

2.4.1 Position estimator

After the experiment described in the preceding chapter, we have identified several causes of the bad performance in addition to the before mentioned problem with the missing integration action. The first of the causes being the exponential filter itself. It can be seen in figure 14 that the filter is delaying the position signal for approximately 0.1 s and that the peaks from erroneous sensor readings are not filtered out entirely. We have also noticed that the altitude sensor generates data only at 10 times per second in spite of the controller being executed 70 Hz (at the main system tick).

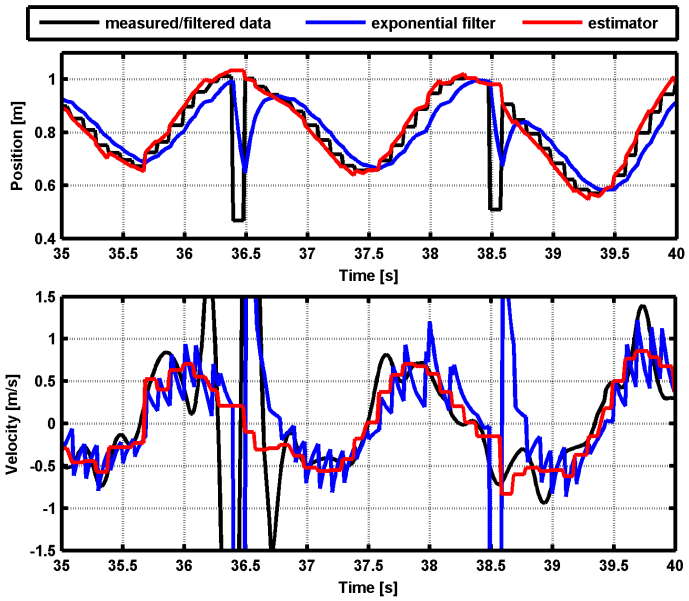


Figure 14: Altitude estimator

This fact has been used to design a *position estimator* component. The component estimates the vertical speed of the drone from the last two data samples and using the estimated speed, it extrapolates the position of the drone until the next data sample (see figure 14). The estimator also includes algorithm for filtration of erroneous signal peaks. The component generates velocity signal that can be directly used by the controller as the D action component defined in (11) is essentially the vertical velocity of the drone.

There are step-changes both in the estimated position and velocity signals. But it does not cause any problems because the signals are not further differentiated in the controller. The estimator provides less delayed position signal due to the estimation of the value of next data sample based on the velocity. And it also provides the velocity signal with less noise than the differentiated exponential filter signal.

2.4.2 Third order model

The other cause of the bad performance of the controller in the experiment described in section 2.3 was an insufficiently accurate model of MAV's behavior. The bottom chart in appendix B shows that the actual acceleration (the red curve) is delayed from the one generated by the used model (the black curve). To increase precision of the model, a 3rd differential order has been added to the model. The added order can be considered to represent the delay caused by the dynamics of the propellers. So now the model is defined by the equations

$$\dddot{p}_{(t)} = \frac{1}{\tau k'_c}(c_{(t)} - k'_o) - \frac{\ddot{p}_{(t)}}{\tau}\,, \tag{13}$$

$$k'_c = 64.8\ \mathrm{m^{-1}s^2}\,, \quad k'_o = 3352.0\,, \quad \tau = 0.31\ \mathrm{s}\,. \tag{14}$$

Again, LSM has been used to identify the constants of the model. The acceleration generated by the new model ($\ddot{p}$) is displayed as the green curve in appendix B.

Even though the output of the 3rd order model fits to the real data better than the output of the previous model (see 2.3), there is still a minor difference between the two signals. In figure 15, the black signal is the second derivative of the filtered real position signal and the red signal is the response of the 3rd order model to the real control signal from the experiment presented in 2.3. The difference (further referred to as *acceleration error*) is probably caused by the fluctuation of external aerodynamic forces.

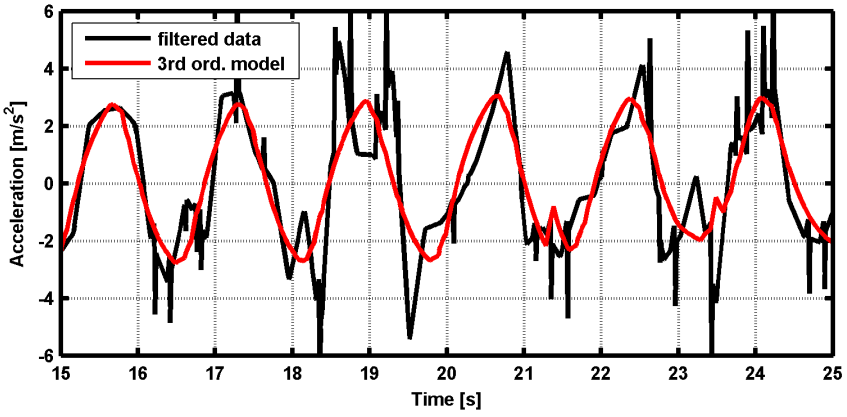


Figure 15: Differences in altitude acceleration

The figure 16 shows the acceleration error values as function of the state variables of the system - acceleration ($\ddot{p}$), velocity ($\dot{p}$) and position (p) along the vertical axis. The figure indicates that there are no significant dependencies of the error signal on the state variables and that it can be considered a purely random Gaussian noise. The root mean square (RMS) of the error values from the investigated experiment is 1.42 $\mathrm{ms^{-2}}$.

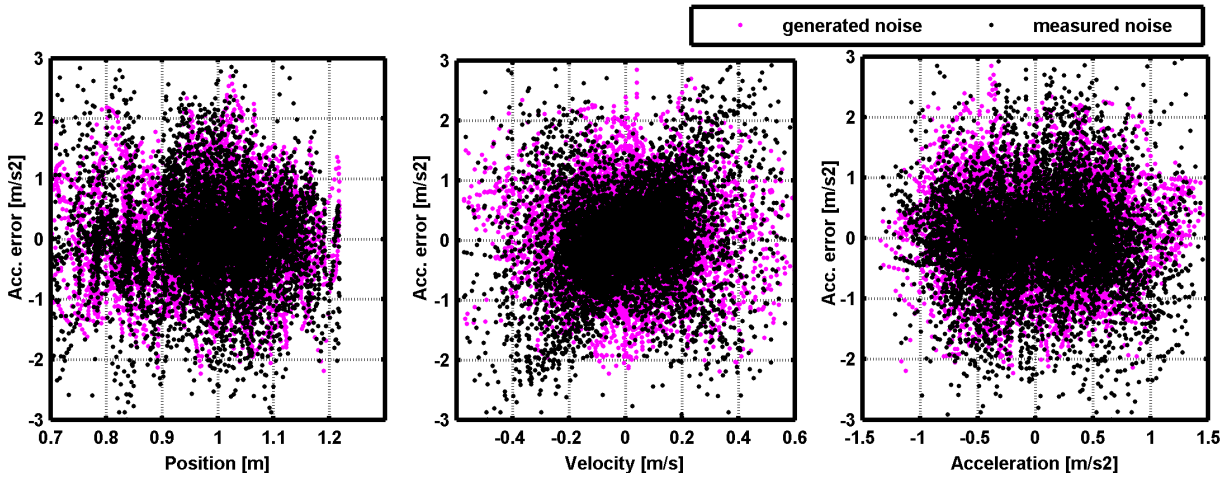


Figure 16: Altitude acceleration error dependencies

We have generated an artificial error signal (using Matlab function `randn`) with the same RMS and compared it to the real acceleration error. In figure 17, the black signals correspond to the real acceleration error and the magenta signasl correspond to the artificially generated error. Total error is computed as integral of the absolute value of the error signal.

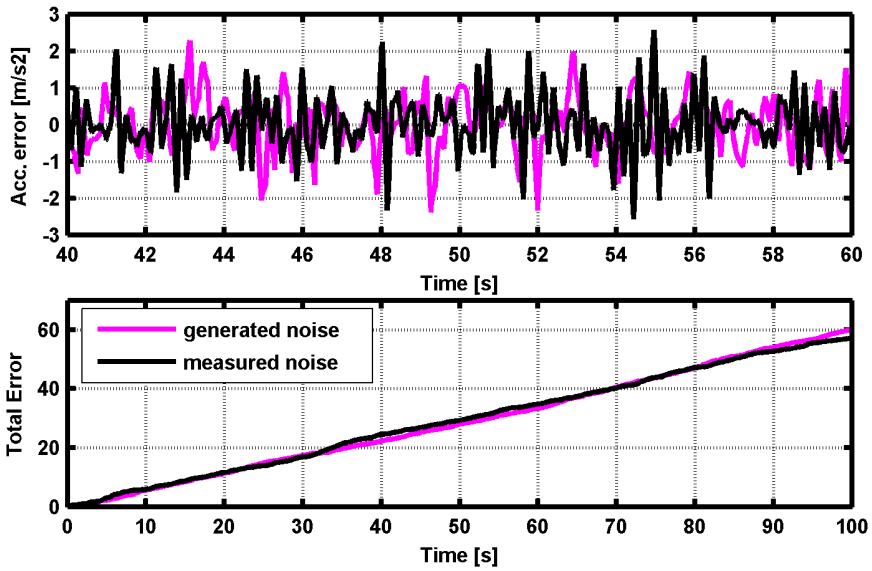


Figure 17: Altitude acceleration error function

It can be seen that the artificial error signal has the same properties as the real error signal. Therefore, we have included the artificial acceleration error to the model of dynamics of the system. This way we obtain the final model of MAV's behavior along the vertical axis

$$\frac{d}{dt}\begin{bmatrix} p \\ \dot{p} \\ \ddot{p}' \end{bmatrix} = \begin{bmatrix} \dot{p}_{(t)} \\ \ddot{p}'_{(t)} + E_{a(t)} \\ \frac{1}{\tau k'_c}(c_{(t)} - k'_o) - \frac{\ddot{p}'_{(t)}}{\tau} \end{bmatrix}, \tag{15}$$

where $\ddot{p}'$ is the acceleration induced by the controller and E_a is the acceleration error. Finally, a quantization algorithm has been added to the model to achieve an output signal sampled at 10Hz, same as the output signal of the PX4Flow sonar sensor (see 2.1).

The response of the 3rd order altitude model with acceleration noise and position quantization is shown in the appendix C. The black signals are the captured data from experiments with a real drone. The red signals are closed-loop responses from simulations using the described MAV model and a corresponding controller model. The left side charts show responses with the original altitude controller and the right side charts show the responses with the controller described in chapter 2.3.

2.4.3 Final altitude controller

The improved model presented in the preceding chapter was used to design a new PID controller with usage of the position estimator (see 2.4.1). The controller can be defined by equations

$$\begin{gathered} c_{(t)} = k_P \cdot P_{(t)} + k_I \cdot I_{(t)} - k_D \cdot D_{(t)}\,, \\ P_{(t)} = (sp_{(t)} - p_{e(t)})\,, \\ I_{(t)} = I_{(t-T)} + T(sp_{(t)} - p_{e(t)})\,, \\ D_{(t)} = \frac{1}{T}(v_{e(t)} - v_{e(t-T)})\,, \end{gathered} \tag{16}$$

$$k_P = 120\,, \quad k_I = 120\,, \quad k_D = 200\,, \tag{17}$$

where T is the sampling period of the controller, p_e is the estimated position signal, sp is the altitude setpoint, v_e is the estimated velocity signal and c is the overall action command of the controller. The values of constants k_p and k_d were again obtain by optimization of the closed-loop response in a Matlab simulation.

The safety saturation of the controller output is ±300. Also the integration action component is saturated to ±200 to prevent the wind-up effect. The regulator was implemented and tested by an experiment with a real drone. The results are shown in appendix D. The controller was able to stabilize the drone's altitude and to follow changes of the altitude setpoint. It also compensated an error intentionally introduced by the pilot at the time of 80 s.

2.5 Summary

Comparison of performance of the original controller, newly implemented controller and manual stabilization is shown in appendix E. Note that the top figure shows position error to make the signals comparable, because the drone flew at different altitude in each experiment. The velocity signals for the original controller and manual control were computed offline after the experiment. Whereas the velocity signal of the new controller is the velocity estimation generated onboard by the altitude estimator.

Table 1: Performance of altitude controllers

	$\bar{p}$ [mm]	σ [mm]	E_{max} [mm]	l [s]
Manual control	1325	63.1	173.3	54
Original controller	984	111.7	373.6	60
New controller	999	28.9	87.5	60

Table 1 shows the statistical summary of the experiments displayed in appendix E, where $\bar{p}$ is the mean value of altitude (the setpoint for controllers was 1 m), σ is the standard deviation of altitude from the mean value, E_{max} is the maximum altitude deviation and l is the length of used dataset in seconds. The table shows that the performance of the newly implemented controller is significantly better than the performance of the original controller and that is is even better than stabilization performed by a human pilot.

3 Position controller design

3.1 Data source

In order to identify the dynamics of the horizontal position of the MAV and to design a horizontal position controller, it has been essential to have the information about the actual position of the drone. The drone has two localization sensors onboard - the Gumstix camera module and the PX4Flow module. However, each of the sensors provides a different position information in a different frame and in a different unit of measurement.

The output of the Gumstix module is the position of the blob relative to its camera given in millimeters. The output of the PX4Flow module is the absolute distance of the sensor from the ground in meters and a 2-dimensional velocity of the sensor relative to the ground in the horizontal plane. Our goal is to obtain the position of the MAV based on the values from the sensors.

3.1.1 Original implementation

Figure 18 shows the horizontal position signals from the sensors as they were received by the Control Board using the implementation of position data processing that was developed previously to the start of the work on this thesis. In the figure, the **y** component of the velocity signal from the PX4Flow module is displayed as the black line in the lower (velocity) chart. Integral of the signal is displayed as the black line in the upper (position) chart of the figure. The red line in the position chart depicts the **y** component of the position signal received from the Gumstix module.

In this original implementation, the PX4Flow signal was filtered by executing the following line of code in the routine receiving the PX4Flow data [5].

```
aileronSpeed = aileronSpeed*0.2 + flow_comp_m_y*0.8;
```

The variable `flow_comp_m_y` contains the value of the **y** component of the velocity received from from the PX4Flow module in ms^{-1} and the variable `aileronSpeed` stores the value of the filtered signal which is displayed as the purple line in the lower chart of figure 18. The integral of the signal is displayed as the purple curve in the position chart of the figure. It can be seen that the filtered PX4Flow signal has nearly the same properties as the received signal, i.e that the performance of the date filter is insufficient.

[5] The code is an implementation of a kind of exponential filter. The problem is, that data from the PX4Flow module are received at variable rate. And therefore the filter does not have an actual time constant property.

