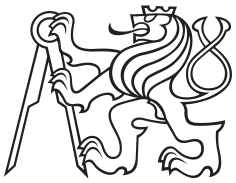
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Identification and Flight Control of Robotic Helicopter

Bc. Jan Švrčina

Supervisor: Ing. Jan Chudoba
Study Program: Cybernetics and Robotics
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Švrčina Jan** Personal ID number: **474447**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Identification and Flight Control of Robotic Helicopter

Master’s thesis title in Czech:

Identifikace a řízení letu robotické helikoptéry

Guidelines:

Make yourself familiar with helicopter Ryze Tello provided by your supervisor, as well as with methods of its control. Also, make yourself familiar with Vicon motion capture system, which will be used as a localization system.
• Identify model of the provided helicopter and
• design a method for a flight control along requested trajectories.
• Implement the designed method and demonstrate its function in several experiments with different trajectories.
• Evaluate quality and precision of the control.

Bibliography / sources:

[1] Jan Gärtner, Udržování formace UAV na základě měření vzájemných vzdáleností, diplomová práce, ČVUT v Praze - FEL, 2021
[2] Pařil David, Autonomous Control of Drone Ryze Tello, bakalářská práce, ČVUT v Praze - FEL, 2021
[3] T. Baca, D. Hert, G. Loianno, M. Saska and V. Kumar, "Model Predictive Trajectory Tracking and Collision Avoidance for Reliable Outdoor Deployment of Unmanned Aerial Vehicles," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 6753-6760

Name and workplace of master’s thesis supervisor:

Ing. Jan Chudoba Intelligent and Mobile Robotics CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **22.09.2023** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **16.02.2025**

Ing. Jan Chudoba	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt

Student’s signature

Acknowledgements

I thank my *alma mater* for the knowledge, skills and lessons I have acquired during my studies, my supervisor, Jan Chudoba, for the provided consultation, resources, and patience and my dear mother, Anna Švrčinová, for the lifelong support of my education.

Děkuji své *alma mater* za znalosti, schopnost a lekce, které jsem během svého studia získal, mému vedoucímu, Janu Chudobovi, za konzultace, zdroje a trpělivost a mé mamince, Anně Švrčinové, za celoživotní podporu mého vzdělávání.

Declaration

I declare that the presented work was developed indepenently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 24. May 2024

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 prací.

V Praze, dne 24. května 2024

Jan Švrčina

Abstract

The goal of this thesis is to identify the model of a small robotic helicopter Tello and to use the motion capture system Vicon to track and control the flight of the helicopter.

We have developed and implemented an off-line model parameter estimation method for identification and on-line asynchronous MHE (moving horizon estimator) and MPC (model predictive control) procedures for real-time control of the helicopter. This thesis presents the theoretical description of the methods used, and in detail, describes the implementation of these methods for a real system.

The results of experiments with the real system are also presented and analyzed, proving the capability of controlling a system with significant input delay.

Keywords: drone, UAV, robotic helicopter, motion tracking, identification, control, feedback, model predictive control, moving horizon estimation

Supervisor: Ing. Jan Chudoba
Jugoslávských partyzánů 1580/3
160 00 Dejvice

Abstrakt

Cílem této práce je identifikace modelu malé robotické helicoptéry Tello a použití systému pro snímání pohybu Vicon na sledování a řízení letu této helikoptéry.

Jsou odvozeny a implementovány off-line metoda pro odhad parametrů modelu při identifikaci a také on-line asynchroní MHE (moving horizon estimator) a MPC (model predictive control) procedury pro řízení helikoptéry v reálném čase. Tato práce prezentuje teoretický popis těchto použitých metod a také detailně popisuje jejich implementaci při použití na reálném systému.

Výsledky experimentů provedených na reálném systém jsou prezentovány a analyzovány, dokazující schopnost řízení systém s výrazným vstupním zpožděním.

Klíčová slova: dron, bezpilotní letoun, robotická helikoptéra, snímání pohybu, řízení, zpětná vazba, MPC, MHE

Překlad názvu: Identificace a řízení letu robotické helikoptéry

Contents

Chapter 1

Introduction

Flight control systems have been in use since the start of aviation, however with the ever-growing availability and capability of unmanned aerial vehicles (UAVs), the research regarding fully autonomous flight control is becoming increasingly more important.

UAVs are most commonly used in package delivery, agriculture, security inspections, and aerial photography. Although a relatively new technology, autonomous drones are highly versatile, attracting significant research interest for innovative applications like autonomous reconnaissance missions and firefighting [1]. Regardless of the specific area, reliable control and accurate localization are prerequisites for autonomous drones.

The goal of this thesis is to develop a solution for the autonomous flight control of a commercially available robotic helicopter with the use of external tracking system for localization. We present generic methods for the identification and control of dynamic systems, and we evaluate the availability of a tailored implementation of these methods for robotic helicopters.

1.1 Key concepts

In this section we describe two key concepts for the autonomous control of UAVs.

1.1.1 Quadrocopter

A quadrocopter is a type of robotic helicopter with four rotors. Figure 1.1 shows a diagram of a quadrocopter drone with numbered rotors. If the drone

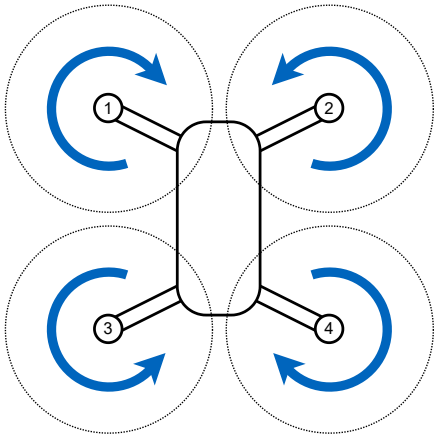


Figure 1.1: Quadrocopter diagram

is stationary, each rotor is rotating at the same speed (assuming the center of gravity is in the middle of the drone). Each rotor produces a lift equal to a quarter of the total weight. The net torque is zero because rotors 1 and 4 spin clockwise, and rotors 2 and 3 spin counterclockwise.

To change the position of the quadrocopter, we will vary the rotors' speeds. Four controls correspond to the control of a conventional helicopter or airplane, diagrams for the first 3 are in figure 1.2.

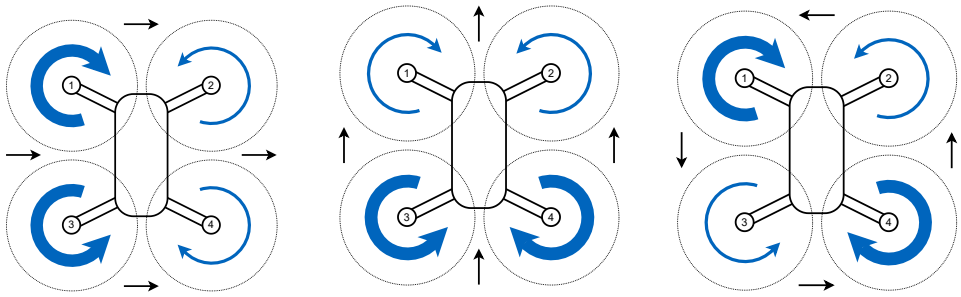
Roll. To move right, we increase the speed of rotors 1 and 3 and decrease the speed of rotors 2 and 4; this will tilt the drone to the right and the net force will accelerate it to the right.

Pitch. To move forward, we decrease the speed of rotors 1 and 2 and increase the speed of rotors 3 and 4; this will tilt the drone forward and the net force will accelerate it forward.

Yaw. To rotate counterclockwise, we increase the speed of rotors 1 and 4 (clockwise) and decrease the speed of rotors 2 and 3 (counterclockwise). The increased drag from clockwise rotors and decreased drag from counterclockwise rotors will result in a net torque when rotating the drone.

Throttle. To increase the altitude, we simply increase the speed of all rotors, increasing the lift.

Remark 1.1. All of the controls are explained in their positive directions. To move to the other direction, we reverse the rotor selection. For altitude decrease, lower the speed across all rotors.



(a) : Roll **(b) :** Pitch **(c) :** Yaw

Figure 1.2: Quadrocopter controls

1.1.2 Closed-loop control

When controlling a dynamic system using naive open-loop control, we simply get a reference (target) of where we want the system to end up and then send the appropriate inputs to the dynamic system. This is approach is simple and often stable (finite input results in a finite output). However, some disturbances usually affect our system, and our model of dynamic systems is often only an approximation.

To compensate for the disturbance and the inexact model, we can measure the output of the system with some sensors and use this measurement to adjust our control input accordingly, diagram in figure 1.3. This concept of closed-loop (feedback) control, allows us to adapt according to the current situation

at the cost of the measurement, and when designing the controller, we need to be mindful of the properties of the closed-loop system.

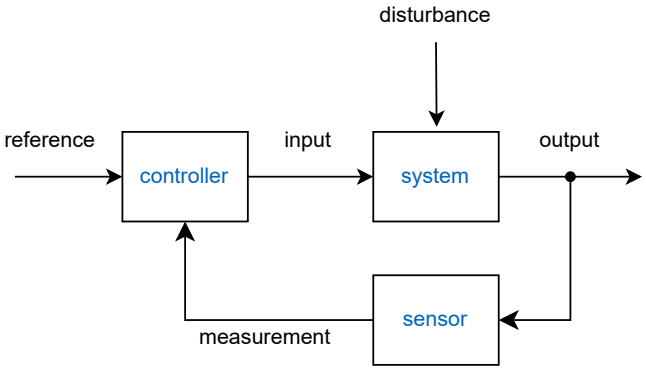


Figure 1.3: Closed-loop control

Chapter 2

Literature review

2.1 Theses working with Tello robotic helicopters

There are previous theses from CTU working with the same drones as ourselves. To list a few, M. Bekhzod [2] was tracking drone position using ArUco markers, J. Ševic [3] is using a drone camera to fly through obstacles, D. Pařil [4] is using the drone camera and odometry to control its position, J. Gärtner [5] is using self-made distance sensors to keep the formation of multiple drones.

Most of the research outside of CTU working with the Tello drones focuses on computer vision. Two examples are T. Saini [6], using the human pose estimation to control the drone and a paper from Hulek et al. [7] discussing positioning using April tags. There is very little research available regarding the control of the Tello drones, while the use of the drone's internal regulators is common.

It should be of note that in these theses or papers, the identification of the drone model (if it was done) was done using ad-hoc methods (usually simple input-step measurements) and the control was done using a PID regulator, either external or internal to the drone. The main improvement in this thesis is the use of a generic model data-driven method for identification and the use of the model-based MPC controller, similar to the current state-of-the-art implementations of UAV flight control [8].

2.2 State of the art

Current state-of-the-art UAV flight control is well summarized in [9]. Current applications use many different approaches. Linear model methods using proportional–integral–derivative (PID) controller [10], linear-quadratic regulator (LQR) or H-inf controllers [11] are used for their computation simplicity and well-developed methods guaranteeing the stability of the closed-loop system.

Non-linear models are approaches such as feedback linearization [12] or backstepping [13] that trade off the simplicity of the model in favor of capturing the non-linear dynamics of the drone, improving the accuracy of the controller, usually at the cost of only local stability and increased computational complexity.

The computationally demanding model predictive methods are used for linear, linearized (linear approximation of the model in each step) or non-linear models [14] with increasing popularity in real-time systems (such as flight control) due to the availability of computational power.

Reinforcement learning approaches that adapt the parameters of the controller (for example, coefficients of the PID) during the flight or directly control the drone [15] are an active area of research but are yet to outperform classic methods of control.

Extensive details for the application of UAVs are in "UAVs Beneath the Surface: Cooperative Autonomy for Subterranean Search and Rescue in DARPA SubT" [16] by the Multi-robot Systems group here at CTU.

