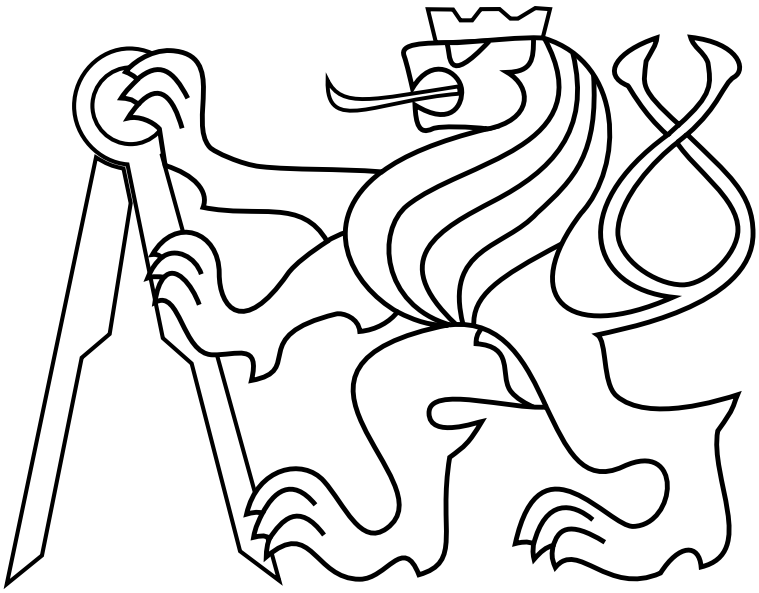
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Tomáš Báča

Model predictive control of micro aerial vehicle using onboard microcontroller

Department of Computer Science

Thesis supervisor: **Dr. Martin Saska**

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 my family for their encouragement and support during my whole studies. Furthermore, I thank my supervisor Martin Saska for his great support throughout this project. Finally, my thanks goes to my friend and colleague Antonín Novák for his excellent expertise and corrections regarding mathematics. This thesis is an outcome of a long lasting and diligent work. The author is grateful for the knowledge and the experience he gained, both theoretical and technical.

Abstract

This thesis deals with autonomous stabilization of unmanned multirotor aircraft along predefined trajectories, using a model predictive control approach. The main focus of this work lies in design and implementation of an embedded platform to allow onboard execution of the controller. We propose a linear dynamical model of the helicopter, identification of its parameter and design of a state observer. A quadratic model predictive controller is derived, implemented, and prepared for the experimental platform, which has been created for this purpose. Presented experiments, conducted both indoors and outdoors, verified the capability of the system to accurately follow a desired trajectory in space.

Keywords: model predictive control, unmanned aerial vehicles, embedded systems

Abstrakt

Tato práce se zabývá návrhem řídicího systému pro vícerotorovou bezpilotní helikoptéru s použitím metody prediktivního řízení. Za cíl si klade vývoj vestavěného systému, který umožní výpočet regulátoru přímo na palubě letounu. Předkládáme model dynamického systému helikoptéry, identifikaci jeho parametrů a následný návrh pozorovatele stavů. Kvadratický prediktivní regulátor je odvozen a implementován do navrženého hardware. Prezentované experimenty, provedené uvnitř i vně budovy, prokázaly schopnost přesného sledování trajektorií v prostoru.

Klíčová slova: prediktivní řízení, bezpilotní letouny, vestavěné systémy

Contents

List of Figures

1 Introduction

Unmanned aerial vehicles capable of vertical takeoff and landing have undergone a big boom in the past few years. It is mainly thanks to the arrival of multirotor helicopters (multicopters). Their capabilities span from being a platform for professional film makers, to toys and hobby products, and military vehicles built for reconnaissance. These vehicles (fig. 1.1) have characteristically several propellers (typically four and more) with a fixed pitch angle. The mechanical design of the machine is usually simple by comparison with a classical helicopter. From the mechanical point of view it has several brushless motors with propellers mounted directly on them. There is a rigid body with a utility platform, motor mounts and landing gear. They can operate in situations where the presence of a person could be hazardous (natural disasters) or they can help with work tasks which would otherwise require a very expensive solution e.g. inspection of high-voltage lines. They also find an application in situations where prior technology would not help — archaeological imaging from low-to-mid altitude.

In most cases, such an aircraft requires a human operator to fly although it can offer a certain level of automatic assistance. Nowadays the global position system (GPS) receiver is embedded in almost every mobile phone thus it is no surprise that it can help with the control of an unmanned aircraft (UAV) when flying outdoors. Commercially available platforms can assist with position hold or even fly to a particular location. Such technology can provide an automatic flight with precision up to 1 m depending on the GPS, external conditions and the vehicle itself. The scientific community has shown that there are methods to increase the precision of UAV control up to centimeters by using a better localization system than GPS (namely Vicon[1]) and implementing advanced algorithms for automatic control [10, 24].

Figure 1.1: An example of multirotor aircraft used for aerial imaging.

[1] Vicon is a motion capture system using multiple cameras capturing the object. http://www.vicon.com/

Although the previously mentioned works required an expensive laboratory hardware and on-the-ground computational power, they can be used as an example of what one could imagine an "automatic assistance" would be capable of when flying indoors. Since the multirotor UAV is a highly unstable machine, feedback control is necessary to even only stop it in the air. UAVs have been a subject of research for a long time. Current intention is to build an automatic UAV that is supposed to work not only in a laboratory but also in uncontrolled indoor environment [1]. A modification of current outdoor solutions for indoor usage brings new challenges and requires different approaches. The main ones are requirement of high precision control when flying indoors while omitting external localization system (like Vicon and GPS) and managing all computations onboard the vehicle itself.

The control mechanism used in today's control systems is usually satisfactory when the precision is roughly defined by the GPS localization and the control settling time is slow. It is often narrowed to PID (proportional-integral-derivative controller) that is relatively easy to compute on common embedded hardware [30, 2]. But when increasing requirements on precision and settling time, such approach seems to suffer if controlling this kind of unstable vehicle (see prior work in chapter 1.2). Also when imposing a requirement not only to stabilize the vehicle (stop it from moving) but to fly through a desired space trajectory, the need for a better control design approach emerges.

One of them is a model predictive control (MPC) which is a method based on the knowledge of a dynamical model of the controlled system. It formulates the computation of control actions as a mathematical optimization problem. Mathematical optimization can solve such problems by finding a minimum of a mathematical function. MPC uses a general prediction of system future states to formulate the objective function to be optimized. There are various types of optimization problems based on the shape of the objective function and the range of its parameters. MPC is usually formulated as a continuous optimization with a convex quadratic objective function where the function itself penalizes squares of deviations from a desired state trajectory. When the optimum is found it guarantees that control actions will lead to an optimal future trajectory with respect to the objective function.

Several important features set the MPC apart from other control techniques such as PID. Firstly, it works with the state space representation of the system and all states (e.g. position, velocity, acceleration, ...) participate as an input for optimization, not only a control error of the tracked state (usually position) as it is with the PID. Although, there are methods for nesting multiple controllers, the MPC offers a direct way to work with all states of the system. The other important feature is the way in which MPC handles constraints. We can impose constraints on inputs, systems states and even more complex ones. Though the optimization task becomes more complicated — linearly constrained convex quadratic optimization, it still has a global optimum which can be found in reasonable time. Another feature is a capability to account in disturbances. When properly modeled and estimated, a disturbance can be used directly by the controller to produce adequate control actions.

As it was previously mentioned, MPC has been already successfully used for control of

UAVs. But the challenging task is to implement it solely on the hardware onboard of the UAV where computational power is very limited. Today microcontrollers[2] can have parameters similar to $\approx$ 200 MHz clock, $\approx$ 200 kB RAM and mainly floating point performance in order of 10^7 operations per second. Surely one could ask, why not to equip the UAV with a dedicated (miniaturized) PC to handle the computation. Our goal is to create a small all-in-one solution which can be mounted on any micro aerial vehicle (MAV) where the potential payload capacity could be only hundreds of grams. In the long term this would allow us to create a swarm of self-sustained MAVs that could solve tasks as remote sensing, localization of dangerous objects or mapping even in indoor and unknown environment.

The engineering part of the thesis had been preceded by a survey of state-of-the-art work (section 1.3) and study of optimization and control algorithms. Then the experimental platform was created. A tricopter (multirotor aircraft with 3 propellers) was built using of-the-shelf parts (see fig. 6.1) and adapted to mount a custom control board. The control board is an electronic platform designed specifically for execution of the MPC controller. The printed circuit board has been manufactured and equipped with variety of microcontrollers to support data logging, execution of MPC and wireless telemetry.

This thesis is structured as follows. Firstly, chapter 2 describes a derivation of a state space model of the helicopter, which is further utilized for state estimation and control. Chapter 3 describes how the dynamical model of the UAV was identified using data from experiments. A concept of a state observer is introduced in chapter 4 with explanation of linear Kalman filter. It is followed by chapter 5 where the MPC controller is derived. Chapter 6 introduces the UAV platform that was developed specifically for experimental verification. Implementation aspects of the control system are discussed in chapter 7 accompanied by important notes on disturbance rejection and offset-free tracking. Experimental results follow in chapter 8, where trajectory tracking and disturbance rejections features were tested on several predefined trajectories, both indoors and outdoors.

1.1 Problem statement

The task of this thesis is to design, build and implement an embedded controller that will utilize the model predictive control approach to drive the aircraft in a way suitable for indoor use, supporting precise trajectory following, and in future will enable its usage in formations and swarms of helicopter. It entails a design and creation of dedicated circuit board which will allow to work with any usual UAV and extends its capabilities to fulfill complex robotic tasks. It should support connecting a variety of external peripherals and sensors to localize the helicopter in space. Firstly, system modeling and identification need to be done to supply a dynamical model for simulation purposes and the MPC itself. It is followed up by a design of a state observer to track the current position of the UAV based on its sensors and to estimate all other states for purpose of the MPC. The design of the model predictive controller requires

[2] A microcontroller is an integrated circuit containing a processor, operating memory, program memory and peripheral controller.

a formulation of optimization task

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^N} \quad & \mathrm{J}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{x}^T\mathbf{c} \\ \text{s.t.} \quad & \text{constrains on } \mathbf{x}, \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the solution that leads to the desired control actions over a certain *prediction horizon*. A proper method for solving the optimization process needs to be chosen and implemented on the embedded hardware of the UAV. Previously mentioned tasks should be accompanied by simulation of the system including modeling of measurements, data filtration and the flight itself. There are minor tasks tied to the implementation and hardware design, mainly resolving communication, signal processing and data handling. The final outcome should be verified using variety of experiments testing its capabilities and finally compared to the prior solution used in the laboratory up to now.

1.2 Previous work

There has been a research on UAV control in laboratory of Intelligent and Mobile Robotics (IMR) group at FEE CTU Prague since 2011. The main focus has been on simulation of relatively localized UAV swarms and formations. Because it was also desired to verify the results experimentally, a development of hardware platform started. The first experiments has been done with AR Drone platform [19] but it was shown that there is a need for a customizable platform capable of lifting heavier payload.

The first iteration of creating the custom platform is depicted in [4]. A custom control board was developed accommodating PID controllers. It could be mounted on a general purpose UAV and allowed connecting a dedicated onboard camera localization system [14]. Experiments showed that the UAV was able to track and follow the position of a ground robot marked for image recognition. It was also tested that localization based solely on the position of a moving object (another robot or UAV) tends to destabilize the vehicle. Experiments were conducted with a heterogeneous formation of two helicopters and one ground robot.

Subsequent work has been done in [42] utilizing the same hardware accompanied by the *px4flow* optical flow sensor. PID controllers were fine-tuned and the system was further improved to allow automatic flight even when the tracked object cannot be seen. This allowed creating formations of multiple UAVs which was not successful with the former system. Experiments were again conducted to test the performance of the system.

Furthermore, the need for a precise execution of indoor experiments led to increasing focus in development of the control system itself. This thesis describes a development of new hardware platform which satisfies demands that arose during the previous work. The system should be capable of executing a MPC controller, transmitting telemetry to ground station as well as logging data onboard in realtime.

1.3 Related work

It is difficult to find results on completely embedded MPC control of UAV. Those who are not only scientifically active in simulating flight control but verify their outcomes experimentally tend to use an external localization system and on-the-ground computational power, since it makes experiments much less demanding on particular hardware implementation. Following survey contains the state-of-the-art of both, embedded and ground computed MPC.

The paper [5] shows an onboard quadratic implementation of MPC running on Pixhawk platform [30]. The control problem with the prediction horizon of length 5 steps is solved by unconstrained optimization. The UAV is supplied with position data gathered from the Vicon system and transmitted to the vehicle wirelessly. Data from presented experiments suggest that the aircraft is capable of tracking position step changes and harmonic trajectory. A correct MPC formulation is presented. The paper lacks description of experiments including multimedia material, which is suspicious since it presents a working and flying autonomous helicopter. The system is not fully onboard and the results can be overcome.

The paper [1] is considered as the current state-of-the-art in the field of embedded MPC implementation on UAV. Authors have proposed a solution based on RMPC (Robust MPC) utilizing an explicit formulation of the optimization task. The RMPC takes a form of a linear objective subject to linear constraints (Linear Programming). The formulation allows a disturbance rejection which is demonstrated by experiments where the UAV follows a trajectory while being under the influence of wind gusts. The experimental platform consists of two vehicles equipped with Atom based PC, one of them being localized by Vicon, the other one equipped with a ultrasonic rangefinder and camera for computing an optical flow. The presented solution also allows to incorporate a collision avoidance directly into the control loop, which is experimentally verified in the paper. Despite the results are impressive, the approach suffers from combination of a trajectory planner and a low-level controller in one system. The overall performance of the controller (its capability to follow the trajectory) is lower than if using our approach, because the optimization takes care of creating the optimal trajectory using state constraints. Theoretical and experimental analyses show that a better approach is to separate the controller from a trajectory planner. The trajectory planner may use an MPC approach too (even non-linear [38]), but its execution rate can be slower than of the controller. See section 7.2 for detailed discussion to this topic.

Another work [29] (prior to [1]) shows an implementation of a quadratic MPC, embedded on tilt-rotor aircraft, utilizing onboard Atom based computer. The position of UAV is estimated from onboard sensor data. The constrained formulation presented in the paper can handle state constraints and allow optimization over prediction horizon of 8 steps. Experiments were conducted in order to show its ability to track desired 3D trajectory and reject position disturbances.

Authors of technical report [7] and later paper [8] managed to implement the explicit

formulation of quadratic MPC using onboard Atom PC. They localize the helicopter using the Vicon system and use a learning based MPC to improve the performance of the system. Their experiments involved a helicopter catching a flying ball.

In [40], a non-linear MPC formulation allowing to control several miniature, single-rotor helicopters in a collision-free way, is presented. An external camera localization system as well as a ground computer station are utilized. The MPC optimizes over 2 s prediction horizon. Their simulation result has been experimentally verified with one miniature helicopter.

University ETH in Zurich has a long tradition of research in the field of UAV control. Some of their videos of helicopter aerobatics have been commonly known even among general public. They have accomplished astounding achievements with UAVs with the Vicon system involved [10, 3]. Their contribution to MPC is published in [25], it also relies on the external localization system as well as the PC ground station.

Almost all of presented works was implemented using a single MPC approach to control the vehicle together with creating the optimal trajectory. We aim to decompose those problems. This thesis presents MPC controller, while our other related work describes the trajectory planning [39, 38]. A purely embedded implementation of MPC can be seen only in [1, 29] but our proposed solution surpasses its capabilities of trajectory tracking by having a significantly longer prediction horizon. Other works [5, 7, 8, 25, 40] utilize an external localization system which limits the usage and testing of such system to laboratory conditions.

1.4 Contribution

We present a control system for UAV that allows the execution of quadratic MPC onboard the aircraft. It consists of a single board equipped with two microcontrollers, telemetry module and SD-card data logger. The system was tested with the *px4flow* optical flow sensor, but allows connecting other sensors and modules. A custom implementation of QMPC enables to control the UAV with 2.2 s prediction horizon. The implementation allows to estimate system disturbances with Kalman filter and reject them directly with the controller. When flying in indoor environment, the system is able to track desired trajectories with errors in order of centimeters. Experiments have shown that the results are similar to current state of the art solutions. It can be implemented in very small UAVs (5 inch propellers) to allow flying in compact formations in indoor environment. Our contribution beyond the state-of-the-art approach is summarized in following points:

- We formulated a system observer and MPC controller for embedded implementation.
- We present a disturbance rejection model delivering an offset-free tracking.
- We designed a hardware platform specifically intended for UAV MPC control relying on onboard sensors and computational resources.

1.5 Mathematical notation

The table 1.1 denotes the basic mathematical notation used in this thesis.

Symbol	Description
lower or uppercase letter, e.g. n, N	a scalar
bold lowercase letter, e.g. $\mathbf{x}$	a column vector
bold uppercase letter, e.g. $\mathbf{A}$	a matrix
$\mathbf{x}^T$, $\mathbf{A}^T$	vector and matrix transpose
underlined vector, e.g. $\underline{\mathbf{x}}$	concatenated vectors $\left(\mathbf{x}_1^T, \mathbf{x}_2^T, ..., \mathbf{x}_n^T\right)^T$
$\mathbf{I}$	identity matrix
$\mathbf{1}$	matrix of ones
$x^{(W)}$	x in coordinate system W
$x_{[t]}$, $\mathbf{x}_{[t]}$	x, $\mathbf{x}$ in the sample time t
$\mathbb{R}$, $\mathbb{N}$	set of real and natural numbers

Table 1.1: Overview of mathematical notation.

2 UAV dynamics

Modeling a system dynamics is an important part of the control design process. Good understanding of system's behavior enables to design a proper controller using approaches studied in the field of control theory. Such controller can reflect known characteristics of the system and can appropriately response to evolution of system states. There are two fundamental approaches to the system modeling. One is based on the knowledge about the physics involved in the system. This knowledge can be used to derive a mathematical model using e.g. Hamiltonian mechanics. Control design of a well known system is usually called Whitebox, or Greybox, depending on how much of the physical process we are able to describe. On the other hand, if the system is unknown, one can experimentally identify a mathematical model which sufficiently represents observed behavior of the system. Such method is usually called Blackbox.

If modeling a classical helicopter with main and tail rotors, a complex model including phenomena as aerodynamics and rotor-blade flapping could be constructed. In many approaches [1, 23], dynamics of a multirotor UAV is simplified to a single rigid-body description, mostly because of smaller fixed-pitch propellers. Further, due to existence of well designed and tested platforms such as Pixhawk [30] or Ardupilot [2], the vehicle can be modeled with the inner feedback loop already closed. By doing that, considering a fixed-pitch quadrotor, we move from a system actuated by thrust of its four propellers to a system, where the inputs are the desired pitch (θ_D), roll (ψ_D), yaw rate ($\dot{\phi}_D$) and collective thrust (U_D). It is assumed that such system can be treated as a decoupled one [23]. Some assumptions and simplifications are made in the following chapter thus the field of modeling in this thesis is the Greybox.

2.1 Attitude dynamics

Several coordinate systems are presented, in which states of the UAV (fig. 2.1) are expressed. The first one is a world coordinate system W with a fixed position in the world. Then there is a rotating world coordinate system R. It is rotated from W around axis W_z by angle ϕ. An inertial frame I follows, in which the attitude angles θ and ψ are measured. It is translated into the geometric center of the UAV. The last one is a body frame B which axis are aligned with the mechanical frame of the UAV.

Assuming a complete decoupling of the system, the attitude dynamics can be modeled using Newton's 2nd law of motion and expressed by following equations

$$\begin{aligned} \ddot{x}^{(W)} &= \frac{U}{m}\left(\sin\psi\cos\phi + \sin\theta\sin\phi\right), \\ \ddot{y}^{(W)} &= \frac{U}{m}\left(\sin\theta\cos\phi + \sin\psi\sin\phi\right), \end{aligned} \tag{2}$$

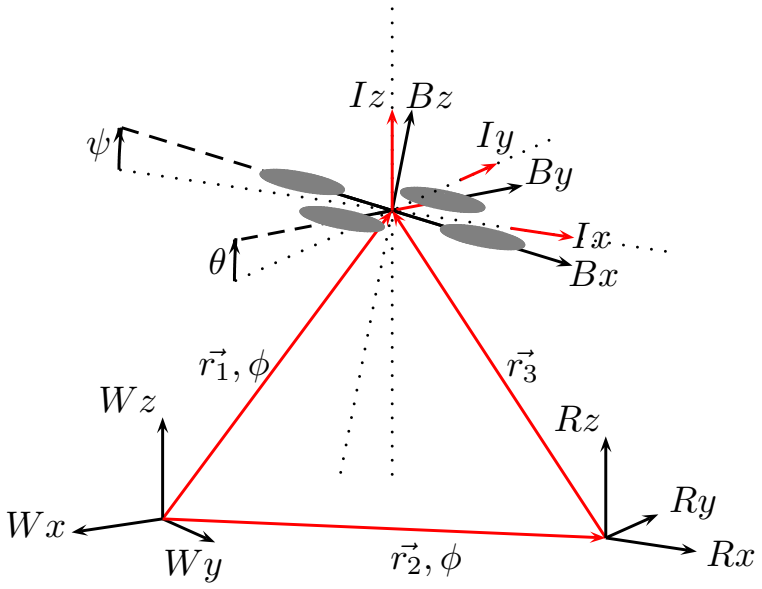


Figure 2.1: UAV and its coordinate systems.

where U is the total thrust force action on a center of gravity[3] and m is the mass of the UAV. It is assumed that desired operating point is a hovering state, where U, m are constants and absolute values of θ, ψ are small. Since our system is not localized using any global localization system and it relies completely on a dead-reckoning in terms of stabilizing the yaw motion[4], all positions and their derivatives in following equations are expressed in system R. Thus the position is no longer a function of ϕ. We can then simplify the equations into the following form

$$\begin{aligned} \ddot{x}^{(R)} &= K_1 \sin\psi, \\ \ddot{y}^{(R)} &= K_1 \sin\theta. \end{aligned} \tag{3}$$

Furthermore, assuming a small input actions during hovering around the operating point, these forms can be linearized. It is done by approximating it by first two terms of the Taylor series. The acceleration of the UAV is directly proportional to its attitude angle providing assumptions previously mentioned in this paragraph. The linearized forms follows as:

$$\begin{aligned} \ddot{x} &= K_1\psi, \\ \ddot{y} &= K_1\theta. \end{aligned} \tag{4}$$

[3] in a direction of the B_z axis

[4] The yaw angle ϕ is stabilized by a feedback loop implemented on a stabilization board using onboard IMU.

2.2 Altitude dynamics

The altitude dynamics can be also modeled using Newton's 2nd law of motion. The following equation describes the relationship between UAV acceleration in the R_z axis and control inputs U, θ, ψ:

$$\ddot{z} = \frac{U}{m}\cos\theta\cos\psi - g^{(W)}. \tag{5}$$

The gravitational acceleration is denoted by $g^{(W)}$. This system is also non-linear as in the case of attitude dynamics, but we need to be more cautious with a potential linearization in this case. If we try to build an altitude controller that is supposed to work not only around a hovering point, but also during take-off and landing, the force U cannot be treated as constant, unlike in eq. (3). The pull force of a propeller can be simplified to a quadratic function of its angular speed [21], which is one of the controlled inputs.