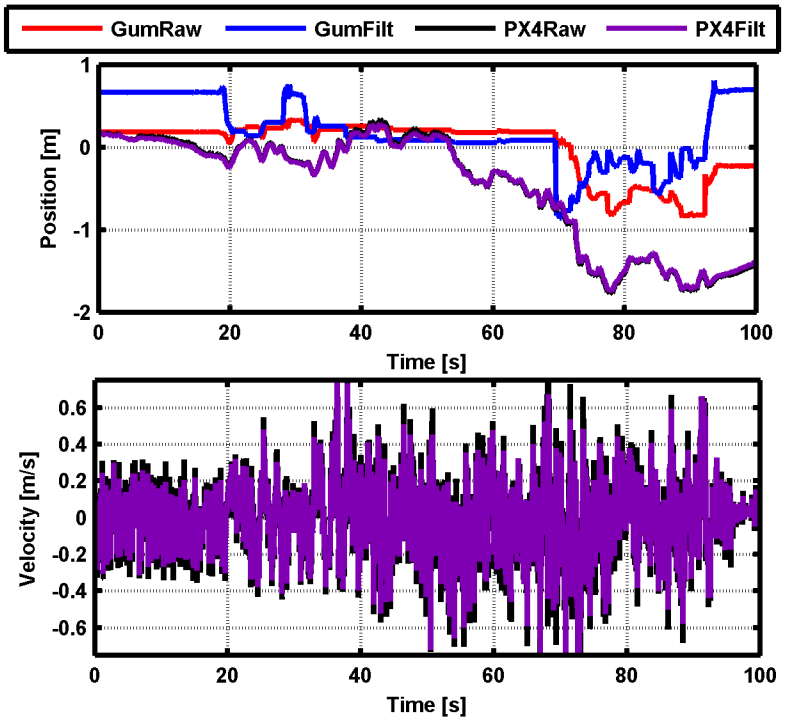


Figure 18: Original position signals [6]

The position signal from the Gumstix module was filtered similarly by executing the line of code below in the routine which receives the data from Gumstix module (also executed at variable rate).

```
yPosGumstix = yPosGumstix*0.4 + zPosGumstixNew*0.6;
```

Note that the variable `yPosGumstix` actually stores the value of the filtered **z** component of the Gumstix position signal. This way the code also performed a sort of coordinates conversion. But there was no coordinates conversion for the PX4Flow signal.

Altogether, the original implementation of the position data processing was hardly readable. It contained ambiguous variable names, worked with signals in different units of measure and there was no well-defined signal corresponding to the position of the drone.

3.1.2 Definition of coordinate systems

We have specified the coordinate systems as shown if figure 19 to be able to convert the localization sensor signals properly. After few experiments we figured out that it would be most convenient to specify the coordinate systems according to the control signals:

- positive *Elevator* value → positive value along x axis (ahead),
- positive *Aileron* value → positive value along y axis (to left),
- positive *Throttle* value → positive value along z axis (up).

This way the drone-fixed coordinate system ($D : \{O_D, x_D, y_D, z_D\}$) was defined. Our reference in the world is the blob mark and the floor. The coordinate system B is fixed to the blob and the coordinate system *0* is its projection onto the floor. Axes of these systems have the same direction as the ones of system D.

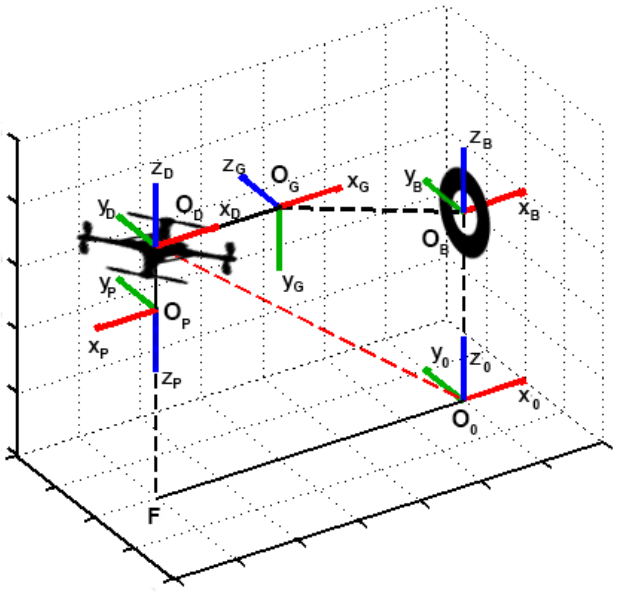


Figure 19: Relation of coordinate systems [7]

The Gumstix module is mounted on the front side of the drone and gives the position of the blob (point O_B) in its coordinate system ($G : \{O_G, x_G, y_G, z_G\}$). The PX4Flow module is mounted on the bottom side of the drone and it provides the absolute distance of the floor (point F) from the PX4Flow sensor along axis z_P and the velocities of the floor relative to the sensor along axes x_P and y_P. The value we want to obtain is the position of the drone in the word, that is the position of point O_D in the coordinate system $0 : \{O_0, x_0, y_0, z_0\}$, which is displayed as the red dashed line in figure 19. In other words, we want to obtain its altitude above the floor and the position (along x_0 and y_0) relative to the blob.

[7] For clarity of the figure the coordinate systems D, G and P are displayed in different locations, but they should originate from the same point ($O_D = O_G = O_P$).

To simplify the problem as much as possible, we have defined the origin of coordinate system D in the origin of system G (in the Gumstix camera). And because the PX4Flow sensor does not give the absolute position along axes x_P and y_P, we can place the origin of the PX4Flow coordinate system into the same point so that $O_D = O_G = O_P$.

What is causing a real problem are the *roll* (rotation around x_D), *pitch* (rotation around y_D) and *yaw* (rotation around z_D) angles of the drone because the information about the angles is not available in the Control Board on the current hardware. So we have to accept a few assumptions about the angles to be able to convert the signals between coordinate systems. First of all, we have no good way of determining the *yaw* angle and so we cannot control it automatically. But the drift in this angle is quite small and it does not affect the MAV movement as much at the other two angles. So it is sufficient if the pilot corrects the angle towards zero once in a time and we assume that this angle is always equal to 0. Furthermore, the trajectories that we are about to perform with the drones consist of translational movement with limited velocity and acceleration, therefore we can assume that the *roll* and *pitch* angles will be relatively small along the whole trajectory.

From the coordinates returned by the Gumstix module only the value along x_G is not affected by the angles (because the distance from the blob is constant when we rotate the drone's body). The value along z_G is affected by the *roll* and *yaw* angles. We can assume that $roll = 0$ and if we consider that the drone is mostly hovering in the same altitude as the blob, we can assume that the effect of the *roll* angle is negligible. The value along y_G is significantly affected both by the *roll* and *pitch* angles and because we are unable to compensate the effects on the current hardware it is the most distorted signal. However, it does not matter because instead of this signal we are using the altitude signal from PX4Flow. If we accept these assumptions, we can write

$$G = \{O_B\}_G \,,$$

$$\{O_G\}_B = -R_{G\to B} \cdot G \approx \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} G \,,$$

$$\{O_D\}_0 = \{O_G\}_B + \{O_B\}_0 \,, \tag{18}$$

where G is the vector returned by the Gumstix module (position of the blob relative to the Gumstix camera), $\{O_G\}_B$ is the position of the Gumstix camera relative to the blob, $R_{G\to B}$ is the rotation matrix from coordinate system G to system B, $\{O_D\}_0$ is the position of the drone in the system 0 and $\{O_B\}_0$ is the position of the blob in system 0 which is actually only its altitude above the ground.

The PX4Flow sensor compensates the *roll* and *pitch* angular rates, but it does not compensate the *yaw* rate. This can be imagined in such way that if the drone moves and rotates, the coordinate system P moves along with it and rotates around the z_P axis, but the axes x_P and y_P remain in the plane parallel to the floor. If we accept the assumption that $roll \approx 0$ we can write the equations

$$
\begin{aligned}
P &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \{F\}_P + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \frac{d}{dt}\{F\}_P \,, \\
P' &= -R_{P\to 0} \cdot P \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} P \,, \\
\{O_D\}_0 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} P' + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \int P' dt + \{F\}_0 \,,
\end{aligned} \tag{19}
$$

where P is the vector returned by the PX4Flow module, that is the absolute altitude above the floor and relative velocities along axes x_P and y_P). To obtain the position of the drone from the PX4Flow data, we have to invert the P vector and convert it to the base of coordinate system 0. Then integrate its first 2 coordinates and add $\{F\}_0$ which is the position of the drone (along x_0 and y_0) at the time the integration is started.

We have changed the code receiving the Gumstix and PX4Flow data to perform the specified coordinates conversion and unit conversion from mm to m for Gumstix. If any of the sensors should be mounted on the drone in a different way, the applied rotation matrix can be easily changed using precompiler defines. Also we have renamed the Gumstix variables to `aileronGumstix`, `elevatorGumstix` and `throttleGumstix` to improve the readability of the code. The filtration of the signals is newly performed in the 70 Hz loop in the position estimator.

3.1.3 Position estimator design

Because the Gumstix altitude signal is distorted and because we want to measure altitude from the ground (not relative to the blob) only the PX4Flow signal is used to estimate altitude as described in chapter 2.4. To determine the position of the drone along axes corresponding to the *Elevator* and *Aileron* control signals, we have designed a simple component to merge the signals from the PX4Flow and Gumstix modules and to cancel out the cons of each signal. Performance of this estimator is shown in figure 20.

The position controller was originally supposed to be based only on the PX4Flow sensor. According to equation 19, it is theoretically possible to determine the drone's absolute

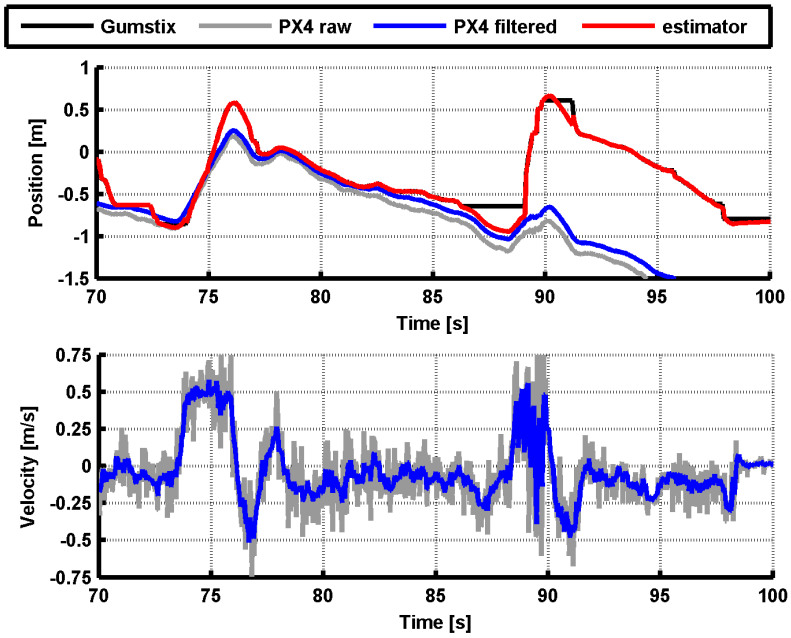


Figure 20: Converted position signals

position from the PX4Flow data if the starting position is known. However, the optical flow signal includes noise and very sensitive to floor texture and lighting conditions. Also the angle compensation might cause distortion. As a result, if we only integrate the PX4Flow data, the estimated position gradually drifts away from the actual position. This may be inconvenient for longer flights, especially in tight places. On contrary, the Gumstix module provides an absolute position, but if the blob gets out of the camera view, there is no position data at all.

The estimator component gives the filtered PX4Flow signals as the estimated velocities for the position controller. We have decided to use the PX4Flow signals only because derivation of the Gumstix signal amplifies its noise and results in a worse signal than the one obtained from PX4Flow. Also errors in blob detection would cause undesirable peaks in the signal.

If the Gumstix module returns valid data, the estimated position is the filtered Gumstix signal. If there is no valid Gumstix data, the estimator integrates the filtered PX4Flow signal. Integrating the filtered PX4Flow velocity results in smoother position signal and it can be seen in figure 20 that this doesn't increase the position drift. Also the exponential filter on the Gumstix data ensures a smoother transition from PX4Flow to Gumstix data.

Due to the position estimator, there is a single more reliable position (and velocity) signal prepared for the controller. If the blob is lost from the camera view, the controller continues to stabilize the position according to the PX4Flow data. If the blob reappears in the picture, the drone fluently returns from the drifted position to the correct one.

3.2 System identification

3.2.1 Data approximation

For the identification of the behavior of the MAV along the horizontal axes, we needed to compute a usable acceleration from the PX4Flow signal. We were not able to reuse the approximation by splines that was used on the sonar data. First of all, it was not possible to approximate the raw velocity signal by subsampling because of the substantial noise. And when the spline approximation has been applied to the signal filtered by a sliding average filter, the derivation of the approximated signal oscillated excessively. So to obtain a smoother acceleration signal, we had to expand the window of the sliding average filter, but this started to flatten the velocity signal. Nevertheless, the acceleration signal was still not sufficiently clear.

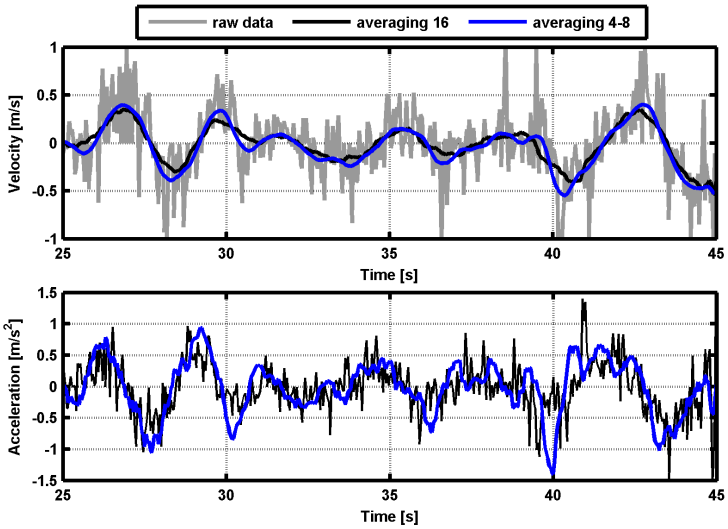


Figure 21: Velocity signal filtration

Finally, we found out that much better approximation can be obtained when the velocity signal is filtered by filter with a smaller window and the resulting acceleration is filtered as well and integrated back to velocity. The difference between approximation methods can be seen in figure 21 where *averaging 16* stands for sliding average filter with a window of ±16 samples and *averaging 4-8* means filtration of velocity with a window of ±4 samples, then filtration of acceleration with window ±8 samples and integrating back to velocity. It can be seen that the second approximation method provides smoother acceleration signal and simultaneously a velocity signal that is closer to the original PX4Flow signal. If we compute the standard deviation of the original signal with respect to the*averaging 16* signal, it is 0.232 ms^{-2} while the deviation with respect to the *averaging 4-8* signal is 0.218 ms^{-2}.

3.2.2 First test with PX4Flow only

As it was mentioned before, we have no good way of determining the *yaw* angle so we cannot control it automatically and it is the responsibility of the pilot to keep the angle around zero. According to the theoretical model, the control of drone position along the *Elevator* and *Aileron* axis should be mutually independent and the behavior should be the same along the two axes. This was confided by the experiments on the real system. With respect to this fact, we have been designing the controller using only the *Aileron* data. When the controller was finished, it was reused to control position along the *Elevator* axis as well.

Because the goal of the thesis is to stabilize to position of the drone using the data from the PX4Flow sensor, we tried to identify the system dynamics in the first experiment using only the PX4Flow data. In figures 22 and 23, there are data from the same flight with the unchanged original controller controlling the *Elevator* axis and with the first new controller controlling the *Aileron* axis. Both of the controllers have a safety saturation of the output signal to ± 75.