Results regarding controller performance, including its precision, robustness, or comparison with other regulators, are primarily collected in simulations. Real-world data is presented for specific applications where the controller is tailored for a selected task.

Chapter 3

Theoretical background

3.1 Dynamical systems

This section will introduce the basics of dynamical systems, such as the state space model with disturbances and discretization, as well as clarify the used notation. It will also introduce the model identification problem, the moving horizon estimator (MHE) and the model predictive controller (MPC), concepts used for the identification and control of the real robotic helicopter.

> The dynamical behavior of systems can be understood by studying their mathematical descriptions. (P. Antsaklis and A. Michel [17])

Most of the introductory descriptions of the dynamical systems will come from the book "A Linear Systems Primer" [17] by Antsaklis and Michel. It is a great book that introduces many critical concepts for studying dynamic systems. Another very verbose source regarding this topic is in [18].

In this thesis, we will discuss systems described by differential or difference equations, "It has become customary in the engineering literature to use the term *dynamical system* rather loosely." (P. Antsaklis and A. Michel [17]).

Definition 3.1. A *signal* is a value representing information in time. It is a mapping from the time domain T (continuous or discrete) to a subset of real (or possible complex) numbers $A \subseteq \mathbb{R}$ [19],

$$x : T \to A. \tag{3.1}$$

Definition 3.2. A *system* is a relation $R \subseteq (U \times O)$ of input $u \in U$ and output $o \in O$ signals [19].[1]

Definition 3.3. The time domain T can be either continuous $t \in \mathbb{R}$ or discrete $k \in \mathbb{Z}$, we then classify the system as a *continuous-time system* or a *discrete-time system* [17].

Definition 3.4. If the output of the system o is only influenced by current and previous inputs [19] [17],

$$u_1(t) = u_2(t),\ \forall t \leq t_0 \implies o_1(t) = o_2(t),\ \forall t \leq t_0, \tag{3.2}$$

the system is *causal*.

Definition 3.5. The system is *time-invariant* if the time-shifted input produces the same time shifted-output [19],

$$u(t) \to o(t) \implies u(t-\tau) \to o(t-\tau). \tag{3.3}$$

3.1.1 State-space model

We describe dynamical systems using mathematical models. The key model we will use to describe our systems will be the *state-space model*. It has a set of inputs $\mathbf{u} \in \mathcal{U}$, outputs $\mathbf{o} \in \mathcal{O}$, and states $\mathbf{s} \in \mathcal{S}$.[2] The input $\mathcal{U} \subset \mathbb{R}^{n_u}$, output $\mathcal{O} \subset \mathbb{R}^{n_o}$, and state spaces $\mathcal{S} \subset \mathbb{R}^{n_s}$ are subsets of real vector spaces with appropriate dimensions n_u, n_o and n_s, also called number of inputs, outputs, and states.

In continuous time, the state space model is described using a set of differential equations, and in discrete time, it uses difference equations. The implicit form is more general than the much preferred explicit form.

Remark 3.6. We are denoting time derivatives using the *dot* notation,

$$\frac{\mathrm{d}a(t)}{\mathrm{d}t} = \dot{a}(t). \tag{3.4}$$

[1] The notation of the input as u is customary throughout the control engineering literature (sometimes a is used as in action), however the output is usually denoted y. We chose the letter o as the notation of the output (observation) so as not to confuse it with the y-axis further in the thesis.

[2] Similarly to the output notation, the state is usually with denoted with the letter x. However, to avoid confusion with the x-axis, we choose the letter s.

Remark 3.7. To make the following equations more readable and compact, we are using the *subscript* time notation for the inputs, outputs, states, and other time-varying variables, these notations are equivalent,[3]

$$a_t = a(t),\ a_k = a(k). \tag{3.5}$$

Definition 3.8. The *implicit form* of the state-space model in *continuous-time* is,

$$\mathbf{F}(\dot{\mathbf{s}}_t,\ \mathbf{s}_t,\ \mathbf{u}_t,\ t) = \mathbf{0}, \tag{3.6}$$
$$\mathbf{G}(\mathbf{o}_t,\ \mathbf{s}_t,\ \mathbf{u}_t,\ t) = \mathbf{0}, \tag{3.7}$$

where $\mathbf{F}(\cdot)$ is the state equation function and $\mathbf{G}(\cdot)$ is the output equation function [20][19].

Definition 3.9. The *explicit form* of the state-space model in *continuous-time* is,

$$\dot{\mathbf{s}}_t = \mathbf{f}(\mathbf{s}_t,\ \mathbf{u}_t,\ t), \tag{3.8}$$
$$\mathbf{o}_t = \mathbf{g}(\mathbf{s}_t,\ \mathbf{u}_t,\ t), \tag{3.9}$$

where $\mathbf{f}(\cdot)$ is the state equation function and $\mathbf{g}(\cdot)$ is the output equation function [20][19].

Definition 3.10. The *explicit form* of the state-space model in *discrete-time* is,

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k,\ \mathbf{u}_k,\ k), \tag{3.10}$$
$$\mathbf{o}_k = \mathbf{g}(\mathbf{s}_k,\ \mathbf{u}_k,\ k), \tag{3.11}$$

where $\mathbf{f}(\cdot)$ is the state equation function and $\mathbf{g}(\cdot)$ is the output equation function [20].

Remark 3.11. Assuming that the system is time invariant, we can omit the time t and time-step k in the state and output functions. Unless otherwise specified, this is assumed in the following usage of space-state models.

Parametrization

Because in real use cases, the exact equations are not known, only their forms derived from physics, biology, economics, et cetera. They are parameterized by the model parameters $\mathbf{p} \in \mathcal{P}$.[4] The parameter space $\mathcal{P} \subset \mathbb{R}^{n_p}$ is, again, a subset of a real vector space with an appropriate dimension of number of parameters n_p. For this thesis, we will only parameterize the state equation function (the output function could also be parameterized).

[3] Sometimes the time notation is forgone completely. However, we would like to keep the time-dependency of the variables clear.

[4] Again, the parameters of the model are often times denoted θ or w, but we choose the letter p to avoid conflicting notation.

Definition 3.12. The state equation for the continuous-time state-space model with parametrization is

$$\dot{\mathbf{s}}_t = \mathbf{f}(\mathbf{s}_t, \mathbf{u}_t, \mathbf{p}). \tag{3.12}$$

Definition 3.13. The state equation for the discrete-time state-space model with parametrization is

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k, \mathbf{u}_k, \mathbf{p}). \tag{3.13}$$

Noise

For the real use of state space models, we introduce the concept of noise. Because there are often external factors that we are unable to control, we include them in the model. These factors are often random or unobservable. The state noise $\mathbf{w}$ influences the state equation, and the output (observation) noise $\mathbf{v}$ influences the output equation [17]. We will use the *additive noise* model [21], where the noise is simply added to the respective equation. With this we arrive at the form of the state space model used in this thesis.

Definition 3.14. The equations for the continuous-time state-space model with parametrization and additive noise are

$$\dot{\mathbf{s}}_t = \mathbf{f}(\mathbf{s}_t, \mathbf{u}_t, \mathbf{p}) + \mathbf{w}_t, \tag{3.14}$$

$$\mathbf{o}_t = \mathbf{g}(\mathbf{s}_t,\ \mathbf{u}_t) + \mathbf{v}_t. \tag{3.15}$$

Definition 3.15. The equations for the discrete-time state-space model with parametrization and additive noise are

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k,\ \mathbf{u}_k,\ \mathbf{p}) + \mathbf{w}_k, \tag{3.16}$$

$$\mathbf{o}_k = \mathbf{g}(\mathbf{s}_k,\ \mathbf{u}_k) + \mathbf{v}_k. \tag{3.17}$$

3.1.2 Discretization

A continuous-time system can be discretized by using the Euler method [22], which is a first-order approximation. It is causal because the next state is only affected by current and past states and inputs, it also introduces a minimal delay caused by discretization. The principle is $\frac{\mathrm{d}\mathbf{s}(t)}{\mathrm{d}t} \approx \frac{\Delta\mathbf{s}(t)}{\Delta t}$, where Δt is the discretization step [23].

$$\frac{\mathrm{d}\mathbf{s}_t}{\mathrm{d}t} = \mathbf{f}(\mathbf{s}_t, \mathbf{u}_t) \approx \frac{\Delta\mathbf{s}_t}{\Delta t} = \frac{\mathbf{s}_{t+\Delta t} - \mathbf{s}_t}{\Delta t}. \tag{3.18}$$

Then the equation 3.18 is rearranged, where $\boldsymbol{\epsilon}_t$ is the discretization error,

$$\mathbf{s}_{t+\Delta t} = \mathbf{s}_t + \Delta t\, \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) + \boldsymbol{\epsilon}_t, \tag{3.19}$$

from this, we can create a discrete-time state-space model approximating the real model.

Definition 3.16. The Euler method for the discretization of the state-space model is

$$\mathbf{s}_{k+1} = \mathbf{s}_k + \Delta t\, \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k). \tag{3.20}$$

There are many more discretization methods, described in chapter 8, Digital control in [24], all of them trading off between accuracy (higher-order methods), computational complexity and introduced delay [23].

3.1.3 Parameter identification problem

As noted, we often only know the form of the state $\mathbf{f}(\cdot)$ and the output equation $\mathbf{g}(\cdot)$, but not the actual values of the parameter $\mathbf{p}$. Parameter identification problem [25], is that we have a set of trajectories $H = \{h_1, \ldots, h_{|H|}\}$, where each trajectory $h = \{(\mathbf{u}_{h,k}, \mathbf{o}_{h,k})\}_{k=1}^{L_h}$ consists of a history of inputs $\mathbf{u}_{h,k}$ and outputs $\mathbf{o}_{h,k}$ (index h denotes the trajectory, k denotes the time-step and L_h is the length of the trajectory, number of time-steps in h) and we want to get the estimate of the model parameters $\hat{\mathbf{p}}$, with the model equations 3.16 and 3.17.

Partially observed Markov process

To develop the estimator for the parameter identification problem, we use the framework of partially observed Markov process (POMP) [26] [27] [28] to describe the randomness of noise influencing the system dynamics.

Definition 3.17. Markov process (MP)[5] is a random process $\{X_t \,|\, k \in T\}$ [29], if the conditional distribution of X_{k+1} given the values of $\{X_0, \ldots, X_k\}$ is only dependent on the last state X_k [30]. This is called the Markov property,

$$\mathrm{P}\left[X_{k+1} \leq x_{k+1} \,|\, X_0 = x_0 \ldots, X_k = x_k\right] = \mathrm{P}\left[X_{k+1} \leq x_{k+1} \,|\, X_k = x_k\right]. \tag{3.21}$$

Definition 3.18. Partially observed Markov process (MP)[5] [31][26] is the extension of the MP, where we do not observe the state X_t directly but only through an observation Y_t, the conditional distribution of Y_t is only dependent on the current state,

$$\mathrm{P}\left[Y_k \leq y_k \,|\, X_0 = x_0 \ldots, X_k = x_k\right] = \mathrm{P}\left[Y_k \leq y_k \,|\, X_k = x_k\right]. \tag{3.22}$$

[5] We are considering a discrete-time Markov process with a continuous state.

MAP estimate

Now that we have formulated the POMP, we will try to fit our problem of parameter identification into it.

Remark 3.19. Since we are working with continuous states, we will be speaking in terms of probability density functions (PDF)[6] noted $p_{X|Y}(x \mid y)$ or $p(x \mid y)$.