2.3 Dynamics of the integrated stabilization

Current UAVs are usually equipped with an attitude stabilization system. If properly tuned and the feedback loop is closed it transforms the system to be controlled by θ_D, ψ_D, U_D, $\dot{\phi}_D$, instead of desired thrust of each motor. Such system can be a cheap and affordable item on a list when building a custom multirotor aircraft. For purpose of this thesis, all four model systems are modeled using first order transfer function (6). Furthermore it is shown (chapter 3) that these models are satisfactory and can be fitted on a measured data reliably. It is assumed the stabilization does not integrate an altitude feedback loop[5]. In that case, the first order system would encapsulate also one of system integrators ($\ddot{z} \to \dot{z}$) considering an altitude speed controller, which is common in some systems [30, 2]. The first order transfer functions are defined as:

$$\begin{aligned}
\frac{\mathcal{L}\{U\}}{\mathcal{L}\{U_D\}} &= \frac{1}{\tau_1 s + 1}, \\
\frac{\mathcal{L}\{\theta\}}{\mathcal{L}\{\theta_D\}} &= \frac{1}{\tau_2 s + 1}, \\
\frac{\mathcal{L}\{\psi\}}{\mathcal{L}\{\psi_D\}} &= \frac{1}{\tau_3 s + 1}, \\
\frac{\mathcal{L}\left\{\dot{\phi}\right\}}{\mathcal{L}\left\{\dot{\phi}_D\right\}} &= \frac{1}{\tau_4 s + 1}.
\end{aligned} \tag{6}$$

[5] Which would require e.g. barometer, GPS, or other sensors.

2.4 State space representation

For the purpose of this thesis the discrete formulation of the dynamical system will be used. It allows to design a proper filtration method and the MPC controller itself. Since now, all differential equations and state space formulation are written in a discrete form with a constant sampling rate $1/\Delta t$. The following form describes a discrete time-invariant system with a main matrix $\mathbf{A}$, and input matrix $\mathbf{B}$:

$$\mathbf{q}_{[t+1]} = \mathbf{A}\mathbf{q}_{[t]} + \mathbf{B}\mathbf{u}_{[t]}, \tag{7}$$

where $\mathbf{x}_t$ is the state vector in the sample time t. Following matrices describe the pitch and roll systems, where state vectors are defined as $\mathbf{q}_x = (x, \dot{x}, \ddot{x})^T$, $\mathbf{q}_y = (y, \dot{y}, \ddot{y})^T$ and inputs as $\mathbf{u}_x = \psi_D$, $\mathbf{u}_y = \theta_D$:

$$\mathbf{A}_{x,y} = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & \Delta t \\ 0 & 0 & P_1 \end{bmatrix}, \mathbf{B}_{x,y} = \begin{bmatrix} 0 \\ 0 \\ P_2 \end{bmatrix}, \tag{8}$$

where Δt is the sampling period, P_1 and P_2 are parameters of the first order transfer from a desired angle to the actual angle of attitude. This description is LTI system and can be directly used for state estimation (using e.g. Kalman filter) and for the MPC controller. For purpose of modeling altitude dynamics including Earth's gravitational pull, an additional input needs to be added to the system which will act as an constant source of acceleration. In the following case, the 2nd input value is always equal to 1. Eq. (9) shows the discrete altitude LTI system for state vector $\mathbf{q}_z = (z, \dot{z}, \ddot{z}, \ddot{z}_u)^T$ and input $\mathbf{u}_z = (U_D, 1)^T$ linearized around a hovering point

$$\mathbf{A}_z = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & \Delta t & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & P_3 \end{bmatrix}, \mathbf{B}_z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & -g \\ P_4 & 0 \end{bmatrix}, \tag{9}$$

where $g \approx 9.8ms^{-2}$ is a gravitational acceleration, P_3 and P_4 are parameters of the first order system $U_D \rightarrow \ddot{z}$. The last system represents the yaw dynamics of the UAV for state vector $\mathbf{q}_\phi = \left(\phi, \dot{\phi}\right)^T$ and input $\mathbf{u}_\phi = \dot{\phi}_D$. Symbols P_5 and P_6 denote parameters of the system $\dot{\phi}_D \rightarrow \dot{\phi}$. The system matrices follows as:

$$\mathbf{A}_\phi = \begin{bmatrix} 1 & \Delta t \\ 0 & P_5 \end{bmatrix}, \mathbf{B}_\phi = \begin{bmatrix} 0 \\ P_6 \end{bmatrix}. \tag{10}$$

3 System identification

A test flight needed to be made, in order to identify parameters $P_1 \dots P_6$, introduced in chapter 2. Several methods exist for system identification for both time and frequency domain. Despite theirs advantages, one have to excite system poles and zeros in a way that could lead to damage of the system and its surroundings. In case of highly unstable UAV, testing by a step response or frequency response is mostly unwanted. Instead of that the model can be fitted to arbitrary data using a mathematical optimization approach. The test flight, which was conducted by a human operator, consisted mainly of hovering above one place with small oscillations in all axis. All data used for following identification were gathered from the onboard *px4flow* optical flow sensor (including ultrasonic rangefinder). The data were received and logged in constant rate and coupled with appropriate control action of the operator. Since all searched parameters are part of the first order system eq. (6), we can write down its discrete differential equation

$$q_{[n]} = P_A q_{[n-1]} + P_B u_{[n-1]}, \tag{11}$$

where P_A, P_B are unknown constants of the transfer function. If supposing that, all sensors are perfect and there are no other effects on the system, this equation should hold for all measured samples. When considering all samples, the set of equations can be written in a matrix form as follows:

$$\mathbf{b} = \mathbf{A}\mathbf{p}, \tag{12}$$

where $\mathbf{A} \in \mathbb{R}^{(m-1)\times 2}$ represents matrix of the set of equations, $\mathbf{b} \in \mathbb{R}^{(m-1)}$ is the left-hand side of the equation, $\mathbf{p} \in \mathbb{R}^2$ is the vector of parameters we are looking for, and m being the number of samples. Parameters $\mathbf{b}$, $\mathbf{A}$, $\mathbf{p}$ are created as follows:

$$\mathbf{A} = \begin{bmatrix} q_{[1]} & u_{[1]} \\ q_{[2]} & u_{[2]} \\ \vdots & \vdots \\ q_{[m-1]} & u_{[m-1]} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} q_{[2]} \\ q_{[3]} \\ \vdots \\ q_{[m]} \end{bmatrix}, \mathbf{p} = \begin{bmatrix} P_A \\ P_B \end{bmatrix}. \tag{13}$$

Due to the fact that the system description does not cover all phenomena and the measured data is noisy, the equation (12) does not necessary hold. But we can still estimate $\mathbf{p}$ in a way that $\|\mathbf{Ap\text{-}b}\|_2^2$ is minimal. Thus the supplied set of equations needs to be overdetermined to provide sufficient accuracy of estimated parameters. The solution is then optimal in terms of least squares of $\mathbf{r}$, where $\mathbf{r} = \mathbf{Ap\text{-}b}$ is a vector of residuals of the equation system. This optimization problem can be solved e.g. by using operator '\' in Matlab.

3.1 Attitude subsystem identification

Four parameters, P_1, P_2 for `x` axis and P_3, P_4 for `y` axis, need to by identified. Assuming the system is decoupled and the low-level stabilization drives both subsystems identically[6], we can identify a model using only one of them. Following experiment was conducted using a manually controlled UAV equipped with the *px4flow* optical flow sensor. Data from the experiment (fig. 3.1) are both unfiltered and captured onboard using a dedicated logging device. Unit of measurement of the input signal represents a difference from a mean PPM[7] signal pulse length, measured using a hardware timer. They are directly proportional to the desired attitude angle of the UAV.

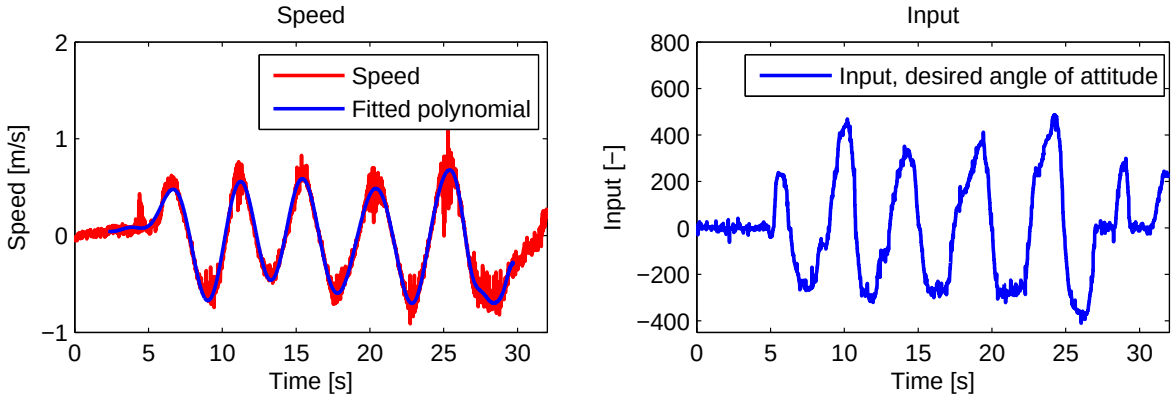


Figure 3.1: Identification data - attitude.

The speed signal has to be differentiated to identify the first order transfer function from the input to the acceleration. Discrete differentiation of the signal was found impractical, due to its large noise component. But when fitting a smooth function to the data, the derivative can be computed analytically. The polynomial function was chosen to fit the data since it can be easily differentiated and it can be found easily. Using the approach described above, constants P_1 and P_2 were obtained as:

$$P_1 = 0.9799, P_2 = 5.0719 \times 10^{-5}. \tag{14}$$

Having these constants, the attitude system can be completed from eq. (8) with particular values:

$$\mathbf{A}_{x,y} = \begin{bmatrix} 1 & 0.0114 & 0 \\ 0 & 1 & 0.0114 \\ 0 & 0 & 0.9799 \end{bmatrix}, \mathbf{B}_{x,y} = \begin{bmatrix} 0 \\ 0 \\ 5.0719 \times 10^{-5} \end{bmatrix}. \tag{15}$$

[6]Further experiments show that both axis can be driven by the same controller utilizing a single model.
[7]Pulse Position Modulation is a common communication protocol used on unmanned vehicles.

The system can be further tested by estimating all states in open-loop from the measured input. We can evaluate its performance empirically by comparing estimates with measured data.

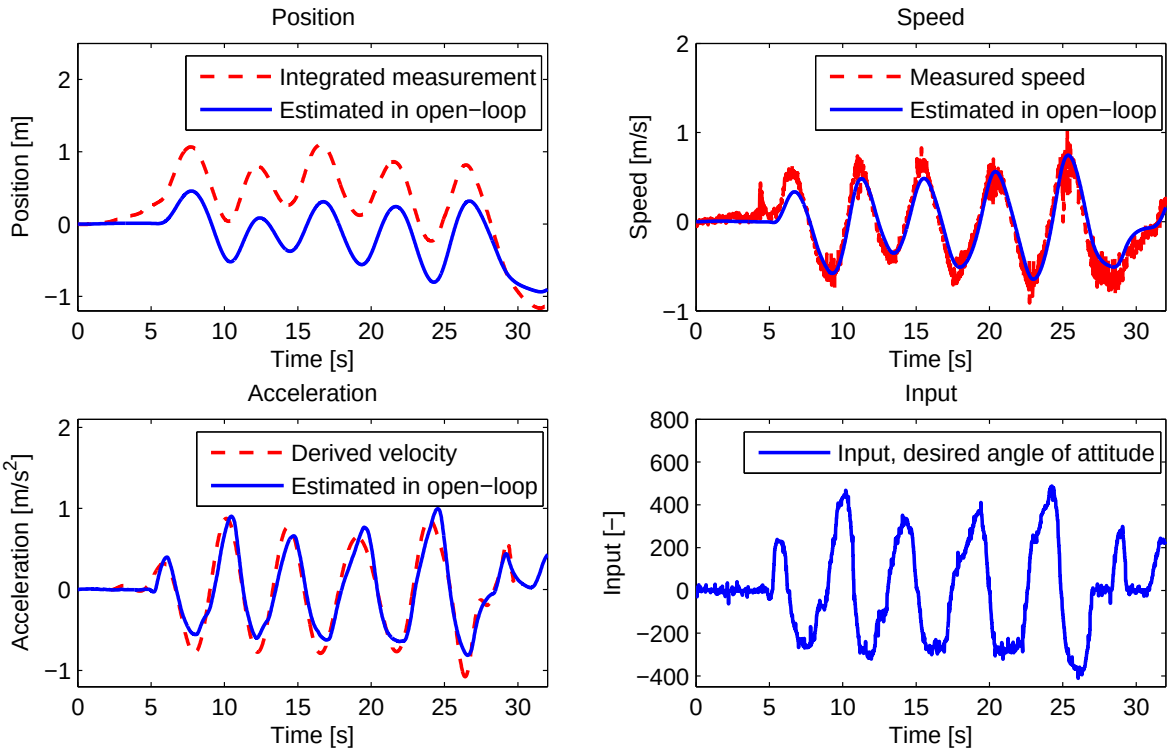


Figure 3.2: Identification - open loop estimation of attitude system states.

As it can be seen in fig. 3.2, states estimated in open-loop are tracking the genuine values without a significant error. Some drift can be seen in the position, after the second integrator. But the model fits the measured data sufficiently, at least for purpose of the control design.

3.2 Altitude subsystem identification

For the purpose of altitude identification, another flight was realized. Again, with an intention to excite the system's dynamics. In this case, the measured state is an altitude. It is supplied by an ultrasonic rangefinder which works in a range from 0.3 m to 4 m. The sensor is implemented on the *px4flow* unit and although the data is sent together with the optical flow, its actual rate is about 20 Hz. Data were again fitted with polynomial and further differentiated into other states. Figure 3.3 shows data on which parameters P_5, P_6 were found. In this particular situation, input signal had to be shifted to cancel the DC[8] component which countered the gravitational acceleration. The input offset was found by another optimization with a cost computed as a sum of sums of quadratic errors of all state open-loop estimations.

[8]Supposing the force caused by the gravity of Earth is a constant, the total thrust signal contains a DC component countering the gravitational pull.

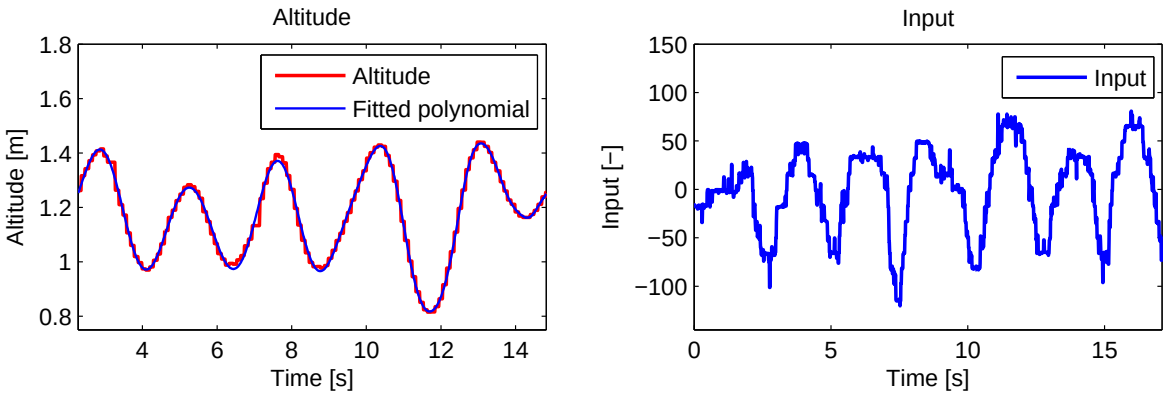


Figure 3.3: Identification data - altitude

The measurement unit of the input signal is again a time difference from a mean PPM pulse measured in discrete steps of a hardware timer. Following parameters were found using the *least squares method* described in the beginning of this chapter:

$$P_5 = 0.9519, P_6 = 0.0012 \times 10^{-5}. \tag{16}$$

Having these constants, the attitude system can be completed from eq. (9) with the particular values

$$\mathbf{A}_z = \begin{bmatrix} 1 & 0.0114 & 0 & 0 \\ 0 & 1 & 0.0114 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.9519 \end{bmatrix}, \mathbf{B}_z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & -g \\ 0.0012 & 0 \end{bmatrix}, \tag{17}$$

where $g \approx 9.8ms^{-2}$ is the gravitational acceleration.

As can be seen in fig. 3.4 the open-loop estimation holds fairly up to the speed state. The altitude drifts heavily in a horizon of seconds, which indicates that the LTI system is not identified as good as in case of the attitude system. It could be due to omitting some physical phenomena e.g. in case of the rotor thrust or because of the linearization itself.

3.3 Yaw subsystem identification

The system could be identified by exactly the same procedure as the previous subsystems. But in our case, the UAV is not equipped with any device to measure its yaw angle or rate. It is stabilized by dead-reckoning from IMU[9] (yaw rate measured by a gyro). The actual data

[9]Inertial Measurement Unit - measures accelerations and rotational speeds of the UAV.

from IMU are present only in integrated stabilization and are not sent to the custom control board. This would require a modification of the used stabilization board.

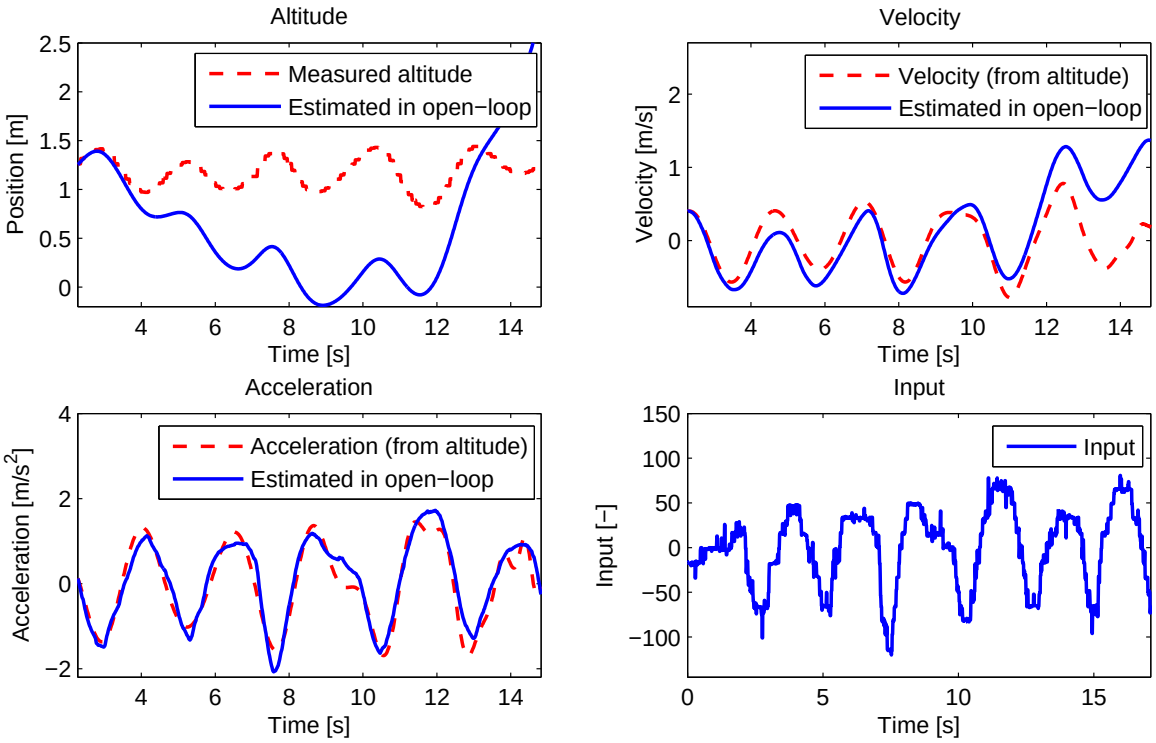


Figure 3.4: Identification - open loop estimation of altitude system states.

3.4 Summary

This chapter discussed the identification of parameters of the dynamical model presented in chapter 2. Mathematical optimization was used to fit a model to measured data in terms of least squares of residuals. We were able to find common parameters for both axis of the attitude model P_1, P_2, and parameters of the altitude model P_3, P_4. Using this information state space representation of both systems was constructed. It can be further used for state estimation and system control as it is shown in following chapters. The identification is considered successful by means of open-loop estimation error as it can be seen in presented figures. The absence of the yaw measurement could be solved by adding e.g. a magnetometer or an additional camera system.

4 State observer

Design of the *model predictive controller*, requires a knowledge about development of all system states. This is a different situation than in the case of previously used PID controller, when only the controlled state needs to be known. There are some situations and corresponding systems, where all states values are relatively easy to obtain, respectively they might be already measured for another reason than the control. For instance cosmic, aerospace or another vehicles, where the typical states are position and its derivatives. But if not all states can be measured or we do not want to measure them, the state observer needs to be implemented. The state observer is a dynamical system that is simulated concurrently with the controlled system. It has the same number of inputs and it is controlled by the same actions as the real system. The order of the observer is the same as the order of the system. Its states are supposed to be observable and they should correspond to the real system's states.

4.1 Open-loop observer

Let us have an LTI system that needs to be observed (assuming $\mathbf{C} = \mathbf{I}$, $\mathbf{D} = \mathbf{0}$ i.e. the output consists directly of all states and there is no direct transfer from the input to the output). The uncertainty in the model is covered by a random variable $\mathbf{w} \in \mathbb{R}^n$, which is usually called a *process noise*. The system description (7) can be extended to

$$\mathbf{x}_{[t+1]} = \mathbf{A}\mathbf{x}_{[t]} + \mathbf{B}\mathbf{u}_{[t]} + \mathbf{w}_{[t]}. \tag{18}$$

The open-loop observer can be constructed by setting up following system (19), where $\hat{\mathbf{x}}$ denotes a vector of estimated state values. If $\mathbf{w}$ has non zero-mean, $\hat{\mathbf{x}}$ drifts from $\mathbf{x}$ since the process noise is integrated over time. Three distinct situations can happen during the execution of the observer:

$$\hat{\mathbf{x}}_{[t+1]} = \mathbf{A}\hat{\mathbf{x}}_{[t]} + \mathbf{B}\mathbf{u}_{[t]} \tag{19}$$

- Estimated states track well corresponding system states. The system was identified perfectly and there is no need for feedback control in this situation.
- Estimated states track well system states, but there is some drift during the time frame of the experiment. The feedback loop is required to provide zero-offset observation.
- States estimated by the observer are completely out of scope of real system states which may suggest that the observer's model is wrong.

Open-loop estimations in chapter 3 indicate that our model is satisfactory, but the open-loop estimator would not probably lead to precise control results.

4.2 Closed-loop observer

The open-loop observer can be basically corrected by closing a feedback loop around the system as follows

$$\hat{\mathbf{x}}_{[t+1]} = \mathbf{A}\hat{\mathbf{x}}_{[t]} + \mathbf{B}\mathbf{u}_{[t]} - \mathbf{L}\left(\mathbf{x}_{[t]} - \hat{\mathbf{x}}_{[t]}\right). \tag{20}$$

There are methods of finding $\mathbf{L}$ such that poles of the observer are desirably placed[10]. One can utilize that placing poles of a state feedback is a dual problem for placing poles of the observer. Practically, a state feedback controller can be tuned using matrices $\mathbf{A}^T, \mathbf{B}^T$ to get $\mathbf{L}^T$ for the observer, hence the duality.