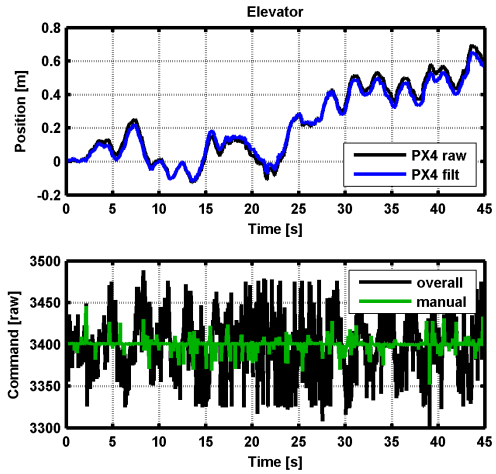


Figure 22: The original controller

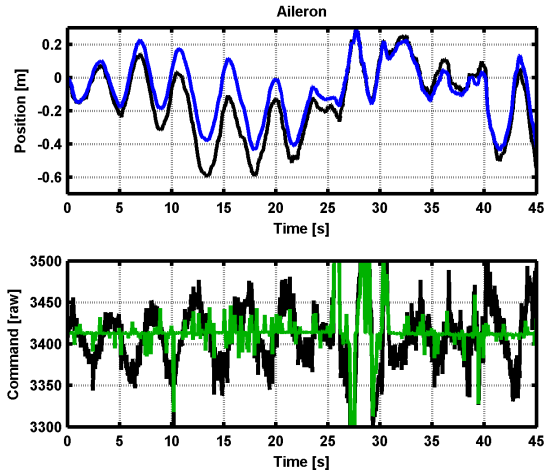


Figure 23: First velocity controller test

When the original position controller has no data from the Gumstix module (the absolute position of the blob), it attempts to regulate the velocity from the PX4Flow sensor to 0. In this mode, the original controller can be simplified to the following code.

```
error = elevatorSpeed;
proportional = 235*error;
derivative = 2*(error-elevatorSpeedPreviousError);
ctrlElevatorOutput = proportional + derivative;
```

Notice that there is no minus sign in the first command, this is because originally there

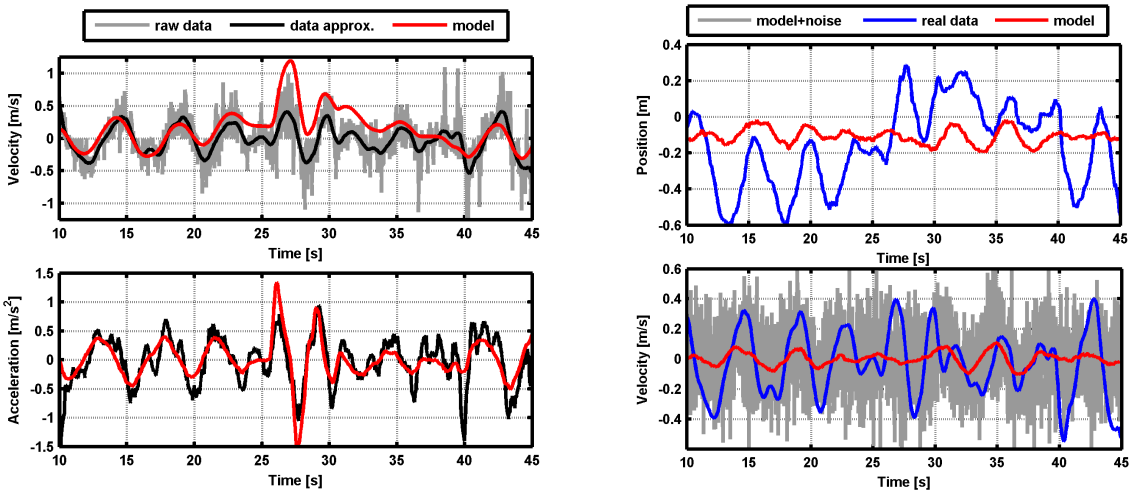
was no coordinates conversion of the PX4Flow output. Probably because of the ineffective filter described in section 3.1, it was not possible to use greater derivative action. This code was executed in the 70 Hz control loop and if we want to obtain the derivative constant corresponding to the proper derivation of the PX4Flow signal (the acceleration) we get approximately $2 \cdot 0.014 = 0.028$. So the derivative action plays nearly no role in the controllers output. Nevertheless, the overal output signal is still significantly noisy. This is again caused by the ineffective input filter.

Using the experience from the altitude controller design, we have modified the original controller and tested the new design with the real MAV right away. The exponential input filter of the controller was improved to decrease the controller output noise. Also an integration action component was added and the proportional component was reduced. The controller is now defined by equations

$$\begin{aligned} c_{(t)} &= k_P \cdot v_{f(t)} + k_I \cdot I_{(t)} \,, \\ I_{(t)} &= I_{(t-T)} - T \cdot v_{f(t)} \,, \end{aligned} \tag{20}$$

$$k_P = 150 \,, \quad k_I = 60 \,, \tag{21}$$

where v_f is the filtered velocity signal and c is the controller output. Performance of the controller is captured in figure 23. Obviously, this setting of the controller was inappropriate because it only increased the oscillations. However, the noise in controller output has been significantly reduced and the flight was visually more still.



(a) Open loop

(b) Closed loop

Figure 24: Simulation based on PX4Flow data

To identify the parameters of the system, the 3rd order model defined by equation (13) was reused. Using Matlab optimatization and LSM, the parameter values (22) have been found. using optimization in Matlab I reused the and got the following parameters.

$$k'_c = 80 \text{ m}^{-1}\text{s}^2 , \quad k'_o = 3412 , \quad \tau = 0.9 \text{ s} \tag{22}$$

Response of the model can be seen as the red signals in figure 24a. The gray signal is the real PX4Flow data and the black-colored signals are derived from it by filtration. The acceleration generated by the model is similar to the real one, even the velocity signal (integrating differences) remains close to the actual velocity. We have also identified the standard deviations of the differences between the generated and actual acceleration and of the noise of the velocity signal to

$$\sigma_v = 0.234 \text{ ms}^{-1} , \quad \sigma_a = 0.317 \text{ ms}^{-2} , \tag{23}$$

where σ_v corresponds to velocity noise and σ_a to acceleration noise. Surprisingly, the performance of the model was significantly better that the real data in a closed loop simulation with the model of the used controller (see figure 24b) even though we have added greater noise both to the simulated acceleration and velocity signal ($\sigma_v = 0.3$ and $\sigma_a = 0.4$).

The only feasible explanation of the different behavior we were able to reach is that the noise signals in the model are purely random with zero mean value and that the PX4Flow output signal probable more severely distorted than with a random noise. Then we realized that the data from the PX4Flow sensor are not entirely correct. The position signal in figure 22 generated by integrating the PX4Flow signal drifts away from 0, however in reality the drone remained in approximately the same position along the *Elevator* axis. And from the position signal in figure 23 it seems that the drone oscillated roughly around the same position along the *Aileron* axis. But at the time of 25 s, it actually drifted more than 1 meter to the right and the pilot had to act to avoid the collision. This can be seen in the lower chart of the figure.

3.2.3 Test using Gumstix module

Due to the reasons described in previous chapter, we have performed another experiment using also the Gumstix module for relative localization. And to obtain more reliable position signal, the position estimator as described in chapter 3.1.3 was implemented. Results of the experiment are shown in appendix F.

From the differences between PX4Flow and Gumstix signals, the most evident is the difference in position. In this experiment the position determined based on the PX4Flow

signal drifted approximately 0.7 meters away from the absolute Gumstix position in 40 seconds. There are slight differences in the velocity signals, especially at the time of 4 s, and the differences in acceleration signals is already significant.

We have generated an acceleration from the Aileron control signal using the 3rd order model and it can be seen that it is more similar to the acceleration from the Gumstix localization module. We have fine-tuned the model parameters using the least squares method and Matlab `fminsearch` function and obtained the following values.

$$\begin{gathered} k_c' = 89.1 \text{ m}^{-1}\text{s}^2 \,, \quad k_o' = 3381.8 \,, \quad \tau = 0.87 \text{ s} \\ \sigma_v = 0.222 \text{ ms}^{-1} \,, \quad \sigma_a = 0.161 \text{ ms}^{-2} \end{gathered} \tag{24}$$

We have also studied the properties of the acceleration and velocity noises in greater detail. Magnitude of noise of the velocity signal (obtained from the PX4Flow module) highly depends on the actual lighting conditions. It is greater if we compare the raw data signal to the velocity derived from the Gumstix position signal than if we compare it to the filtered PX4Flow signal. In this experiment, the noise relative to the Gumstix signal was 0.222 ms^{-1} and relative to the filtered PX4Flow signal it was 0.186 ms^{-1}. But the shape of the noise is practically same as the one of a simulated Gaussian noise.

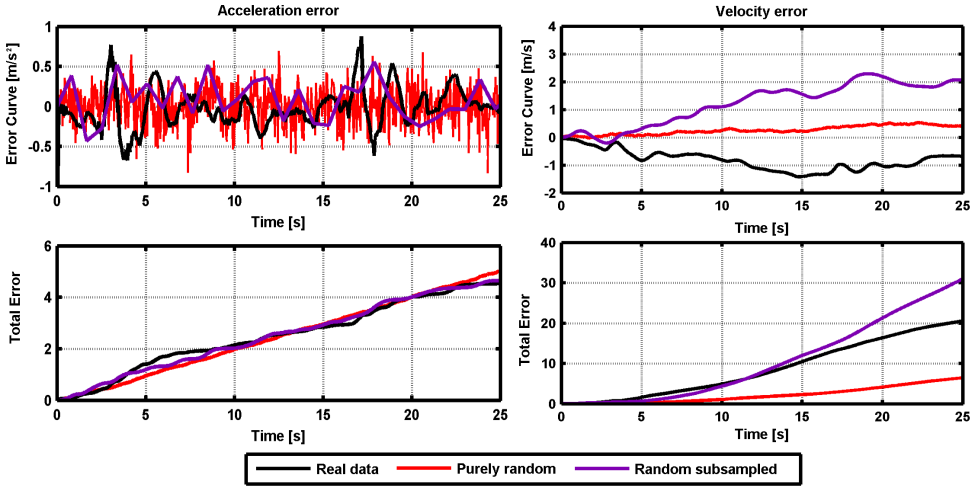


Figure 25: Position acceleration error

On contrary, the acceleration noise is greater when compared to the PX4Flow signal suggesting that the signal is actually a little distorted. In figure 25, where Total Error is integration of the absolute value of the error function, it can be seen that a generated random noise with the same standard deviation (the red signals) generates the same acceleration error as the real noise. But the noise signal is visually very different.

The result of the relatively high frequency of the generated noise is that if we integrate the acceleration error signal to obtain the error in velocity, we get a very small velocity error compared to the real one. To generate a better acceleration error signal, we subsampled the random noise to 1 sample per second and linearly interpolated the values. The result can be seen as the purple curves in figure 25.

Using the new system model parameters and error signals, we have performed a closed-loop simulations with models of the used controllers and compared it with the real behavior. Results can be seen in appendix G. In the experiment on the left side of the appendix, the controller was unable to keep the drone at constant position, the drone drifted away and so the pilot had to return it to the original position once in a while. Apparently, the simulated position of the drone oscillated and drifted similarly as the real data. Also the reactions to pilot's corrections and the shape of controller output are similar.

The experiment on the right side was performed later with the newly designed position controller and autonomously generated setpoint according to the predefined trajectory. Model parameter were set to the values identified from the data. And it can be seen that the closed loop response of the model is again very similar to the response of the real system even with a different controller.

3.3 Controller design

Given that the controller should be based primarily on the PX4Flow sensor there was a question how to handle constant position stabilization. When the absolute position from the Gumstix module is not available, the estimated position signal is obtained as integration of the PX4Flow velocity signal and it may and does drift away from the real position. If we use this position estimation signal in the controller and the pilot interacts to correct the drone's position, the drone will return to the integrated (bad) position. For this reason two controller types were implemented - *velocity controller* and *position controller*.

3.3.1 Velocity controller

First of the two types is the *velocity controller* which only tries to regulate the velocity of the drone along Aileron and Elevator axis to zero and it does not use any position information at all which enables the pilot to change the position of the drone arbitrarily.

$$v_{f(t)} = v_{f(t-T)} + \frac{T}{\tau_f}(v_{(t)} - v_{f(t-T)}) \tag{25}$$

$$a_{f(t)} = a_{f(t-T)} + \frac{T}{\tau_f}(\frac{v_{f(t)} - v_{f(t-T)}}{T} - a_{f(t-T)}) \tag{26}$$

$$\begin{gathered} c_{(t)} = k_P \cdot P_{(t)} + k_I \cdot I_{(t)} + k_D \cdot D_{(t)} \\ P_{(t)} = -v_{f(t)}\,, \quad I_{(t)} = I_{(t-T)} - T \cdot v_{f(t)}\,, \quad D_{(t)} = -a_{f(t)} \end{gathered} \tag{27}$$

$$k_P = 250\,, \quad k_I = 10\,, \quad k_D = 30\,, \quad \tau_f = 0.05 \text{ s} \tag{28}$$

The equations above define the newly implemented velocity controller. In the equations, $T = 0.014222$ is the sampling period, $v_{(t)}$ is the PX4Flow velocity signal and $c_{(t)}$ is the controller output added to the Aileron or Elevator control signal. Velocity signal is filtered according to the equation (25) and acceleration signal is obtained and filtered according to equation (26). The constants of the controller (28) were obtained by closed loop response optimization in Matlab using the model described in section 3.2.3 and then tuned on the real system to reach the desired behavior. Especially the choice of τ_f proved to be critical because too low value caused twitches of the attitude of drone's body and destabilized it. Too high value caused excessive oscillations due to the delay of the filtered signal.

Table 2 shows statistical comparison of the velocity controllers and manual control performances. The used velocity signals are obtained using the approximation method described in 3.2.1, σ is the standard deviation of velocity from zero, E_{max} is the maximum deviation and l is the length of used dataset. Part of the dataset is displayed in appendix H. While the filtration of the velocity signal decreased the noise of the controller's output signal (as can be seen in figure 23), the noise was increased by using the derived acceleration signal. So eventually the shape of the new controller output is similar to the one of the original controller, but the usage of the acceleration signal provides more timely feedback thus improving performance of the controller.

Table 2: Performance of velocity controllers

	σ [mm]	E_{max} [mm]	l [s]
Manual control	64	175	70
Original controller	93	338	40
Implemented controller	81	203	40

An interesting result is that while the implemented altitude controller gives better performance than a human pilot, we were unable to improve the velocity controller enough to match the performance of the pilot. We believe that this is caused by the fact that the controller is limited by the quality of the PX4Flow optical flow signal (which is quite noisy and otherwise distorted) while the pilot virtually sees all the real state variables - position, velocity and angle of attitude (which corresponds to the acceleration).

3.3.2 Position controller

The second type of controller is a *position controller* which tries to regulate the estimated position of the MAV to the position setpoint. The setpoint may be changed dynamically either by the pilot or autonomously (trajectory following). Using this controller, the drone is locked to the estimated position (which may not be correct) for the whole time the controller is on. So with this controller, it is not possible to correct the position of the drone. However the drift should be slower than with the velocity controller. Furthermore, if the blob mark appears in the Gumstix camera view the drone fluently transfers to the correct position.

At first, we tried to use a plain PID controller with input signal filtration but we were unable to stabilize the position of the drone using this controller. The drone kept oscillating as it is displayed in figure 26. So we decided to reuse the working velocity controller and add a fourth action component based on the position signal. This makes the controller a kind of state feedback controller which can be defined by equations

$$\begin{aligned} c_{(t)} &= k_p \cdot P_{(t)} + k_I \cdot I_{(t)} - k_v \cdot v_{f(t)} - k_a \cdot a_{f(t)} \,, \\ P_{(t)} &= (sp_{(t)} - p_{e(t)}) \,, \quad I_{(t)} = I_{(t-T)} + T(sp_{(t)} - p_{e(t)}) \,, \end{aligned} \tag{29}$$

$$k_p = 85 \,, \quad k_I = 5 \,, \quad k_v = 180 \,, \quad k_a = 10 \,, \tag{30}$$

where $sp_{(t)}$ is the position setpoint, $p_{e(t)}$ is the position signal from the estimator (see chapter 3.2.3) and $v_{f(t)}$ and $a_{f(t)}$ are the velocity and acceleration signals computed from the PX4Flow signal according to the equations (25) and (26) respectively. The parameters (30) of the controller were again tuned on the model of the system and fine-tuned during the flight experiments to achieve the optimal behavior of the drone.