Because the state-space model (equation 3.16) has the Markov property ($\mathbf{s}_{k+1}$ is only based on $\mathbf{s}_k$ and independent noise $\mathbf{w}_k$) and observations $\mathbf{o}_k$ are following the POMP definition (equation 3.17, only based on the current state $\mathbf{s}_k$ and independent noise $\mathbf{v}_k$), we can formulate the following probability density functions,

$$p(\mathbf{s}_{k+1} \mid \mathbf{s}_k,\, \mathbf{u}_k,\, \mathbf{p}) = p(\mathbf{w}_k + \mathbf{f}(\mathbf{s}_k,\, \mathbf{u}_k,\, \mathbf{p})), \tag{3.23}$$

$$p(\mathbf{o}_k \mid \mathbf{s}_k,\, \mathbf{u}_k) = p(\mathbf{v}_k + \mathbf{g}(\mathbf{s}_k,\, \mathbf{u}_k)). \tag{3.24}$$

With these, we now formulate the maximum a posteriori (MAP) estimation [26] of the parameter $\mathbf{p}$, for a set of trajectories with known states $\tilde{H} = \{\tilde{h}_1, \ldots, \tilde{h}_{|\tilde{H}|}\}$, where each trajectory $\tilde{h} = \{(\mathbf{u}_{\tilde{h},k}, \mathbf{s}_{\tilde{h},k})\}_{k=1}^{L_{\tilde{h}}}$ consists of a history of inputs $\mathbf{u}_{\tilde{h},k}$ and states $\mathbf{s}_{\tilde{h},k}$,

$$\hat{\mathbf{p}} = \arg\max_{\mathbf{p}} \tilde{L}(\mathbf{p},\, H_s) = \arg\max_{\mathbf{p}} p(\mathbf{p} \mid \tilde{H}) = \arg\max_{\mathbf{p}} \frac{p(\tilde{H} \mid \mathbf{p}) \overbrace{p(\mathbf{p})}^{\text{prior of } \mathbf{p}}}{\underbrace{p(\tilde{H})}_{\text{independent of } \mathbf{p}}} \tag{3.25}$$

The prior term $p(\mathbf{p})$ is some prior information about the parameters $\mathbf{p}$, like previous measurements or expert knowledge (lower or upper bound, expected value, previous measurements, etc.). The most complicated is the term $p(\tilde{H} \mid \mathbf{p})$. First, we split it for each trajectory $\tilde{h}$. We assume the trajectories are independent,

$$p(\tilde{H} \mid \mathbf{p}) = \prod_{\tilde{h} \in \tilde{H}} p(\tilde{h} \mid \mathbf{p}). \tag{3.26}$$

The probability of the trajectory $\tilde{h} = \{(\mathbf{u}_{\tilde{h},k}, \mathbf{s}_{\tilde{h},k})\}_{k=1}^{L_{\tilde{h}}}$ is a joint probability of each of the states and inputs

$$p(\tilde{h} \mid \mathbf{p}) = p(\mathbf{u}_{\tilde{h},1},\, \mathbf{s}_{\tilde{h},1}, \ldots,\, \mathbf{u}_{\tilde{h},L_{\tilde{h}}},\, \mathbf{s}_{\tilde{h},L_{\tilde{h}}} \mid \mathbf{p}) \tag{3.27}$$

[6] Probability density $p_X(x)$ (in literature often denoted $f_X(x)$) is defined as $\mathrm{P}[a \leq X \leq b] = \int_a^b p_X(x)\mathrm{d}x$, further information in [29].

Now we use the chain rule of conditional probability (remark 3.21), remove the conditional terms using independence (remark 3.22) and use Markov property of the states $\mathbf{s}_{\tilde{h},k}$ to simplify the states,

$$p(\tilde{h} \,|\, \mathbf{p}) = \underbrace{p(\mathbf{s}_{\tilde{h},1})}_{\text{initial state}} \overbrace{\prod_{k=1}^{L_{\tilde{h}}-1} p(\mathbf{s}_{\tilde{h},k+1} \,|\, \mathbf{s}_{\tilde{h},k},\, \mathbf{u}_{\tilde{h},k},\, \mathbf{p})}^{\text{Markov property}} \underbrace{\prod_{k=1}^{L_{\tilde{h}}} p(\mathbf{u}_{\tilde{h},k})}_{\text{indep. of } \mathbf{p}}. \tag{3.28}$$

Remark 3.20. In equation 3.28, we are only using $L_{\tilde{h}} - 1$ time-steps because that the number of state transitions that we have. The initial state is also independent of $\mathbf{p}$.

Until now, we assumed that we measured the states $\mathbf{s}$ fully and exactly. However that is not the case for the actual model, as we only have the observations $\mathbf{o}$. Because of this, we can only use an estimate of the state, so we need the likelihood not only for the parameters $\mathbf{p}$ but also all of the states for each trajectory $H_s = \{\{\mathbf{s}_{h,k}\}_{k=1}^{L_h}\}_{h \in H}$,

$$\begin{aligned}(\hat{\mathbf{p}}, \hat{H}_s) &= \arg\max_{\mathbf{p},\, H_s} L(\mathbf{p}) = \arg\max_{\mathbf{p},\, H_s} p(\mathbf{p},\, H_s \,|\, H) = \\ &= \arg\max_{\mathbf{p},\, H_s} \frac{p(H,\, H_s \,|\, \mathbf{p}) \overbrace{p(\mathbf{p})}^{\text{prior of } \mathbf{p}}}{\underbrace{p(H)}_{\text{independent of } \mathbf{p},\, H_s}}.\end{aligned} \tag{3.29}$$

We can again split the trajectories and define the joint probability for the trajectory,

$$p(h,\, h_s \,|\, \mathbf{p}) = p(\mathbf{u}_{h,1},\, \mathbf{o}_{h,1},\, \mathbf{s}_{h,1},\, \ldots,\, \mathbf{u}_{h,L_h},\, \mathbf{u}_{h,L_h},\, \mathbf{s}_{h,L_h} \,|\, \mathbf{p}) \tag{3.30}$$

Now, we start splitting away the parts of the joint probability using the chain rule of conditional probability (remark 3.21) and removing conditional terms using independence (remark 3.22). Similarly to equation 3.28, we take away the $\mathbf{u}_{h,k}$, and next we use the Markov property to split away the states $\mathbf{s}_{h,k}$, which leaves us with the observations $\mathbf{o}_{h,k}$, only dependent on the current state and input, the final equation is then

$$p(h,\, h_s \,|\, \mathbf{p}) = \underbrace{p(\mathbf{s}_{h,1})}_{\text{initial state}} \overbrace{\prod_{k=1}^{L_h-1} p(\mathbf{s}_{h,k+1} \,|\, \mathbf{s}_{h,k},\, \mathbf{u}_{h,k},\, \mathbf{p})}^{\text{state transition}} \times \\ \times \underbrace{\prod_{k=1}^{L_h} p(\mathbf{o}_{h,k} \,|\, \mathbf{s}_{h,k},\, \mathbf{u}_{h,k})}_{\text{observations indep. of } \mathbf{p}} \overbrace{\prod_{k=1}^{L_h} p(\mathbf{u}_{h,k})}^{\text{indep. of } \mathbf{p}} \tag{3.31}$$

Remark 3.21. The chain rule of conditional probability is that for random events $A_1, \ldots, A_n$ [29],

$$\mathrm{P}\left[A_1, \ldots, A_n\right] = \mathrm{P}\left[A_1\right] \mathrm{P}\left[A_2 \,|\, A_1\right] \mathrm{P}\left[A_3 \,|\, A_1,\, A_2\right] \ldots \mathrm{P}\left[A_n \,|\, A_1, \ldots, A_{n-1}\right]. \tag{3.32}$$

Remark 3.22. We are removing conditional terms for the independent variables because if $A \perp B$, then

$$\mathrm{P}\left[A \,|\, B\right] = \mathrm{P}\left[A\right]. \tag{3.33}$$

Putting it all together while omitting the terms that do not include $\mathbf{p}$ or $\mathbf{s}_{h,k}$ (*argmax* operator is invariant to multiplication with positive values [32]), we get

$$(\hat{\mathbf{p}}, \hat{H}_s) = \underset{\mathbf{p},\, H_s}{\arg\max}\, L(\mathbf{p},\, H_s) = \underset{\mathbf{p},\, H_s}{\arg\max}\; \overbrace{p(\mathbf{p})}^{\text{prior of } \mathbf{p}} \times \\ \times \prod_{h \in H} \left(\underbrace{p(\mathbf{s}_{h,1})}_{\text{initial state}} \overbrace{\prod_{k=1}^{L_h-1} p(\mathbf{s}_{h,k+1} \,|\, \mathbf{s}_{h,k},\, \mathbf{u}_{h,k},\, \mathbf{p})}^{\text{state transitions}} \underbrace{\prod_{k=1}^{L_h} p(\mathbf{o}_{h,k} \,|\, \mathbf{s}_{h,k},\, \mathbf{u}_{h,k})}_{\text{observations}} \right). \tag{3.34}$$

Working with products is complicated, so we rewrite this term in summation using the log transform. Even though the initial state term $p(\mathbf{s}_{h,1})$ could be useful if we had a guess in what state the system is at the start of the trajectory, we omit it as well (prior for the initial state is uniform distribution over $\mathcal{S}$),

$$(\hat{\mathbf{p}}, \hat{H}_s) = \underset{\mathbf{p},\, H_s}{\arg\max}\, L(\mathbf{p}, H_s) = \underset{\mathbf{p},\, H_s}{\arg\max} \log\left(L(\mathbf{p}, H_s)\right) = \underset{\mathbf{p},\, H_s}{\arg\max} \log(p(\mathbf{p})) + \\ + \sum_{h \in H} \left(\sum_{k=1}^{L_k - 1} \log\left(p(\mathbf{s}_{h,k+1} \,|\, \mathbf{s}_{h,k},\, \mathbf{u}_{h,k},\, \mathbf{p})\right) + \sum_{k=1}^{L_k} \log\left(p(\mathbf{o}_{h,k} \,|\, \mathbf{s}_{h,k},\, \mathbf{u}_{h,k})\right) \right). \tag{3.35}$$

We need to know the properties of the disturbances $\mathbf{w}_k$ and $\mathbf{v}_k$ in equations 3.23 and 3.24 to optimize this. The parts of the functions $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are deterministic. It is only the probability of the random $\mathbf{w}_k$ and $\mathbf{v}_k$ that we are trying to maximize. By rewriting the equations 3.16 and 3.17 we get

$$\mathbf{w}_{h,k} = \mathbf{s}_{h,k+1} - \mathbf{f}(\mathbf{s}_{h,k},\, \mathbf{u}_{h,k},\, \mathbf{p}), \tag{3.36}$$

$$\mathbf{v}_{h,k} = \mathbf{o}_{h,k} - \mathbf{g}(\mathbf{s}_{h,k},\, \mathbf{u}_{h,k}). \tag{3.37}$$

If we assume that noise $\mathbf{w}_k$ and $\mathbf{v}_k$ have multivariate normal distribution [29] with $\mathbf{0}$ mean and some covariance matrices $\mathbf{\Sigma}_s$ and $\mathbf{\Sigma}_o$, and we also assume that the parameters $\mathbf{p}$ have multivariate normal distribution with $\mathbf{p}_0$ mean (this is the prior for the model parameters) and covariance matrix $\mathbf{\Sigma}_p$, we can rewrite the equation 3.35 as

$$\begin{aligned}(\hat{\mathbf{p}}, \hat{H}_s) = \underset{\mathbf{p},\, H_s}{\arg\max} & -(\mathbf{p} - \mathbf{p}_0)^T \mathbf{\Sigma}_p^{-1} (\mathbf{p} - \mathbf{p}_0) - \\ & - \sum_{h \in H} \left(\sum_{k=1}^{L_h - 1} \mathbf{w}_{h,k}^T \mathbf{\Sigma}_s^{-1} \mathbf{w}_{h,k}^T + \sum_{k=1}^{L_h} \mathbf{v}_{h,k}^T \mathbf{\Sigma}_o^{-1} \mathbf{v}_{h,k} \right) - \\ & - \log\left(\sqrt{(2\pi)^{n_p} \left|\det(\mathbf{\Sigma}_\mathbf{p})\right|} \right) - \\ & - \sum_{h \in H} L_h \log\left(\sqrt{(2\pi)^{n_p} \left|\det(\mathbf{\Sigma}_\mathbf{v})\right|} \right) - \\ & - \sum_{h \in H} (L_h - 1) \log\left(\sqrt{(2\pi)^{n_p} \left|\det(\mathbf{\Sigma}_\mathbf{w})\right|} \right).\end{aligned} \tag{3.38}$$