4.3 Kalman filter

In this thesis, the Kalman filter (KF) was implemented to estimate all states of the helicopter and to filter the measured data from sensors. The Kalman filter is variant of the closed-loop iterative estimator. If well tuned, it corrects a drift in estimated states. It is widely used not only as an observer but also for its great filtering capabilities. A hypothesis about estimated states takes form of a normal distribution with mean vector $\hat{\mathbf{x}}$ and covariance matrix $\hat{\mathbf{\Sigma}}$. It presumes a dynamical model in the form (18), where $\mathbf{w} \in \mathbb{R}^n$ is drawn from normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{R})$, $\mathbf{R} \in \mathbb{R}^{n \times n}$. Furthermore, we presume a linear sensor model

$$\hat{\mathbf{z}}_{[t]} = \mathbf{P}\hat{\mathbf{x}}_{[t]} + \mathbf{v}_{[t]}, \tag{21}$$

where $\hat{\mathbf{z}}_{[t]} \in \mathbb{R}^p$ is the measurement vector, $\mathbf{P} \in \mathbb{R}^{p \times n}$ is a matrix that maps the state vector to the measurement and $\mathbf{v} \in \mathbb{R}^p$ is the measurement noise which is drawn from $\mathcal{N}(\mathbf{0}, \mathbf{Q})$, $\mathbf{Q} \in \mathbb{R}^{p \times p}$. A real measurement $\mathbf{z}_{[t]} \in \mathbb{R}^p$ is modeled using the sensor model by adding a sample of the measurement noise to all estimated states and transforming it by $\mathbf{P}$ to a measurement vector. The filter updates a state vector $\hat{\mathbf{x}}_{[t]}$ and its covariance matrix $\hat{\mathbf{\Sigma}}_{[t]}$ between iterations. The feedback withing the filter is then constructed using the inverse sensor model with the real measurement $\mathbf{z}_{[t]}$.

	vector	covariance	mapping to $\hat{\mathbf{x}}$	mapping to $\hat{\mathbf{z}}$
estimated states	$\hat{\mathbf{x}}$	$\hat{\mathbf{\Sigma}}$	$\mathbf{I}$	$\mathbf{P}$
measurement	$\hat{\mathbf{z}}$	$\mathbf{Q}$		$\mathbf{I}$
model (process)	$\mathbf{x}$	$\mathbf{R}$		

Table 4.1: Overview of vectors and matrices used in Kalman filter.

[10] Assuming the system can be converted to the *controlled canonical form*.

4.3.1 Prediction phase

There are two stages of the KF algorithm, which are shown in the block diagram in figure 4.1. The first one is the *prediction phase* (22) which can be easily related to the open-loop observer (19). Here, the new state vector is estimated using the model (18), previous state $\hat{\mathbf{x}}_{[t-1]}$ and the input $\mathbf{u}_{[t-1]}$. Its covariance $\hat{\mathbf{\Sigma}}_{[t]}$ is modified using the model and the process noise covariance $\mathbf{R}$.

$$\begin{aligned} \hat{\mathbf{x}}_{[t]} &\leftarrow \mathbf{A}\hat{\mathbf{x}}_{[t-1]} + \mathbf{B}\mathbf{u}_{[t-1]} \\ \hat{\mathbf{\Sigma}}_{[t]} &\leftarrow \mathbf{A}\hat{\mathbf{\Sigma}}_{[t-1]}\mathbf{A}^T + \mathbf{R} \end{aligned} \tag{22}$$

4.3.2 Correction phase

The second phase is the *correction phase* (23), where the state vector is updated using the measurement $\mathbf{z}_{[t]}$. The covariance $\hat{\mathbf{\Sigma}}_{[t]}$ is again modified, but this time using the noise measurement covariance $\mathbf{Q}$. Following assignment statements denote the correction phase of the Kalman filter

$$\begin{aligned} \mathbf{K}_{[t]} &\leftarrow \hat{\mathbf{\Sigma}}_{[t]}\mathbf{P}^T \left(\mathbf{P}\hat{\mathbf{\Sigma}}_{[t]}\mathbf{P}^T + \mathbf{Q}\right)^{-1}, \\ \hat{\mathbf{x}}_{[t]} &\leftarrow \hat{\mathbf{x}}_{[t]} + \mathbf{K}_{[t]}\left(\mathbf{z}_{[t]} - \mathbf{P}\hat{\mathbf{x}}_{[t]}\right), \\ \hat{\mathbf{\Sigma}}_{[t]} &\leftarrow \left(\mathbf{I} - \mathbf{K}_{[t]}\mathbf{P}\right)\hat{\mathbf{\Sigma}}_{[t]}, \end{aligned} \tag{23}$$

where $\mathbf{K}_{[t]}$ is the Kalman gain, $\mathbf{I}$ is the identity matrix and $\mathbf{P}$ is the matrix that maps estimated states to the measurement. The first part consists of computing a Kalman gain $\mathbf{K}_{[t]}$. It has a direct correspondence to the matrix $\mathbf{L}$ in (20) thus it determines the feedback effect in the observer. The gain can be fine tuned by carefully setting $\mathbf{R}$ and $\mathbf{Q}$. The $\mathbf{Q}$ can be found by observing the noise parameters of measurements. It means that the process noise can be independently set. The table 4.1 denotes the important vectors and matrices that are used in the filter. Practically there is a trade-off between trusting fully to the model and trusting fully to sensors. When setting $\mathbf{R}$ the filtered values have to be observed and the Kalman filter should be set basically by tuning the ratio between elements of $\mathbf{R}$ and $\mathbf{Q}$. The desired outcome usually is to eliminate the measurement noise while still preserving the zero-offset tracking (with sufficient transient response) of all estimated states.

4.4 Summary

The Kalman filter was presented in order to estimate all system states and filter measured data. Its performance can be shown on subsequence of data (forward movement) used in

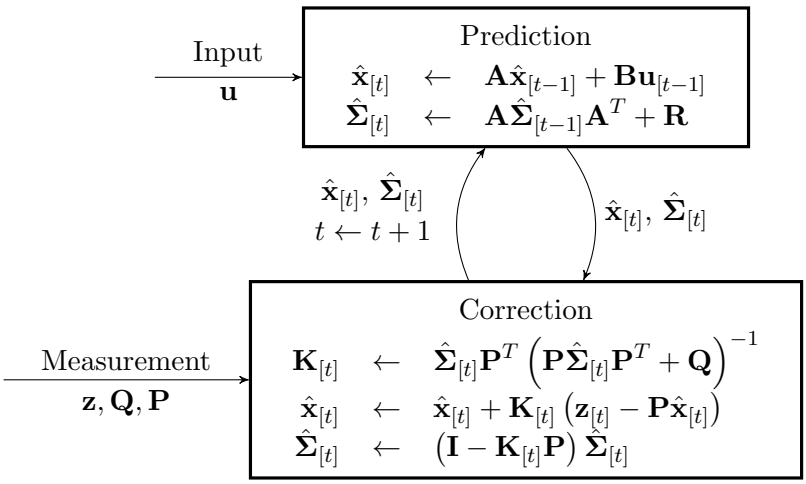


Figure 4.1: Block diagram of the Kalman filter.

chapter 3.1. The figure 4.2 shows the velocity of UAV measured by onboard sensor. Among others, it is able to estimate all other states including the position and the acceleration which is essential for further use of model predictive controller. It can be tuned to estimate noisy signal with relatively small delay compared to e.g. exponential filter[11]. For particular parameters of KF that were used in the final implementation see chapter 7 where also an extended form of LTI system is presented to provide a disturbance estimation. For an intuitive description of KF in more details see [15].

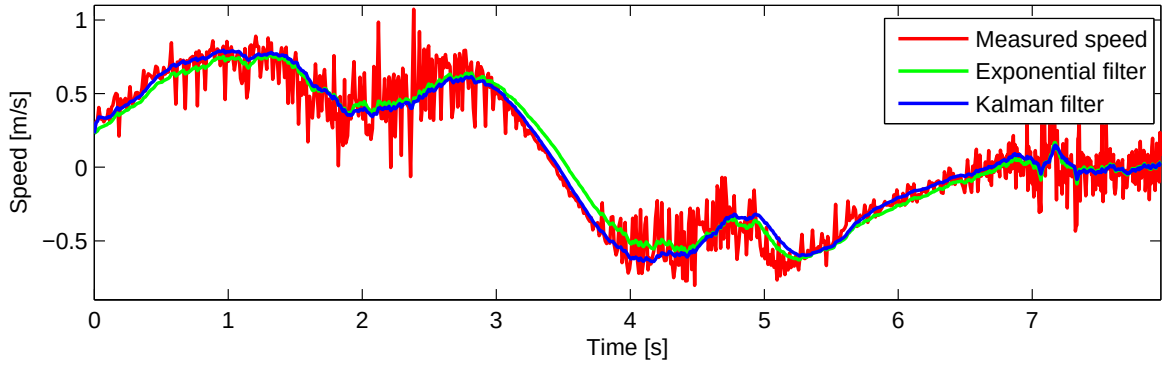


Figure 4.2: Kalman filter tracking a noisy singal of UAV forward speed. See the difference between filtered values, which might have a significant effect on the control.

[11] The exponential filter is a method where filtered values are computed as $\hat{\mathbf{x}}_{[t]} = \alpha\hat{\mathbf{x}}_{[t-1]} + (1-\alpha)\mathbf{P}\mathbf{z}_{[t]}$.

5 Model predictive control

Model predictive controller (MPC), also known as *receding horizon controller*, is an uncommon technique of controlling unmanned aircraft. Mostly because it is a challenging task to implement it into an embedded hardware, unlike other feedback loop controllers usually used on UAVs, namely PID (proportional-integral-derivative controller) and full-state feedback. The origins of MPC can be found in control of chemical plants, where the time constants of such dynamical processes are relatively high (up to order of hours) thus the computational demand is not so limiting. Also the constraint handling, an inherent property of MPC, is widely used while driving chemical processes. Since then the MPC started to spread on faster systems as the hardware become more powerful. Nowadays it is used to drive systems with sampling in order of milliseconds and tens of hertz loop rate.

The control loop itself is built upon repeatedly optimizing a cost function (usually called *objective function*) with decision variables that represent a desired input action. It is usually a function of all states, desired trajectory and system inputs over a certain time horizon, often called *prediction horizon*. It penalizes (by increase of cost values) the difference between predicted and desired state trajectory. It also penalizes the control action itself, which can be interpreted as penalizing the energy used for controlling the system. In other words, the objective function returns a scalar value that quantifies whether the controller drove the system well. Such function can have extrema. Our goal is to find its minimum which corresponds to states changing according to our desired trajectory. The minimum can be local or global, depending on the function itself. It is usually desirable to find the global minimum, since it corresponds to the optimal control action with respect to the constructed objective function.

There are several classes of continuous optimization problems depending on the type of the objective function and constraints. Problems based on linear or quadratic objective function subjected to linear inequality constraints are common. These are historically well studied cases with known methods for solving them. The problem is usually called *Linear Programming* (LP) if optimizing linear function, or *Quadratic Programming* (QP) if optimizing a quadratic function. Since the MPC can be formulated as LP or QP, the control design problem is then basically reduced to solving a QP or LP program and the main focus is left on system modeling and fine-tuning of free parameters of the objective function. The optimization task itself is usually left on dedicated solver that it specialized on the particular function type. In this thesis, we consider only the linear MPC i.e. controlling an LTI system proposed in chapter 3. The MPC can be formulated as LP or QP, depending on what type of Euclidean norm is used for computing the distance between two states. When using the ℓ^1 or ℓ^∞ norm, the formulation leads to a linear program. One can formulate the LP in a way that minimizes the maximal deviation from desired trajectory - this formulation is usually called a *robust MPC* (RMPC). We will focus on the QP formulation (ℓ^2 norm, QMPC) since the quadratic penalization can be more expressive than the linear one — it penalizes large errors more and additionally provides a deadband in the controller.

5.1 System prediction

For the purpose of MPC, it is essential to be able to predict a series of system's states $\underline{\mathbf{x}} = (\mathbf{x}_{[0]}^T, \mathbf{x}_{[1]}^T, ..., \mathbf{x}_{[M-1]}^T)^T$ based on the initial state $\mathbf{x}_{[0]}$ and inputs $\underline{\mathbf{u}} = (\mathbf{u}_{[0]}^T, \mathbf{u}_{[1]}^T, ..., \mathbf{u}_{[M-1]}^T)^T$, where M is the length of the prediction horizon. Let us consider a discrete, linear, time-invariant system with n states and k inputs, assuming $\mathbf{C} = \mathbf{I}$ and $\mathbf{D} = \mathbf{0}$ (again, assuming there is no direct transfer from the input to the output and the output consists directly of all states)

$$\mathbf{x}_{[t+1]} = \mathbf{A}\mathbf{x}_{[t]} + \mathbf{B}\mathbf{u}_{[t]}, \tag{24}$$

where $\mathbf{x}_{[t]} \in \mathbb{R}^n$ is the state vector in the sample time t, $\mathbf{u}_{[t]} \in \mathbb{R}^k$ is the input vector in the sample time t, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the system matrix and $\mathbf{B} \in \mathbb{R}^{n \times k}$ is the input matrix. First two prediction steps from $\mathbf{x}_{[0]}$ can be formulated as

$$\begin{aligned} \mathbf{x}_{[1]} &= \mathbf{A}\mathbf{x}_{[0]} + \mathbf{B}\mathbf{u}_{[0]}, \\ \mathbf{x}_{[2]} &= \mathbf{A}\mathbf{x}_{[1]} + \mathbf{B}\mathbf{u}_{[1]} = \mathbf{A}^2\mathbf{x}_{[0]} + \mathbf{A}\mathbf{B}\mathbf{u}_{[0]} + \mathbf{B}\mathbf{u}_{[1]}. \end{aligned} \tag{25}$$

The prediction can be further generalized for any time step as follows:

$$\mathbf{x}_{[t+2]} = \mathbf{A}^2\mathbf{x}_{[t]} + \mathbf{A}\mathbf{B}\mathbf{u}_{[t]} + \mathbf{B}\mathbf{u}_{[t+1]} \tag{26}$$

The recurrent relation can be used to get the prediction in any future time step. Moreover, it can be put in the matrix form for all future time steps within the prediction horizon. Matrices denoted in (27) are basic building blocks of QMPC formulation presented on following page.

$$\underbrace{\begin{bmatrix} \mathbf{x}_{[1]} \\ \mathbf{x}_{[2]} \\ \vdots \\ \mathbf{x}_{[M-1]} \end{bmatrix}}_{\underline{\mathbf{x}}} = \underbrace{\begin{bmatrix} \mathbf{A} \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^{M-1} \end{bmatrix}}_{\hat{\mathbf{A}}} \mathbf{x}_{[0]} + \underbrace{\begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}\mathbf{B} & \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{M-1}\mathbf{B} & \mathbf{A}^{M-2}\mathbf{B} & \dots & \mathbf{B} \end{bmatrix}}_{\hat{\mathbf{B}}} \underline{\mathbf{u}} \tag{27}$$

Thus it can be simplified into a form using matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ as

$$\underline{\mathbf{x}} = \hat{\mathbf{A}}\mathbf{x}_{[0]} + \hat{\mathbf{B}}\underline{\mathbf{u}}. \tag{28}$$

5.2 Problem formulation - QMPC

The objective function for QMPC is formulated as sum of squares of weighted control errors combined with weighted control actions. In our case the last control error is weighted differently than all the previous errors, which is denoted by the second summand in (29). By doing that, we can prioritize the final error in the horizon and force the controller to converge to the desired trajectory.

$$\mathrm{V}\left(\underline{\mathbf{x}}, \underline{\mathbf{u}}\right) = \frac{1}{2} \sum_{i=0}^{M-2} \left(\mathbf{e}_{[i]}^T \mathbf{Q} \mathbf{e}_{[i]} + \mathbf{u}_{[i]}^T \mathbf{P} \mathbf{u}_{[i]} \right) + \frac{1}{2} \mathbf{e}_{[M-1]} \mathbf{S} \mathbf{e}_{[M-1]} \tag{29}$$

The control error is denoted by $\mathbf{e}_{[t]} = \mathbf{x}_{[t]} - \tilde{\mathbf{x}}_{[t]}$ in the time t, $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is the state weighting matrix, $\mathbf{P} \in \mathbb{R}^{k \times k}$ is the input weighting matrix, and $\mathbf{S} \in \mathbb{R}^{n \times n}$ is the matrix weighting the final state values. Matrices $\mathbf{Q}$, $\mathbf{S}$ need to be positive semi-definite ($\mathbf{Q}, \mathbf{S} \succeq 0$) and matrix $\mathbf{P}$ need to be positive definite ($\mathbf{P} \succ 0$) to ensure that the function $\mathrm{V}\left(\underline{\mathbf{x}}, \underline{\mathbf{u}}\right)$ is strictly convex. Moreover, elements of $\underline{\mathbf{x}}$ and $\underline{\mathbf{u}}$ need to satisfy the system's dynamics (24). Furthermore by inducing (28) into (29) the objective function can be rewritten into matrix form

$$\mathrm{J}(\underline{\mathbf{u}}) = \frac{1}{2} \underline{\mathbf{u}}^T \underbrace{\left(\hat{\mathbf{B}}^T \hat{\mathbf{Q}} \hat{\mathbf{B}} + \hat{\mathbf{P}} \right)}_{\hat{\mathbf{H}}} \underline{\mathbf{u}} + \underline{\mathbf{u}}^T \underbrace{\left(\hat{\mathbf{Q}} \hat{\mathbf{B}} \right)^T \left(\hat{\mathbf{A}} \mathbf{x}_{[0]} - \underline{\tilde{\mathbf{x}}} \right)}_{\hat{\mathbf{c}}}, \tag{30}$$

where $\underline{\tilde{\mathbf{x}}} = (\tilde{\mathbf{x}}_{[0]}^T, \tilde{\mathbf{x}}_{[1]}^T, ..., \tilde{\mathbf{x}}_{[M-1]}^T)^T$ is the reference trajectory for all states consisting of a state vector for each step of the prediction horizon. $\hat{\mathbf{Q}} \in \mathbb{R}^{nM \times nM}$ and $\hat{\mathbf{P}} \in \mathbb{R}^{kM \times kM}$ are weighting matrices denoted in (31). Matrices $\hat{\mathbf{H}} \in \mathbb{R}^{kM \times kM}$ and $\hat{\mathbf{c}} \in \mathbb{R}^{kM}$ then define the quadratic form.

$$\hat{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} & \dots & \vdots \\ \mathbf{0} & \dots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{S} \end{bmatrix}, \hat{\mathbf{P}} = \begin{bmatrix} \mathbf{P} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{P} & \dots & \vdots \\ \mathbf{0} & \dots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{P} \end{bmatrix}. \tag{31}$$

Finally the optimization task can be formulated as minimizing the objective function $\mathrm{J}(\underline{\mathbf{u}})$ subject to constraints on $\underline{\mathbf{u}}$ (which will be discussed in the following paragraphs). This problem is solved repeatedly for new $\mathbf{x}_{[0]}$ in each control step

$$\begin{aligned} \min_{\underline{\mathbf{u}} \in \mathbb{R}^{kM}} \quad & \mathrm{J}(\underline{\mathbf{u}}) = \frac{1}{2} \underline{\mathbf{u}}^T \hat{\mathbf{H}} \underline{\mathbf{u}} + \underline{\mathbf{u}}^T \hat{\mathbf{c}} \\ \text{s.t.} \quad & \text{constrains on } \mathbf{u}. \end{aligned} \tag{32}$$

5.3 Constraints

Two types of constraints are usually imposed on the searched solution if optimizing the quadratic form (32). The first type is often related to the physical limitations of the controlled system. System actuators (particularly the physical ones) may not be designed to accept an arbitrary input signal. For example a servo-motor has a maximum allowed electric current and rotational speed or control surface of an airplane operates within a certain angles of freedom. Let us focus on this type of input constraints defined as *box constraints* i.e. decision variables need to lie within a closed hypercube.

5.3.1 Input constraints

Input box constraints allow the controller to find a solution (control actions) that satisfies the input limitations of system actuators. We can find an analogy in adding a saturation on control outputs of PID controller. They can be modeled by set of inequalities taking following form

$$\mathbf{b} \leq \underline{\mathbf{u}} \leq \mathbf{g}, \tag{33}$$

where $\mathbf{b} \in \mathbb{R}^{kM}$ and $\mathbf{g} \in \mathbb{R}^{kM}$ are constraint vectors denoting the lower and upper bound on inputs. Such constrained optimization task can be easily solved as it can be seen in chapter 5.5. Though it seems more practical to tune parameters of MPC (matrices $\hat{\mathbf{Q}}$ and $\hat{\mathbf{P}}$) in a way, that the controller won't naturally produce such control action even when starting from improbable initial conditions. This approach leads to proper control actions anyway so the input constraints can serve as a protection mechanism.

5.3.2 State constraints

The ability to constrain particular state values is one of the main features of the MPC. It can be used to drive the system within some safety region of state variables and to find a control action that won't push the system into unwanted states (sometime irreversibly, e.g. in chemical processes). A convenient example can be found also in a field of unmanned vehicles that can usually operate only within a certain region of velocities and accelerations. By inducing state constraints to the optimization task, it suddenly becomes more difficult to solve. These constraints can be also modeled as a set of linear inequalities using the prediction equation (28). Vectors $\underline{\mathbf{v}} \in \mathbb{R}^{nM}$ and $\underline{\mathbf{w}} \in \mathbb{R}^{nM}$ denote the lover and upper bounds on states. State constraints create the general linear inequalities of the quadratic programming.

$$\underline{\mathbf{v}} \leq \hat{\mathbf{A}}\mathbf{x}_{[0]} + \hat{\mathbf{B}}\underline{\mathbf{u}} \leq \underline{\mathbf{w}} \tag{34}$$

5.3.3 Other constraints

Several different types of constraints can be declared to improve or change the behavior of the system. One particularly useful example is limiting the rate of change of input within the prediction horizon. This leads to smoother input signals and can prevent some stress on actuators of the system. Another possibility is to restrict the monotonicity of particular state values over the horizon [31]. This can lead to suppression of overshoots and limiting reactions of unstable zeros (systems with non-minimum phase characteristics) of the system (initially, the system tends to move in the opposite direction than intended). Finally, one can create constraints that force the control signal to lie out of a deadzone of an actuator. Deadzones are inconvenient actuator nonlinearities that can make the control design otherwise very painful.

5.4 Move blocking – reducing complexity of MPC

Until now, we have considered that each decision variable of the optimization task directly corresponds to a value of input signal in a particular time of the prediction horizon. One could ask whether it is necessary to optimize over the input signal with the same density of signal changes at the beginning as at the end of the prediction horizon. Simulation and experiments show that when coming to the end of the horizon, the optimized input tends to take form of a constant function, assuming the initial condition is sufficiently near to the desired trajectory. Also, supposing that the model of the UAV is not perfect and that the control loop (see chapter 5.7) uses only first few steps of the input signal, there might be an open-loop prediction error too high to payoff for a densely distributed variables. The move blocking technique allows to project a smaller number of variables to cover a longer prediction horizon. The transformation is denoted by following equation

$$\underline{\mathbf{u}} = \underbrace{\begin{bmatrix} \begin{bmatrix}1\\\vdots\\1\end{bmatrix} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \begin{bmatrix}1\\\vdots\\1\end{bmatrix} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \begin{bmatrix}1\\\vdots\\1\end{bmatrix} \end{bmatrix}}_{\mathbf{U}} \underline{\mathbf{u}}_r, \tag{35}$$

where $\mathbf{U} \in \{0,1\}^{kM \times kN}$, $N \in \mathbb{N}$ is the number of decision variables and $\underline{\mathbf{u}}_r \in \mathbb{R}^{kN}$ is the reduced input vector. The MPC task can be then simply modified by creating a new matrix $\hat{\mathbf{B}}_r = \hat{\mathbf{B}}\mathbf{U}$ to solve the optimization with, and then simply project the variables on the whole

horizon. The distribution of variables can vary. One can distribute them evenly or assign a larger portion of them at the beginning of the horizon. This technique can also improve system stability by prolonging the horizon when maintaining a similar computational complexity. Although, the optimization does not control the system precisely (withing the prediction) the objective function is still in play in every system step. The chapter 7.2.3 discuses the particular setting used during the implementation and experiments.