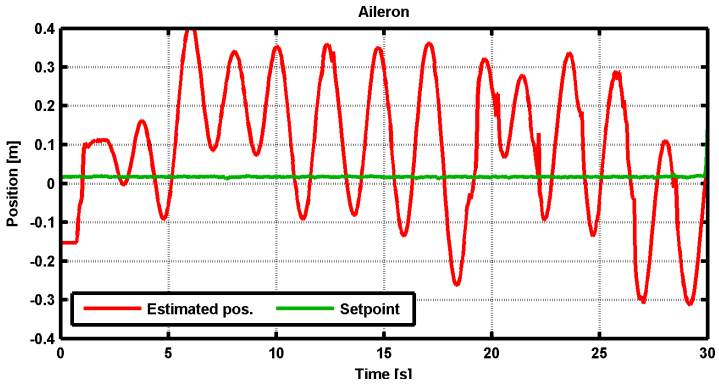


Figure 26: First position controller test

The implemented controller is able to stabilize the drone on a spot and compensate errors, e.g. introduced by the manual offset as in figure 27. Using the controller, the drone is also able to follow a moving target. In the experiment captured in figure 28, the blob was slowly moved away from the drone and then slowly pushed towards it to the original position. In this case, the gray signal (obtained only by integrating the PX4Flow signal) represents the approximate absolute position while the red signal (output of the position estimator equal to the filtered Gumstix signal) represent the relative position from the blob and the controller tries to regulate the estimated position to the setpoint. In the time of 180 s the blob was moved rapidly to verify if this can destabilize the drone. It can be seen from the PX4Flow signal that the drone kept its stable position because the controller uses the absolute velocity signal from PX4Flow only and not the velocity relative to the blob. Similar experiment can be seen in video `3_blob_following` on the attached DVD.

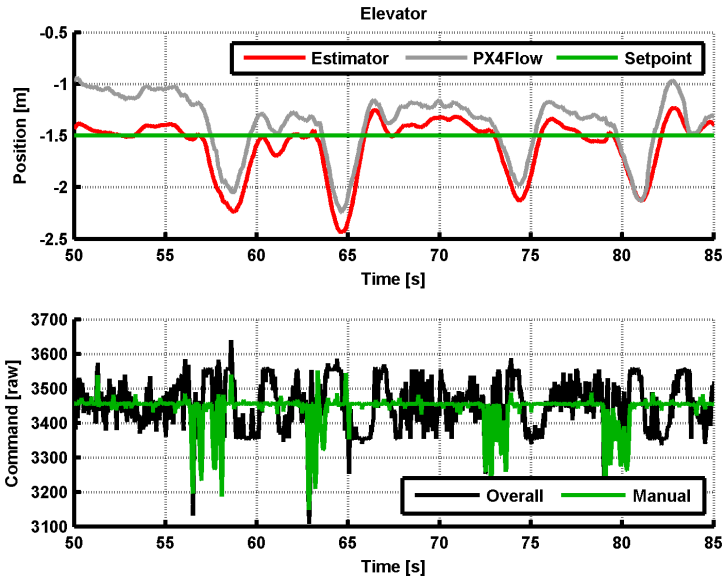


Figure 27: Manual errors compensation

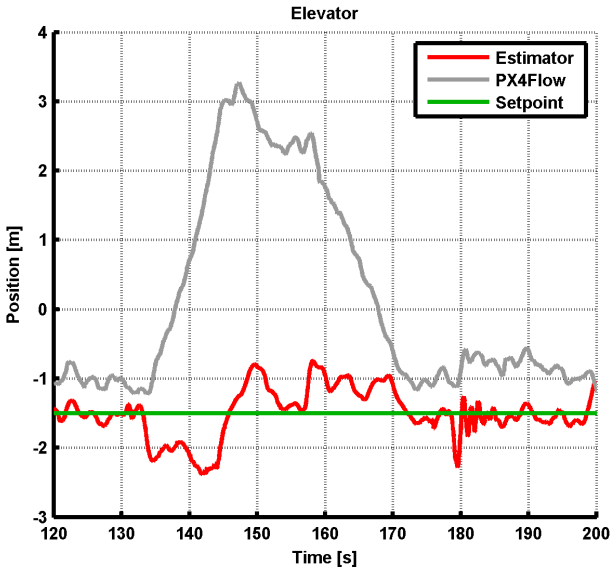
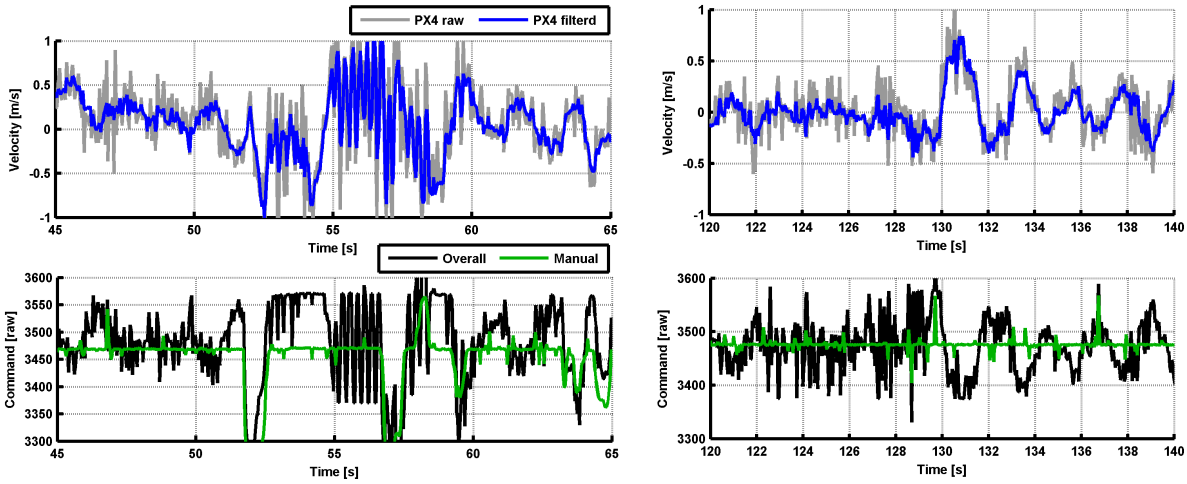


Figure 28: Following a moving target

3.3.3 Velocity limitation

The controller worked well for small position deviations. But when there was a relatively big change of the position setpoint, the system reached higher velocity and became unstable - the drone's attitude started to oscillate rapidly. The behavior can be seen in figure 29a where the pilot manually interacted to drag the drone away from the blob at the time around 52 s. As a result, the controller gave a full positive output for longer than 2 seconds reaching a relative high velocity and became unstable continuing the movement with high velocity. So the pilot had to act again to stop the drone from colliding with the blob. It can be seen that when the velocity decreased, the system became stable again.



(a) No limitation

(b) With velocity limit

Figure 29: Higher velocity problem

We have identified the problem to be caused by the PX4Flow sensor. It is obvious from the data that the sensor is unable to properly measure velocities over 0.5 ms^{-1}. Probably because at these speeds, there is a completely different picture in the PX4Flow camera view, the optical flow cannot be computed and the sensor returns invalid (too low) velocity. The difference (in velocity and acceleration) is amplified by the controller causing the drone to rapidly change the attitude. This causes another rapid change in the PX4Flow signal (there might also be some problem with PX4Flow's internal angle compensation in high velocities) and the closed-loop system starts to oscillate.

To prevent this problem from occurring and also to improve the safety of autonomous flights (see the rule number 2 in 1.3), we have implemented a modification of the controller limiting the velocity of the motion. We found out that the equations defining the controller (29) can be easily converted to a layered controller structure defined by

$$\begin{aligned} c_{(t)} &= k_v(v_{d(t)} - v_{f(t)}) + k_I \cdot I_{(t)} - k_a \cdot a_{f(t)} \\ v_{d(t)} &= \tfrac{k_p}{k_v}(sp_{(t)} - p_{e(t)}) \quad I_{(t)} = I_{(t-T)} + T(sp_{(t)} - p_{e(t)}) \end{aligned} \tag{31}$$

where v_d is the controller's desired velocity signal which is saturated to ± 0.33 ms^{-1} in the final controller implementation. The result of this modification can be seen in figure 29b where the blob mark was revealed in a distance from the drone in approximately 128 s. The drone moved slowly and safely to the new position in front of the blob.

3.4 Summary

The responses of the newly implemented position controller are statistically better than the ones of the original controller. However, we found out that the quality of the performance considerably varied between the individual experiments depending on the current conditions. So the difference was not always so evident as with the altitude controller. The conditions may include the wind or drought, texture of the ground surface and for indoor experiments, mostly the lighting conditions. Even the charge state of the battery and slight differences in the drone's hardware setup or balance may have had their influences.

The results are summarized in table 3 which shows the used controllers, lighting conditions, the standard deviation of estimated position from setpoint (σ) and length of the dataset (l). All experiments were performed with a static blob in front of the drone and with a constant setpoint. Part of the two dataset with good light conditions (direct sunlight) is displayed in appendix I, all the signals are the real measured data.

At the attached DVD there is video `1_original_controllers` capturing a flight with the original position and altitude controller and video `2_new_controllers` capturing a flight with the newly implemented controllers. Both videos were shoot at the same day with the same conditions (moderate light). In the second video the controller type was switched from *position controller* to *velocity controller* at the time of approximately 26 s. After a while when the drone drifted away (at 48 s) the control was switched back to the *position controller* and it can be seen that the drone returned back to the correct position.

Table 3: Performance of position controllers

Position ctrl.	Altitude ctrl.	Light conditions	σ [mm]	l [s]
Original	Original	moderate	193	80
Original	New	good	130	90
New	New	bad	149	60
New	New	moderate	113	70
New	New	good	62	65

As the main benefit of the new controller we see the robustness and reliability of the implementation gained due to data filtration and position estimation based on fusion of data from multiple sensors. As a result, the drone is able to stabilize its position and recover from undesired position changes (caused by wind, pilot introduced errors, etc.) even without seeing the blob mark and thus having an absolute position reference (if we overlook the slow drift). Furthermore, if the blob appears in the camera view, the drone fluently corrects its position relatively to the blob and it is stable even if the blob is quickly moved (for instance, if it is mounted to another drone). And most importantly, this controller was a base enabling to implement advanced features as autonomous landing and takeoff, autonomous trajectory following and formation flying.

The video `3_blob_following` is an example showing the robustness of position stabilization. The first part of the video captures the drone being able to follow the blob (stabilize its position relative to the blob mark). At the time from 0:39 to 0:47 the drone cannot see the blob and stabilizes its position according to the PX4Flow data only. It can be seen that the drone doesn't become unstable even if the blob's position oscillates or if it is moved relatively fast (time 0:58 to 1:10). At the end of the video, the drone successfully corrects its position (relative to the blob) after a while when the blob is hidden from the Gumstix camera and moved to another position.

3.4.1 Drawbacks

The primary (and many times the only one) source of feedback data for the controller is the P4Flow module and I see this concept as one of the main bottlenecks of the whole system. Because the quality of the controller performance is thus tightly bound to the quality of the PX4Flow signal which is not always good. It limits the speed of drone's motion and makes the quality of stabilization dependent on the light conditions. It might be possible to improve the quality of the velocity signal by using velocity derived from the Gumstix position signal (when available) but this would probably destabilize the system while following a moving target as described in [14] which is very undesirable.

Also the controller is currently implemented on a board that was originally designed for simple communication purposes only and not at all for complex computational tasks. The control MCU has only two serial communication lines and so the information about the angles of the attitude of the drone's body (from the stabilization board) can't be routed to the control board with the current hardware. Nevertheless, using this information could significantly improve the performance of the controller.

With this small 8-bit MCU that doesn't have a FPU, it was basically impossible to implement a more sophisticated controller type than a plain PID-like one. Actually, at the end of work on this thesis the program memory was completely full and I even had to reimplement code parts from float operations to integer operations where it was feasible,

to be able to “fit” all the new features to the memory. Also the MCU’s execution time is completely used up. Most of the tasks are executed at rate of 70 Hz but some had to be moved to a slower rate when all functionalities are turned on. However the manufacturer of the PX4Flow module declares that the sensor is able generate optical flow data up to 250 Hz in ideal conditions so there is a possibility that the quality of stabilization could be improved if we were able to process the input data (and possible also compute the output action signals) at higher rates.

For these reasons, we believe that the implemented controller reached the hardware limit, that it is the best solution that was possible at the current situation and that further improvements would require a hardware rework.

4 Advanced features

4.1 Landing and takeoff

4.1.1 Autonomous landing

When we had a reliable altitude and position controller, it was possible to start developing autonomous lading/takeoff functionality. The autonomous landing can be turned on by a switch on the RC transmitter. When it is turned on, the *altitude controller* stabilizes to velocity along vertical axis to slow descent while the *velocity controller* stabilizes the drone to zero velocity in horizontal plane. When the controller detects that the drone is on the ground, the output of the controller is set to the lowest possible value.

The *velocity controller* is the same as described in chapter 3.3.1. The output of the *altitude controller* is computed according the following discrete equations in the autonomous landing mode.

$$\begin{aligned} c_{(t)} &= k_v(v_d - v_{e(t)}) + k_I \cdot I_{(t)} \,, \\ I_{(t)} &= I_{(t-T)} + T(v_d - v_{e(t)}) \,, \end{aligned} \tag{32}$$

$$k_I = 120 \,, \quad k_v = 180 \,, \quad v_d = -0.4 \text{ ms}^{-1} \,. \tag{33}$$

In these equations $c_{(t)}$ is the controller output and $v_{e(t)}$ is the estimated velocity along vertical axis generated by the altitude estimator component described in chapter 2.4. The constants of the controller (33) where v_d is the desired descent speed, were selected experimentally.

The performance of the lading altitude controller can be seen and compared to the manual landing in appendix J. The dataset was shortened (the flight phase marked by the gray line in the middle of the charts was cut out) to display both the takeoff and landing. For manual flight the displayed position signal is a spline approximation computed offline and the velocity signal is derived from it. For the autonomous flight the displayed position and velocity signals are the signals computed onboard by the altitude estimator. Note that the landing phase is initiated by the short decrease of the of the controllers output to value about 3230, not by the drop under 2900 (which is actually the end of the landing phase).

4.1.2 Autonomous takeoff

The autonomous takeoff functionality was actually already implemented by the *altitude* and *velocity* controllers. All that is needed to start an autonomous takeoff is to set the manual offset (high enough to make the altitude controller able to takeoff), turn on the controllers and possibly turn off the autonomous landing.