Turning the sign inside *argmax* we get *argmin*, and we can omit the last 3 terms as they do not have any optimized variables, using weighing vectors $\mathbf{c}_p$, $\mathbf{c}_s$, and $\mathbf{c}_o$ instead of the inverses of the covariance matrices (equivalent if the matrices are diagonals with reciprocal values of the weighing vectors),[7]

[7] We call them *weighing vectors* to denote how much weight each of the terms will carry in the final loss function, i.e., how important each of the terms are.

$$
\begin{aligned}
(\hat{\mathbf{p}}, \hat{H}_s) &= \underset{\mathbf{p},\, H_s}{\arg\min}\, \mathcal{L}(\mathbf{p},\, H_s) \\
\mathcal{L}(\mathbf{p},\, H_s) &= \mathbf{c}_p^T(\mathbf{p} - \mathbf{p}_0)^{\circ 2} + \sum_{h \in H} \left(\sum_{k=1}^{L_h - 1} \mathbf{c}_s^T \left[\mathbf{s}_{h,k+1} - \mathbf{f}(\mathbf{s}_{h,k},\, \mathbf{u}_{h,k},\, \mathbf{p})\right]^{\circ 2} + \right. \\
&\qquad \left. + \sum_{k=1}^{L_h} \mathbf{c}_o^T \left[\mathbf{o}_{h,k} - \mathbf{g}(\mathbf{s}_{h,k},\, \mathbf{u}_{h,k})\right]^{\circ 2} \right).
\end{aligned} \tag{3.39}
$$

Remark 3.23. We are using the Hadamard (elementwise) power, so for matrices $\mathbf{A}$ and $\mathbf{B}$ and their coefficient $[\mathbf{A}]_{i,j}$ and $[\mathbf{B}]_{i,j}$,

$$\mathbf{A} = \mathbf{B}^{\circ 2}, \tag{3.40}$$

$$[\mathbf{A}]_{i,j} = [\mathbf{B}]_{i,j}^2. \tag{3.41}$$

The loss function $\mathcal{L}(\mathbf{p},\, H_s)$ is, in short, trying to minimize the observation error while also trying to minimize the error of the state transitions, the prior of the parameters helps when there is not enough information that could be gained from the trajectories and also helps regularize the variables during the optimization process. How to optimize this function will be discussed in the following section, "Non-linear least-squares optimization".

3.1.4 Moving horizon estimator

In the last subsection, "Parameter identification problem", we arrived at an estimator that estimates not only parameters $\mathbf{p}$ but also all of the state trajectories H_s. The state estimates seemed like a byproduct, but the moving horizon estimator (MHE) [33] uses the same principle of the POMP to get an estimate of the state $\mathbf{s}$ that can be used in control, an alternative to algorithms like the Kalman filter [34] or the extended Kalman filter [35].

The principle of the MHE is that in each time-step it solves non-linear least squares problem (linear if the $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ functions are linear) defined in 3.24.

Definition 3.24. The MHE optimization problem is, with the horizon L and weighing vectors $\mathbf{c}_p$, $\mathbf{c}_o$, $\mathbf{c}_s$, given the history of the inputs $\{\mathbf{u}_k \in \mathcal{U}\}_{k=0}^{L}$, the history of the observations $\{\mathbf{o}_k \in \mathcal{O}\}_{k=1}^{L}$, as well the previous last state estimate $\mathbf{s}_0 \in \mathcal{S}$ and a prior of the parameters $\mathbf{p}_0 \in \mathcal{P}$, we want to optimize

the following problem over the parameters $\mathbf{p}$ and the states $\{\mathbf{s}_k\}_{k=1}^{L}$[33][23],

$$\min_{\mathbf{p},\,\{\mathbf{s}_k\}_{k=1}^{L}} \mathbf{c}_p^T \Delta\mathbf{p}^{\circ 2} + \sum_{k=1}^{L} \mathbf{c}_o^T \Delta\mathbf{o}_k^{\circ 2} + \sum_{k=1}^{L} \mathbf{c}_s^T \Delta\mathbf{s}_k^{\circ 2}, \tag{3.42}$$

$$\Delta\mathbf{p} = \mathbf{p} - \mathbf{p}_0, \tag{3.43}$$

$$\Delta\mathbf{o}_k = \mathbf{o}_k - \mathbf{g}(\mathbf{s}_k,\, \mathbf{u}_k), \tag{3.44}$$

$$\Delta\mathbf{s}_k = \mathbf{s}_k - \mathbf{f}(\mathbf{s}_{k-1},\, \mathbf{u}_{k-1},\, \mathbf{p}). \tag{3.45}$$

Remark 3.25. We are only estimating L states, state $\mathbf{s}_0$ from the previous step is reused as prior. In most uses of the MHE we also change the prior $\mathbf{p}_0$ to the last estimated parameters $\mathbf{p}$. In a situation where the disturbances do not follow our assumption of normality and zero mean, this is not recommended, as this would make the estimator biased and inconsistent, shifting the prior in each iteration, thus hindering the controller's performance.

We can use the solution of this problem, mainly the last term $\mathbf{s}_L$, which is the current estimated state of the system and the estimated parameters $\mathbf{p}$ for the control of our system.

It would seem redundant to estimate the model parameter $\mathbf{p}$, but this allows us to deal with parameter drift, a phenomenon where the parameters of the model might change compared to the initial measurements, for example, when a battery of a drone that runs low, comparatively, makes the lift of the drone lower with the same inputs [23].

3.1.5 Model predictive control

Now that we have gotten a relatively good estimate of how our system behaves (estimated $\mathbf{p}$ for the function $\mathbf{f}(\cdot)$) and also roughly know in what state it is (estimated $\mathbf{s_k}$), we can use this information to control our dynamic system.

The model predictive control (MPC) [36][37] is the counterpart to the MHE. As the MHE was looking into the past, the MPC will, figuratively, look into the future. This is a control technique, an alternative to the proportional-integral-derivative (PID) controller [24][38] or the linear-quadratic-regulator (LQR) [39][40], giving some input $\mathbf{u}$ to get our system from our current state $\mathbf{s}_0$ to some desired state $\mathbf{s}^*$.

The PID or the LQR techniques require a linear $\mathbf{f}(\cdot)$, or a linearized version is used. This is one of the main advantages of the MPC controller, it does

not impose such requirements on the state function $\mathbf{f}(\cdot)$, we can work with the non-linear function directly.

Definition 3.26. The MPC optimization problem is, with the horizon L, weighing vectors $\mathbf{c}_u$, $\mathbf{c}_s$ and $\mathbf{c}_t$, given state-space model with $\mathbf{f}(\cdot)$ and $\mathbf{p} \in \mathcal{P}$, last input $\mathbf{u}_{-1} \in \mathcal{U}$, initial state $\mathbf{s}_0 \in \mathcal{S}$ and a target state $\mathbf{s}^* \in \mathcal{S}$, we want to optimize the following problem over the set of actions $\{\mathbf{u}_k \in \mathcal{U}\}_{k=0}^{L-1}$ [36][23],

$$\min_{\{\mathbf{u}_k\}_{k=0}^{L-1}} \overbrace{\sum_{k=0}^{L-1} \mathbf{c}_u^T \Delta \mathbf{u}_k^{\circ 2}}^{\text{action term}} + \overbrace{\sum_{k=1}^{L-1} \mathbf{c}_s^T \Delta \mathbf{s}_k^{\circ 2}}^{\text{stage term}} + \overbrace{\mathbf{c}_t^T \Delta \mathbf{s}_L^{\circ 2}}^{\text{terminal term}}, \tag{3.46}$$

$$\Delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}, \tag{3.47}$$

$$\Delta \mathbf{s}_k = \mathbf{s}_k - \mathbf{s}^*, \tag{3.48}$$

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k, \mathbf{u}_k, \mathbf{p}). \tag{3.49}$$

Remark 3.27. From equation 3.49, you can see there is a strict recursive relationship between the states, so each state $\mathbf{s}_k$ is a function of the initial state $\mathbf{s}_0$ and the previous inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$.

Now, we take an overview of what each of the terms in the optimization is trying to do. The first action term is trying to minimize the differences between consecutive actions,[8] making our controller more steady. Fast changes in the input would often disrupt our model (for example, going from a standstill to fast speed could cause a wheel slip, it is better to accelerate more steadily).

The second state term is there to help move our system along the trajectory to the target or possibly to reduce velocities if they were part of the states vector. The last term is the terminal term, which usually has higher weights as we want to be as close to the target as possible at the end of our horizon.

[8] Often the MPC action term is defined in absolute values of the actions $\mathbf{u}$ and not their differences, this is to minimize the inputs and save energy.

3.2 Non-linear least-squares optimization

In equations 3.39, 3.42, and 3.46, we have developed three non-linear least-squares optimization problems. This section will explain the methods used to solve these problems.

3.2.1 Modeling

The non-linear least-squares optimization problem [41] is a problem, where we are minimizing the objective

$$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2 = \min_{x_1,\ldots,x_m} \frac{1}{2}\sum_{i=1}^{n} r_i(x_1, \ldots, x_m)^2, \tag{3.50}$$

where $\mathbf{r} : \mathbb{R}^m \to \mathbb{R}^n$ is a vector residual function composed of residual functions $\mathbf{r} = \begin{bmatrix} r_1(\mathbf{x}) & \ldots & r_n(\mathbf{x}) \end{bmatrix}^T$, each with the arguments $\mathbf{x} = \begin{bmatrix} x_1 & \ldots & x_m \end{bmatrix}^T$, which are the optimization variables. Sometimes there are lower bound lb_j and upper bounds ub_j conditions imposed on the variables $lb_j \leq x_j \leq ub_j$.

Jacobian. Most of the numerical methods used to solve these problems make the use of the Jacobian matrix $\mathbf{J}_\mathbf{r}(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^{n\times m}$ [42],

$$\mathbf{J}_\mathbf{r}(\mathbf{x}) = \begin{bmatrix} \frac{\partial r_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial r_1(\mathbf{x})}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_n(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial r_n(\mathbf{x})}{\partial x_m} \end{bmatrix}. \tag{3.51}$$

3.2.2 Solving

When trying to find a solution to the equation $r(x) = 0$, we could, sometimes, get an analytical solution. Oftentimes, we are unable to get an analytical (closed-form) solution, and we have to resort to numerical methods.