5.5 Solving QMPC - unconstrained

The quadratic function in our task is a convex function due to matrices $\mathbf{Q}$, $\mathbf{S}$ and $\mathbf{P}$ which are designed to be positive semi-definite ($\mathbf{Q}, \mathbf{S} \succeq 0$) and positive definite ($\mathbf{P} \succ 0$). Using this assumption we can find a global minimum by translating it (by a vector $\underline{\mathbf{u}}^*$) to the origin and thus changing it to a *quadratic form*. The quadratic form with a semi-definite matrix has a minimum in the origin. The translation can be found by completing the function to the square

$$\begin{aligned}\frac{1}{2}\underline{\mathbf{u}}^T\hat{\mathbf{H}}\underline{\mathbf{u}} + \underline{\mathbf{u}}^T\hat{\mathbf{c}} &= \frac{1}{2}(\underline{\mathbf{u}} - \underline{\mathbf{u}}^*)^T\hat{\mathbf{H}}(\underline{\mathbf{u}} - \underline{\mathbf{u}}^*) \\ &= \frac{1}{2}\underline{\mathbf{u}}^T\hat{\mathbf{H}}\underline{\mathbf{u}} - \frac{1}{2}\underline{\mathbf{u}}^T\hat{\mathbf{H}}\underline{\mathbf{u}}^* - \frac{1}{2}\underline{\mathbf{u}}^{*T}\hat{\mathbf{H}}\underline{\mathbf{u}} + \frac{1}{2}\underline{\mathbf{u}}^{*T}\hat{\mathbf{H}}\underline{\mathbf{u}}^* \\ &= \frac{1}{2}\underline{\mathbf{u}}^T\hat{\mathbf{H}}\underline{\mathbf{u}} - \underline{\mathbf{u}}^T\hat{\mathbf{H}}\underline{\mathbf{u}}^* + \frac{1}{2}\underline{\mathbf{u}}^{*T}\hat{\mathbf{H}}\underline{\mathbf{u}}^*.\end{aligned} \tag{36}$$

By comparing parts of the same degree we get following equations

$$\begin{aligned}\hat{\mathbf{c}} &= -\hat{\mathbf{H}}\underline{\mathbf{u}}^*, \\ 0 &= \frac{1}{2}\underline{\mathbf{u}}^{*T}\hat{\mathbf{H}}\underline{\mathbf{u}}^*.\end{aligned} \tag{37}$$

Finally the solution can be found in a closed-form as $\underline{\mathbf{u}}^* = -\hat{\mathbf{H}}^{-1}\hat{\mathbf{c}}$.

5.6 Solving QMPC - constrained

Firstly, we will see how to optimally solve QMPC constrained only by the input constraints (33). Secondly, we will shortly discuss several QP algorithms for solving QMPC constrained by general linear inequalities (produced by (34)). Since the work done in this thesis focuses on implementing the first case, only a short survey of QP algorithms is presented.

5.6.1 QMPC with input constraints

Input box constrains are a special case of general linear inequalities of QP. Thus they create a convex set of feasible solutions. When the global optimum of a convex quadratic form is not a feasible solution, the actual optimum satisfying the constraints is located on the facet of the convex polytope of the feasible set [9]. One can find it by projecting the unconstrained optimum orthogonally on the feasible set. In the case of box constrains imposed on decision variables this can be done by using median function. It is possible to project variables independently on each other because box constrains are orthogonal and perpendicular to their corresponding axis.

$$\underline{\tilde{\mathbf{u}}}_i^* \leftarrow \text{median}\left(\mathbf{b}_i, \underline{\mathbf{u}}_i^*, \mathbf{g}_i\right), \forall i = 0, ..., kN \tag{38}$$

One could ask why this process leads to the optimal solution on a feasibility set. A simple proof can be constructed. The figure 5.1 shows an analogous example for two dimensions. It is assumed that the function J($\underline{\mathbf{u}}$) is convex. When the global optimum $\underline{\mathbf{u}}^*$ is found using the closed-form solution and it satisfies all constraints, the task is finished. Otherwise there is a constrained optimum $\underline{\tilde{\mathbf{u}}}^*$ located in the feasibility set which can be found by (38). If there would be another, different, better solution on the feasible set, we could get to it by following facets of the feasibility set polyhedron [9]. That means the step from $\underline{\tilde{\mathbf{u}}}^*$ would lower the objective value of J($\underline{\mathbf{u}}$) thus there would be a negative gradient in the direction. But when using (38), every step of projecting variables violating constraints is a step over a positive gradient (as any step from the global optimum of a convex function). This would lead to a claim that there is a local maximum in $\underline{\tilde{\mathbf{u}}}^*$ which is contradictory to the assumption.

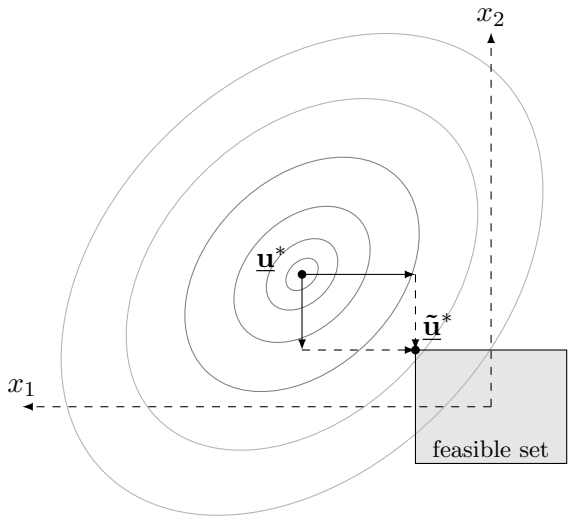


Figure 5.1: An illustrative example of 2-dimensional quadratic form with box constraints.

5.6.2 QMPC with state constraints

As it was mentioned previously, the state constraints are the most general of linear inequalities. Solving such quadratic program is a difficult task, compared to the one constrained only by input constraints. The author of [28] compares several iterative methods for solving constrained model predictive control. The first one is the **Active set method**. It uses the fact that the optimum is attained on the boundary of the feasible set. Throughout the iterations, the algorithm walks along the facets of the convex polytope in a similar way as the *simplex algorithm* does for linear programming. The QP with equality constraints (or unconstrained) is solved every step. The number of iterations depends on the number of constraints active in the optimum.

Another method presented in [28] and used in [44] is the **Fast gradient method**. It is a modification of the classical gradient descend. But instead of moving against the gradient vector, it rather projects it to the feasible set. Its computational demands strongly depend on the projection itself which means that it is usually used if only a certain type of constraints is used e.g. box, simplex or Euclidean norm ball. The method also suffers from bad convergence if the quadratic form is not well conditioned.

The last method discussed in [28] is the **Interior point method** using logarithmic barrier function. It constructs a new unconstrained optimization problem based on the original one which can be solved by e.g. Newton's method. The new task is only an approximation which precision can be controlled. It requires to be started from a feasible solution and the computational complexity strongly depends on the conditionality of the quadratic form.

5.7 The MPC control loop

The optimization task is solved repeatedly, if using MPC for realtime control. Only the first elements of $\underline{\mathbf{u}}$ are used for control, until another iteration with updated $\mathbf{x}_{[0]}$ is done. The number of elements used depends on how well the system was modeled and identified and how fast the optimization can be done. One have to set the complexity (number of variables, length of prediction horizon) in such way, that the optimization is done fast enough so the system could react adequately to disturbances and trajectory changes.

5.8 Summary

This is the end of the theoretical introduction for implementation of MPC into embedded hardware of unmanned helicopter. We have discussed how to formulate a control task as a mathematical optimization problem using quadratic programming. The input and state constraints were presented as well as the move blocking technique for reducing the number of variables. At last, methods for solving unconstrained and constrained MPC were described.

6 Hardware and Software platform

In order to test the embedded implementation of MPC on real aircraft, either an existing platform has to be used, or a new one created. Creating a custom hardware is a tedious job, but it is an investment that may repay itself by providing complete control over platform's parameters and design. The key decision has been done to improve the existing design of custom control board (see author's Bc. thesis [4] for more details) emphasizing possibility to reuse the code that has been already developed. A new version has been designed by incorporating experience and observations from prior development. Following chapter firstly describes the UAV testbed used for experimental validation of the control design, then there is the new control board presented with overview of its features and capabilities, and lastly the software structure is discussed including realtime operating system and custom matrix library used for UAV control.

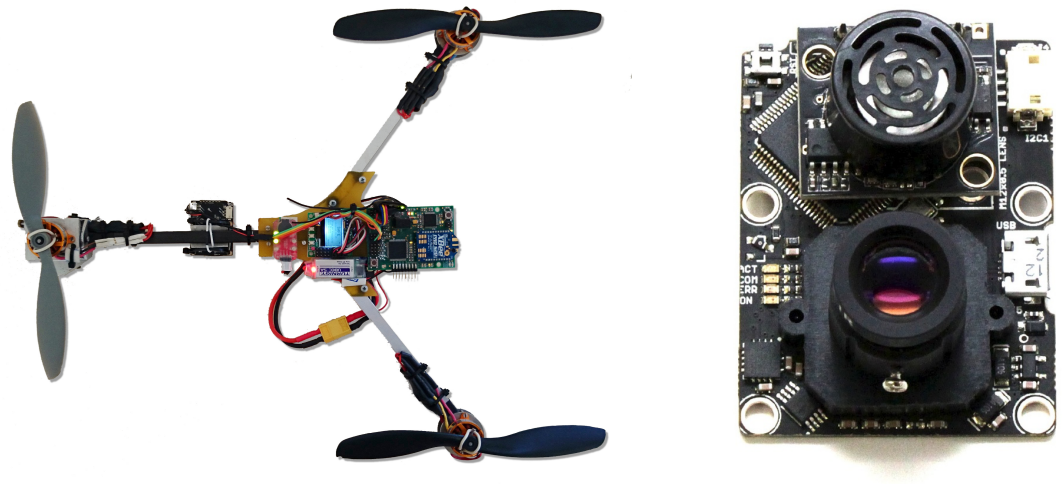
6.1 UAV platform

The UAV is a custom built tricopter (fig. 6.1a) with one propeller mounted on a tilting mechanism. It has a capability to pitch, roll and yaw just as any other multirotor UAV. The yaw control is supplied by the tilting mechanism while pitch and roll can be controlled by changing the ratio between rotational speed of all motors. All propellers are mounted directly on brushless motors, each one of them controlled by an individual ESC (electronic speed controller). The platform is capable of lifting payload of 150 g while its weight is 450 g. Its flight time is 7 minutes on average. Propellers are 5×3.8 inch in dimensions, mounted on motors by rubber bands to increase safety.

The aircraft is equipped with the *KK2* board that provides the basic stabilization of pitch and roll angles (θ, ψ) and yaw rate ($\dot{\phi}$). It is a low-cost ($\approx$ 30 USD) commercial product with open-source software. It utilizes 3-axis MEMS gyroscopes and accelerometers to estimate θ, ψ, $\dot{\phi}$ and allow the vehicle to be controlled as an RC model. It incorporates a set of nested PID controllers for both attitude axis. They can be easily tuned using built-in display and buttons. It can handle various types of multirotor aircraft including the tricopter. Another important module is the *px4flow* device, which is the only sensor used for localization of UAV 3D pose in space (see section 6.2 for more information).

6.2 px4flow sensor

The vehicle is localized in 3D space by the *px4flow* sensor [11, 17] (fig. 6.1b), developed and produced by PixHawk [30]. It encapsulates two sensors — a camera for computing an optical flow and an ultrasonic rangefinder for measuring a distance from the ground. It provides an information about its velocity relative to the ground computed by the correlation of two consecutive images from the camera (the same principle as employed in most computer mice). The velocity is internally compensated from rotational motion by built-in gyroscope and



(a) Tricopter aircraft. **(b)** The px4flow sensor.

Figure 6.1: Tricopter platform with px4flow optical flow sensor.

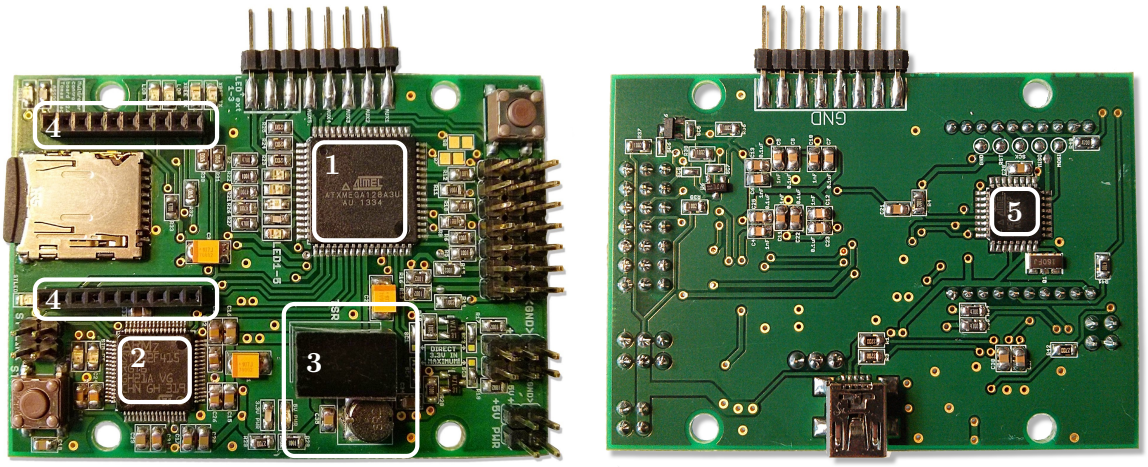
finally scaled to absolute values using the altitude measured by the ultrasonic sensor. The sensor is able to measure velocities up to 0.5 ms^{-1} when flying in 1 meter altitude in good light conditions. The altitude is measured from 0.3 m to 4 m. Data is sent in frequency around 70 Hz over UART (universal asynchronous receiver-transmitter) using MAVLink protocol [11].

6.3 Custom control board v.2

The control board v.2 (see fig. 6.2) is a significant improvement of the previous version [4] that comprised only of a single 8-bit Atmel microcontroller. After 1 year of using the old control board, designed within Bc. thesis of author of this thesis, in research of Department of Cybernetics, CTU in Prague, following requirements initiated a development of the second version. The platform should support variety of connections for external sensors and modules, mainly via UART and i^2c. It should support onboard data logging that is necessary for debugging and capturing data for system identification. Another requirement is a presence of a telemetry module. The UAV should be able to send short packets of data to another helicopter and to the ground station (laptop). The main motivation is to allow simple telemetry data being displayed on laptop while conducting an experiment. This should limit the number of unsuccessful experiments by offering a simple way to detect misbehaving sensors etc. Additionally, was required a tool for sending simple commands from PC to the UAV. And the last and most important demand was to support execution of the model predictive controller, being the goal of this thesis.

The board itself was built upon a standard (for UAVs) square mounting pattern (45 × 45 mm). It is designed in such way that allows mounting another board with dimensions 50×50 mm on its top while not obscuring connectors, buttons and radio antenna. The board

contains a 3.3 V, 1 A switching power supply that powers all its components. For an electrical schematic see Appendix C, for layouts of the printed circuit board see Appendix D. Following sections contain brief description of all key parts of the control board. Figure 6.4 shows a block diagram of all modules on the UAV.



(a) board's top **(b)** board's bottom

Figure 6.2: Custom control board v.2, key components are placed at follows: 1 – xMega, 2 – STM, 3 – switching power supply, 4 – socket for XBee, 5 – data logging MCU.

6.3.1 xMega main unit

The first of two used microcontrollers is 8-bit AVR, ATxMega128a3u. It was decided to distribute software tasks (MPC, Kalman filter and others) onto two separate units. The controllers and estimators are isolated in the co-processing ARM microcontroller, allowing potential students to develop on xMega without worrying about damaging the control system. Another reason is to reuse the low-level code (communication handling) to be reused from the previous version of the control board. The xMega MCU (microcontroller unit) is designated for handling all communication and other minor tasks. It is one of the most powerful MCUs in AVR 8-bit family with 32 MHz clock and 8 kB of SRAM memory. One of its greatest features are 7 separate UARTs. 3 of them are used for communication with other onboard parts, 4 of them are left free for connecting external devices. One is equipped with an optional level converter for connecting 5 V devices. Additionally, there are two i^{2}c lines and PPM[12] input/output for communicating with *KK2* and RC receiver (both with optional level converters). The software is developed using C programming language and uploaded to the microcontroller using a dedicated hardware programmer.

[12]Pulse position modulation is a communication protocol commonly used on UAVs and RC models.

6.3.2 ARM coprocessing unit

The second MCU onboard is a powerful 32-bit ARM device produced by STMicroelectronics — STM32F415RGT6. It is built upon ARM Cortex M4 with FPU (floating point unit) which allows a native work with floating point numbers. It has a powerful processor working on 168 MHz accompanied by 192 kB of RAM. This MCU is designated solely for computing Kalman filter and MPC. It is incorporated in such a way, that it only serves as an external coprocessor for xMega — there are no peripherals connected to it. There are several reasons for choosing the architecture where the more powerful MCU does not serve for all purposes although it has enough resources for it. The first one is the backwards compatibility with previously developed software. The xMega can execute it without many changes. The second one is the fact that it is more complicated to develop on the ARM STM, than the simpler AVR xMega MCU (The system as a whole is designed withing this thesis as an open platform for another students to develop and test their work.). Since the MPC and Kalman filter are important pieces of the program and their unwanted modification could make the machine dangerous, it was decided to conserve it on a separate MCU. The controller and KF are then used from the xMega MCU by a form of API. Important feature of STM is its floating point unit. Custom benchmarks has shown that it is capable of making $\approx 6 \times 10^6$ floating point operations per second which is a noticeable difference comparing it to the xMega's $\approx 3 \times 10^4$.

6.3.3 XBee telemetry module

When searching for a suitable wireless communication module, one cannot miss the family of XBee devices [43]. Built upon ZigBee standard, they can be set up to maintain one of several communication network topologies e.g. star or mesh. There are many different versions of XBee, based on its capabilities and frequency used. All of them support the same connection socket so they can be easily swapped for another type when necessary. Currently, we use XBee Pro S2B that works on 2.4 GHz ISM band. Practical tests shown that it is not well suited for any real-time critical data transfers since there is a significant delay ($\approx$ 150 ms) and its throughput is $\approx$ 20 kbit/s. The XBee module is used to transmit telemetry data to the ground station and allows communication between multiple UAVs. The communication protocol was developed by another student within his Bc. thesis [16].

6.3.4 OpenLog data logging module

Since we were not interested in creating our own data logger, we have integrated an existing, open source solution — OpenLog [13]. It has been designed to serve as an external SD card logging device connected by UART. Because its design is very simple, it was directly integrated into the control board. One can then set it up to receive a stream of data from the xMega MCU once the UAV is turned one. In our system, it handles logging 30 bytes with rate 70 Hz which is sufficient for debugging and system identification.

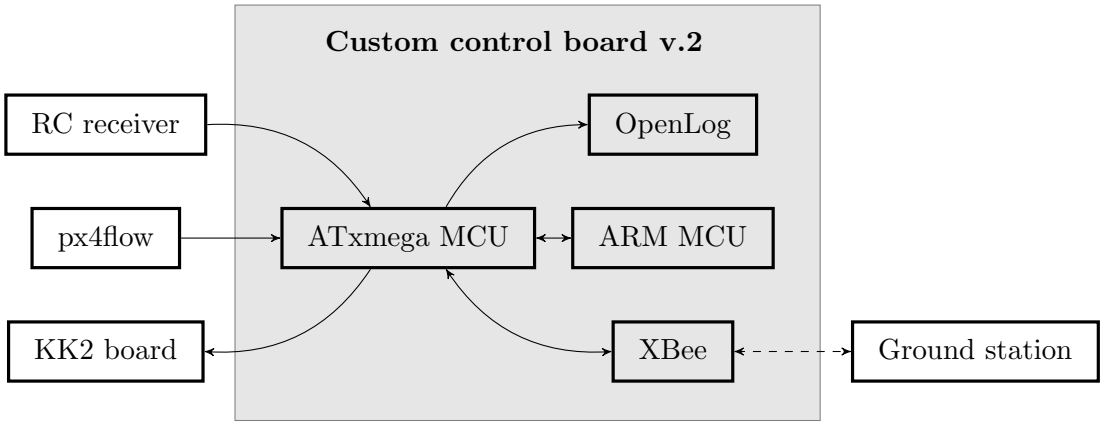


Figure 6.3: Block diagram of modules on the UAV.

6.4 FreeRTOS and tasks

A program for MCU such as xMega and STM can be created in two ways. The most straightforward one is to develop the bare application that will be directly executed on the processor while utilizing all of its computational resources. It is up to the creator of the program to manage concurrent processes, interact with hardware and supply fast communication responses in parallel with long-running calculations. In previous work, the control software was developed exactly in this way. Its benefit is obvious. The programmer has a complete control over the hardware, since only his code is executed on the CPU. But when the application gets complicated, an operating system can be used to take care of allocating computational resources for different parts of the program. There is a family of operating systems intended for real-time applications called Real-Time Operating Systems (RTOS).

The reader should not confuse the RTOS with the notion of operating system usually used on personal computers. RTOS are special software solutions developed with different criteria. The main one is its scheduling which ensures that each application is given its time slot in a deterministic and defined time. The scheduling usually works based on hardware timers and interrupts. Context switch in RTOS is usually very fast and it happens relatively often to allow real-time processing of incoming data. One of widely used RTOS is the FreeRTOS [41]. An Open Source solution with existing ports for both MCUs used in the control board. It offers a capability to create so called *tasks*, an analogy of processes in classical operating systems. Tasks are separate pieces of program with its own CPU context and memory stack.

The RTOS takes care of switching between tasks based on given priority allowing user to develop apparently multitask application. There are new issues coming up with this concept, which are the problems of synchronization and sharing resources. The FreeRTOS incorporates *queues* and *semaphores* which are supposed to be used for exchanging data between tasks and for their synchronization. We have incorporated the FreeRTOS on both MCUs creating separate tasks for handling communication and computations. See figure 6.3 for view on tasks

and the information flow between them.

Since our system requires execution of controllers and estimators in order of tens Hz and it has to handle communication in rate in order of hundreds Hz, the context switch rate was chosen in order of magnitude faster — 1000 Hz. But still some operations require even faster execution and reaction times (e.g. reception and creation of pulse-position signal or buffering raw data from peripherals). That is why a part of the control software remains executed directly on interrupt basis, independently of the FreeRTOS. Due to safety constraints, the system is able to hand over the UAV to the human operator even when the FreeRTOS and the control system do not work properly or even freezes. Both implementations on our MCUs require around 4 kB which leaves around 4 kB for our code.

6.5 Tasks on xMega MCU

The software on xMega consists of 4 different tasks running equally in parallel. *CommTask* is designated to handle incoming and outgoing communication. Since shared access to all communication media would be difficult to maintain, all other tasks communicate with peripherals via the CommTask. *ControllersTask* handles computations with regards to system controllers. *LogTask* is responsible for periodic data logging to OpenLog module. Finally, *MainTask*, despite its name, handles minor jobs as trajectory following and flight state automaton (modeling different flight modes of the UAV).

6.6 Tasks on STM MCU

Tasks on STM have much clearer designation. There is also *CommTask* serving as a communication mediator. The second one is *KalmanTask* which handles complete computation of the Kalman filter. Finally, *MPCTask*, which takes care of computing the model predictive controller.

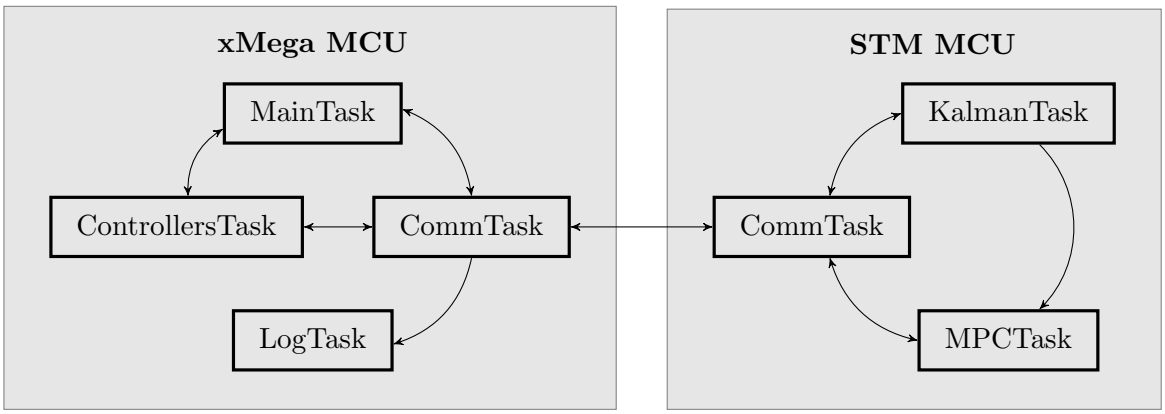


Figure 6.4: Block diagram of information flow between tasks of xMega and STM MCUs.