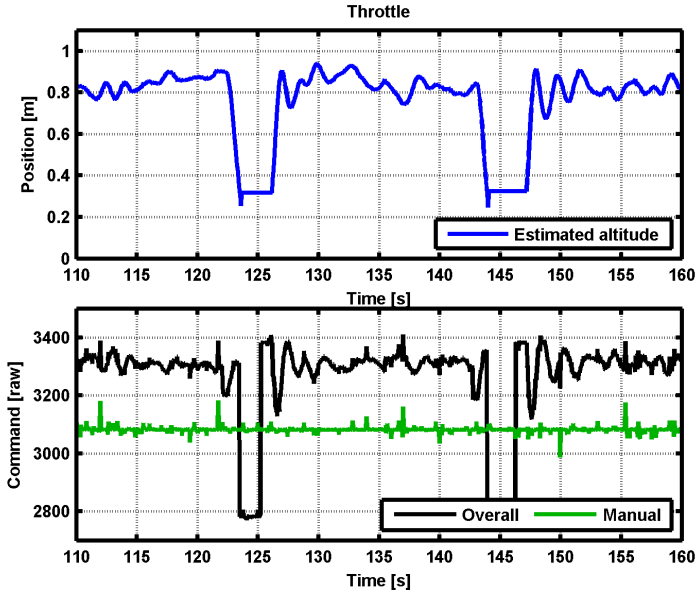


Figure 30: Autonomous takeoffs and landings

We have only modified the altitude controller similarly as the position controller to limit the desired velocity. The data from the sonar sensor is usually reliable and the system behaves well even at relatively high speeds along the vertical axis. So it is merely a safety measure. The output of the takeoff (and altitude) controller can then be computed according to the equations

$$\begin{aligned} c_{(t)} &= k_v(v_{d(t)} - v_{e(t)}) + k_I \cdot I_{(t)} \,, \\ v_{d(t)} &= \tfrac{k_p}{k_v}(sp_{(t)} - p_{e(t)}) \,, \\ I_{(t)} &= I_{(t-T)} + T(sp_{(t)} - p_{e(t)}) \,, \end{aligned} \tag{34}$$

$$k_p = 180 \,, \quad k_I = 120 \,, \quad k_v = 200 \,, \tag{35}$$

where sp is the altitude setpoint, p_e and v_e are the estimated position and velocity signals respectively and v_d is the desired velocity which is saturated to ± 0.8 ms^{-1}. In figure 30, there is a row of two autonomous landing and two autonomous takeoffs displayed. It can be seen that the pilot did not control the altitude (constant manual offset) and the controller

performed the landings and takeoffs when requested. Autonomous takeoff and landing can also be seen in video `4_autonomous_takeoff_and_landing.mov`, the flight is controlled by the velocity controller (regulation to zero horizontal speed).

In appendix J, it can be noticed that the vertical speed overshoots to approximately 1 ms^{-1} in the takeoff phase and to -0.5 ms^{-1} in the landing phase. The quality of vertical speed regulation could probably be improved, if a vertical acceleration signal was derived (similarly as the vertical velocity) and used in the control loop. But the displayed performance was completely satisfactory for the tasks of autonomous takeoff and landing so we did not consider this a real problem and invested time into implementation of other features.

4.1.3 Landing state machine

On the other hand, what needed to be secured was the synchronization of controllers cooperation in takeoff and landing phases. For example, if the drone would be placed on the ground in a distance in front of the blob and the altitude and position controllers were turned on, it would result in a straight forward takeoff which is a risky maneuver and it would be difficult for the controllers to stabilize the drone in the target spot. Furthermore, takeoff is a relatively fast maneuver and the PX4Flow sensor may not give proper optical flow data for the stabilization.

Therefore we want the drone to perform purely vertical takeoffs and start the horizontal movement after its altitude is stabilized. Similarly, we want the drone to perform purely vertical landings. In other words we want it to stabilize its horizontal velocity to zero and then initiate the landing phase. To ensure this behavior I have designed a state machine displayed in figure 31.

Standard takeoff procedure: The takeoff starts with the drone on the ground (state OnGround of the landing state machine), motors are armed, autonomous mode is enabled (so the outputs of controllers are added to the control signals), *landing request* is turned on (LR = 1) so that controller keeps the drone on the ground and the *Throttle* manual offset is set high enough to make the controller able to takeoff [8].

When the *landing request* is turned off (LR = 0), the state machine transits to the state Takeoff. The *velocity* and *altitude* controller are turned on in this state and so the drone takes off vertically. Once the altitude of the drone is stabilized the machine transits to state Flight. The altitude is considered stabilized if the difference between altitude setpoint and the estimated altitude is less than 0.1 m and the estimated vertical velocity is in the range ± 0.2 m/s for longer than 0.5 s.

[8] All of these states can be controlled by the pilot using the switches and levers on the RC transmitter.

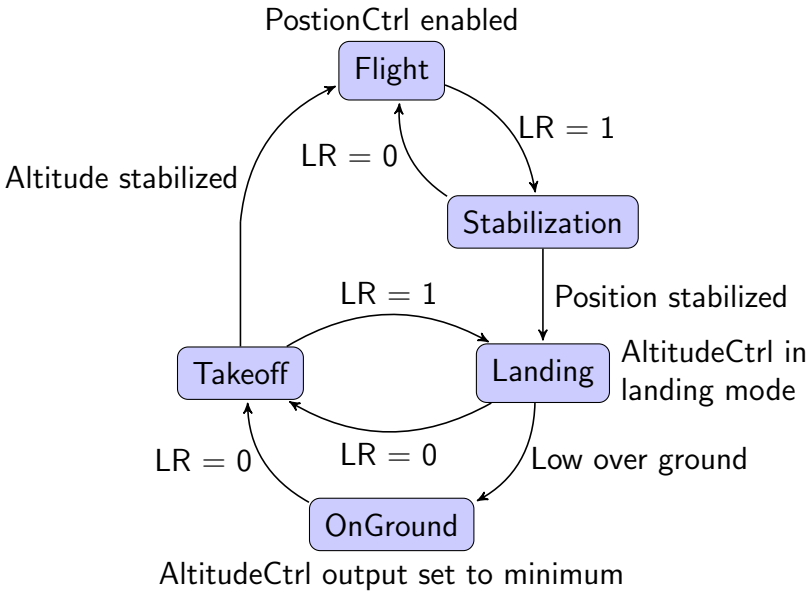


Figure 31: Landing state machine

In state Flight the *position* controller is enabled and if it is also turned on by the pilot, the drone starts to regulate its horizontal position towards the setpoint (that might be dynamically changing).

Standard landing procedure: The landing phase starts with the state machine in state Flight. In this state the drone might be following a dynamically changing setpoint. When the pilot turns on the landing request the machine transits to state Stabilization in which the controller type is switched from *position* controller (if turned on) to the *velocity* controller regulating the drone's vertical speed to zero.

When the position of the drone is stabilized the machine transits to state Landing. Because the optical flow signal from the PX4Flow module is quite noisy and because the position of the drone always oscillates a little bit, it is not easy to detect when it was stabilized. So the transit is currently initiated 1 s after the *velocity* controller is turned on.

In the state Landing the *altitude* controller is switched to the landing mode in which it regulates the vertical velocity while the *velocity* controller is still on, so the drone performs a purely vertical landing. While in the beginning of the landing phase it is essential to regulate the vertical speed to a slow descent to prevent a hard landing, in the end of the landing phase (low over the ground) the control output should be quickly decreased, to prevent the drone from bouncing off the ground back to the air or rolling over.

The lowest altitude value that the sonar sensor is able to measure is 0.3 m over the ground. It also returns this value if it is unable to measure the ground distance properly,

so there are occasional negative peaks with this value in the sensor signal. The state of the landing state machine is therefore changed to OnGround when the value of the ground distance is less than 0.35 m for longer than 0.1 s. In this state the output of the *altitude* controller is set to the possible value to ensure a proper finish of the landing phase and to prevent the drone from taking off again. After the landing the pilot usually turns of the autonomous mode and disarms to motors or he can initiate another autonomous takeoff by turning off the *landing request*.

Other transitions: There are a few non-standard transitions that take place when the value of the *landing request* changes during the takeoff/landing phase. If it is turned on in the state Takeoff the drone lands back to the ground. If it is turned off in the state Stabilization (before the position was stabilized) the drone stays in the air (state machine returns back to state Flight). And if the *landing request* is turned off in state Landing the altitude controller is switched to standard mode returning the drone into the air (state Takeoff).

4.1.4 Safety

The takeoff maneuver is relatively fast and the PX4Flow sensors do not give valid data in the initial takeoff phase when the drone is low above the ground. Therefore, it takes some time for the controllers to recover from the autonomous takeoff maneuver and to stabilize the MAV. As a result of this fact, the takeoff may not be straight vertical if the drone starts from an uneven surface or if the Stabilization board is not properly calibrated. So a safe standoff distance of a drone from obstacles or other drones should be kept.

On the other hand, the implemented autonomous landing procedure is robust and the landing is always nearly vertical. The state OnGround of the landing state machine makes the drone able to safely land on rugged or slightly inclined surface. So this enables future implementation of the safety rule 5 (see 1.3). If there was enough free program memory on the Control Board, it would be possible to implement an automatic motor disarming procedure as well.

4.2 Trajectory following

The trajectory following is functionality is implemented by a component that autonomously changes setpoints of the *position* and *altitude* controllers. The trajectory is generated by linear interpolation from anchor points stored in RAM memory on the Control Board. Each point consists of 3 position coordinates and a time value which corresponds to a special trajectory following timer that is reset each time the trajectory following is turned on.

The trajectory following component is designed in such a way that it would be possible to change the trajectory during flight. For example to load an updated trajectory from a path planning system running on a ground computer. However, the communication channel from a ground computer via the WiFi link of the onboard Gumstix microcomputer to the Control Board is currently not implemented. So at the time being, the trajectory has to be programed into the program FLASH memory of the MCU on the Control Board and it is loaded to the RAM memory when the board is powered on or restarted. With all the features described in this thesis implemented on the Control Board, there was nearly no free program memory, so the trajectory was limited to 10 anchor points plus the *starting point* which used as a setpoint for controllers when trajectory following is turned off.

Figure 32: Drone autonomously following a trajectory

The trajectory following component can be turned on by the pilot using the RC transmitter. Nevertheless, it starts to operate only if the *position* controller is enabled and if the landing state machine is in the Flight state. This enables to turn on the trajectory following mode while the MAV is on the ground so that it will start following trajectory right after an autonomous takeoff once the altitude is stabilized.

When the drone reaches the final point of the trajectory, it stays at the point and waits for a pilot command. If the trajectory following mode is turned off at the end or in the middle of a trajectory, the drone returns to the starting point. If the *position* controller

is turned off while in the trajectory following mode, the *velocity* controller takes over the control and the MAV stabilizes itself in the current position. Similarly, if an autonomous landing is requested, the drone stabilizes its horizontal position and then performs the landing.

The first experiment in which we tested the trajectory following functionality is captured in video `5_autonomous_trajectory_following.mov`. From the measured data displayed in figure 33 it can be seen that the drone successfully managed to follow the testing trajectory.

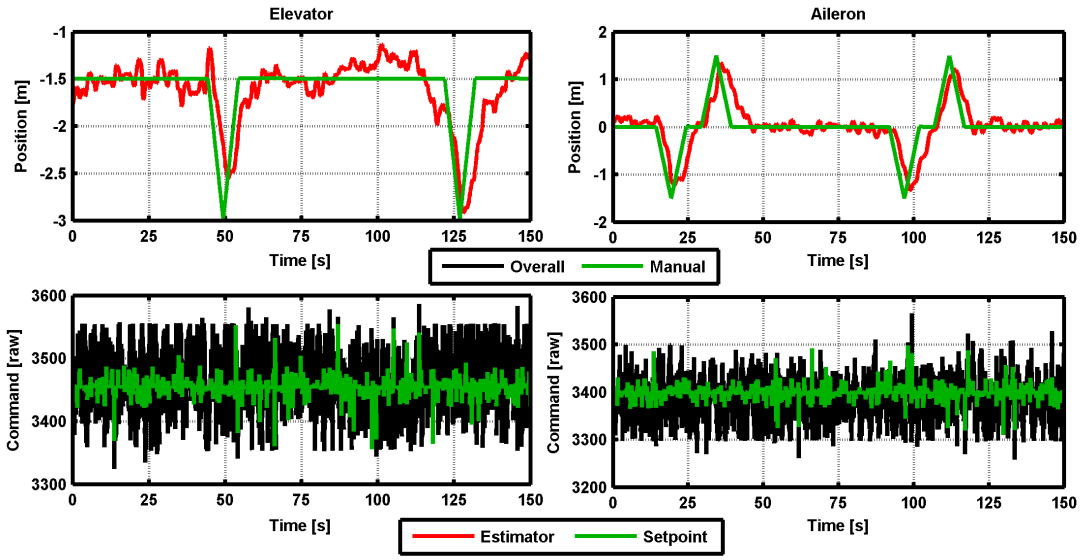


Figure 33: Trajectory following test

We have also performed experiments in which a drone was supposed to follow a trajectory precomputed by a planing software. The trajectory incorporated an obstacle avoidance while in forward motion. Data from one of the experiments can be seen in appendix K where the generated setpoint signals correspond to the linearization of the precomputed trajectory (one anchor point per 5 seconds). Position estimation was done based on the PX4Flow data only as there was no reference blob.

Table 4: Performance of trajectory following

	σ_{th} [mm]	E_{th} [mm]	σ_{el} [mm]	E_{el} [mm]	σ_{ai} [mm]	E_{ai} [mm]	l [s]
testing	35	171	251	972	304	875	150
experiment 1	51	172	251	685	308	651	60
experiment 2	44	151	306	669	200	506	65

The table 4 summarizes quality of trajectory following from the experiments. The first row corresponds to the experiment with testing trajectory. In experiments 1 and 2, MAV followed the precomputed trajectory. The two experiments were performed at the same day but in windy conditions. Therefore, the results slightly differ. In the table, values with subscript $_{th}$ correspond to the *throttle* (altitude) axis, values with subscript $_{el}$ correspond to the *elevator* (forward) axis and values with subscript $_{ai}$ correspond to the *aileron* (sideway) axis. σ is the standard deviation of the estimated position from the setpoint signal, E is the maximum deviation of the same signals and l is the length of the dataset.

4.2.1 Trajectory limitations

The main limitations of the trajectories that can be followed by the MAV using the implemented solution are:

- **Number of anchor points** limited by the amount of free space in Control Board's program memory. The limit is currently 10 points.
- **Altitude of flight** which is limited by the measuring range of the sonar sensor on the PX4Flow module. The recommended lowest value is 0.5 m. Concerning the upper limit, an altitude of 2 m has been tested to be feasible and safe.
- **Maximum horizontal velocity** which is currently explicitly limited to 0.33 m/s so that the position controller would be always able to stabilize the drone using the PX4Flow signal (see section 3.3.3).
- Position along elevator axis must be negative if the blob mark is to be used as a position reference. Distance of at least 1 m in front of the blob is advised.

4.3 Formation flights

At the end of work on this thesis, we have performed several experimental flights with formations of multiple MAVs. Most of the experiments were part of a reportage shooting of Czech Television. Due to the lack of time, I was able to obtain data from only one experiment which is captured in the video `8_dynamic_formation.mov` on the attached DVD. There were two drones autonomously flying in the experiment. The *leading* drone was following a precomputed trajectory based on the PX4Flow sensor data in the same way as described in previous chapter. The *following* drone was following the *leading* drone. More specifically, it had a constant position setpoint relative to the blob attached to the *leading* drone.

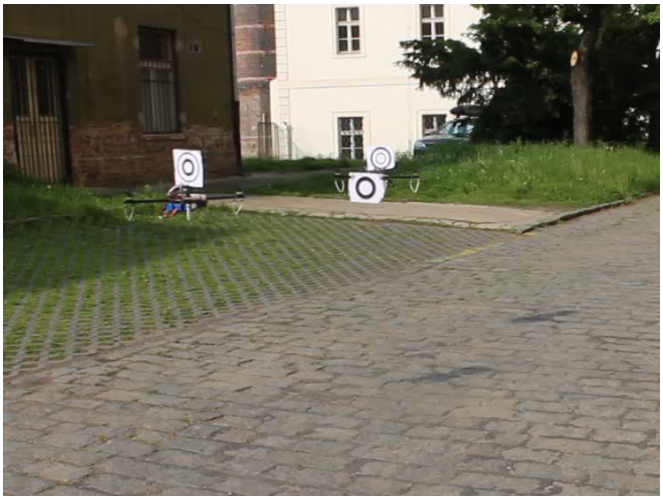


Figure 34: Two drones following trajectory in a formation

The data from the *following* drone can be seen in appendix L where the position signals from the Gumstix module (red lines) correspond to the relative position against the *leading* drone and the PX4Flow signals roughly correspond to its absolute position. The Gumstix altitude signal is shifted 1 m up to make it comparable to other signals. The green solid signals consist the setpoint of the *following* drone and the green dashed signals display the trajectory followed by the leading drone [9].