Gauss-Newton method. The most basic numerical method for this kind of problem is the Gauss-Newton method [43]. In each step we approximate

the function with a line and solve for the intercept with the x-axis,

$$r(x) \approx r(x_k) + \frac{\partial r(x_k)}{\partial x}\Delta x_{gn}, \tag{3.52}$$

$$0 = r(x_k) + \frac{\partial r(x_k)}{\partial x}\Delta x_{gn}, \tag{3.53}$$

$$\Delta x_{gn} = \left(\frac{\partial r(x_k)}{\partial x}\right)^{-1} r(x_k), \tag{3.54}$$

$$x_{k+1} = x_k + \Delta x_{gn} \tag{3.55}$$

For the multidimensional case of $\mathbf{r}(\mathbf{x}) = \mathbf{0}$

$$\mathbf{r}(\mathbf{x}) \approx \mathbf{r}(\mathbf{x}_k) + \mathbf{J}_\mathbf{r}(\mathbf{x}_k)\Delta\mathbf{x}_{gn}, \tag{3.56}$$

$$\mathbf{0} = \mathbf{r}(\mathbf{x}_k) + \mathbf{J}_\mathbf{r}(\mathbf{x}_k)\Delta\mathbf{x}_{gn}. \tag{3.57}$$

$$\tag{3.58}$$

Remark 3.28. Since solving for $\mathbf{r}(\mathbf{x}) = \mathbf{0}$ is usually impossible (overdetermined system of equations), its better to think about the optimization as trying to get as close as possible to the solution,

$$\min_\mathbf{x} L(\mathbf{x}) = \min_\mathbf{x} \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2. \tag{3.59}$$

The major problem with the Gauss-Newton method is the inverse of the derivative. First, if it is small, we will overshoot and possibly diverge. Second, the Jacobian $\mathbf{J}$ usually is not square (if $n \neq m$), so inverting directly might not be possible).

Remark 3.29. From now on, we will use simplified notating of $\mathbf{J} = \mathbf{J}_\mathbf{r}(\mathbf{x}_k)$ and $\mathbf{r} = \mathbf{r}(\mathbf{x}_k)$, denoting Jacobian and the value of the residual function at each step.

To combat the non-square $\mathbf{J}$, we can formulate the equations as a pseudo-inverse [44],

$$\mathbf{J}^T\mathbf{J}\Delta\mathbf{x}_{gn} = -\mathbf{J}\mathbf{r}. \tag{3.60}$$

Levenberg-Marquardt method. An improvement to combat the overshooting was suggested by Levenberg (1944) [45], later improved on by Marquardt (1963) [46], to use the dampening term $\mu\mathbf{I}$ [47],[9] adjusting the equation to

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\Delta\mathbf{x}_{lm} = -\mathbf{J}^T\mathbf{r}. \tag{3.61}$$

[9]The identity matrix is denoted $\mathbf{I}$. It has an appropriate dimension $m \times m$.

The dampening parameter μ has several effects:

- For all $\mu > 0$, the coefficient matrix is positive-definite. This ensures that $\Delta\mathbf{x}$ is a descent direction.
- For large values of μ we get $\Delta\mathbf{x}_{lm} \approx -\frac{1}{\mu}\mathbf{J}$, i.e. a short step in the steepest direction (this is similar to a gradient descent step [48]). This is good if the current iterate is far from the solution.
- If μ is very small, then $\Delta\mathbf{x}_{lm} \approx \Delta\mathbf{x}_{gn}$, which is a good step in the final stages of the iteration when the current solution $\mathbf{x}_k$ is close to the optimal solution $\mathbf{x}^*$. If $\mathbf{r}(\mathbf{x}^*) = \mathbf{0}$ (or very small), then we get (almost) quadratic final convergence.

(K. Madsen [47])

Remark 3.30. The solution to equation 3.61 can be rewritten as ordinary least squares [47][49],

$$\min_{\Delta\mathbf{x}} \frac{1}{2}\left\| \begin{bmatrix} \mathbf{J}_\mathbf{r}(\mathbf{x_k}) \\ \sqrt{\mu}\mathbf{I} \end{bmatrix} \Delta\mathbf{x} + \begin{bmatrix} \mathbf{r}(\mathbf{x}_k) \\ \mathbf{0} \end{bmatrix} \right\|^2 . \tag{3.62}$$

The choice of μ in each step can greatly improve the required number of iterations. The initial value of μ should be related to the size of the elements in $\mathbf{H}_0 = \mathbf{J}_0^T\mathbf{J}_0$,

$$\mu_0 = \tau \max_i \left[\mathbf{H}_0\right]_{i,i} . \tag{3.63}$$

The parameter τ is given by the user.[10] To update the μ we can calculate the gain ration ρ [50],

$$\rho = \frac{\|\mathbf{r}(\mathbf{x}_k + \Delta\mathbf{x})\|^2 - \|\mathbf{r}(\mathbf{x}_k)\|^2}{\|\mathbf{J}_\mathbf{r}(\mathbf{x}_k)\Delta\mathbf{x} + \mathbf{r}(\mathbf{x}_k)\|^2 - \|\mathbf{r}(\mathbf{x}_k)\|^2}. \tag{3.64}$$

- If $\rho > \varepsilon$, then $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}$, else $\mathbf{x}_{k+1} = \mathbf{x}_k$. This is to ensure that we improve the objective. If not, we are not taking the step.
- If $\rho > \eta_1$, then $\mu_{k+1} = 2\mu_k$. Large ρ signifies that the current solution $\mathbf{x}_k$ is very far from optimal solution $\mathbf{x}^*$, we want to take a steepest direction step.
- Else if $\rho < \eta_2$, then $\mu_{k+1} = \frac{1}{2}\mu_k$. Small ρ signifies that we are close to the solution $\mathbf{x}^*$, we want to take a Gauss-Newton step.
- Else $\mu_{k+1} = \mu_k$.

[10] For good starting positions of $\mathbf{x}_0$, $\tau = 1e^{-6}$, increasing if the confidence in the initial guess is low [47].

With the iteration of the algorithm defined, we need to determine the termination of the algorithm. The first used termination condition is function change tolerance [51], this is to most common termination,

$$\frac{|L(\mathbf{x}_k) - L(\mathbf{x}_{k-1})|}{L(\mathbf{x}_k)} \leq tol_f. \tag{3.65}$$

We can also define the parameter tolerance termination, this is useful when the value of $L(\mathbf{x}_k)$ is very low [47],

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}_k\|} \leq tol_x. \tag{3.66}$$

There can also be external terminations, like the maximum number of iterations or maximum computation time.

Factorization

In each of the iterations of the optimization, we are solving an ordinary least-squares problem. Since we might need many iterations and the problem could be quite large, we need an efficient solution [47]. This is done by factorization of the Jacobian $\mathbf{J}$.[11]

QR decomposition. Since we are trying to solve the standard least-squares problem,

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2, \tag{3.67}$$

we can rewrite the solution (assuming the system of equations is not underdetermined) as

$$\mathbf{A}^T\mathbf{A}\mathbf{x}^* = \mathbf{A}^T\mathbf{b}, \tag{3.68}$$

to solve this matrix equation, we would need the inverse of $\mathbf{A}^T\mathbf{A}$ (matrix $\mathbf{A}$ needs to have full rank). Directly calculating the inverse is very complicated, Gram (1883) [52] and Schmidt (1907) [53] proposed the QR decomposition method to decompose the $\mathbf{A} \in \mathbb{R}^{n\times m}$ matrix into factors, [12]

$$\mathbf{A} = \mathbf{Q}\mathbf{R}, \tag{3.69}$$

[11] For Levenberg-Marquardt method this is already the extended Jacobian $\begin{bmatrix} \mathbf{J_r}(\mathbf{x}_k)^T & \sqrt{\mu}\mathbf{I} \end{bmatrix}^T$.

[12] Sometimes this method is called the Gram-Schmidt method after its two authors.

where $\mathbf{Q} \in \mathbb{R}^{n \times m}$ is a matrix with orthonormal columns and $\mathbf{R} \in \mathbb{R}^{m \times m}$ is upper triangular matrix [44], we can rewrite the equation 3.68 as

$$(\mathbf{QR})^T \mathbf{QR}\mathbf{x}^* = \mathbf{R}^T \mathbf{Q}^T \mathbf{QR}\mathbf{x}^* = \mathbf{R}^T \mathbf{Q}\mathbf{b} = (\mathbf{QR}^T)\mathbf{b}. \tag{3.70}$$

Now we use the fact that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and that $\mathbf{R}$ has rank n,

$$\mathbf{R}\mathbf{x}^* = \mathbf{Q}^T\mathbf{b}. \tag{3.71}$$

We can solve this system of equations by backward substitution, as $\mathbf{R}$ is upper triangular.

Cholesky decomposition. If the number of rows n is much bigger than the number of columns m of the matrix $\mathbf{A}$, it might be more efficient to calculate the Cholesky decomposition [54] of the Hermitian $\mathbf{H} = \mathbf{A}^T\mathbf{A} = \mathbf{R}^T\mathbf{R}$, where $\mathbf{R} \in \mathbb{R}^{m \times m}$ is upper triangular. We can then use this to solve the equation,

$$\mathbf{A}^T\mathbf{A}\mathbf{x}^* = \mathbf{R}^T\mathbf{R}\mathbf{x}^* = \mathbf{A}^T\mathbf{b}, \tag{3.72}$$

$$\mathbf{R}^T\mathbf{w} = \mathbf{A}^T\mathbf{b}, \tag{3.73}$$

$$\mathbf{R}\mathbf{x}^* = \mathbf{w}. \tag{3.74}$$

Remark 3.31. The $\mathbf{R}$ matrix in the Cholesky method (equations 3.72, 3.73 and 3.74) is the same as with the QR method (equations 3.70 and 3.71) [51][41].

So to get a solution we first back-substitute to equation 3.73 to get $\mathbf{w}$ and again equation 3.74. So the trade-off between QR and Cholesky decomposition is that, QR is more numerically stable and only requires one back-substitution, but Cholesky decomposition is faster, especially for $n >> m$ [55][56].

Sparse matrices. In most problems, the Jacobian matrix $\mathbf{J}$ is very sparse (most of the entries in the matrix are zero), because only a few variables x_j are arguments of the residual $r_i(\cdot)$.

Because of this, we can use the sparse matrix system where instead of saving all the zeros in the computer memory, we only save the value and position of the non-zero elements.

This can be leveraged by algorithms for sparse matrices, significantly improving memory and computation requirements, namely the sparse Cholesky decomposition [57], allowing us to solve huge least-squares problems.

3.2.3 Deriving Jacobians

The final Jacobian is going to be very sparse, we only need to derive the Jacobian for variables directly influencing the residual function, the rest are going to be zero.

MHE and model identification. For the first term, the parameter prior, the Jacobian is very simple,

$$\frac{\partial \Delta \mathbf{p}}{\partial \mathbf{p}} = \mathbf{I}_{n_p \times n_p}. \tag{3.75}$$

For the observation terms, the only variable there is the state $\mathbf{s}_k$,

$$\frac{\partial \Delta \mathbf{o}_k}{\partial \mathbf{s}_k} = -\frac{\partial \mathbf{g}(\mathbf{s}_k, \mathbf{u}_k)}{\partial \mathbf{s}_k}. \tag{3.76}$$

For the state transition terms, there are 3 variables, $\mathbf{s}_{k+1}$, $\mathbf{s}_k$, $\mathbf{p}$,

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{s}_k} = \mathbf{I}_{n_s \times n_s}, \tag{3.77}$$

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{s}_{k-1}} = -\frac{\partial \mathbf{f}(\mathbf{s}_{k-1}, \mathbf{u}_{k-1}, \mathbf{p})}{\partial \mathbf{s}_k}, \tag{3.78}$$

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{p}} = -\frac{\partial \mathbf{f}(\mathbf{s}_{k-1}, \mathbf{u}_{k-1}, \mathbf{p})}{\partial \mathbf{p}}. \tag{3.79}$$

MPC. For the model predictive control problem, we are going to derive the action term, and the state terms (stage and terminal terms are the same, only scaled). The Jackobian blocks for the action terms are

$$\frac{\partial \Delta \mathbf{u}_k}{\partial \mathbf{u}_k} = \mathbf{I}_{n_u \times n_u}, \tag{3.80}$$

$$\frac{\partial \Delta \mathbf{u}_k}{\partial \mathbf{u}_{k-1}} = -\mathbf{I}_{n_u \times n_u}. \tag{3.81}$$

For the state terms, we will need to use the chain rule to calculate the Jacobians, each state $\mathbf{s}_k$ is a function of the initial state $\mathbf{s}_0$ and the previous inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$, so for $k > i \geq 0$,

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{u}_i} = \frac{\partial \mathbf{f}(\mathbf{s}_{k-1}, \mathbf{u}_{k-1}, \mathbf{p})}{\partial \mathbf{s}_{k-1}} \cdots \frac{\partial \mathbf{f}(\mathbf{s}_{i+1}, \mathbf{u}_{i+1}, \mathbf{p})}{\partial \mathbf{s}_{i+1}} \frac{\partial \mathbf{f}(\mathbf{s}_i, \mathbf{u}_i, \mathbf{p})}{\partial \mathbf{u}_i}. \tag{3.82}$$

Chapter 4

Technical solution

This chapter will explain the details of the technical solution to the initial problem of identification and control of a robotic helicopter. It will give basic description of the hardware used, introduce the model used for our dynamic system and describe the software solution in detail.