6.7 CMatrixLib - ANSI C matrix library

Implementation of the Kalman filter (section 4.3) and the model predictive controller (section 5) requires to deal mainly with vector and matrix operations. Although they can be implemented directly using the programming language, this way is impractical for code debugging and it discards the clarity of mathematical matrix notation. There is a large selection of matrix libraries [26] available for variety of programming languages. But since both software for STM and xMega are developed using ANSI C programming language and our platforms are specific (the architectures of these processors are specific, thus the library should be supplied by means of its source code) we decided to develop our own matrix library. Another reason that supports this decision is to maintain as much control over the executed code as possible, especially with regards to memory allocation, which is a critical issue for microcontrollers (and considering we are dealing with matrices and vectors with dimensions in order of hundreds). Our library supports basic matrix and vector operations with floating point values as well as basic algebra operators, such as matrix inversion and computation of determinant. It was developed with intention to minimize memory stack footprint by requiring prior memory allocation for subresults. There is also a possibility to create a matrix using a constant data located within the program memory. CMatrixLib, as we named it, is released on a community website GitHub [27] under GNU General Public License together with a proper documentation.

6.8 Summary

This section described the platform used for development, testing and evaluation of model predictive controller onboard the UAV. The aircraft uses a classical tricopter design utilizing built-in stabilization board. We proposed a custom control board for handling signal processing and controller computation onboard while supplying wireless telemetry and data logging. Two different microcontrollers were used to allow a decomposition of the cod. Computationally intensive tasks (Kalman filter and MPC) are located on 32-bit ARM coprocessor, while communication and other tasks are left on 8-bit xMega MCU. Both systems utilize FreeRTOS, Real-Time operating system. RTOS Tasks were presented with brief description of their purpose. In order to implement KF and MPC on these microcontrollers, a CMatrixLib matrix library was developed and published as an Open Source project on community website `https://github.com/klaxalk/CMatrixLib`.

7 Implementation aspects

Following section shows how the theoretical concept of Kalman filter and MPC (see sections 4.3 and 5) were transferred to the real hardware. Although its implementation is simple and straightforward if using e.g. Matlab, the transfer of the technology into embedded hardware brings new challenges. We propose practical refinements that allow execution of MPC on underpowered hardware of the UAV. We present an approach that exploits the structure of the dynamical system to introduce off-set free tracking with the original MPC formulation. Subsequently, we discuss how to take an advantage of objective function structure to optimize it while having a small memory footprint. Lastly, parameters of MPC are tuned to meet the performance requirements (namely computation rate). Furthermore, the simulation results are presented to allow comparison with experiments.

The final setup comprises of two separate MPC controllers driving both attitude axis. Since the PID altitude controller developed during previous work [42] is satisfactory for experiments with multiple UAVs, implementation of the MPC for the altitude system was postponed for future work. Because both attitude axis are controlled identically (due to decoupled system description), following chapters contain figures mostly for forward motion of the UAV. The yaw subsystem is left uncontrolled due to absence of data for estimating angle ϕ but considering its stability and the order of the dynamical system, appropriate PD controller could be designed to suite our needs.

7.1 Implementing Kalman filter

The KF, described in section 4.3, was firstly simulated in Matlab. In order to provide good estimate of all states, the measurement noise $\mathbf{Q}$ has to be identified and the process noise parameters in $\mathbf{R}$ tuned. Practically, setting of kalman filter is done by tuning the ratio between elements of $\mathbf{R}$ and $\mathbf{Q}$. For the system defined by (8), $\mathbf{R}$ and $\mathbf{Q}$ were empirically identified as follows:

$$\mathbf{R}_{x,y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{Q}_{x,y} = \begin{bmatrix} 150 \end{bmatrix}. \tag{39}$$

The remaining parameters for KF are $\mathbf{C}_{x,y} = [0, 1, 0]$ and LTI system matrices defined as (15). Tuning the filter is a very subjective process where one has to take into account several criteria, often contradictory. All states need to be estimated as accurate as possible while having them to converge as fast as possible (increasing values of $\mathbf{R}$). On the other hand, it is supposed to eliminate the measurement noise (increasing values of $\mathbf{Q}$) that brings a possibility of an incorrect estimate. The rule of thumb is to give a priority to convergence by finding a noise level, which is the highest tolerable level that allows smooth control. The corresponding ratio (39) was found roughly in simulation and lately fine-tuned with hardware.

The filter was firstly implemented solely on the xMega MCU. Because the MCU allowed its execution only in a rate around 50 Hz while dedicating all its resources, it was moved on the STM MCU, where the computation is done under 1 ms for both axis.

7.1.1 Estimating state disturbances

The Kalman filter can be also used for estimation of disturbances, if they can be modeled as states in the LTI system. In our case, these are uncontrollable and unmeasurable states, that have a link into the path between the system input and a measured state. Specifically, all disturbances in the system of the UAV can be combined into one force acting on its body. In practice, these are for example the wind disturbances and IMU calibration offset. The force can be expressed as an *parasitic* acceleration in the respective attitude axis.

Let us improve the attitude LTI system by adding two new states — $\ddot{x}_d$ representing the disturbance in acceleration and $\ddot{x}_u$ representing the contribution in acceleration from the control input. The state $\ddot{x}$ that was previously considered as the only acceleration now combines the two previously mentioned states. The diagram of the newly proposed system is depicted in figure 7.1. $u(s)$ is the Laplace image of the input signal $u(t)$, and $x(s)$, $\dot{x}(s)$, $\ddot{x}(s)$, $\ddot{x}_u(s)$, $\ddot{x}_d(s)$ are Laplace images of $x(t)$, $\dot{x}(t)$, $\ddot{x}(t)$, $\ddot{x}_u(t)$, $\ddot{x}_d(t)$ respectively. Furthermore, the disturbance can be estimated by the KF by setting the process noise matrix $\mathbf{R}$. If we set relatively high variances for states $\ddot{x}_u$, $\ddot{x}$ compared to state $\ddot{x}_d$ the filter will estimate $\ddot{x}_d$ by computing discrepancies between estimations from u and corrections from measured $\dot{x}$.

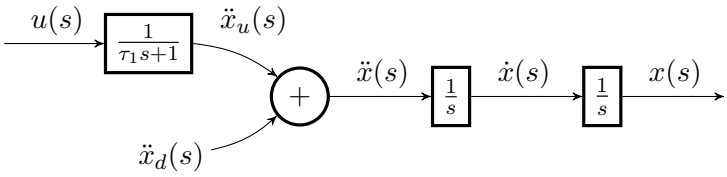


Figure 7.1: Diagram of the LTI system with the disturbance estimation.

Matrices for the new discrete attitude LTI system state as follows:

$$\mathbf{A}_{x,y} = \begin{bmatrix} 1 & 0.0114 & 0 & 0 & 0 \\ 0 & 1 & 0.0114 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0.9799 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{B}_{x,y} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 5.0719 \times 10^{-5} \\ 0 \end{bmatrix}, \tag{40}$$

where the state vectors are $\mathbf{q}_x = (x, \dot{x}, \ddot{x}, \ddot{x}_u, \ddot{x}_d)^T$ and $\mathbf{q}_y = (y, \dot{y}, \ddot{y}, \ddot{y}_u, \ddot{y}_d)^T$. This state representation was used in the final implementation and during experiments. The parameters of KF can be tuned that the settling time is fast enough to compensate wind disturbances.

The diagonal of the process noise matrix was set to $\text{diag}(\mathbf{R}) = (1, 1, 1, 1, 0.04)^T$ in the final implementation. Estimated disturbances can be further used in the control loop to eliminate control offset (see section 7.2.2). It can be also used for detecting abnormal flight conditions as a part of a failure detection system.

7.2 Implementing QMPC

Two approaches to model predictive control can be seen in literature. It can be either used for trajectory planning or for real-time control solely. Given the dynamical model, set of constraints and roughly defined (even infesible) desired trajectory, MPC optimization can produce a feasible result (satisfying constraints and the dynamical model) when minimizing a certain objective. In this form, MPC can be used as an offline trajectory planner [38]. On the other hand, if using the MPC for real-time control of the UAV, we suppose that the desired trajectory is already known and that is feasible. It has a significant impact on the particular formulation of our MPC. If the trajectory is feasible (meaning no state constraints are violated) and it initiates in the current state of the UAV, no state constraints are violated (and thus required) in the MPC formulation. Otherwise, the trajectory is infeasible. Based on these assumptions, we propose an implementation of *input constrained QMPC*. Since the feasibility of the desired trajectory may not be guaranteed or the trajectory may not start with the current state of the UAV, we propose a simple *input governor* (see section 7.2.1), which is a recommended technique [31] instead of implementing constraints.

Since optimizing the objective is easier if it is constrained only by input constraints, we can afford to optimize over a larger prediction horizon. It allows the UAV to track dynamic trajectories with greater precision since the controller can be more proactive with regards to future changes in the desired trajectory.

7.2.1 Input governor

In order to provide a trajectory tracking mechanism that does not violate state constraints, two requirements have to be considered. Firstly, the trajectory itself should satisfy the system dynamics and imposed constraints and secondly the trajectory should start with the current state of the UAV. The *input governor* is a system that modifies the desired trajectory in such a way, that it satisfies these requirements. This modification does not need to be optimal since our aim is not to produce optimal desired trajectories but to find optimal control actions to drive the system through known trajectories. The governor produces a feasible trajectory from the current state of the UAV. The only constraint we impose is a maximum speed of the UAV, since the *px4flow* sensor effectively measures speed only up to 0.35 ms^{-1}. Our input governor limits the rate of change of the desired position trajectory by that value while ensuring the trajectory initiates in the current state of the UAV.

7.2.2 Offset-free tracking

The MPC formulation as stated in section 5.2 does not provide offset-free tracking. In the case of external disturbances or imperfect behavior of the integrated stabilization system, this would lead to steady state control error. Several techniques exist to solve the problem. The classical one is a *delta input formulation of MPC* [6]. Morover, an additional integral feedback loop could be created around the MPC controller. But since our estimator is able to observe the disturbance within the system, the classical formulation of MPC is in fact able to control the system without a steady state error [31]. Moreover it has some other useful properties, e.g. no windup issues (like with classical integral feedback) and the controller (considering its separation from the estimator) can be completely stateless.

With this approach, the controller is able to compensate not only small and steady disturbances, but it creates adequate actions even for sudden and momentary disturbances. See section 8.5.2 for results of the experiments.

7.2.3 Controller parameters

The controller has several parameters whose settings have an impact on its performance. Some of them are tied up by execution rate of the MPC, namely the number of decision variables and the length of the prediction horizon. After an experiment, we have converged to horizon length of 200 steps (2.2 s) with 20 decision variables in the objective. They are distributed in an exponential way over the horizon. First 10 variables directly correspond to first 10 control actions. Another 9 variables cover control actions evenly up to the 100th one. The last variable sets the control action for the last 100 steps. See figure 7.2 for an example of the optimized control action.

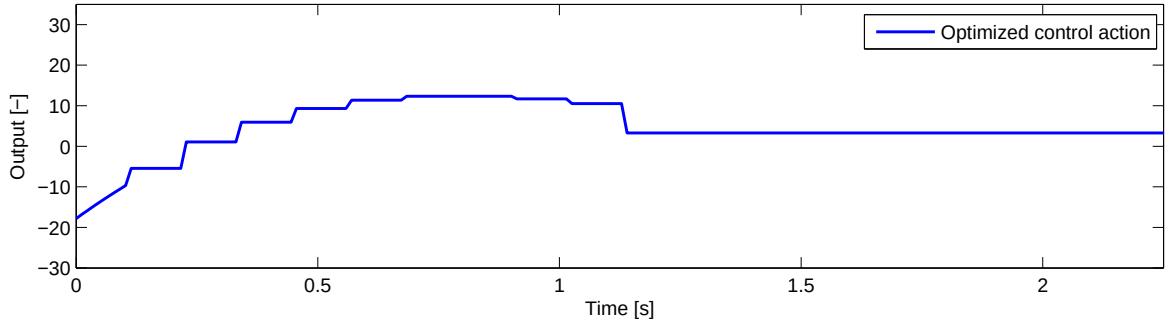


Figure 7.2: Control action produced by 20 decision variables and stretched to 2.2 s prediction horizon.

Our system presumes desired trajectories in the form of position states only. There is no need to require the information about the desired velocity of the UAV, unless its speed is the only state that should be controlled without demanding the control of UAV's position. One can specify not to penalize certain states by setting respective elements of matrices $\mathbf{Q}$,

Horizon length	200 steps, 2.2 s
Number of variables	20
Move blocking	exponential-like
Constraints	input constraints
MPC rate	30 Hz

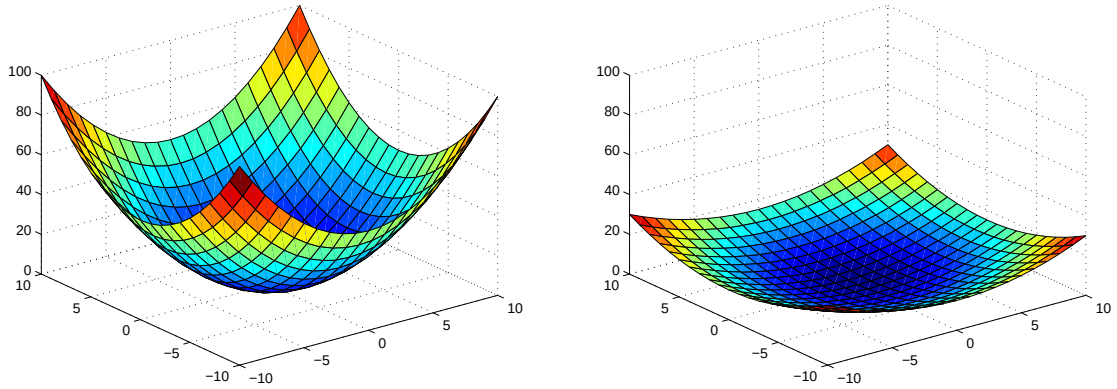
Table 7.1: Parameters of our MPC implementation.

$\mathbf{S}$ and $\mathbf{P}$ (consequently merged to $\hat{\mathbf{Q}}$, $\hat{\mathbf{P}}$) to zero. By doing that, the control error of the state does not take part in the objective. This leaves us basically three parameters to set in order to tune the performance of the controller, supposing the desired trajectory consists of only position state. These parameters are weights for penalizing position errors and control actions as defined in section 5.2, specifically tuned as follows

$$\mathbf{Q} = \begin{bmatrix} 1.07 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{S} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P} = \begin{bmatrix} 0.0000025 \end{bmatrix} \tag{41}$$

Tuning of MPC controller is a different process than tuning of PID controller, which was used in the previous system for UAV control by Department of Cybernetic of CTU. Parameters of the MPC controller (diagonals of matrices in (41)) have a meaningful interpretation since they correspond to a particular state, and the effect of changing the parameters can be observed on the states. A common rule of thumb is to firstly normalize the diagonal of $\mathbf{Q}$ according to a *'usual'* range of state values (e.g. if the range of the velocity state is 0.1 ms^{-1} then the corresponding element of $\mathbf{Q}$ is set to $1/0.1$). In the beginning, this helps to normalize the penalties, since each state can hypothetically have different typical range of values. Furthermore the parameter $\mathbf{P}$ needs to be set in such way that the system produces control actions that do not violate input constraints when started from an ordinary initial condition. On can set $\mathbf{P}$ in such a way that the actions lie within a desired range most of the time. When increasing the input penalization, generated control actions tend to be smaller, which means that the system is more conservative with actions. Finally, one can tune elements $\mathbf{Q}$ so the states are estimated properly.

There is an important phenomenon that needs to be taken into account. The matrix $\hat{\mathbf{H}}$ (defined in (30)) determines the shape of the quadratic form in the objective. A condition number of $\hat{\mathbf{H}}$ determines whether the optimum of the objective is sharply defined or whether it lies within a wide plateau. See figure 7.3 for an illustrative example of 2-dimensional quadratic form. In the case of ill-conditioned quadratic form, there are many solutions with the objective value near the optimum, which can be interpreted as there are many different sequences of control actions that can lead to similar outcomes. We should avoid constructing such function, because locating its minimum may lead to numerical difficulties during calculations.



(a) well-conditioned quadratic form **(b)** ill-conditioned quadratic form

Figure 7.3: Illustrative example of 2-dimensional quadratic form and its.

Optimizing it using a closed-form as described in section 5.5 can be affected by numerical instabilities when calculating the matrix inversion $\hat{\mathbf{H}}^{-1}$. Also, iterative methods tend to suffer from slow convergence on such plateaus [9]. Fortunately, it has a simple solution. We can regularize the matrix $\hat{\mathbf{H}}$ by increasing penalization $\hat{\mathbf{P}}$ since it directly increases values on its diagonal and thus increases its eigenvalues (supposing $\hat{\mathbf{P}} \succ 0$). The interpretation of this process is simple. We limit the freedom of input actions by regularizing $\hat{\mathbf{H}}$, which leads to better numerical stability of computation.

7.2.4 Optimizing onboard

When implementing the *input constrained MPC* into embedded hardware, the structure of the problem can be exploited in following way. Since matrix $\hat{\mathbf{H}}$ does not depend on the desired trajectory nor the initial condition, it can be precomputed offline. This also holds for its inversion $\hat{\mathbf{H}}^{-1}$ and matrices $\hat{\mathbf{B}}$ and $\hat{\mathbf{A}}$. Thanks to that we can store them in a ROM (read-only memory, designated for the program), which supports execution even on a microcontrollers with a small amount of RAM.

The workflow of testing and tuning different parameters of the controller was built upon a simulation in Matlab. One part of the simulation is a script that can generate a `C` code, with all matrices precomputed, that can be easily inserted into sources for the STM microcontroller. This allowed relatively easy tuning and testing cycle.

7.3 Summary

This chapter described some aspects of implementation of the Kalman filter and MPC controller. Particular settings of Kalman filter has been presented and followed by the exten-

sion of the LTI system to allow estimating state disturbances. Then, we discussed the two fundamental approaches to use MPC as a trajectory planner and as a low-level controller. The input governor system is presented to supply the controller with a feasible trajectory followed by particular settings of our MPC implementation. Finally, we discussed a memory allocation of MCP matrices which is necessary for final implementation.

8 Experiments

This section presents an evaluation of the proposed control system. Firstly, simulation results are shown to be subsequently compared real-world experiments. We tested the system in various situations including tracking constant reference, step response and sine trajectory. Another experiments were conducted to verify the disturbance rejection feature. This section also includes observations on drift of the position estimator with an absolute localization system. Finally, the system's performance is compared to the previous work. Most of the experiments are captured in the compilation video `http://youtu.be/lPy7w-GUbw4`, which is also located on the enclosed CD.

8.1 Simulating MPC

Figure 8.1 shows the step response of the system, simulated without the input governor. The proaction can be seen before the step in the reference trajectory, which is enabled due to the predictive nature of the controller.

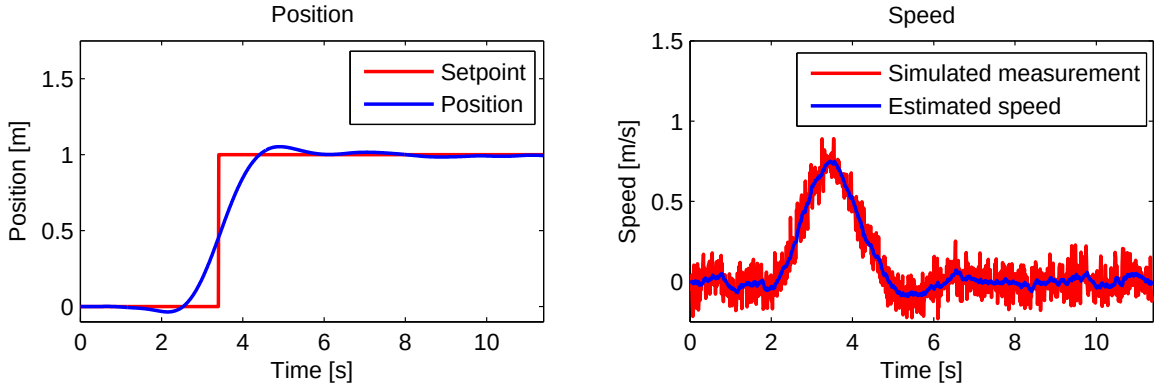


Figure 8.1: Simulating position step response without the input governor (forward motion).

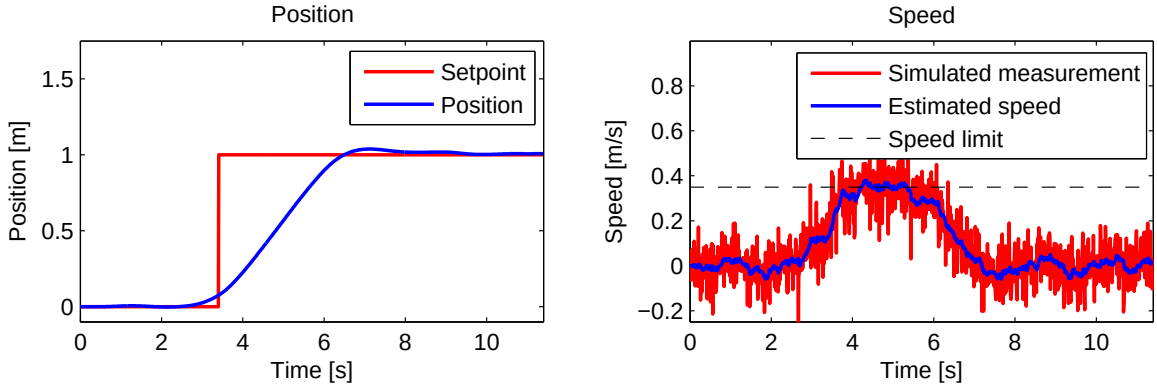


Figure 8.2: Simulating position step response with the input governor (forward motion).

Since the desired *unit step* trajectory is not feasible, it should be firstly transformed using the input governor. In our case, we limit the UAV's speed to 0.35 ms^{-1} (which is due to maximum speed that the *px4flow* sensor can measure reliably). Figure 8.2 shows the simulation of step response with the input governor. See that the speed lies roughly under the limit. Figure 8.3 shows the simulation of UAV tracking sine trajectory. Prior work [4, 42] and related work [5] demonstrated that tracking such trajectory is difficult without a notable lag. Our simulation shows that MPC with long enough prediction horizon achieves better results. See section 8.4 for experimental validation of sine trajectory.

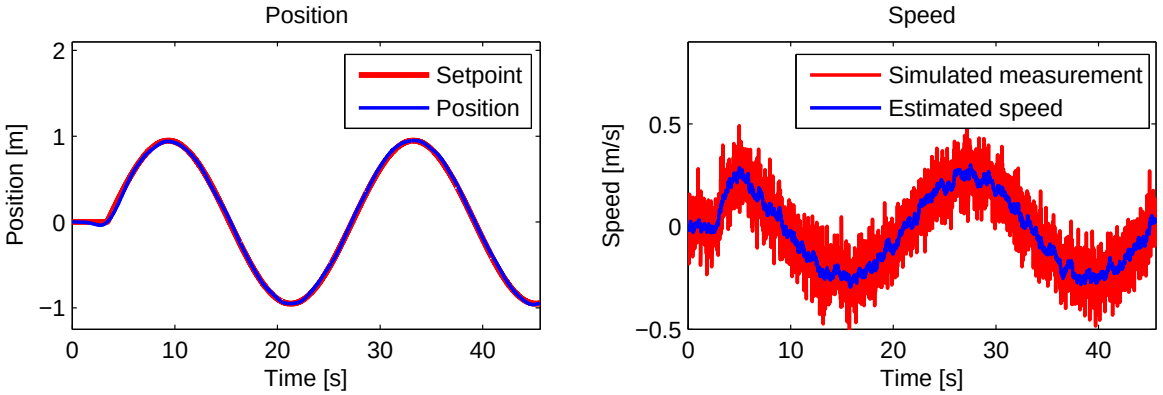


Figure 8.3: Simulating tracking of feasible sine trajectory (forward motion).