The altitude controller of the *following* MAV is regulating its absolute altitude above the ground to 1 m while the altitude controller of the *leading* MAV follows the assigned trajectory. As a result the relative altitude measured by the Gumstix module corresponds to (1 - leader altitude). In other words, if the leading drone ascends, the following drone relatively descends.

[9] The leader setpoint signals were not measured during the experiment. Measured signal from the trajectory following experiment described in 4.2 were reused. The synchronization with other signals is estimated.

The position controller of the *following* drone regulates its relative position to the constant setpoint value that is 1.5 m behind the *leading* drone. On the elevator chart, it can be seen that the drone manages to follow the *leading* drone along the forward axis. In the aileron chart a similar effect as the one with the altitude can be seen. But in this case, the controller tries to follow the sideway position of the *leading* MAV. The aileron PX4Flow signal is quite different from the desired leader trajectory. This is because the leader is not precisely following the trajectory and because the follower reacts to the change of its position with a delay. Furthermore, the position signals are distorted by integration of the noise of both leader's and follower's PX4Flow signal.

Figure 35: Preparations of experiments

Figure 36: Demonstrational experiments

Further experiments were performed as a demonstration of MAV swarms for the shooting of Czech Television. In the experiment captured in video `6_static_formation_of_5_drones.mov`, there is formation of 3 autonomously stabilized MAVs joined by two manually operated drones. The stabilization was performed based on the PX4Flow data only. A similar experiment can be seen in video `7_static_formation_plus_ugv.mov` where there is again a formation of 3 autonomously controlled drones and another manually operated drone takes over from a UGV.

Finally, there is an experiment (video `9_dynamic_formation_over_slope.mov`) in which two drones autonomously follow a trajectory in a formation up a slope. This is possible because the altitude of the drones is controller relatively to the surface beneath them. The leading drone is following the precomputed trajectory described in 4.2 and the other drone is following the leading drone. On the top of the slope both MAVs performed an autonomous landing.

Figure 37: Two MAVs flying up a slope autonomously

4.3.1 Formation limitations

The main limit in formation flights is the distance between formation members that should be large enough to avoid collisions. The same applies to the distance from obstacles. A spacing distance of 1.5 m has been tested to be safe. A greater spacing is advised if the members of a formation are supposed to perform an autonomous takeoff or landing at the same time as the maneuvers may not be perfectly vertical depending on external conditions (e.g. wind, surface inequalities) and the neighboring drones may collide.

If the drones are supposed to fly in a tight formation or to collectively follow a trajectory, they should be also stabilized relatively to each other. For this stabilization a visual link consisting of a blob on one drone and a Gumtix module on the other drone is needed. The MAVs need to be in such relative position that the blob mounted to the *leading* MAV would be in the view of the Gumstix camera of the *following* MAV and they must be close enough so that the Gumstix module would be able to recognize the blob pattern [10]. A distance of 3 m has been proved feasible. Currently, the Gumstix camera is

[10] The Caspa camera has fixed focus and if the image of the blob is blurry, it cannot be recognized.

mounted on the MAVs in such way that it has wider viewing angle along the vertical axis which enabled to perform the experiment over a slope.

If the drones are autonomously following a trajectory, the limitations described in chapter 4.2.1 also apply. Furthermore, if the leading drone moves in a direction towards the following drone, the motion should be slow enough so that the following drone had enough tome to stabilize the formation. It has been tested that the leading MAV can move straight away from following MAVs at the top speed limit (see 3.3.3) and the following MAVs are able to follow the leading one without any problems.

5 Conclusion

All the assignment items of this thesis have been fulfilled. In spite of the lack of on-board computation power, a functional altitude and vertical position control system has been implemented. The system uses the data from the PX4Flow sensor as the primary feedback and it enables to perform autonomous takeoff and landing maneuvers and to autonomously follow a given trajectory. The trajectory can be changed during flight, but due to the absence of a communication link from a ground computer to the MAV's control board, the trajectories have to be programmed into the control board's FLASH memory before a flight.

The implemented system was verified both by simulations with a model of MAV's dynamics identified as a part of this thesis and by a series of real experiments with up to three autonomously flying MAVs. The limitations of trajectory following and formation flying with the use of the implemented controller have been described and it has been shown that the implemented control system gives statistically better position stabilization performances than the system that was used previously to this thesis. Furthermore, the newly implemented system is more reliable and provides more robust moving reference tracking.

During the work on the thesis, I have experienced the pitfalls of working with real hardware and I have learnt that working with it and performing the experiments is very time consuming. I have also learnt that a real-world dynamic system may behave considerably differently than a theoretical model and that it is a difficult task to identify an exact cause of the difference.

References

[1] Intelligent and Mobile Robotics Group at CTU. *Swarm Robotics*. `http://imr.felk.cvut.cz/Swarm/Swarm`, 2011.

[2] University of Pennsylvania. *General Robotics Automation Sensing and Perception Laboratory*. `https://www.grasp.upenn.edu/`.

[3] Swiss Federal Institute of Technology. *Institute for Dynamic Systems and Control*. `http://www.idsc.ethz.ch/`.

[4] J. Eckert, R. German, and F. Dressler. *On autonomous indoor flights: High-quality real-time localization using low-cost sensors. IEEE International Conference on Communications*, 2012.

[5] S. Joshi, A. Kelkar, and J. Wen. *Robust attitude stabilization of spacecraft using nonlinear quaternion feedback. IEEE Transactions on Automatic Control*, 40(10):1800–1803, October 1995.

[6] Taeyoung Lee. *Robust Adaptive Attitude Tracking on SO(3) With an Application to a Quadrotor UAV. IEEE Transactions on Control Systems Technology*, 21(5):1924–1930, September 2013.

[7] S. Esteban and D. Rivas. *Singular Perturbation Control of the Longitudianal Flight Dynamics of an UAV*. In *UKACC International Conference on Control 2012*, September 2012.

[8] Ashfaq Ahmad Mian and Wang Daobo. *Output Feedback Control of a Quadrotor UAV Using Neural Networks. Chineese Journal of Aeronautics*, 21:261–268, March 2008.

[9] Fabian L. Mueller, Angela P. Schoellig, and Raffaello D' Andrea. *Iterrative Learning of Feed-Forward Corrections for High-Performance Tracking*. In *IEEE International Conference on Intelligent Robots and Systems 2012*, October 2012.

[10] T. Dierks and S. Jagannathan. *Output Feedback Control of a Quadrotor UAV Using Neural Networks. IEEE Transactions on Neural Networks*, 21(1), January 2010.

[11] Jian Han, Chang hong Wang, and Guo xing Yi. *Cooperative Control of UAV Based on Multi-Agent System*. In *IEEE Conference on Industrial Electronics and Applications*, 2013.

[12] MikroKopter. *MK Basicset L4-ME*. `https://www.mikrocontroller.com/index.php?main_page=product_info&cPath=80&products_id=434`.

[13] Jan Faigl, Tomáš Krajník, Jan Chudoba, Libor Přeučil, and Martin Saska. *Low-Cost Embedded System for Relative Localization in Robotic Swarms*. In *IEEE International Conference on Robotics and Automation*, 2013.

[14] Tomáš Báča. *Control of Relatively Localized Unmanned Helicopters*. Bachelor Thesis, Czech Technical University in Prague, 2013.

[15] PX4. *PX4Flow*. `http://pixhawk.org/modules/px4flow`.

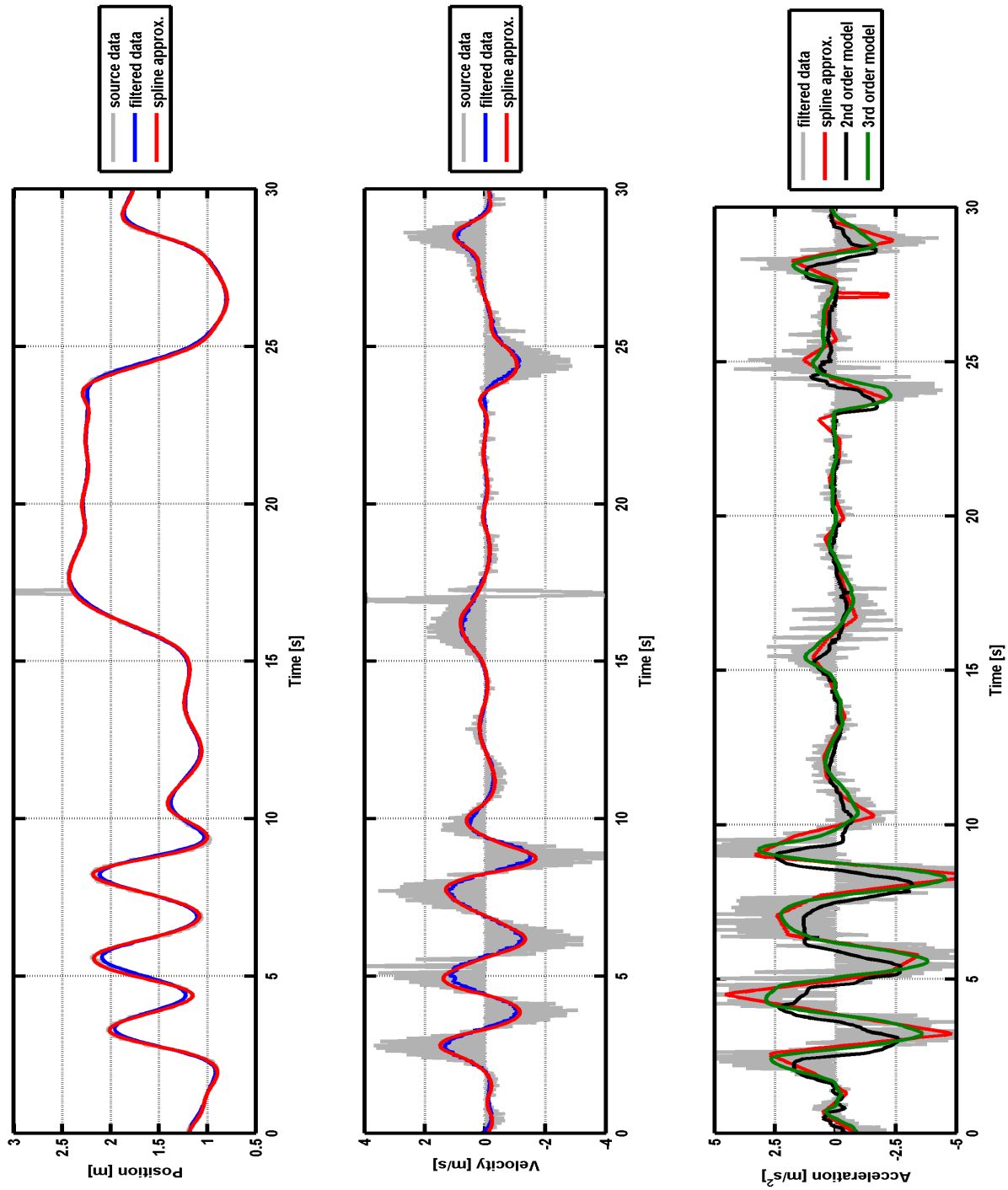
[16] Gumstix. *Overo COMs*. `https://store.gumstix.com/index.php/category/33/`.

Appendix A Contents of the attached DVD

Folder or File	Description
captured_data/	data from selected experiments
control_board_sources/	sourcecodes for MAV Control Board
original/	version prior to this thesis
implemented/	version implemented in the thesis
matlab_sourcecodes/	scripts used for data analysis and simulations
thesis_sourcecodes/	LaTeX sourcecodes of this thesis
videos/	videos of the experiments described in this thesis
Endrych_DT_2014.pdf	electronic version of this thesis

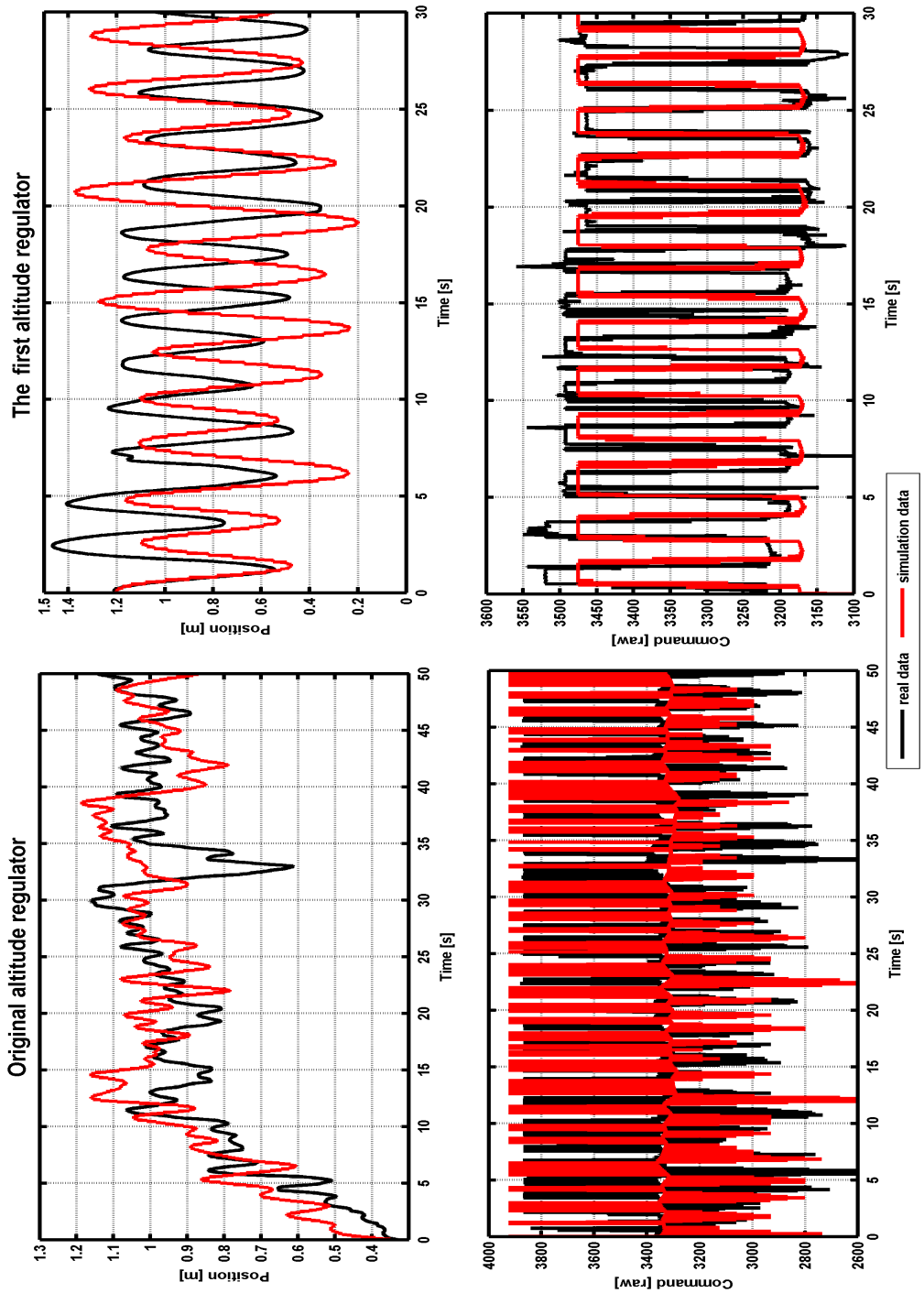
Appendix B System identification - altitude

This appendix is described in chapter 2.2.