4.1 Hardware

4.1.1 Robotic helicopter Tello

The robotic helicopter we are going to identify and control is the Tello drone by Ryze [58], figure 4.1 shows a photograph of the drone with attached markers. It is a lightweight drone (only 87 grams), with internal regulators that use internal vision positioning system to stabilize itself. Its control inputs are roll, pitch, yaw and throttle, which have the same meaning as in the introduction to the quadrocopter (section 1.1.1). The inputs are target velocities, and the internal regulators of the drone adjust the rotor speeds to match the inputs, corrected by the vision positioning system.

Figure 4.1: Tello drone with tracking markers

4.1.2 Motion capture system Vicon

To localize the drone, we are using the Vicon motion capture system. The principle is that many infra-red cameras (figure 4.2 show a photograph of said camera) on the edges of the tracking area are capturing the reflections of the markers, small reflective silver balls, and using these pictures, and positions of the cameras it extracts the positions of the markers.

Figure 4.2: Vicon infra-red camera

4.2 Drone model

> Truth...is much too complicated to allow anything but approximations. (John von Neumann [59])

We are using a first-order model of the robot helicopter system. This is because the internal regulators of the drone accept inputs as scaled target velocities. The model equations are then the following,

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \lambda_h \sin(\psi(t)) & \lambda_h \cos(\psi(t)) & 0 & 0 \\ -\lambda_h \cos(\psi(t)) & \lambda_h \sin(\psi(t)) & 0 & 0 \\ 0 & 0 & 0 & \lambda_v \\ 0 & 0 & \lambda_a & 0 \end{bmatrix} \begin{bmatrix} roll(t) \\ pitch(t) \\ yaw(t) \\ throttle(t) \end{bmatrix}, \tag{4.1}$$

$$\psi(t) = \theta(t) + \theta_0 \tag{4.2}$$

The states of the model are:

- $x(t)$ [m]: positing in the x-axis
- $y(t)$ [m]: positing in the y-axis
- $z(t)$ [m]: position in the z-axis (vertical)
- $\theta(t)$ [rad]: the orientation angle around the z-axis

The outputs (observations) of the system are the same as the states.

The inputs are:

- $roll(t) \in [-1, 1]$: scaled horizontal velocity moving left-right
- $pitch(t) \in [-1, 1]$: scaled horizontal velocity moving forward-backward
- $yaw(t) \in [-1, 1]$: scaled angular velocity
- $throttle(t) \in [-1, 1]$: scaled vertical velocity

The parameters of the model are:

- λ_h [m/s]: horizontal velocity coefficient
- λ_v [m/s]: vertical velocity coefficient
- λ_a [rad/s]: angular velocity coefficient
- θ_0 [rad]: heading offset, the difference between the heading of the drone and the measured orientation

4.2.1 Discretization

Since we are measuring the system positions in discrete time-steps, and we can also only control the drone in discrete time-steps, it makes sense to discretize our model using the Euler method (section 3.1.2) with time-step Δt, the discrete state-space model we will be using then is

$$\mathbf{s}_{k+1} = \mathbf{s}_k + \Delta t \overbrace{\begin{bmatrix} [\mathbf{p}]_1 \sin(\psi_k) & [\mathbf{p}]_1 \cos(\psi_k) & 0 & 0 \\ -[\mathbf{p}]_1 \cos(\psi_k) & [\mathbf{p}]_1 \sin(\psi_k) & 0 & 0 \\ 0 & 0 & 0 & [\mathbf{p}]_2 \\ 0 & 0 & [\mathbf{p}]_3 & 0 \end{bmatrix}}^{\mathbf{f}(\mathbf{s}_k, \mathbf{u}_k, \mathbf{p})} \mathbf{u}_k, \tag{4.3}$$

$$\psi_k = [\mathbf{s}_k]_4 + [\mathbf{p}]_4 \tag{4.4}$$

$$\mathbf{o}_k = \mathbf{s}_k, \tag{4.5}$$

where

$$\mathbf{s}_k = \begin{bmatrix} x(k) & y(k) & z(k) & \theta(k) \end{bmatrix}^T, \tag{4.6}$$

$$\mathbf{u}_k = \begin{bmatrix} roll(k) & pitch(k) & yaw(k) & throttle(k) \end{bmatrix}^T, \tag{4.7}$$

$$\mathbf{p} = \begin{bmatrix} \lambda_h & \lambda_v & \lambda_a & \theta_0 \end{bmatrix}^T. \tag{4.8}$$

4.2.2 Input delay

We are introducing input delay to our model. This means that for delay D a sent action (command) $\mathbf{a}_k \in \mathcal{U}$ at time-step k is going an effective system input $\mathbf{u}_{k+D}$ at time-step $k + D$.

4.3 Software

This section will describe the practical part of the master thesis, the software solution to track and control the robotic helicopter. All of the source code and data, including instructions for building and running the programs, are accessible from `https://github.com/svrc-jan/master-thesis/`, the tree of the repository for the most relevant files is in appendix B.

Both the model identification and the control loop programs are written in the `C++` language [60], with emphasis on computation speed and re-usability for other dynamic models using templates. Data visualization and model identification bootstrap for estimator variance is done using `Python` language scripts [61], for visualization namely the `matplotlib` library [62].

4.3.1 Tello communication

We use a separate Wi-Fi connection [63] to communicate with the drone. The basic object for communicating with the Tello drone, based on the Tello SKD [64] was provided by the thesis supervisor.

It provides an interface for establishing a UDP [65] connection to the drone using Wi-Fi network interface, as well as basic commands to take-off, land and set target inputs of roll, pitch, yaw, and throttle (*target* input because of the internal regulators of the drone using the internal vision positioning system).

Connection is done through a network interface (Wi-Fi adapter) connected to the drone, the IP address of the drone is `192.168.10.1`, and the port used is `8889`, as described in the Tello SDK.

4.3.2 Vicon communication

To get positional data we use the motion tracking system Vicon. The tracking software is running on a separate computer, and the data are being sent over to the control computer using our local network. This setup is provided by the thesis supervisor.

The Vicon Tracker application, running on the server computer, publishes data to *localhost*. Using the Vicon API, described in Vicon DataStream SDK, we can set up a client application to recieve selected data in real-time. Using a local area network (LAN), we establish a TCP [66] connection from the server computer (tracking) to the client computer (controlling), by default on the port `51602`, we then send over real-time data for the selected object(s).

Since the communication is asynchronous and blocking (waiting for the server to send data), it is run in a separate thread, managed by a handler, which opens and closes the connection and provides the latest position from the Vicon system.

4.3.3 Vicon filter

The data from the tracking software is not perfect. To combat the issues of data provided by the Vicon system, we have developed a filter.

The first problem is the angle wrapping of the orientation angle. Since angles are wrapped to $(-\pi, \pi]$ but we want to have a continuous signal when we go above or below the bounds, the following unwrapping formula for the angle θ (θ_{raw} is the value from Vicon) around the reference θ_{ref} (current filtered angle) is used,

$$\theta_{unwrap} = \theta_{ref} + ((\theta_{raw} - \theta_{ref} + \pi) \bmod 2\pi) - \pi. \tag{4.9}$$

The second problem is that the data from the Vicon system is noisy. There are erroneous readings, outliers that would incorrectly shift our position estimation and thus hinder our controller's performance. The tracking imperfections are exacerbated by the fact that our tracked object is relatively small (maximum distance between the markers is less than 10cm), and some of the markers are not clearly visible from all angles (rotor guards, propellers, or the drone's body are often in the way).

To avoid this, there are thresholds thr_h, thr_v and thr_a each limiting how big of a difference $\Delta x = x_{raw} - x_{ref}$, between the current raw observation from Vicon $\mathbf{o}_{raw}$ and the reference $\mathbf{o}_{ref}$ is valid, we also know minimal altitude so we filter the minimal valid z_{floor}.

- If $\|\Delta x, \Delta y\| \leq thr_h$ and $|\Delta z| \leq thr_v$ and $z \geq z_{floor}$ then $\mathbf{o}_{filt} \leftarrow \mathbf{o}_{raw}$.
  - If $|\theta_{unwrap} - \theta_{ref}| > thr_a$ then $\theta_{filt} \leftarrow \theta_{ref}$.
- Else give reference observation $\mathbf{o}_{filt} \leftarrow \mathbf{o}_{ref}$.

Because the orientation θ values are much more prone to erroneous readings, we do the angle filtering in a separate step. If we were to use the angle conditions with the translation conditions, we would lose a lot of observations with the correct position but an incorrect orientation.

After each step of the filtering, we save the output of the filter as the new reference $\mathbf{o}_{k,ref} \leftarrow \mathbf{o}_{k-1,filt}$. The currently used threshold values are adaptive, growing with the number of steps that the filter is "holding" the reference n_{hold} (not changing the values), this is to ensure that if a long period of erroneous readings happens, the filter can catch up to the new values that might be far from the current reference,

$$thr_h = 0.5(3 + n_{hold}), \tag{4.10}$$
$$thr_v = 0.5(3 + n_{hold}), \tag{4.11}$$
$$thr_a = 0.05(3 + n_{hold}). \tag{4.12}$$

4.3.4 Logging

To record the trajectories for analysis and system identification, we created a simple logger, saving data in comma-separated-values (CSV) format, saving tag (position, input or target), time-step, and values, an example:

```
pos,0,-0.9077,-2.0248,1.1274,0.0305
input,0,-0.0407,-0.0627,-0.0318,0.1484
target,0,-1.0000,-2.0000,1.3000,0.0000
pos,1,-0.9081,-2.0243,1.1291,0.0286
input,1,-0.0400,-0.0627,-0.0280,0.1457
target,1,0.0000,-2.0000,1.6000,0.0000
pos,2,-0.9081,-2.0243,1.1291,0.0286
input,2,-0.0387,-0.0607,-0.0274,0.1443
target,2,0.0000,-2.0000,1.6000,0.0000
```

We also created a complimentary parser to load selected log values into an `Eigen` matrix [67] (`Eigen` is a `C++` library for linear algebra).

4.3.5 Ceres solver

We are using the open-source `Ceres` solver [51] developed by Google to model and solve the non-linear least-squares optimization problems. Main advantages of using this solver are:

- The ability to handle large problems (our parameter identification problem has over a hundred thousand optimization variables and residuals, each iteration of the optimization takes mere seconds to compute) leveraging the sparsity of the Jacobian.
- Automatic differentiation of the Jacobian. If required, manual or numerical options are also available.
- Re-usability of the model (both MPC and MHE build the model only once and its being rerun with new data at 50 Hz frequency on a personal computer almost without delay).
- There is a wide variety of solver features, such as factorization methods (QR, Cholesky, and their sparse versions), optimization methods (trust-region or line-search), and multi-threading, available, making it suitable for almost any problem that can be formulated within the framework.