The disturbance rejection ability based on the disturbance estimation is one of key points of the system. It should be able to deal with constant disturbances (cause by bad trimming, offset in *KK2* stabilization) as well as momentary disturbances (possibly caused by wind). Although simulations showed that the disturbance estimation can be tuned arbitrarily to match a desired settling time of the estimate, in practice, there is a limit (see discussion in section 8.5.1). Figure 8.4 shows a simulation of disturbance rejection with parameters tuned using the real UAV.

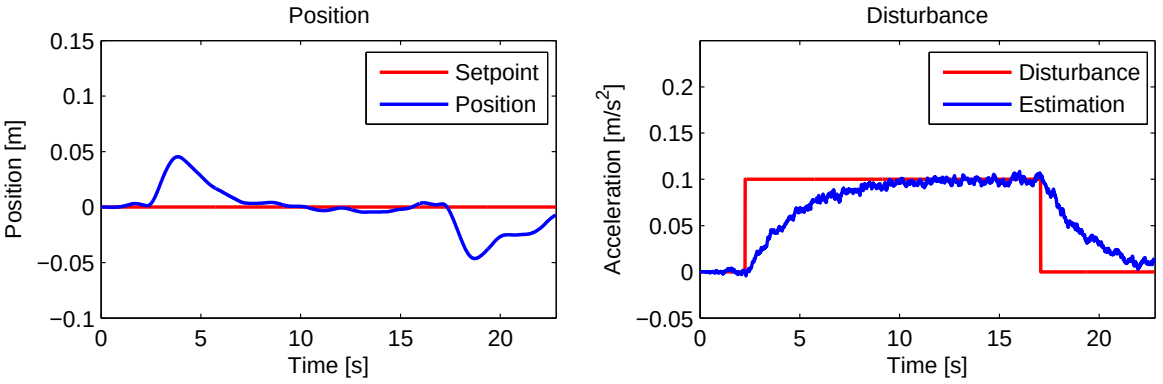


Figure 8.4: Simulation of disturbance rejection (forward motion).

8.2 Tracking constant setpoint

The first experiment was realized with the aim to test the UAV capability of tracking a constant reference, i.e. hovering above one place. Statistically, we are interested in the standard deviation σ and the maximum deviation Δ_{max}. Figure 8.5 shows its performance in good light conditions with no wind disturbances. The aircraft was capable of tracking the reference with standard deviation $\sigma = 4.8$ cm and maximum deviation $\Delta_{max} = 14.4$ cm.

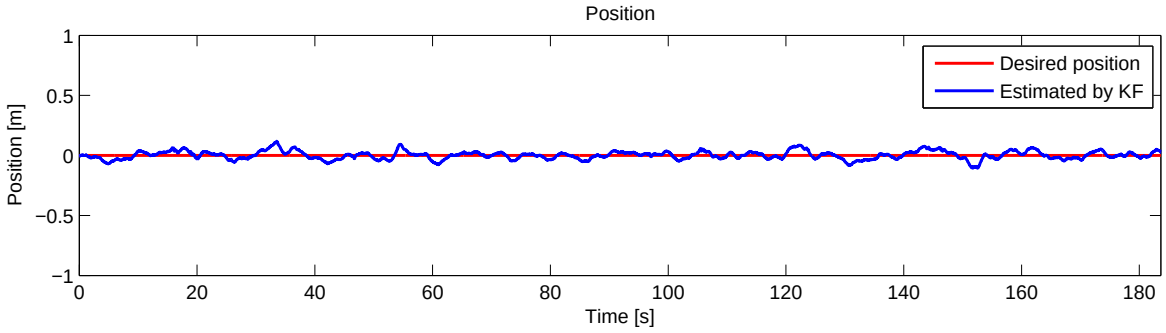


Figure 8.5: Experiment of tracking static trajectory (forward motion).

8.3 Measuring of estimation drift

One could ask, what is the relevance of previously presented data, since the position is estimated onboard using only velocity data and the model. The truth is that the position is not absolute and although the controller seems to stay around the setpoint, the absolute position in the space may drift away. In order to measure such drift we implemented the Whycon camera localization system [20, 14] using calibrated camera was employed. It was used to measure the absolute position of the UAV while conduction a flight. The figure 8.6 shows the position drift during 1 minute flight. The maximum measured deviation was ≈ 10 cm. It can be seen, that the estimated position slowly drifts away from the absolute one, and then lately drifts back. Similar effect can be observed in figure 8.7, where the UAV is tracking a sine trajectory. In general, the drift is larger when the UAV is moving closer to the saturation of the *px4flow* sensor.

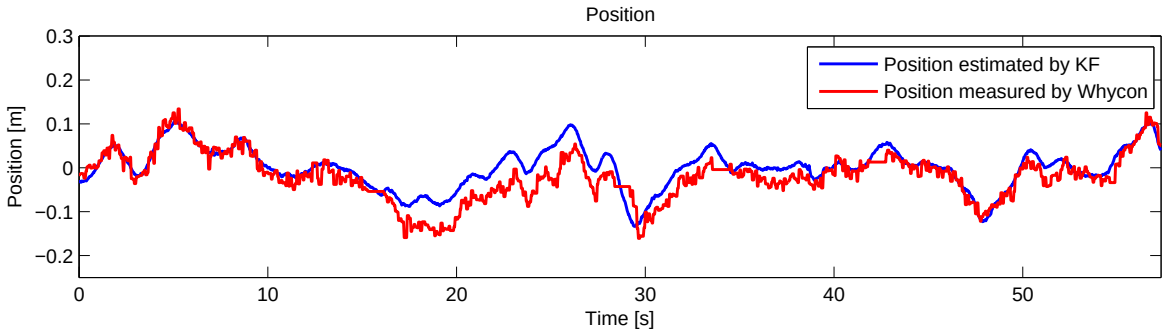


Figure 8.6: Experiment of measuring a position drift (forward motion).

The signal noise from the *px4flow* sensor is usually larger at higher velocities. At last, there are other conditions that need to be met to eliminate the position drift — good light conditions and good vibration isolation of the sensor. Otherwise the drift is ≈ 10 cm/min or larger. Our observations correspond with results presented by authors of the system in [17].

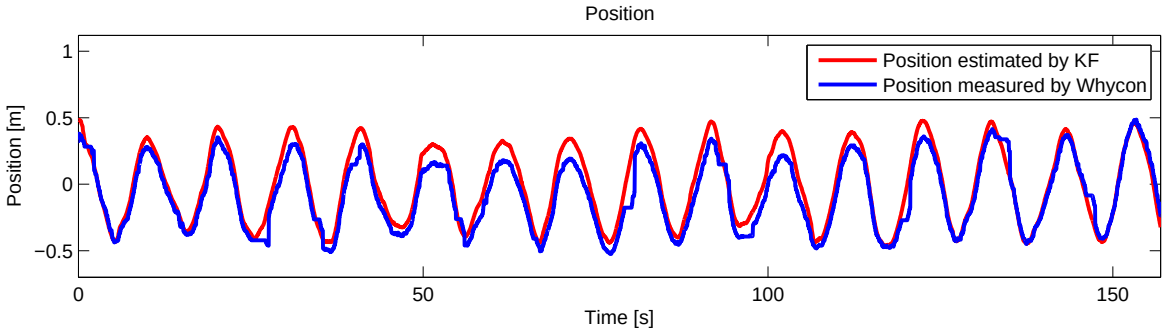


Figure 8.7: Experiment conducted to measure a position drift while tracking a sine trajectory (forward motion).

8.4 Tracking dynamic trajectory

Another experiment tested the system capability of tracking dynamic trajectories. In the first experiment, response of the UAV to a *unit step* response was tested. As it can be seen in figure 8.8, the UAV does not overshoot the setpoint, but it rather makes a proaction which is a typical characteristic of the model predictive controller. The speed lies within the set boundaries due to input governor. Figure 8.9 shows the performance of the system during tracking a circular trajectory. The trajectory was precomputed with constant speed 0.25 ms^{-1}. Figure shows position and speed for both attitude axis. The experiment can bee seen in the compilation video which can be found at `http://youtu.be/lPy7w-GUbw4` or in the enclosed CD (`video1.mp4`). The maximum deviation from the desired trajectory (forward motion) was $\Delta_{max} = 16.3$ cm and the standard deviation $\sigma = 4.3$ cm.

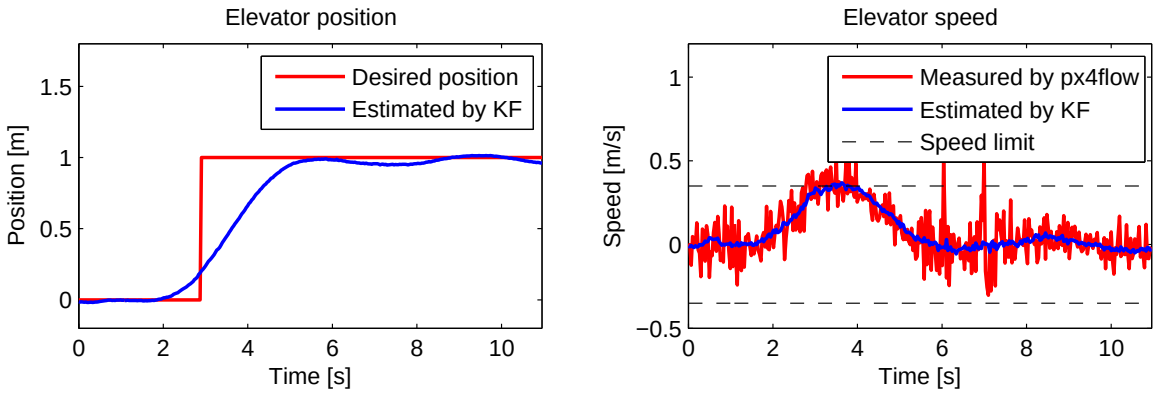


Figure 8.8: Experiment of tracking the *unit step* response.

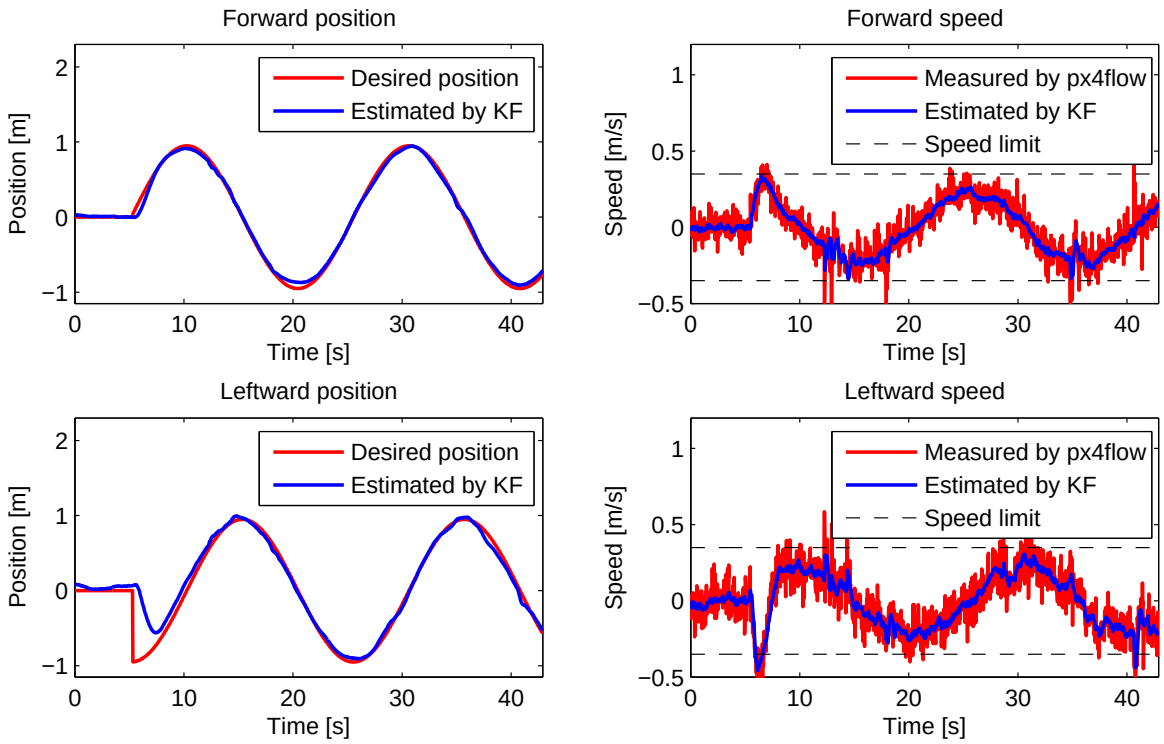


Figure 8.9: Experiment of tracking circular trajectory. Amplitude 0.95 m, period 20 s, speed 0.25 ms^{-1}.

8.5 Disturbance rejection

Different set of experiments was conducted to test the disturbance rejection capability. Since we dealt with the real hardware where the dynamical model is only an approximation of the real system, we cannot suppose the same performance during the experiments as it was observer in simulations. The difference was mostly observed during tuning of the distur-

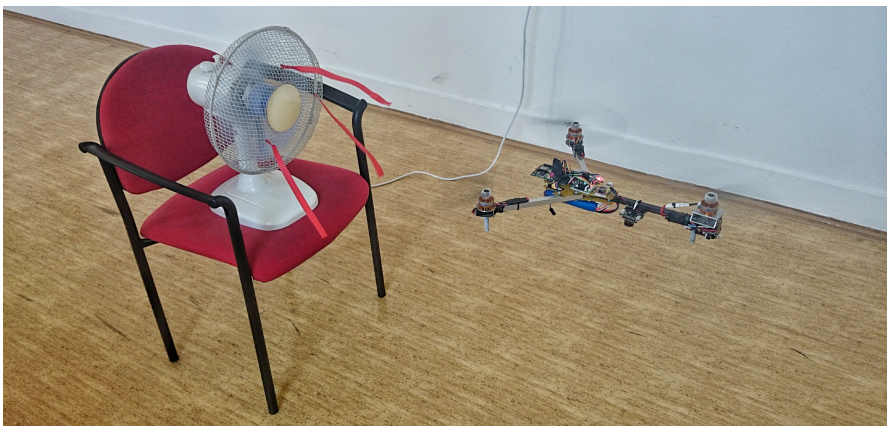


Figure 8.10: Experimental setup with 40 W fan to test the disturbance rejection feature.

bance estimator. By decreasing the settling time of the estimated disturbance, the system becomes unstable due to oscillations being induced into the estimate which led to worse performance. We found sufficient parameters that allows to estimate usual disturbances, such as bad trimming or stabilization offset, in order of seconds.

8.5.1 Persistent wind disturbances

The UAV was tested for its capability to reject wind disturbances by using a 40 W fan pointed to the UAV (see figure 8.10). To put it into context, it is the same power as one of UAV motors. The figure 8.11 shows the forward motion of the aircraft. The grey area indicates when the fan was turned on. The estimated disturbance settled in ≈ 10 s which allowed the UAV to eliminate the resulting *steady state* error. Notice the non-zero disturbance when the fan is turned off, which is caused by already mentioned trimming and stabilization offset.

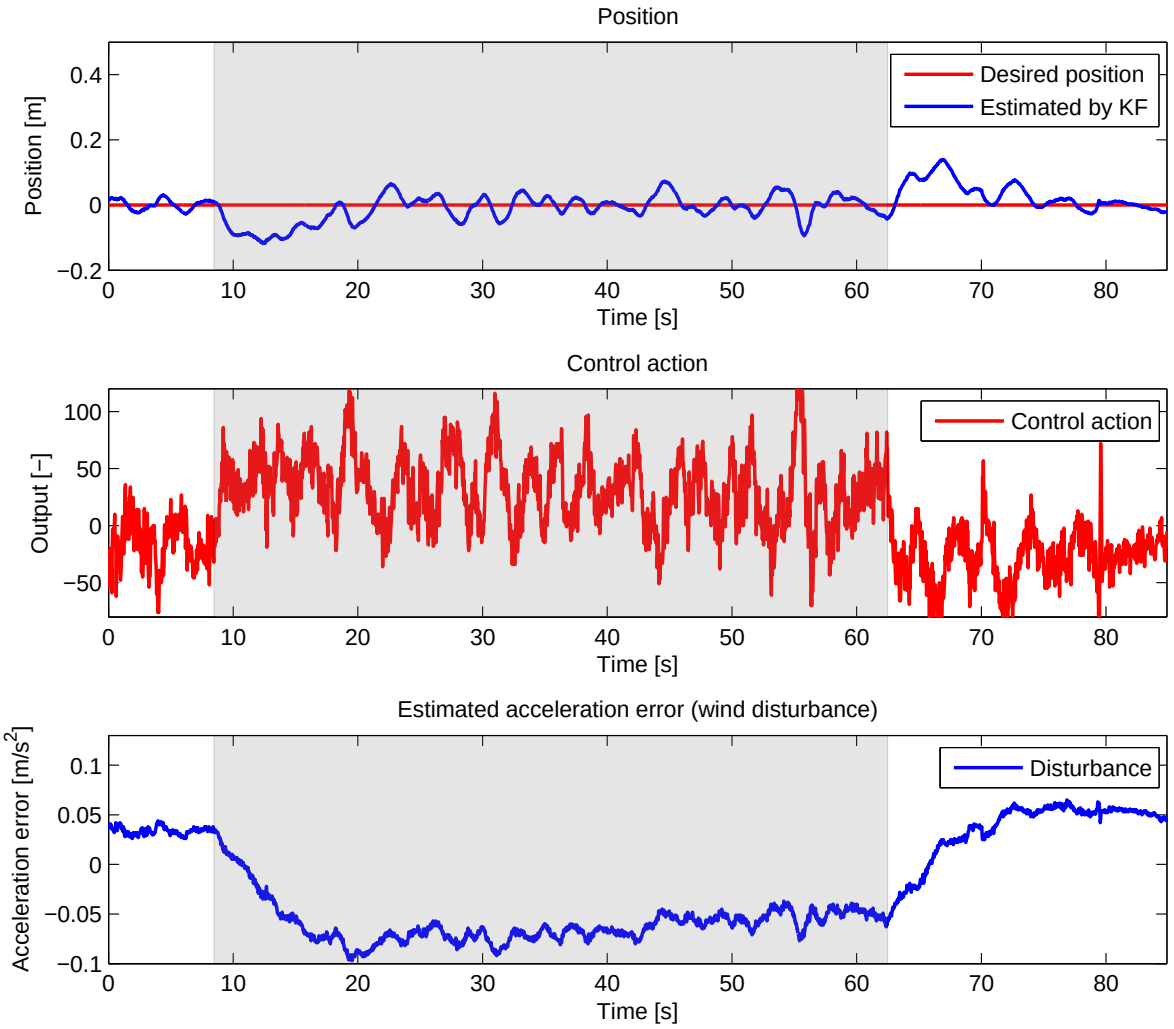


Figure 8.11: Experiment of tracking constant setpoint while being under the influence of wind.

8.5.2 Momentary disturbances

Following experiment, shown in the compilation video, aimed to test the system's capability of rejecting momentary disturbances. Figure 8.12 shows a forward motion of the UAV, which is repeatedly dragged by hand. These disturbances were short and sudden, and their estimate has relatively short raise time, comparing to those in the previous experiment. Gray areas denote the moments when the UAV was dragged away. One can spot overshoots in the position signal. They can be explained by the remaining disturbance in the estimate. The controller accounted for a disturbance, that was not actually effecting the UAV at the moment (due to the settling time of the estimate). See Appendix E for additional data from experiments with disturbances.

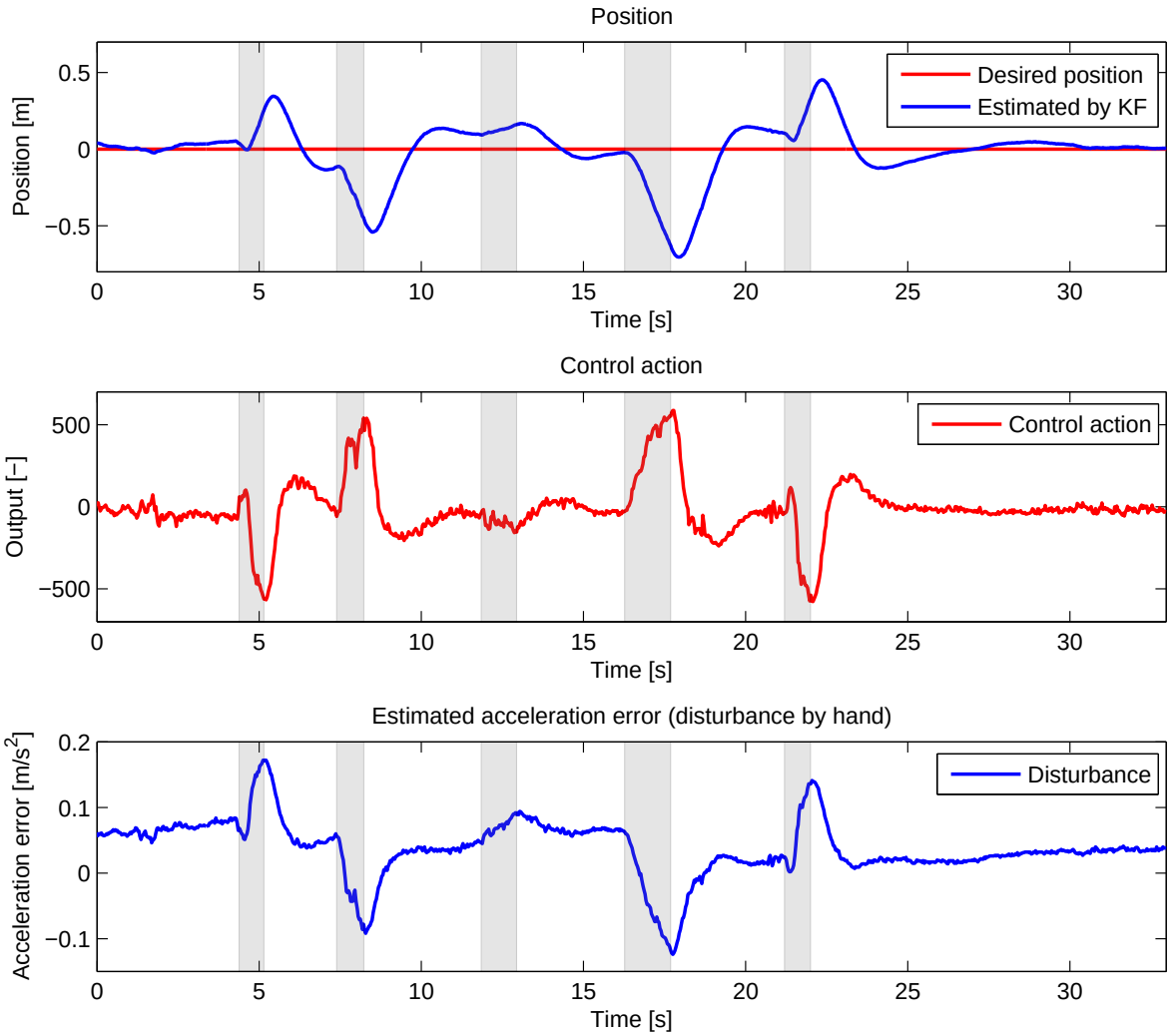
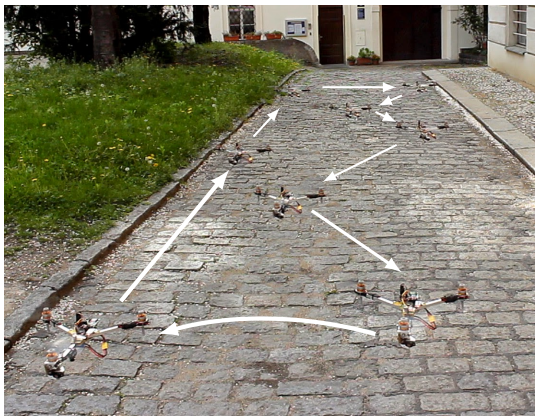


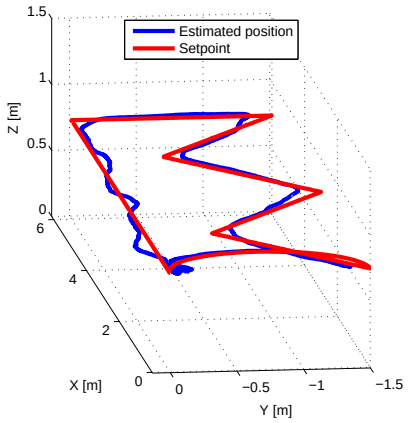
Figure 8.12: Experiment of tracking constant setpoint while being under the influence of momentary disturbances. The UAV was dragged by hand which is indicated by the gray areas.