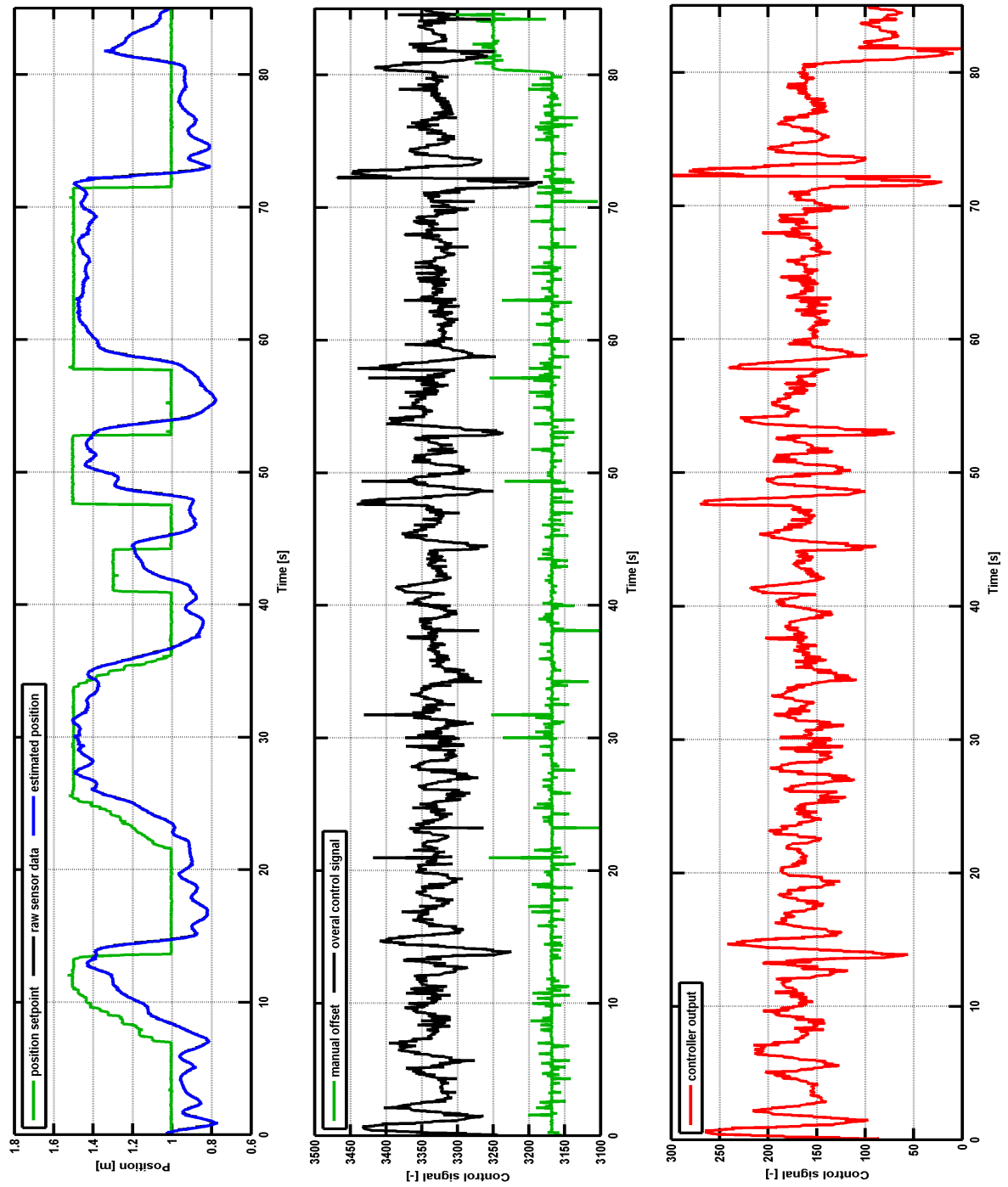
Appendix C Altitude model responses

This appendix is described in chapter 2.4.2.



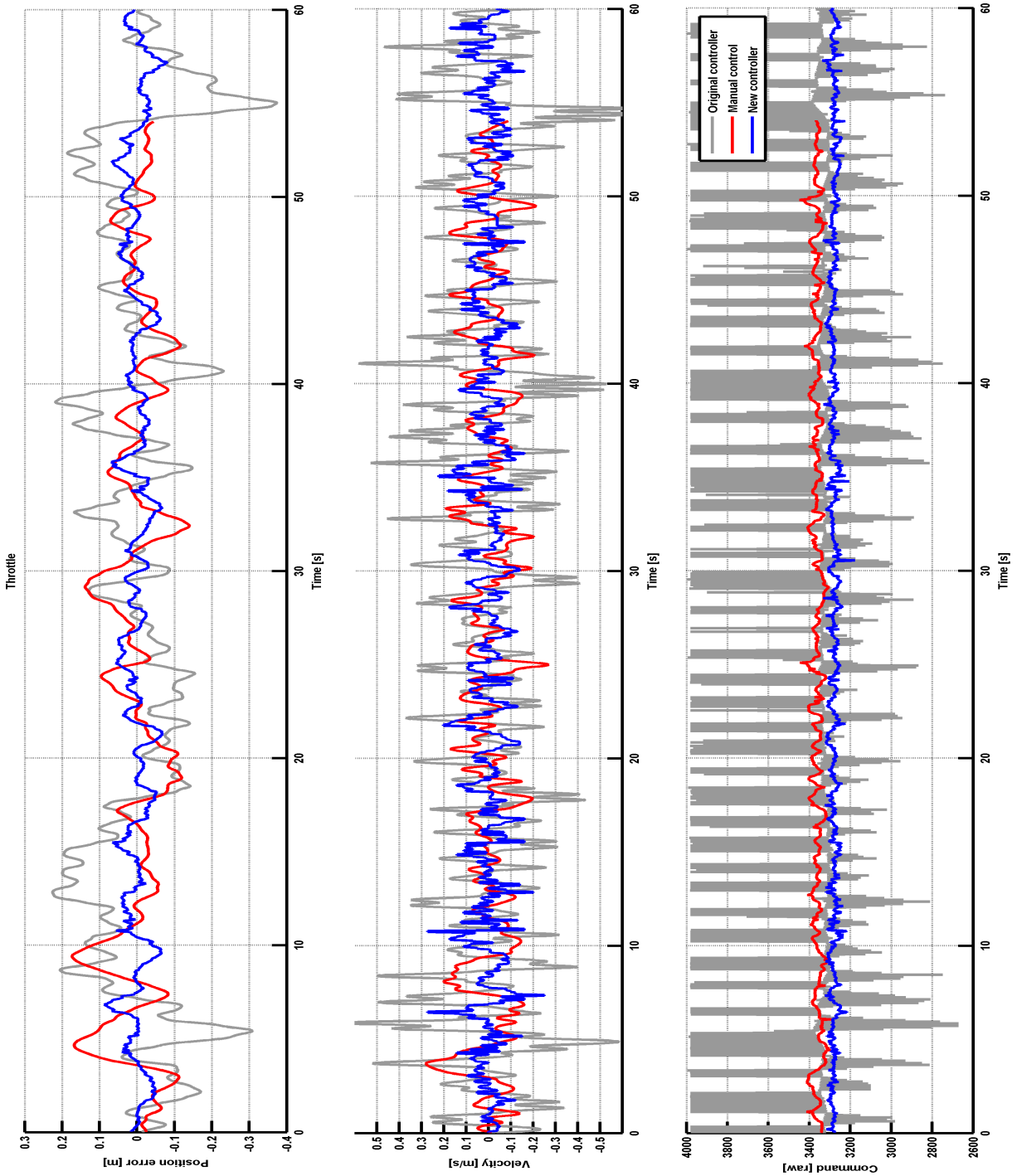
Appendix D New altitude controller performance

This appendix is described in chapter 2.4.



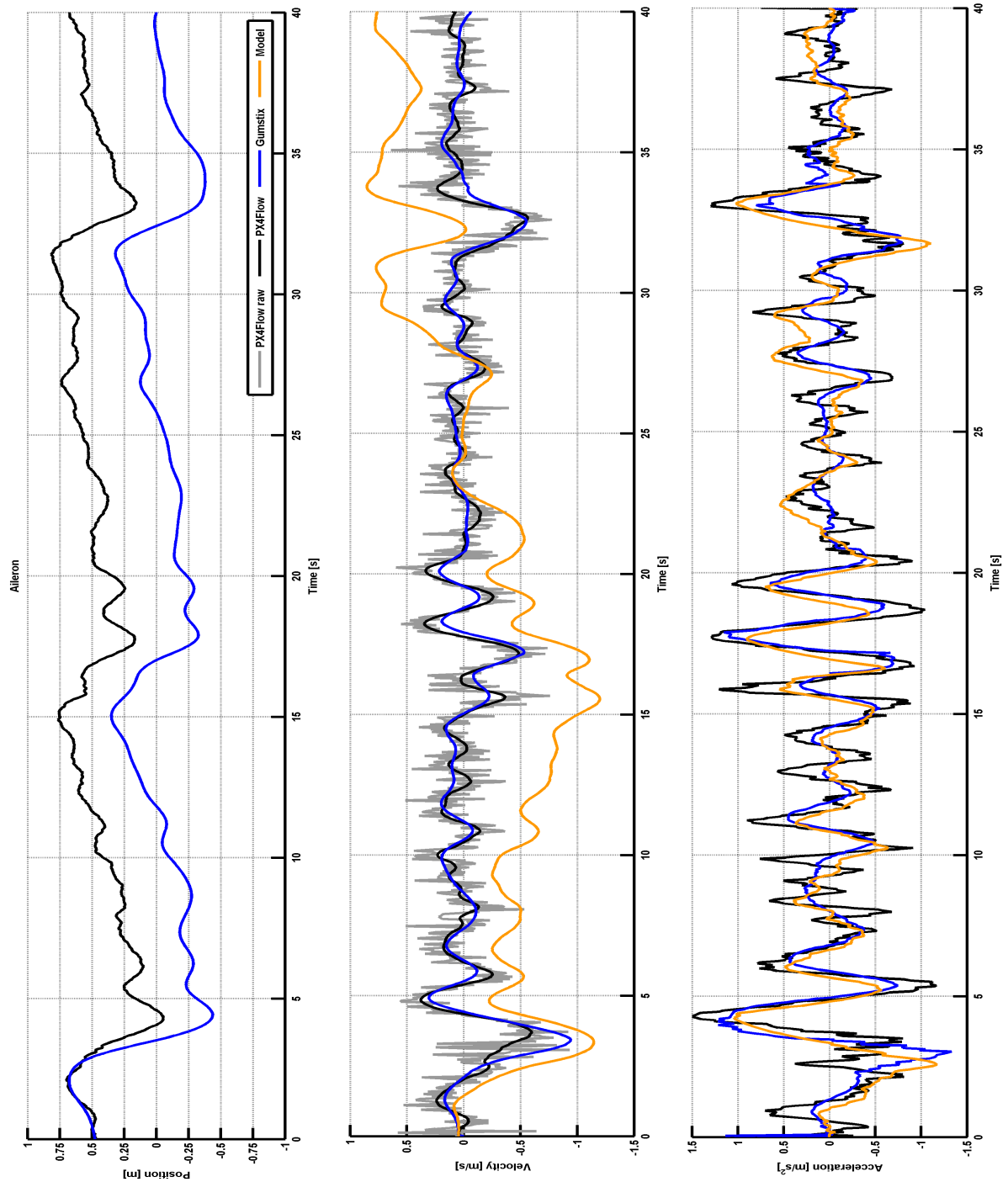
Appendix E Comparison of altitude controllers

This appendix is described in chapter 2.5.



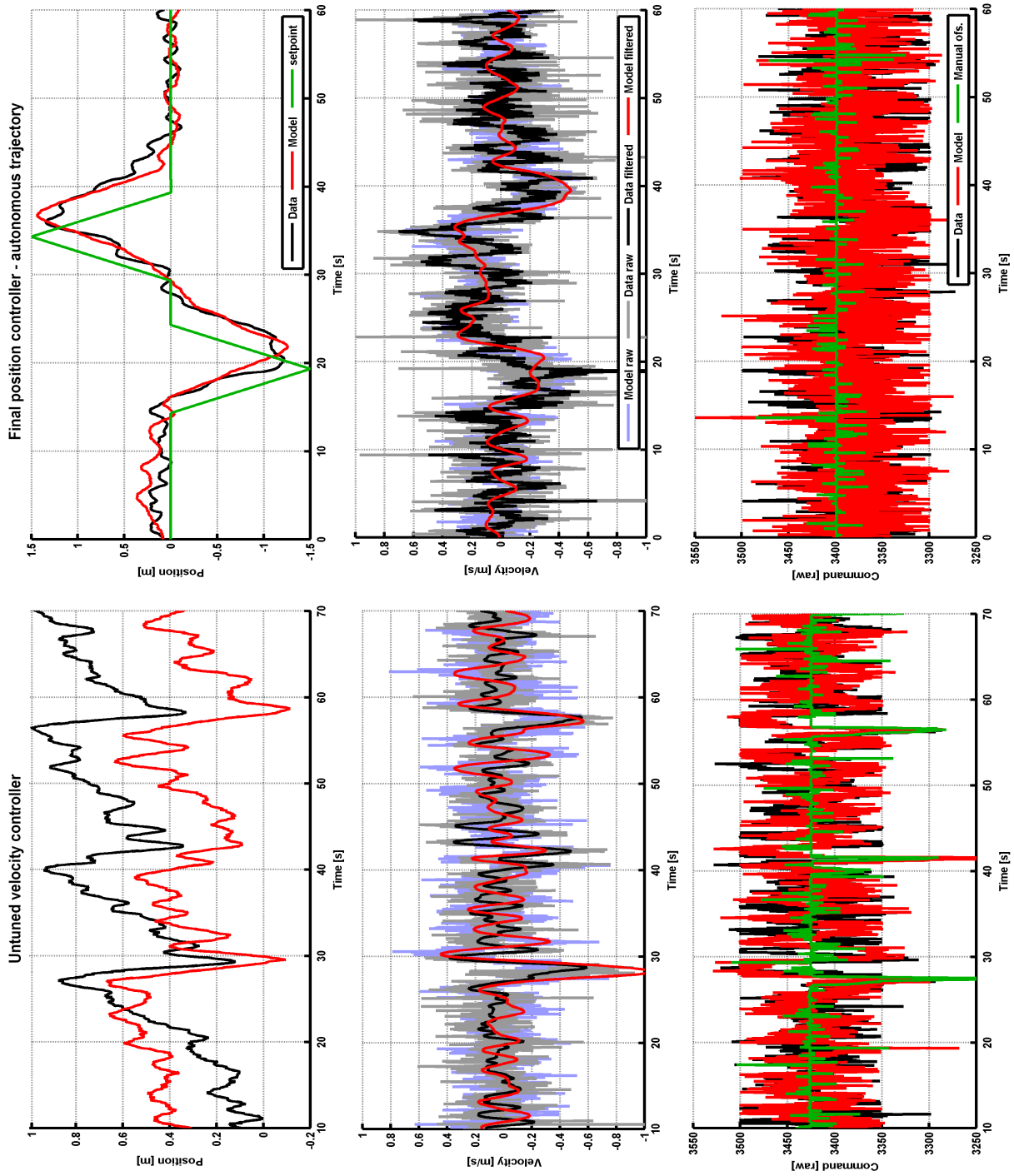
Appendix F System identification - vertical position

This appendix is described in chapter 3.2.3.