During the initial stages of the thesis, some alternatives were tried out. The best alternative was the `Acados` framework [68], which required heavy-handed definitions for the `CasADI` solver when using `C++` language. The `Ceres` solver only requires a template residual function definition (example of a residual block for state term for model identification is listing 4.2) and pointers to where the variables are to be saved.

Most of the other alternatives either did not support non-linear models, were too slow due to using Python or numerical differentiation, or required licensed optimization solvers.

4.3.6 Model template

To make the system identification, MHE and MPC parts of the code reusable for other models, all of them use template functions and classes. The model template specifies the input n_u, output n_o, state n_s and parameter dimensions n_p, the lower and upper bounds for the inputs and parameter and template

state $\mathbf{f}(\cdot)$ and output functions $\mathbf{g}(\cdot)$ only using basic arithmetic and functions defined in the standard math library (such as sin, cos and pow) making it suitable for the automatic differentiation, the source code for our state equation function in listing 4.1 is equivalent to the drone state equation 4.1.

```
template<typename Tds, typename Ts,
         typename Tu, typename Tp>
bool Simple_drone_model::state_eq(
    Tds *ds, const Ts *s, const Tu *u, const Tp *p)
{
    ds[0] = Tds(p[0]*(cos(s[3]+p[3])*u[1]
                + sin(s[2]+p[3])*u[0]));
    ds[1] = Tds(p[0]*(sin(s[3]+p[3])*u[1]
                    - cos(s[2]+p[3])*u[0]));
    ds[2] = Tds(p[1]*u[3]);
    ds[3] = Tds(p[2]*u[2]);

    return true;
}
```

Listing 4.1: C++ code for the state equation

4.3.7 System identification

In the last chapter, section 3.1.3 we formulated the MAP estimator to estimate the model parameters. We are going to be be using the non-linear least-squares form in equation 3.39, with the adjustment that we divide the discretized equation 4.3 by Δt. This is so we do not have to re-scale the weighing coefficient if we change the discretization step.

The logs of the trajectories used for identification are provided in the aforementioned CSV format and problem specifics are in a provided JSON configuration file, full list of configuration options in appendix C.1. Since we are accounting for the input delay, the program for the identification shifts the inputs accordingly, padding the start with zero inputs.

One thing to speed up the computation is to provide the initial solution, so we set the states to the observations and the parameters to its priors.

To verify if the states' estimations are valid (trajectory is following the observations but not over-fitting to outliers), all of the states' estimates are saved.

Jacobian derivation. The Jacobian is differentiated automatically but to verify that it is well defined for all values, we will derive it manually,

$$\frac{\partial \Delta \mathbf{o}_k}{\partial \mathbf{s}_k} = -\mathbf{I}_{4\times 4}, \tag{4.13}$$

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{s}_k} = \frac{\mathbf{I}_{4\times 4}}{\Delta t}, \tag{4.14}$$

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{s}_{k-1}} = -\begin{bmatrix} \frac{1}{\Delta t} & 0 & 0 & -\lambda_h q_2(k-1) \\ 0 & \frac{1}{\Delta t} & 0 & \lambda_h q_1(k-1) \\ 0 & 0 & \frac{1}{\Delta t} & 0 \\ 0 & 0 & 0 & \frac{1}{\Delta t} \end{bmatrix}, \tag{4.15}$$

$$\frac{\partial \Delta \mathbf{s}_k}{\partial \mathbf{p}} = -\begin{bmatrix} q_1(k-1) & 0 & 0 & -\lambda_h q_2(k-1) \\ q_2(k-1) & 0 & 0 & \lambda_h q_1(k-1) \\ 0 & throttle(k-1) & 0 & 0 \\ 0 & 0 & yaw(k-1) & 0 \end{bmatrix}, \tag{4.16}$$

$$q_1(k) = \cos(\psi(k))pitch(k) + \sin(\psi(k))roll(k), \tag{4.17}$$

$$q_2(k) = \sin(\psi(k))pitch(k) - \cos(\psi(k))roll(k), \tag{4.18}$$

$$\psi(k) = \theta(k) + \theta_0. \tag{4.19}$$

Since the discretization step Δt is non-zero and positive, the calculated values of the Jacobian are always finite, and it is continuous for all states, inputs and model parameters, essential conditions for the numerical stability of the optimization.

Source code. To demonstrate how the mathematical model transitions to the source code, we provide the implementation of the state `Ceres` residual block in listing 4.2, this is equivalent to the state term in equation 3.39.

The residual block is created with pointers to the data, as such it can be rerun without rebuilding the model (for example, with new input $\mathbf{u}$), and the template residual function takes the variable and residual pointers as arguments. As you can see in listing 4.2, the template of the model `M` provides the state equation and all the required dimensions (sizes). As such this residual block can be used for any finite state-space model.

```
template<typename M>
struct State_res
{
    State_res(const double *u, const double dt,
        const double *C) : u(u), dt(dt), C(C) {}

    template <typename T>
    bool operator()(const T* const s_curr,
        const T* const s_next,
        const T* const p, T* res) const
    {
        T ds[M::s_dim];
        M::state_eq(ds, s_curr, this->u, p);

        for(int i = 0; i < M::s_dim; i++) {
            res[i] = this->C[i]*(ds[i] +
                (s_curr[i] - s_next[i])/this->dt);
        }

        return true;
    }

    static CostFunction* Create(
        const double* u, const double dt, const double *C)
    {
        return (new AutoDiffCostFunction<State_res,
            M::s_dim, M::s_dim, M::s_dim, M::p_dim>
        (
            new State_res(u, dt, C))
        );
    }

    const double *u;
    const double dt;
    const double *C; // cost multiplier
};
```

Listing 4.2: C++ code for the state Ceres residual block

4.3.8 Moving horizon estimate

To estimate the model state we are using the MHE as defined in the previous chapter, section 3.1.4, specifically equation 3.42. Because we can rerun the optimization model in `Ceres`, at the start of the control program, we build the MHE problem for a fixed horizon length specified in the configuration (full list of configuration options in appendix C.2).

Since it takes a non-trivial amount of time to solve the optimization problem, the estimation is done in a handler thread to prevent blocking the main

thread. At each step in the main thread, we first get the last calculated state estimate $\hat{\mathbf{s}}_{k-1}$ from the handler, which is going to be used for controlling the system, and after that we post a request to the handler with the current observation $\mathbf{o}_k$, and the previous input $\mathbf{u}_{k-1}$. This introduces a delay of one time-step into the control loop, but it is necessary to deal with the real-time limitations of the optimization.

The handler thread is waiting for the request, checking if the time-step is greater than the current solution. If yes, it adds data (inputs and observations) from all unprocessed requests in the queue after shifting the previous data and solution. The fact that we are reusing the previous estimates as the initial solution of the optimization greatly helps to improve the computation speed and allows us to get a good solution within the 20 millisecond sampling period.

If we do not get a solution fast enough, we just reuse the last state estimate value. Since we can set a computation time limit, it is almost guaranteed to get a solution within the required time frame, though it is important to balance the horizon length so that the precision of the solution is sufficient. Longer horizon might produce better, more robust estimates, but will also take longer to compute, so we might not do enough solver iterations to get a sufficiently precise solution.

Jacobian derivation. The Jacobian formulas are the same as for the system identification, equations 4.13 to 4.19.

4.3.9 Model predictive control

To control our system, we are using the MPC as defined in the previous chapter, section 3.1.5, specifically equation 3.46. We can leverage the same advantages of the `Ceres` solver as with MHE, rerunning the model that is built for a specific horizon length (full list of configuration options in appendix C.3) and reusing the previous solution. In the same manner as the MHE, the MPC optimization is run in a separate handler to prevent blocking the main thread.

As with the MHE, the main thread is first getting the calculated input $\mathbf{u}$ for the current step that is used for controlling the system and then we post a request with the new $\mathbf{s}_0$, $\mathbf{p}$, and $\mathbf{s}^*$. The handler thread also works similarly, shifting the previous solution and handling the last posted request.

Lagged inputs. Since solving the MPC is more demanding than the MHE (Jacobian is less sparse; state is influenced by all previous inputs), it happens more often that the computation does not converge withing the allocated time. Normally we only use the first input when calculating the MPC, but in this case we can use the later inputs as well. However, to prevent the system from moving too far from the initial solution, we scale the inputs by a coefficient $\alpha = \exp(-lag_{mpc}/10)$, so if the control is not keeping up, we slow down. It is important to balance the horizon length with the computational capacity available to prevent the MPC from lagging behind.

Target clipping. Since our horizon is limited, if we were to get a target state $\mathbf{s}^*$ far from our current state, we would not be able to get to it within our horizon, which would cause the inputs to dramatically increase. We would have to increase the input weights to balance it out, but that would make the system too slow when we are close to our target. Because of this, it is better to clip the distance to our target by some value d_{max},

$$\mathbf{s}^*_{clip} = \min(d_{max}, \|\mathbf{s}^* - \mathbf{s}_0\|)\frac{\mathbf{s}^* - \mathbf{s}_0}{\|\mathbf{s}^* - \mathbf{s}_0\|} + \mathbf{s}_0. \tag{4.20}$$

Doing this allows us to have small input weights and make the system move with approximately constant speed (the clipped target keeps moving to the real target in each step).

Jacobian derivation. Similarly to the system identification, we are using automatic differentiation, but verifying the Jacobian manually is useful. We are deriving the formula from equation 3.82, and we only need the separate parts of the chain,

$$\frac{\partial \mathbf{f}(\mathbf{s}_k, \mathbf{u}_k, \mathbf{p})}{\partial \mathbf{s}_k} = \begin{bmatrix} 1 & 0 & 0 & -\Delta t \lambda_h q_2(k) \\ 0 & 1 & 0 & \Delta t \lambda_h q_1(k) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.21}$$

$$\frac{\partial \mathbf{f}(\mathbf{s}_k, \mathbf{u}_k, \mathbf{p})}{\partial \mathbf{u}_k} = \begin{bmatrix} \Delta t \lambda_h \sin(\psi(k)) & \Delta t \lambda_h \cos(\psi(k)) & 0 & 0 \\ \Delta t \lambda_h \cos(\psi(k)) & \Delta t \lambda_h \sin(\psi(k)) & 0 & 0 \\ 0 & 0 & \Delta t \lambda_v & 0 \\ 0 & 0 & 0 & \Delta t \lambda_a \end{bmatrix} \tag{4.22}$$

$$q_1(k) = \cos(\psi(k))pitch(k) + \sin(\psi(k))roll(k), \tag{4.23}$$

$$q_2(k) = \sin(\psi(k))pitch(k) - \cos(\psi(k))roll(k), \tag{4.24}$$

$$\psi(k) = \theta(k) + \theta_0. \tag{4.25}$$

4.3.10 Adjusting for input delay

Systems with delay in the control loop tend to overshoot, oscillating around the target. If we would want to limit the overshoot, we would have to dampen the system (make the inputs smaller) but for a large delay this could cause us to limit the controller so severely that it would be unusable in practice.