8.6 Outdoor experiment — longer trajectory

The last experiment was conducted to present the UAV capability of autonomous operation not only in indoor (laboratory) conditions, but also in a general outdoor area. The UAV was supposed to track a precomputed trajectory with 1 minute duration. Figure 8.13a shows a photo that illustrates the course of the experiment[13] while figure 8.13b shows the plot of onboard data including desired and estimated position. The experiment was conducted under the influence of mild and steady breeze. The video from the experiment can be found on url `http://youtu.be/iqgL1H_DCmU` or on the CD (`video2.mp4`).



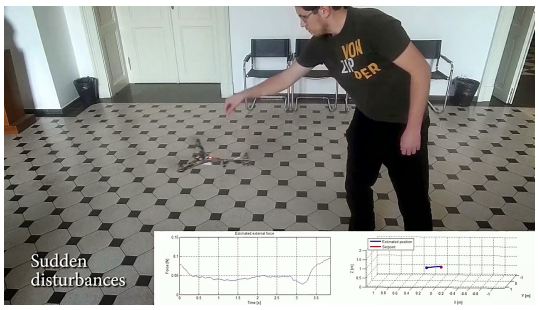
(a) image stitched from the video



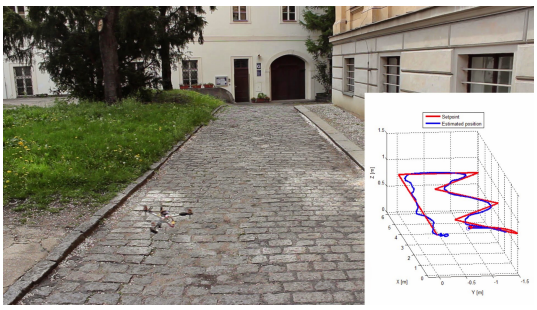
(b) visualization of logged data

Figure 8.13: Experiment with trajectory conducted outdoors. For detailed data see Appendix E.

Most of previously presented situations can be seen in the compilation video. It is accompanied by data logged onboard the UAV, that are rendered in the video (see figure 8.14a). The estimated position is also fitted with the estimated force acting on the UAV. The outdoor video is also accompanied with the position plot (figure 8.14b).



(a) `video1`: `http://youtu.be/lPy7w-GUbw4`



(b) `video2`: `http://youtu.be/iqgL1H_DCmU`

Figure 8.14: Two videos with onboard data rendered — a) compilation video, b) long trajectory video.

[13] The photo was created by stitching multiple images from the video.

8.7 Performance and comparison

Results presented in this section were gathered from three experiments — tracking constant trajectory (section 8.2), tracking circular trajectory (section 8.4), and the last experiment of tracking a longer trajectory in outdoor environment (see the previous section). The first two experiments were performed indoor and all of them were conducted under good lighting conditions. Table 8.1 shows the statistical evaluation of trajectory tracking where σ_x, $\Delta_{x,max}$ and σ_y, $\Delta_{y,max}$ denote the standard and maximum deviation for the lateral and forward axis respectively, and l indicates the time duration of the experiment. One can see that the performance of the system in the indoor environment is slightly better than in the outdoor environment, but overall the results are better when comparing to the previous work [42]. Although the performance of tracking constant trajectory is similar, the results of tracking dynamical trajectory are in order of magnitude better in terms of the standard deviation. According to results of thesis [42] it rarely went under 60 cm.

	σ_x [cm]	$\Delta_{x,max}$ [cm]	σ_y [cm]	$\Delta_{y,max}$ [cm]	l [s]
constant trajectory	4.0	15.1	3.3	11.8	184
circular trajectory	4.3	14.9	3.9	8.5	92
outdoor experiment	8.7	23.5	7.0	21.6	55

Table 8.1: Performance of the proposed solution.

The result of the proposed system are similar to results of previously used PID controller, if tracking a constant setpoint. If tracking a dynamical trajectory, the MPC outperforms the PID controller significantly.

8.8 Summary

The last section of this thesis demonstrated the performance of the system in practice. The implemented Kalman filter was able to estimate external disturbances, which significantly improved performance of the system during offset-free tracking. The UAV was able to effectively counteract wind disturbances. Moreover, it was able to follow a desired trajectory with precision suitable for use in GPS-denied and indoor environment. Statistically, the results were significantly better than in the previous work.

9 Conclusion

In this thesis, we have developed a hardware and software solution that allows execution of the model predictive controller onboard of micro aerial vehicles. The dynamical model of the helicopter has been derived and its parameters have been numerically and experimentally identified. We have designed a system including the Kalman filter as the state estimator and the quadratic MPC which optimizes control actions over the prediction horizon of 2.2 s. The proposed system has been successfully implemented into the embedded hardware. The controller has been verified by numerical simulations and tested in various experiments. Many experiments have been conducted both indoors and outdoors to test different scenarios including tracking various trajectories and disturbance rejection. The entire assignment of this thesis has been fulfilled successfully. According to the assignment, following tasks have been completed:

- The dynamical system of the UAV has been analyzed and its model was constructed.
- A Kalman filter has been implemented to estimate states and disturbances.
- A model predictive controller has been derived and implemented on the experimental micro aerial vehicle.
- The experimental aircraft has been constructed including the custom control board, which has been designed and manufactured for the purpose of this thesis.
- Experiments have been conducted that verified the capabilities of the solution to follow dynamical trajectories in indoor and outdoor environments.

This work also created opportunities for other students to work on their theses [18, 16]. The first work aimed to control a group of UAVs synchronously using the XBee modules, in the second one, a user interface for the ground station was developed and a failure detection system based on the proposed estimator was designed. The platform proposed in this thesis will be further used for a research of UAV swarms and formations within the Multi-Robot System group of FEE CTU.

Finally, let us mention some contributions of the presented work beyond the assignment of this thesis and related publications, in which author of this thesis contributed as a co-author. The experimental platform was used during the research of a visual feature tracking system [12]. The results with multirobotic formations have been published in [39]. The last results dealing with application of the proposed system in multi-robot scenarios have been submitted to Autonomous Robots journal [22].

Relevant information and results in the field of MAV control and multi-robot systems in general, which were achieved by other members of Multi-robot Systems group, can be found in [35, 37, 32, 33, 36, 34].

9.1 Future work

During the development of this thesis, several ideas and needs emerged that specify our future work. Since the main bottleneck of the system is its dependence on sensor data, additional onboard sensors should be mounted to increase the precision and robustness of the estimated UAV position. Barometer and magnetometer could be added to the custom control board and their data fused by the Kalman filter. The IMU, which is already present in the stabilization board, could be also used for the position estimation.

Regarding controllers, the MPC shall be implemented to control also the altitude, although there is not such need for precise trajectory tracking. Furthermore, an additional MPC could be added to allow optimal onboard trajectory planning. It would require to solve an optimization problem with more complex constraints.

10 Bibliography

[1] Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Robust explicit model predictive flight control of unmanned rotorcrafts: Design and experimental evaluation. In *Control Conference (ECC), 2014 European*, pages 498–503. IEEE, 2014.

[2] Ardupilot. http://ardupilot.com/, February 2015.

[3] Federico Augugliaro, Ammar Mirjan, Fabio Gramazio, Matthias Kohler, and Raffaello D'Andrea. Building tensile structures with flying machines. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3487–3492. IEEE, 2013.

[4] Tomáš Báča. Control of relatively localized unmanned helicopters. Bachelor's thesis, České vysoké učení technické v Praze, 2013.

[5] Moses Bangura, Robert Mahony, et al. Real-time model predictive control for quadrotors. *19th World Congress, The International Federation of Automatic Control*, 2014.

[6] Francesco Borrelli and Manfred Morari. Offset free model predictive control. In *Proc. IEEE Conference on Decision and Control*, pages 1245–1250, 2007.

[7] Patrick Bouffard. On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments. Technical report, DTIC Document, 2012.

[8] Patrick Bouffard, Anil Aswani, and Claire Tomlin. Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 279–284. IEEE, 2012.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[10] Dario Brescianini, Markus Hehn, and Raffaello D'Andrea. Quadrocopter pole acrobatics. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3472–3479. IEEE, 2013.

[11] PX4FLOW Smart Camera. https://pixhawk.org/modules/px4flow, April 2015.

[12] J. Chudoba, M. Saska, T. Baca, and L. Preucil. Localization and stabilization of micro aerial vehicles based on visual features tracking. In *Proceedings of 2014 2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, volume 1, pages 611–616, Danvers, 2014. IEEE Computer society.

[13] Open Source Hardware Datalogger. https://github.com/sparkfun/OpenLog, April 2015.

[14] J. Faigl, T. Krajník, J. Chudoba, L. Přeučil, and M. Saska. Low-Cost Embedded System for Relative Localization in Robotic Swarms. In *ICRA2013: Proceedings of 2013 IEEE International Conference on Robotics and Automation*, pages 985–990, Piscataway, 2013. IEEE.

[15] Ramsey Faragher et al. Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE Signal processing magazine*, 29(5):128–132, 2012.

[16] Jiří Fiedler. Synchronized control of group of helicopters using direct communication. Bachelor's thesis, České vysoké učení technické v Praze, expected in June 2015.

[17] Dominik Honegger, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1736–1741. IEEE, 2013.

[18] Martin Klučka. User interface and failure detection for support of multi-mav experiments. Bachelor's thesis, České vysoké učení technické v Praze, expected in June 2015.

[19] T. Krajnik, M. Nitsche, S. Pedre, L. Preucil, and M. Mejail. A Simple Visual Navigation System for an UAV. In *International Multi-Conference on Systems, Signals and Devices*, page 34, Piscataway, 2012. IEEE.

[20] Tomas Krajnik, Matias Nitsche, Jan Faigl, Petr Vanek, Martin Saska, Libor Preucil, Tom Duckett, and Marta Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3-4):539–562, 2014.

[21] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 2011.

[22] J. Chudoba L. Preucil T. Krajnik J. Faigl J. Thomas G. Loianno V. Kumar M. Saska, T. Baca. System for stabilization of micro aerial vehicle swarms using onboard visual relative localization. *Submitted to Autonomous Robots journal*, 2015.

[23] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics & Automation Magazine*, (19):20–32, 2012.

[24] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *Robotics & Automation Magazine, IEEE*, 17(3):56–65, 2010.

[25] Mark W Mueller and Raffaello D'Andrea. A model predictive controller for quadrocopter state interception. In *Control Conference (ECC), 2013 European*, pages 1383–1389. IEEE, 2013.

[26] List of numerical libraries. http://en.wikipedia.org/wiki/List_of_numerical_libraries, April 2015.

[27] CMatrixLib on Github.com. https://github.com/klaxalk/CMatrixLib, April 2015.

[28] Mikuláš Ondřej. Quadratic programming algorithms for fast model-based predictive control. Bachelor's thesis, České vysoké učení technické v Praze, 2013.

[29] Christos Papachristos, Kostas Alexis, and Anthony Tzes. Model predictive hovering-translation control of an unmanned tri-tiltrotor. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5425–5432. IEEE, 2013.

[30] Pixhawk. https://pixhawk.org/, February 2015.

[31] J Anthony Rossiter. *Model-based predictive control: a practical approach*. CRC press, 2013.

[32] M. Saska, J. Chudoba, L. Preucil, J. Thomas, G. Loianno, A. Tresnak, V. Vonasek, and V. Kumar. Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In *Proceedings of 2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, volume 1, pages 584–595, Danvers, 2014. IEEE Computer society.

[33] M. Saska, Z. Kasl, and L. Preucil. Motion Planning and Control of Formations of Micro Aerial Vehicles. In *Proceedings of The 19th World Congress of the International Federation of Automatic Control*, pages 1228–1233, Pretoria, 2014. IFAC.

[34] M. Saska, T. Krajnik, J. Faigl, V. Vonasek, and L. Preucil. Low Cost MAV Platform AR-Drone in Experimental Verifications of Methods for Vision Based Autonomous Navigation. In *Proceedings of 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 4808–4809, Piscataway, 2012. IEEE.

[35] M. Saska, T. Krajnik, V. Vonasek, Z. Kasl, V. Spurny, and L. Preucil. Fault-Tolerant Formation Driving Mechanism Designed for Heterogeneous MAVs-UGVs Groups. *Journal of Intelligent and Robotic Systems*, 73(1-4):603–622, January 2014.

[36] M. Saska, T. Krajnik, V. Vonasek, P. Vanek, and L. Preucil. Navigation, Localization and Stabilization of Formations of Unmanned Aerial and Ground Vehicles. In *Proceedings of 2013 International Conference on Unmanned Aircraft Systems*, pages 831–840, New York, 2013. Springer.

[37] M. Saska, J. Vakula, and L. Preucil. Swarms of Micro Aerial Vehicles Stabilized Under a Visual Relative Localization. In *ICRA2014: Proceedings of 2014 IEEE International Conference on Robotics and Automation*, pages 3570–3575, Piscataway, 2014. IEEE.

[38] M. Saska, V. Vonasek, T. Krajnik, and L. Preucil. Coordination and Navigation of Heterogeneous MAV-UGV Formations Localized by a 'hawk-eye'-like Approach Under a Model Predictive Control Scheme. *International Journal of Robotics Research*, 33(10):1393–1412, September 2014.

[39] M. Saska, V. Vonásek, T. Báča, and L. Přeučil. Ad-hoc heterogeneous (mav-ugv) formations stabilized under a top-view relative localization. In *In Proceedings of workshops of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IEEE)*, 2013.

[40] Satoshi Suzuki, Takahiro Ishii, Yoshihiko Aida, Yohei Fujisawa, Kojiro Iizuka, and Takashi Kawamura. Collision-free guidance control of small unmanned helicopter using nonlinear model predictive control. *SICE Journal of Control, Measurement, and System Integration*, 7(6):347–355, 2014.

[41] FreeRTOS Free Real-Time Operating System. http://www.freertos.org/, April 2015.

[42] Endrych Václav. Control and stabilization of an unmanned helicopter following a dynamic trajectory. Master's thesis, České vysoké učení technické v Praze, 2014.

[43] Xbee PRO wireless module. www.digi.com/zb_module_dk, April 2015.

[44] Pablo Zometa, M Kogel, Timm Faulwasser, and Rolf Findeisen. Implementation aspects of model predictive control for embedded systems. In *American Control Conference (ACC), 2012*, pages 1205–1210. IEEE, 2012.

Appendix A CD Content

In Table A.1 are listed names of directories on CD.

Directory name	Description
thesis	Master's thesis in pdf format
thesis_sources	latex source codes
src/STM	sources for STM32F4
src/xMega	sources for ATxMega128A3U
src/Matlab	matlab scripts for simulation and identification
src/CMatrixLib	CMatrixLib matrix library
src/hardware	Eagle files and material for PCB reproduction
videos	videos from experiments

Table A.1: CD Content

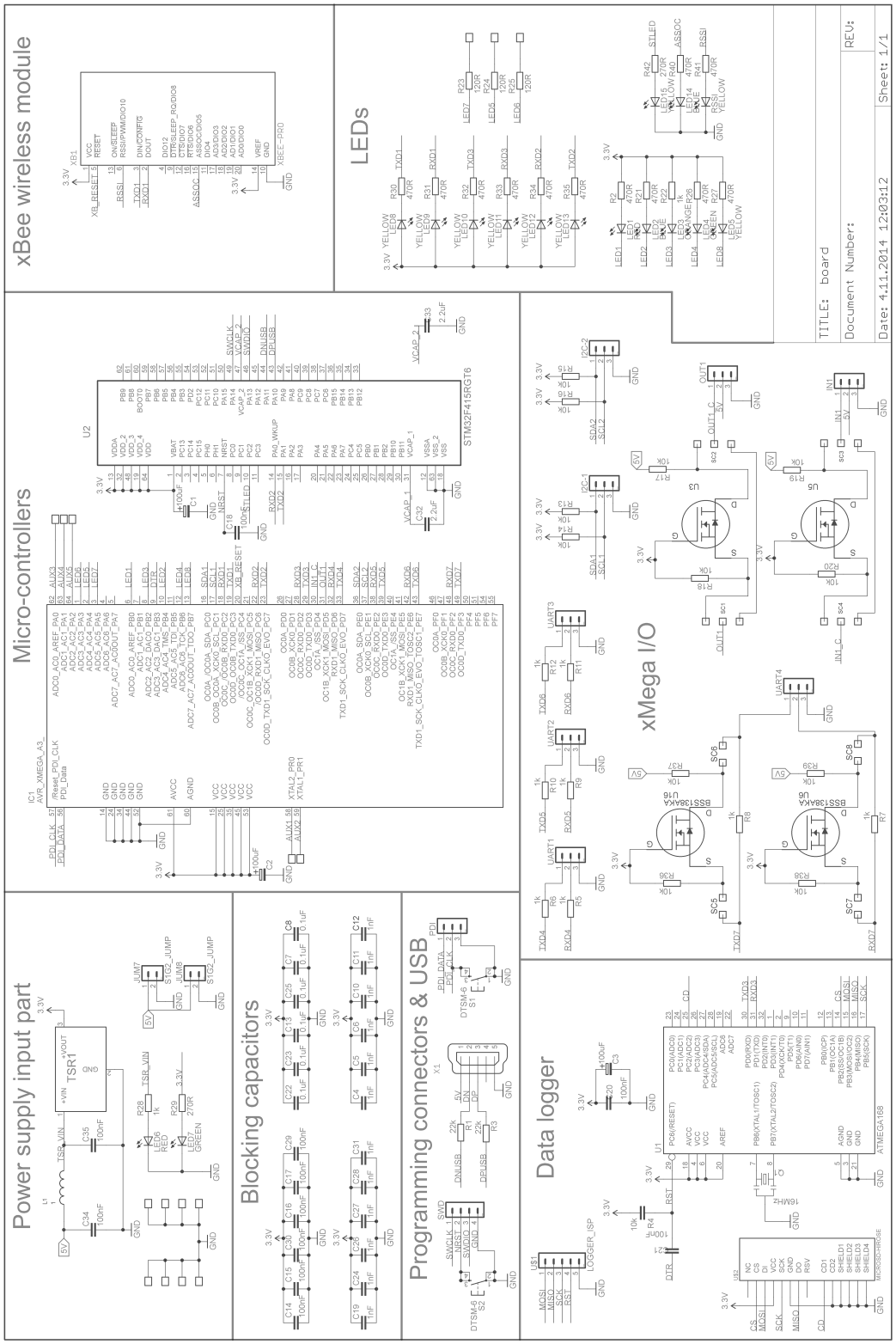
Appendix B List of abbreviations

In Table B.1 are listed abbreviations used in this thesis.

Abbreviation	Meaning
ANSI C	a standard for C programming language
API	application programming interface
AVR	Atmel's micro-controller architecture
ARM	RICS processor architecture
EKF	extended Kalman filter
ESC	electronic speed controller
FPU	floating-point unit
GNU	GNU's not Unix
GPS	global positioning system
IMU	inertial measurement unit
i^2c	two-wire serial interface
KF	Kalman filter
KK2	name of used stabilization board
LP	linear programming
LTI	liner time-invariant
MCU	microcontroller unit
MAV	micro aerial vehicle
MEMS	micro-electro-mechanical system
MPC	model predictive controller
PC	personal computer
PCB	printed circuit board
PID	proportional-integral-derivative controller
PPM	pulse-position modulation
QMPC	quadratic model predictive control
QP	quadratic programming
RAM	random access memory
RC	remotely controlled / remote controller
RMPC	robust model predictive controller
RTOS	real-time operating system
SRAM	static random access memory
STM	STMicroelectronics company
UART	universal asynchronous receiver transmitter
UAV	unmanned aerial aircraft

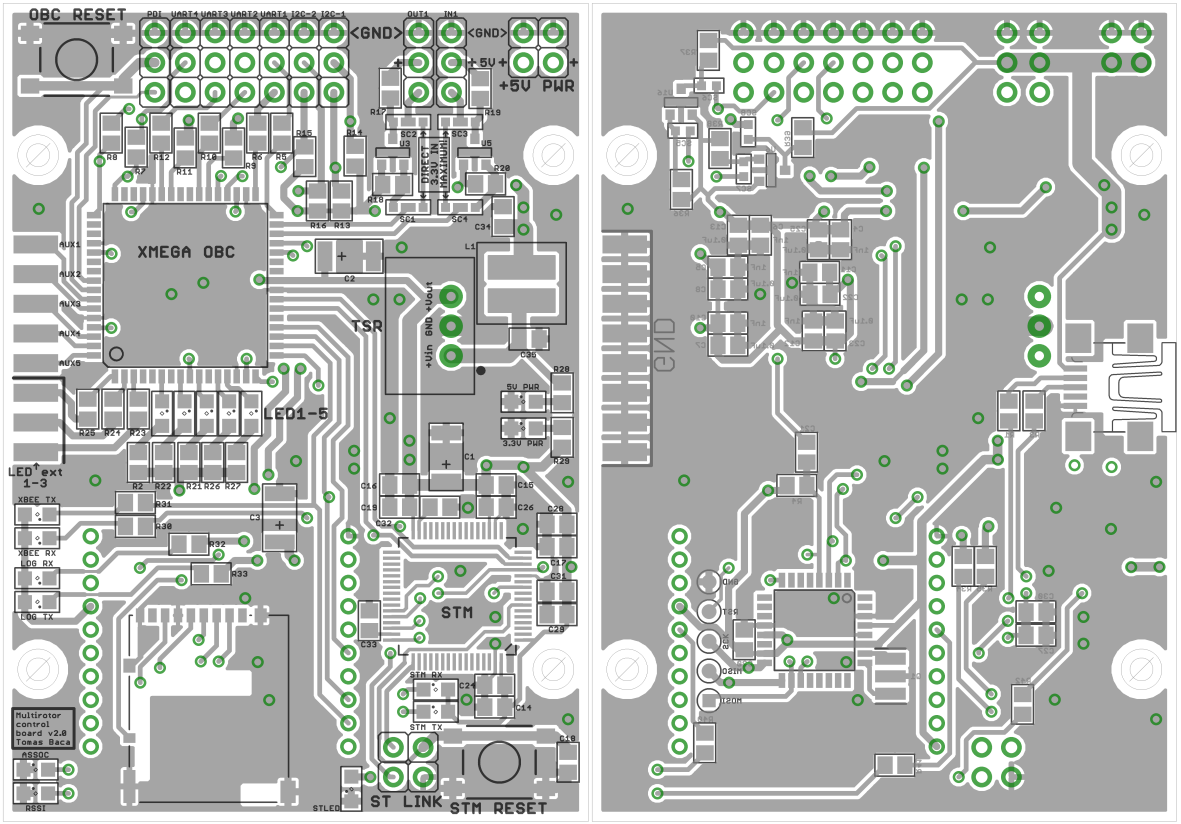
Table B.1: Lists of abbreviations

Appendix C Custom control board schematic



Electrical schematic of the custom control board v.2.