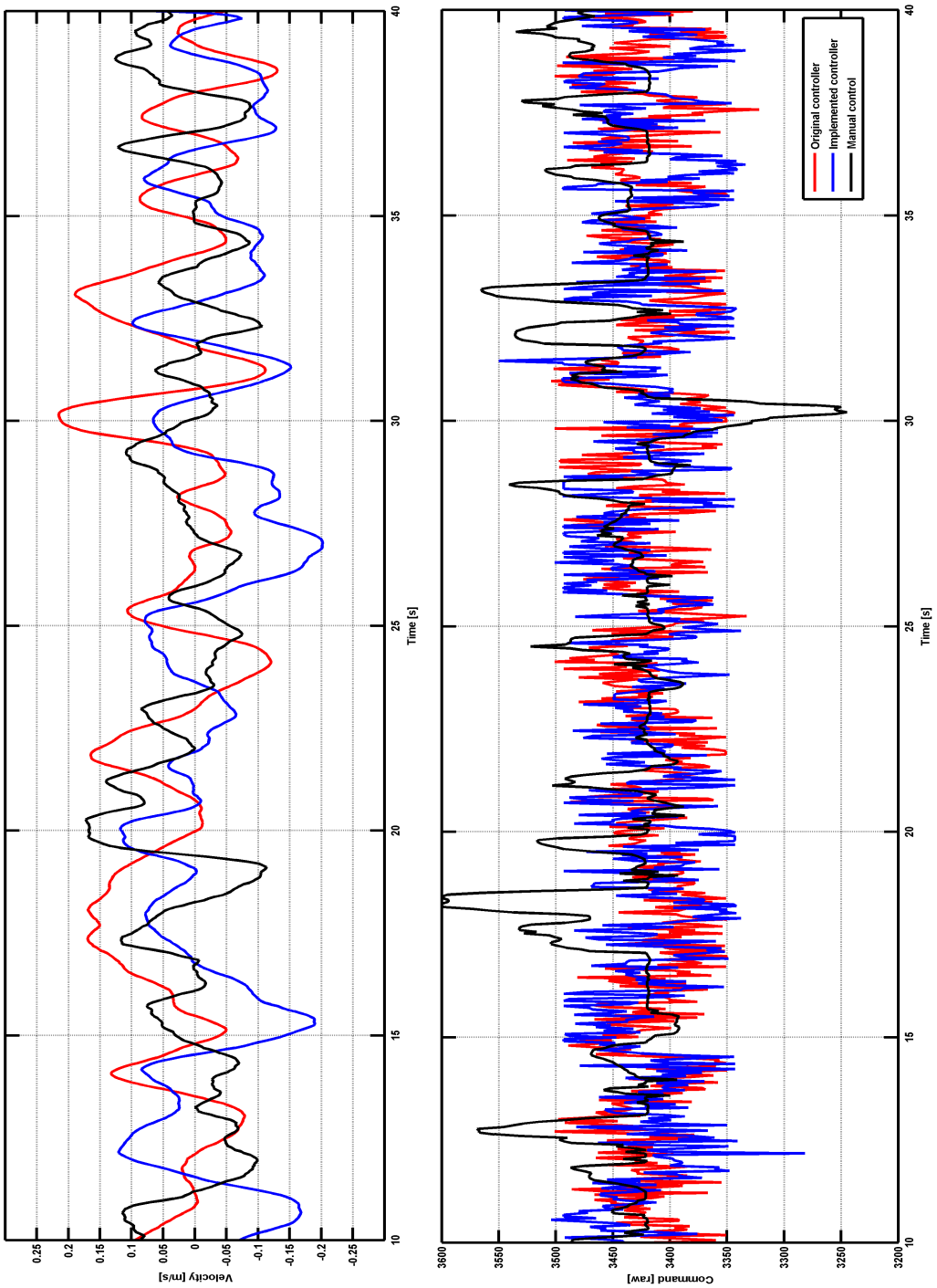
Appendix G Position model responses

This appendix is described in chapter 3.2.3.



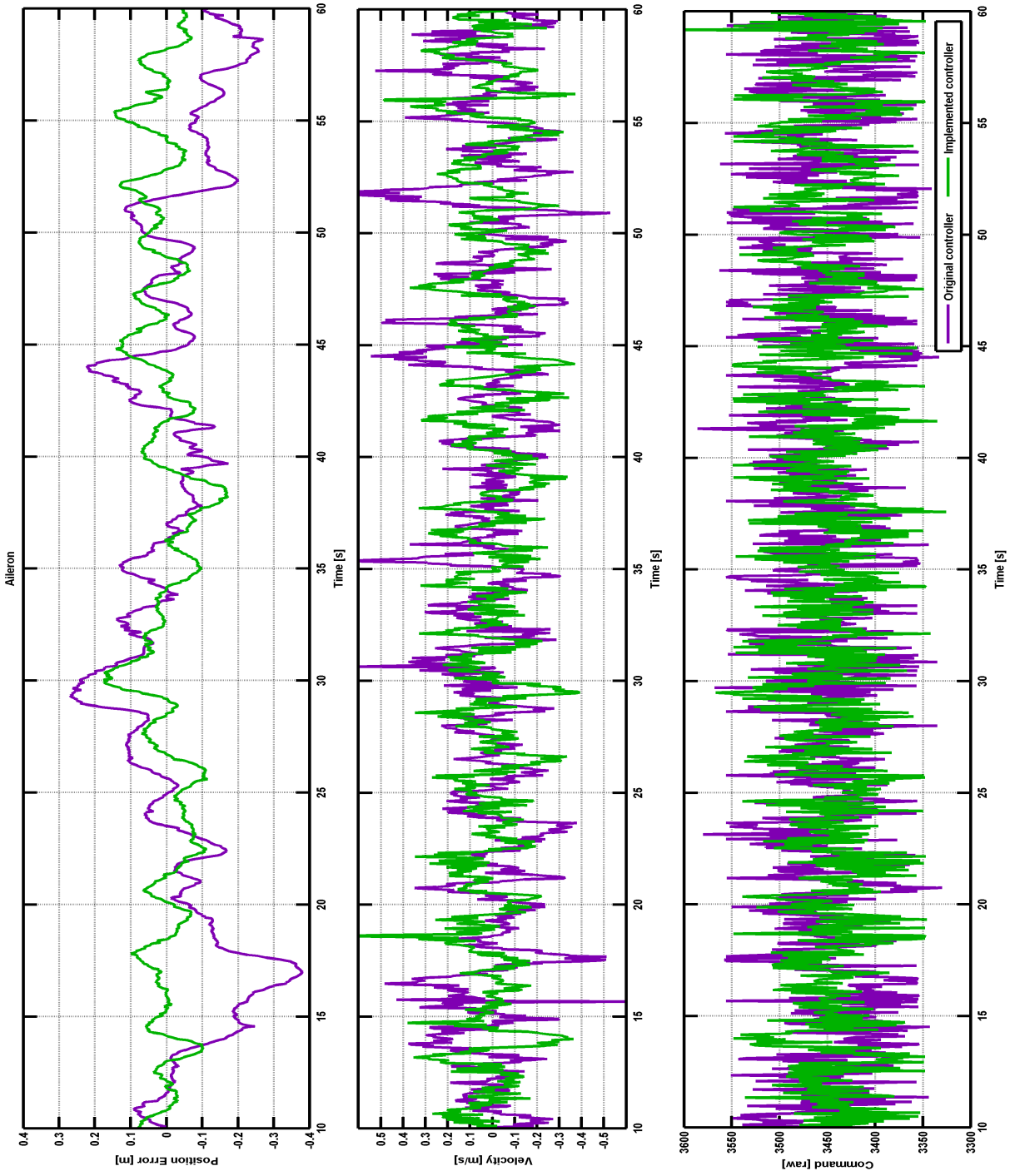
Appendix H Velocity controllers comparison

This appendix is described in chapter 3.3.1.



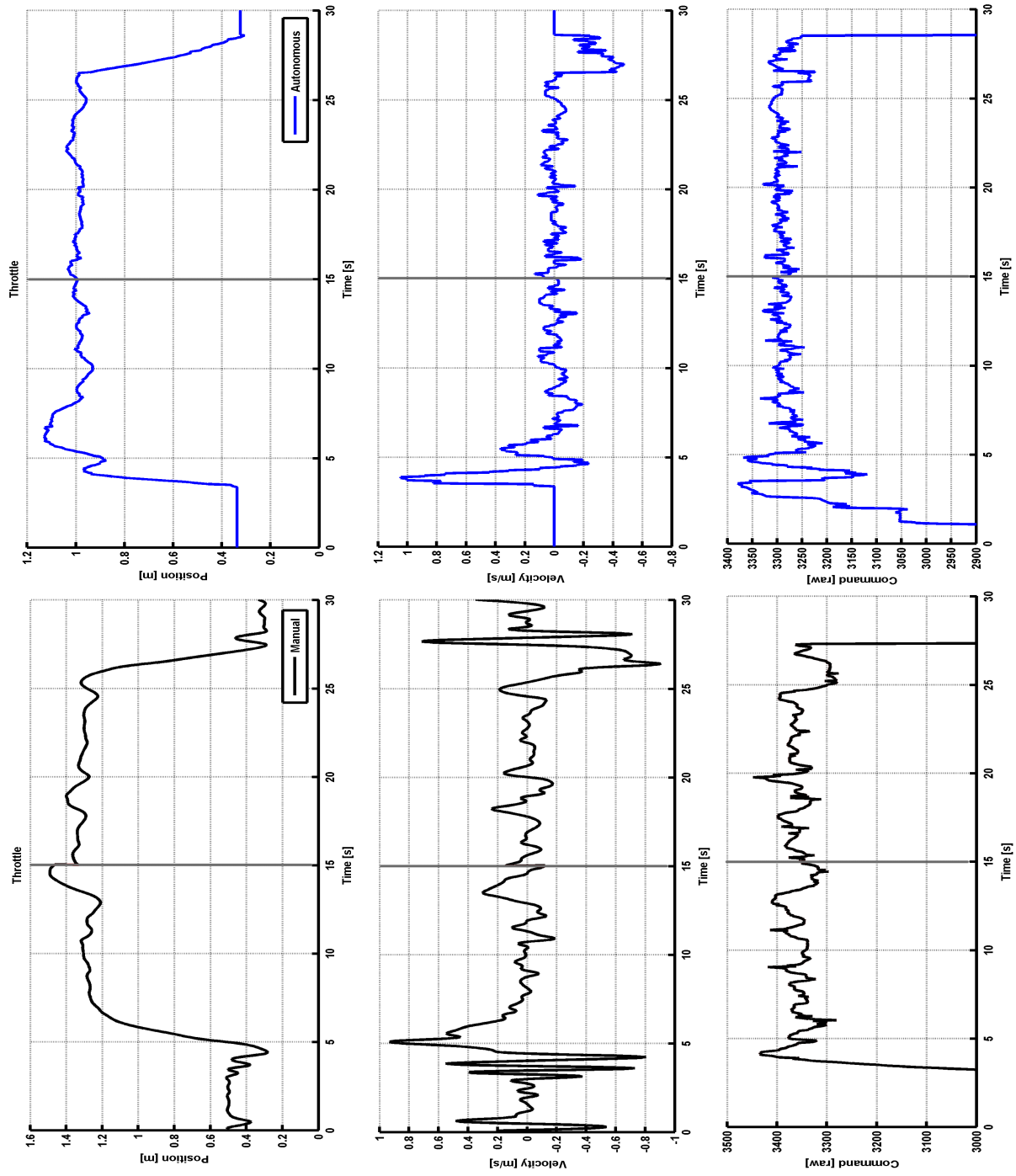
Appendix I Position controllers comparison

This appendix is described in chapter 3.4.



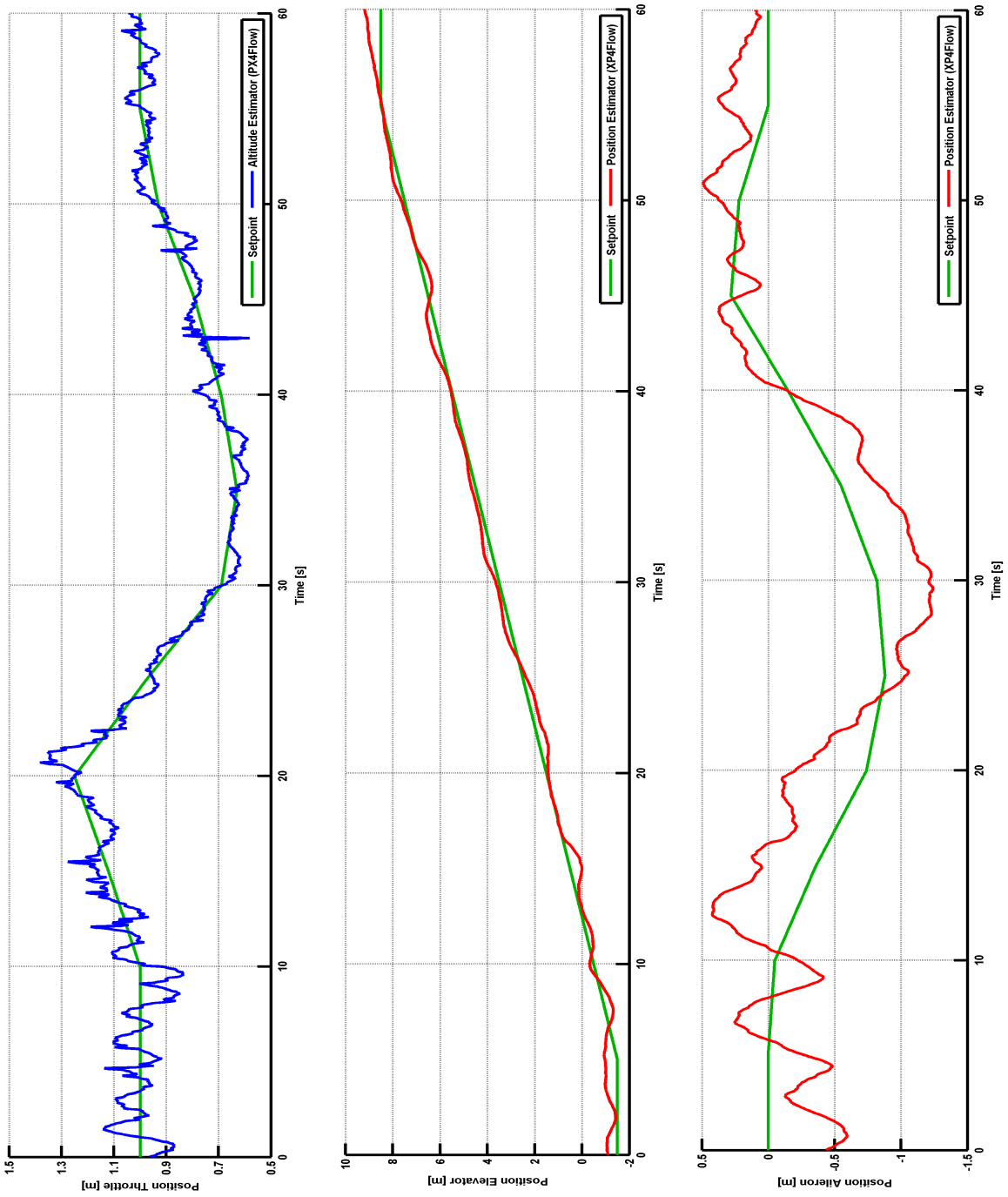
Appendix J Autonomous takeoff and landing

This appendix is described in chapter 4.1.



Appendix K Autonomous trajectory following

This appendix is described in chapter 4.2.



Appendix L Following the leading drone

This appendix is described in chapter 4.3.