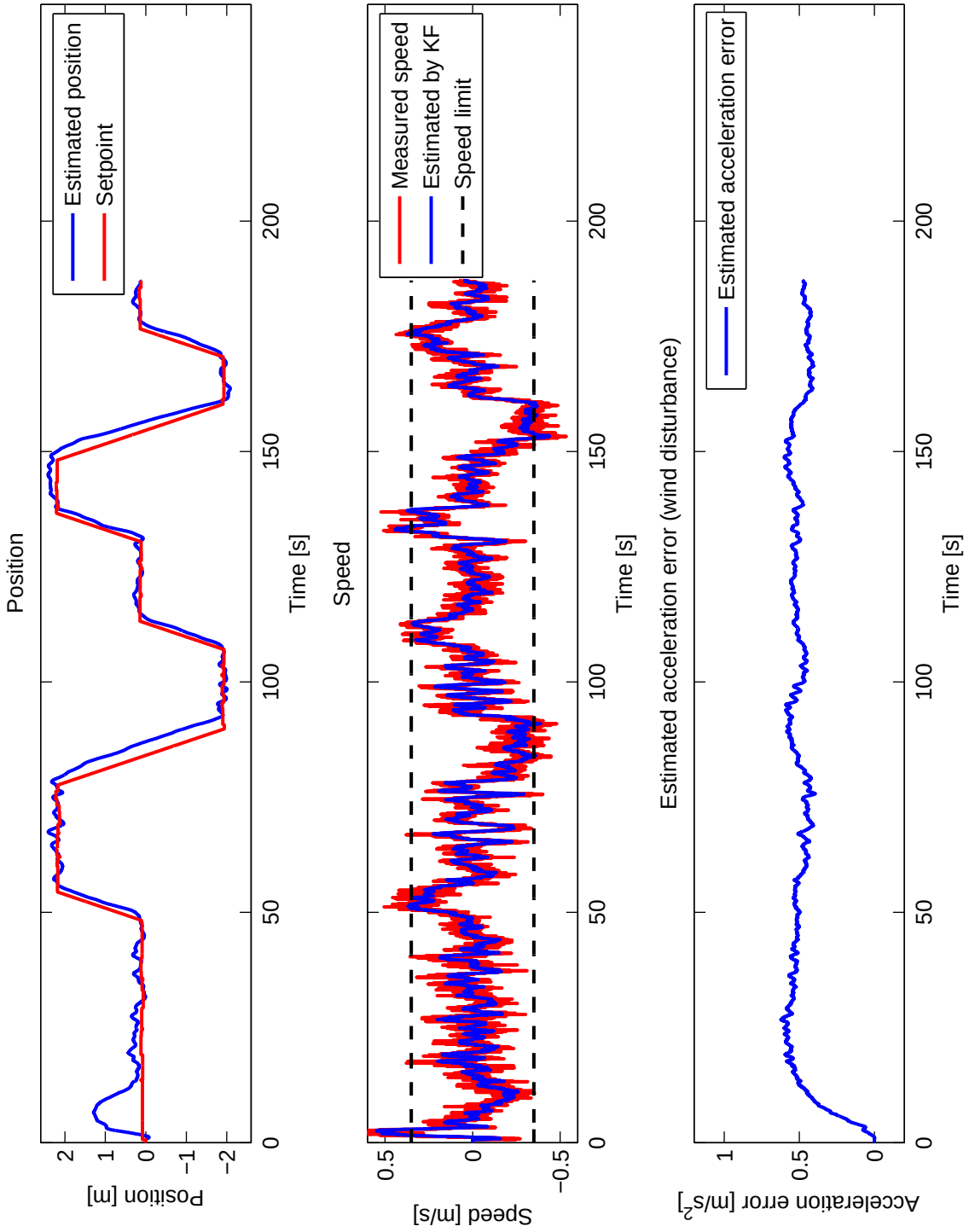
Appendix D PCB layouts



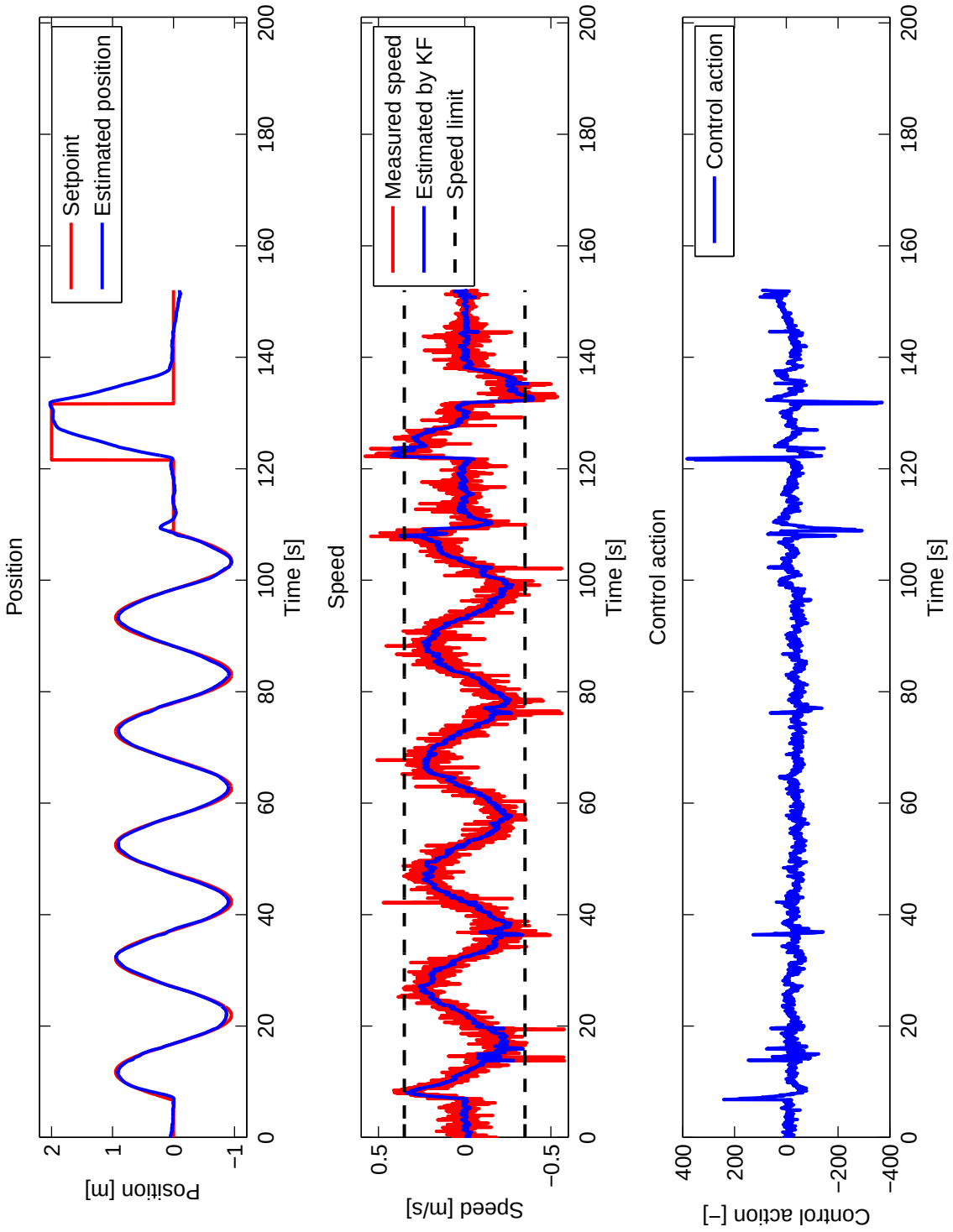
(a) Board's top layer. **(b)** Board's bottom layer.

Printed circuit board layouts of the custom control board v.2.

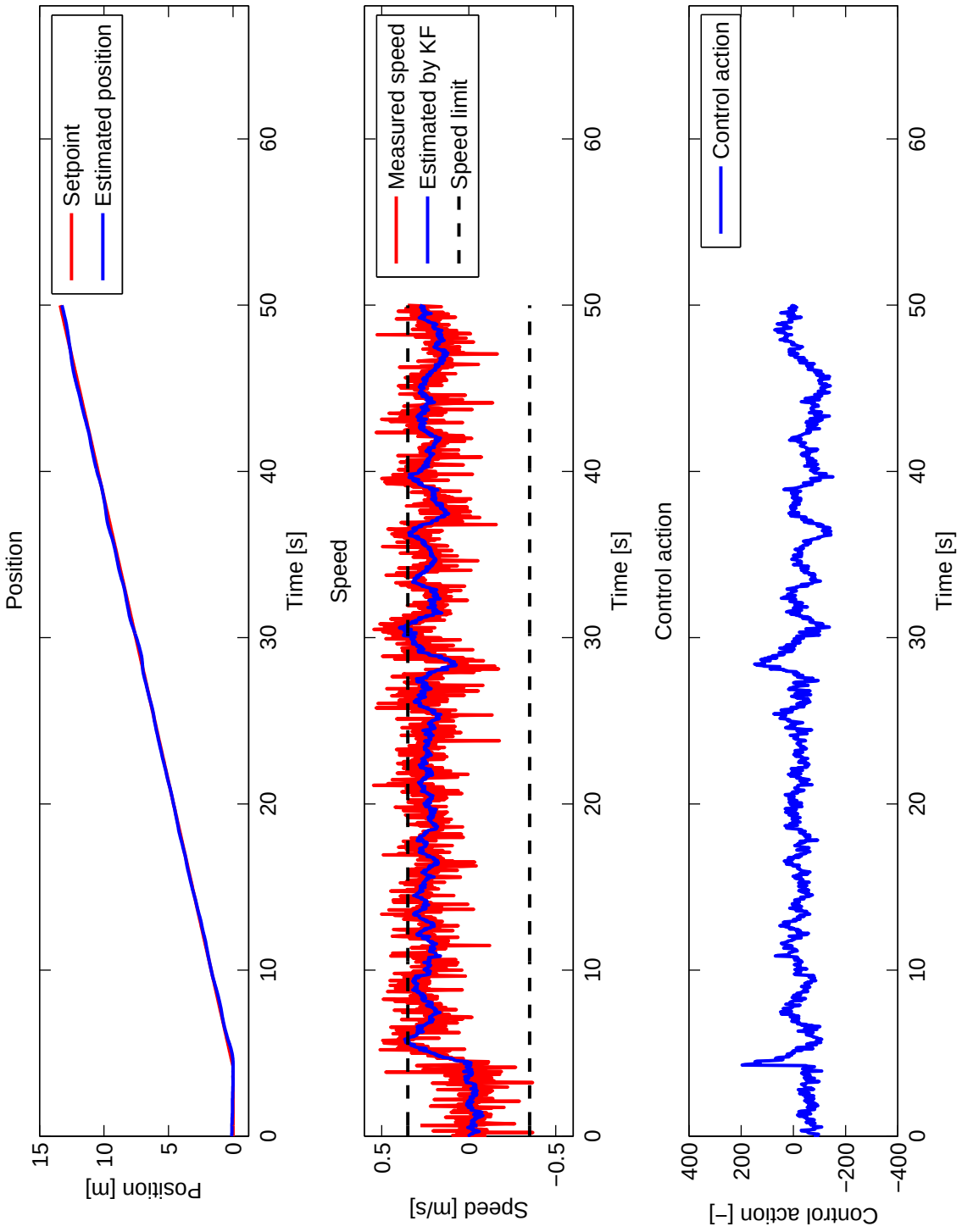
Appendix E Additional experimental result



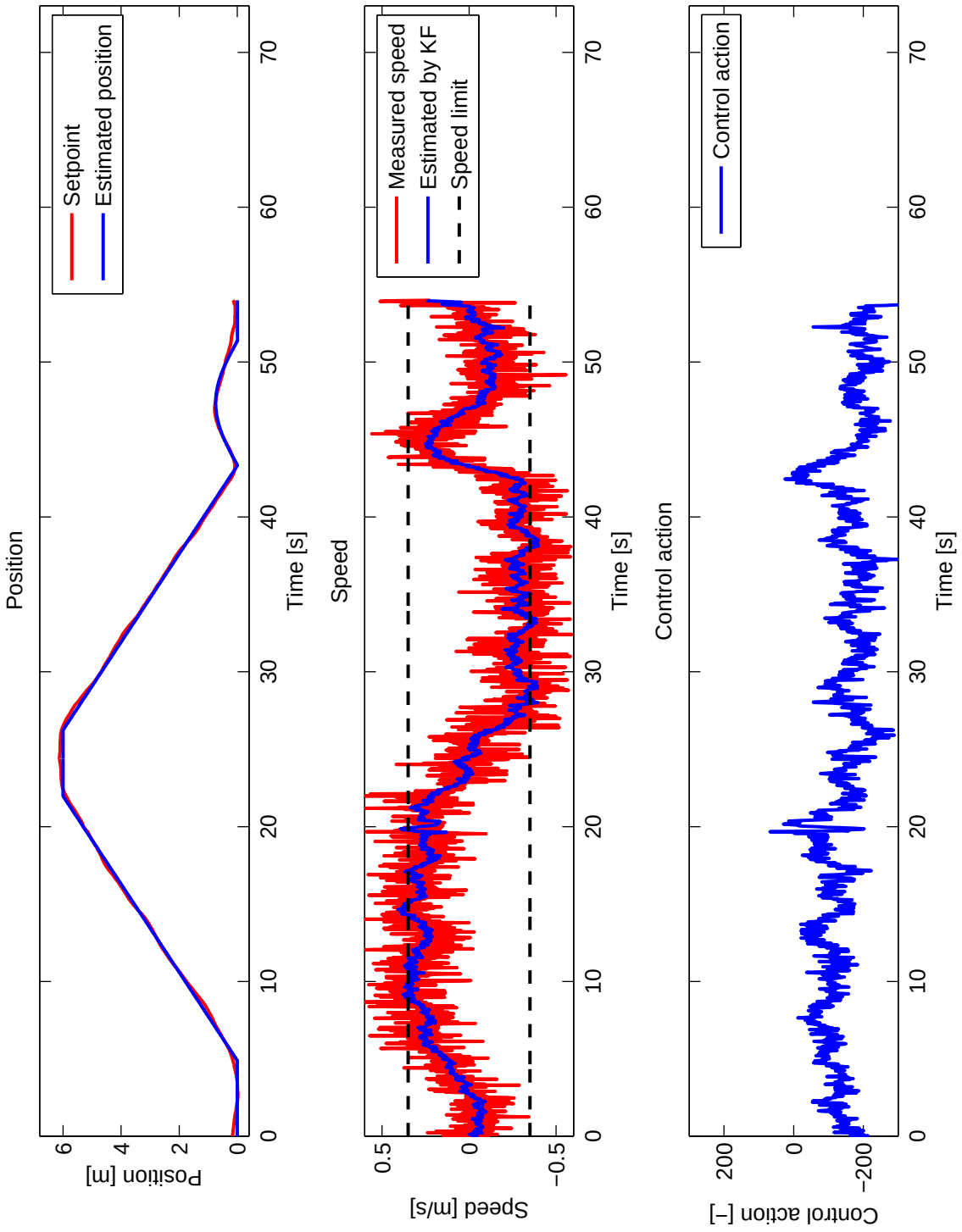
Data from experiment conducted during a strong wind ($5 - 10$ ms^{-1} according to weather forecast). Notice the control lag behind the setpoint — the desired setpoint was changing by 0.35 ms^{-1} which was the actual speed limit in the input governor. Since the system does not integrate the control error, there are no windup issues. Otherwise, there would be unwanted overshoots.



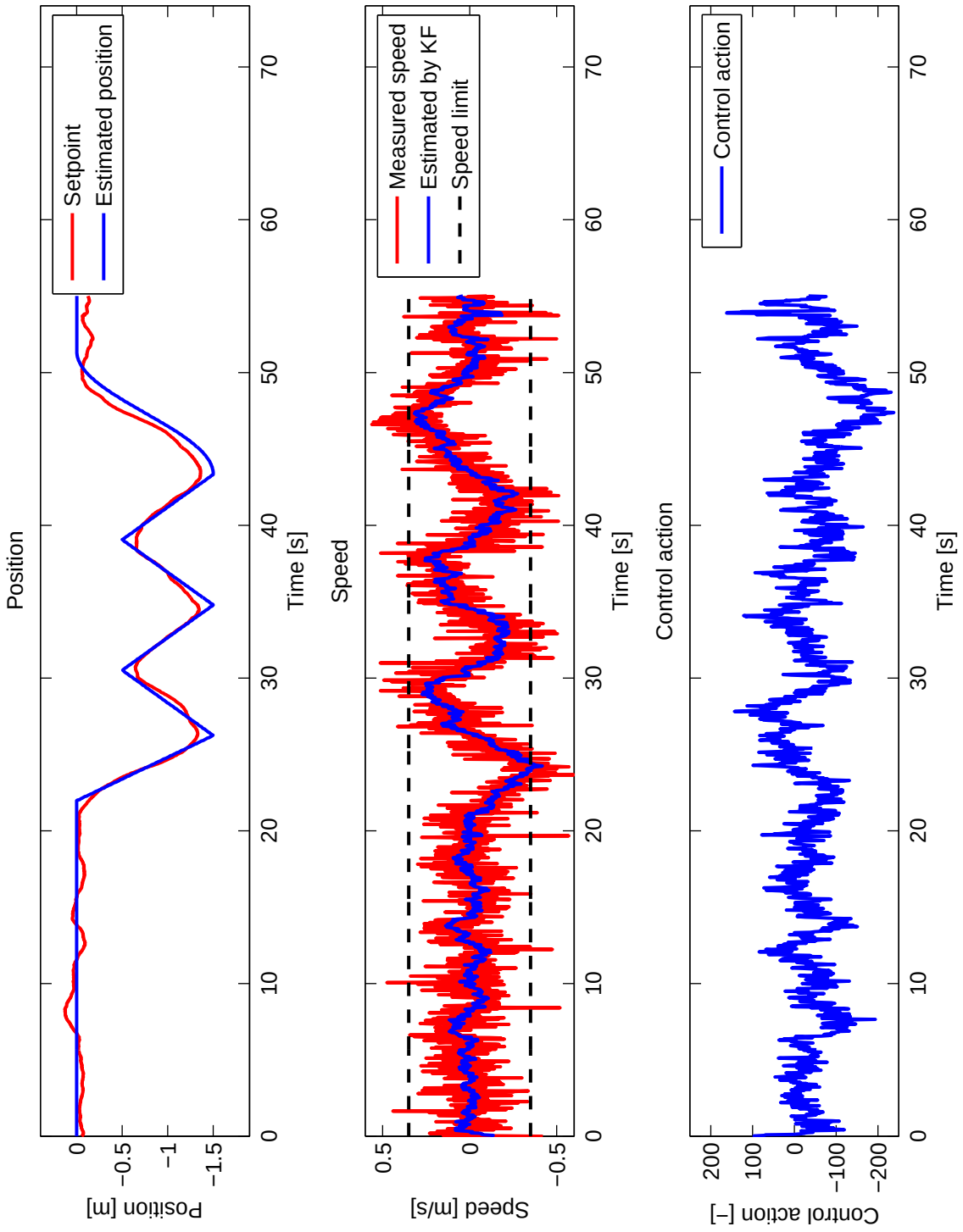
Data for forward axis from the experiment showed in the `video1`, `https://youtu.be/lPy7w-GUbw4`.



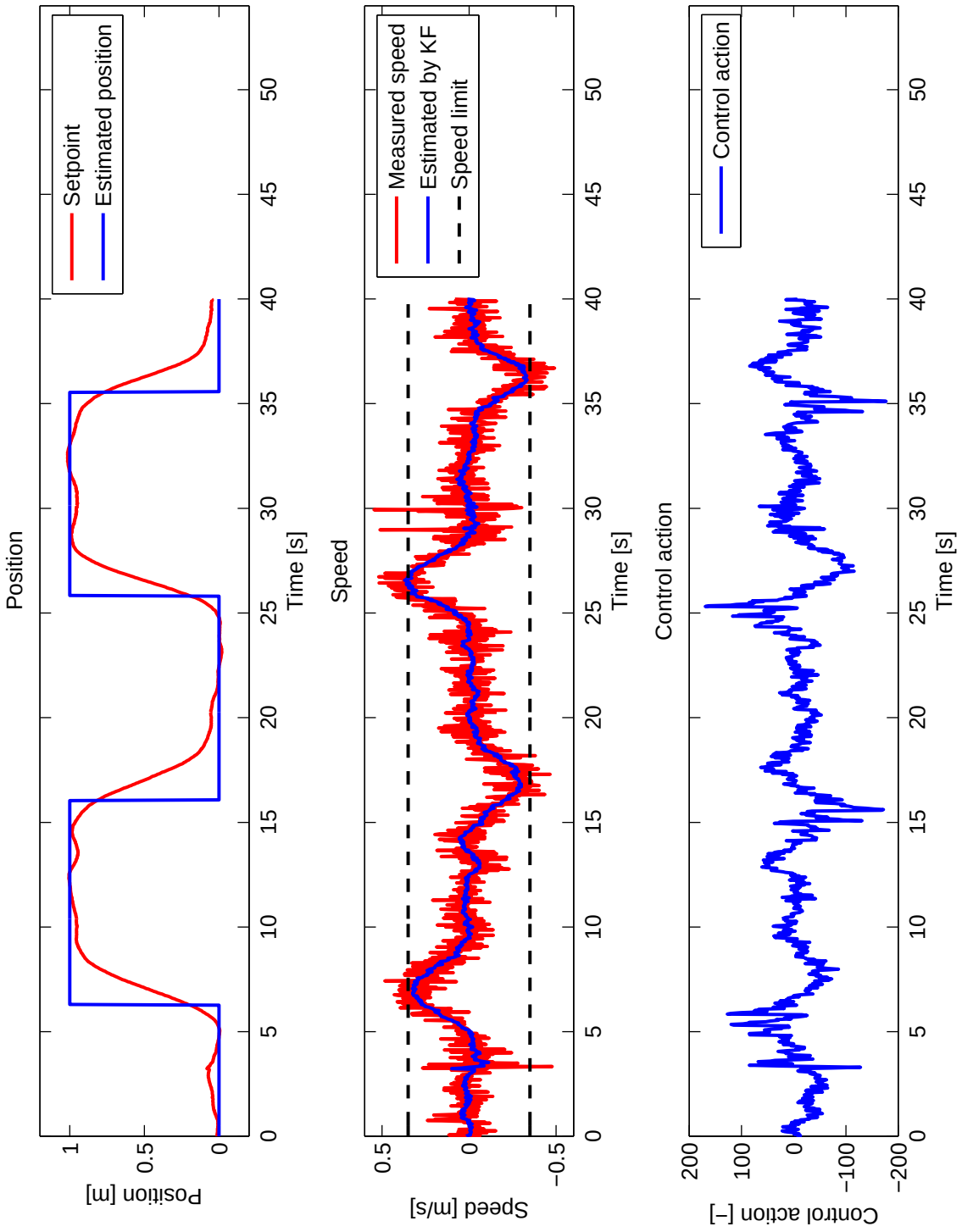
UAV tracking a ramp trajectory.



Data for forward axis from the experiment showed in the `video2`, `https://youtu.be/iqgL1H_DCmU`.



Data for the lateral axis from the experiment showed in the `video2`, `https://youtu.be/iqgL1H_DCmU`.



UAV tracking trajectory containing unit steps. This experiment was conducted outdoors.