> The empirical evaluation shows that the controller of the UAV becomes increasingly less stable when the state estimate is delayed for more than 300ms. (M. Petrlík et al. [16])

To deal with the problem of input delay D, we are keeping a FIFO queue of previous actions $\{\mathbf{a}_{k-D-1}, \ldots, \mathbf{a}_k\}$ where $\mathbf{a}_k$ is the current action (command). This allows us to send the correct input $\mathbf{u}_{k-1} = \mathbf{a}_{k-D-1}$, into the MHE. By itself, this does not allow us to adjust the state of the MPC. To do that we try to predict the state using the recursive formula, starting from the MHE state estimate $\tilde{\mathbf{s}}_{k-1} = \hat{\mathbf{s}}_{k-1}$ and using parameter estimate $\hat{\mathbf{p}}$,

$$\tilde{\mathbf{s}}_{k+1} = \mathbf{f}(\tilde{\mathbf{s}}_k, \mathbf{a}_{k-D}, \hat{\mathbf{p}}). \tag{4.26}$$

This is a "leap of faith", we are relying on the fact that both $\hat{\mathbf{s}}_{k-1}$ and $\hat{\mathbf{p}}$ are relatively close to the real values, and the disturbances are relatively small. If this was not the case, the estimate of the final state $\tilde{\mathbf{s}}_{k+D+1}$ used to control the system could be very much off-target, and the controller's performance might be much worse, or the closed-loop system might even be unstable.

This is especially true for higher-order systems for which a small change in the initial state can mean great change in the later states [69]. This is one of the motivations for us to use a simple first-order model of the system, making the prediction less sensitive to the quality of the estimation.

4.3.11 Control loop

All of the separate parts (Tello communication, Vicon communication, Vicon filter, logging, MHE, and MPC) are combined into one program whose main thread is the control loop. In this loop, in each time-step we do the following (data flow diagram in figure 4.3):

1. Get raw observation $\mathbf{o}_{k,raw}$ from Vicon.
2. Get filtered observation $\mathbf{o}_{k,filt}$ from Vicon filter using $\mathbf{o}_{k,raw}$.
3. Get state $\hat{\mathbf{s}}_{k-1}$ and parameter estimate $\hat{\mathbf{p}}$ from MHE.
4. Get action $\mathbf{a}_k$ from MPC.
5. Send action $\mathbf{a}_k$ to Tello.
6. Post request to MHE with delayed action $\mathbf{a}_{k-D-1}$, and current filtered observation $\mathbf{o}_{k,filt}$.
7. Predict the state estimate $\tilde{\mathbf{s}}_{k+D+1}$ using actions $\{\mathbf{a}_{k-D-1}, \ldots, \mathbf{a}_k\}$, latest state $\hat{\mathbf{s}}_{k-1}$ and parameter estimate $\hat{\mathbf{p}}$.
8. Post request to MPC with predicted state $\tilde{\mathbf{s}}_{k+D+1}$, parameter estimate $\hat{\mathbf{p}}$ and current target $\mathbf{s}^*$.
9. If we are within tolerance of the current target $\mathbf{s}^*$ set the next target.
10. Log latest estimate $\hat{\mathbf{s}}_{k-1}$, action $\mathbf{a}_k$, and target $\mathbf{s}^*$.

Remark 4.1. Even if the delay D is set to 0, we still predict from $\tilde{\mathbf{s}}_{k-1}$ to $\tilde{\mathbf{s}}_{k+1}$ with 2 actions, this is due to the asynchronous calculation of the MHE and the MPC, each introducing a lag of 1 time-step.

We loop with the assigned discretization step Δt of 20 milliseconds until a signal to quit is received from the user, which commands Tello to land, joins all the threads, closes all connections, and ends the program.

To simplify the launch of the program, we provide it with a JSON configuration file (similarly to model identification, MHE and MPC) with all information required for connections, logging, Vicon filter, MHE and MPC setups. Full list for configurations for the control program is in appendix C.4.

As an auxiliary tool and a safety precaution, the program also allows a switch to manual control of the drone using the keyboard.

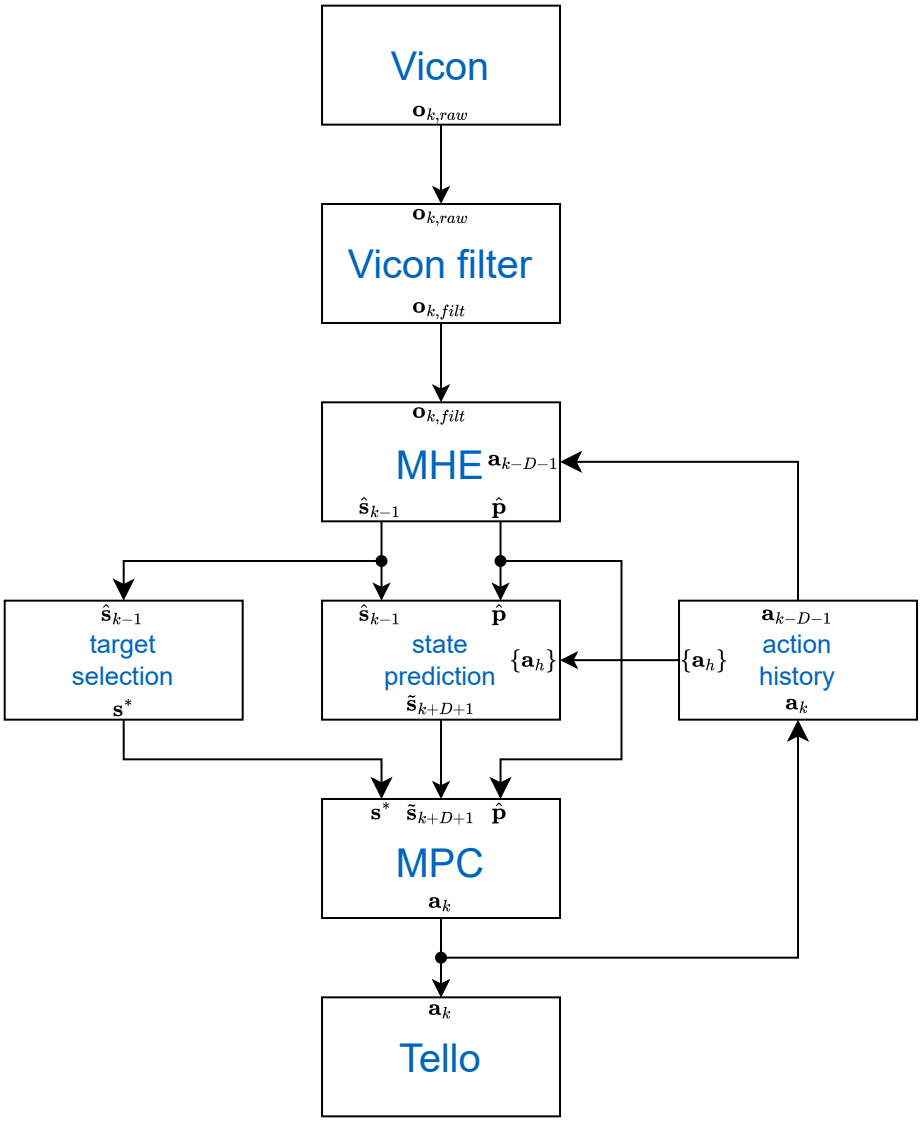


Figure 4.3: Data flow of control loop

Chapter 5

Results

5.1 System identification

To identify the model of our system, we will use the filtered observations of six manually flown trajectories and the recorded inputs, totaling 27832 time-steps. We are using only the filtered values (not values from MHE) and manual flying to remove any bias we could introduce by giving initial estimates for the model parameter values and to get sharp changes in the inputs to better estimate the input delay.

5.1.1 Input delay

During the initial testing of the drone capabilities, we came to the same finding as the other users of the Tello drone, the wireless communication introduces a significant input delay. To find how much delay there is, we plot out the observations and the cumulative inputs, in figure 5.1 we can see the positions (blue), the commutative inputs (orange) and the shifted cumulative input by 35 time-steps (green), time-step Δt is 20 milliseconds.

Definition 5.1. *Cumulative input* is the sum of the expected changes of a state derived from the model, using state equation 4.1. For the x-axis the cumulative input is the following,

$$\widehat{\Delta x}(k_0, k_1) = \sum_{k=k_0}^{k_1} \Delta t \lambda_h (\sin(\theta(k) + \theta_0) roll(k) + \cos(\theta(k) + \theta_0) pitch(k)). \tag{5.1}$$

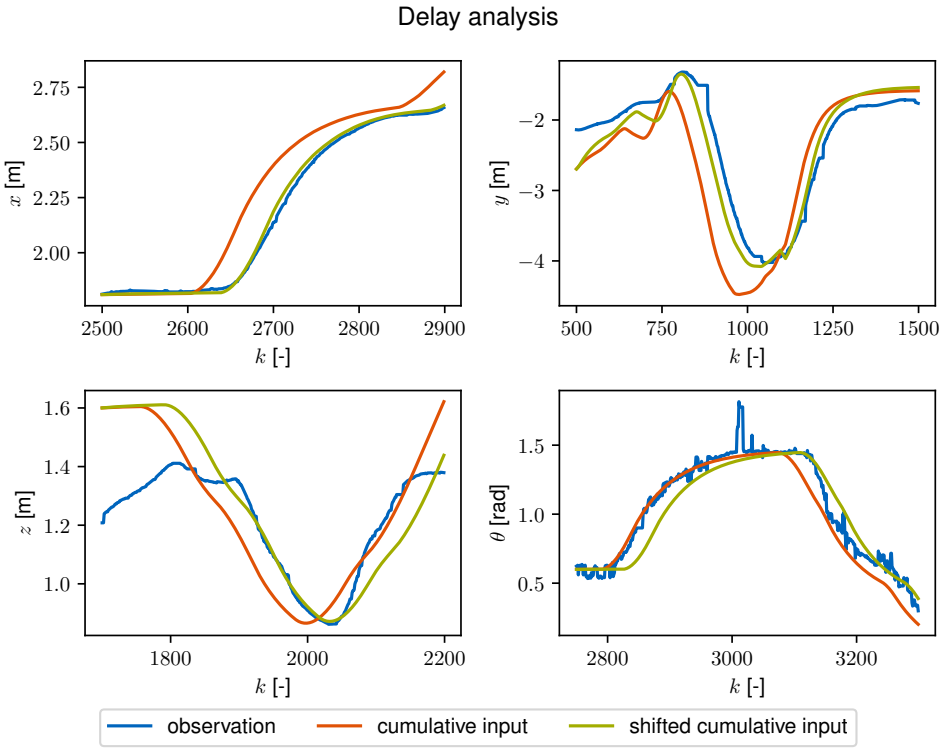


Figure 5.1: Delay analysis plot

Remark 5.2. In figure 5.1, the cumulative inputs are scaled and shifted to match the changes in position, we are *not* trying to estimate the parameters of the model, just the value of the input delay.

You can see that shifted input (corresponding to input delay) matches the observations better than not-shifted input, except for the angle rise. Since we use the value of the delay in the length of the prediction, we use a conservative estimate of 30 time-steps, which is equal to 0.6 seconds (time-step Δt is 20 milliseconds).

5.1.2 Model parameters

To estimate the model parameters, we are going to use the MAP estimate that was formulated and derived in section 3.1.3, with its implementation details in section 4.3.7. The wireless communication was not lossless, and sometimes, the drone did not react to the commands. To limit this negative effect and allow for the use of the bootstrap method for estimation, we split the six trajectories evenly into sub-trajectories of minimal length of 300 time-steps (6 seconds).

This produces a history of 89 trajectories ($|H| = 89$, $304 \leq L_h \leq 319$). If some part of the trajectory is tainted by a loss of connection or some other adverse conditions, smaller part of the dataset would be affected. When tuning the weighting coefficients of the estimate (used values are in appendix D), it is important to get state coefficients $\mathbf{c}_s$ as high as possible while still following the trajectory, this will result in smooth estimates that produce good estimations of the model parameters $\mathbf{p}$. If the state coefficients are too low, most of the model will be explained by observations and the state equations become irrelevant. This finding is concurrent with the elementary analysis of the system behavior using step increases in the inputs.

To verify if the state estimates are following the trajectory, we plot out the estimated states over the observations, like in figure 5.2.

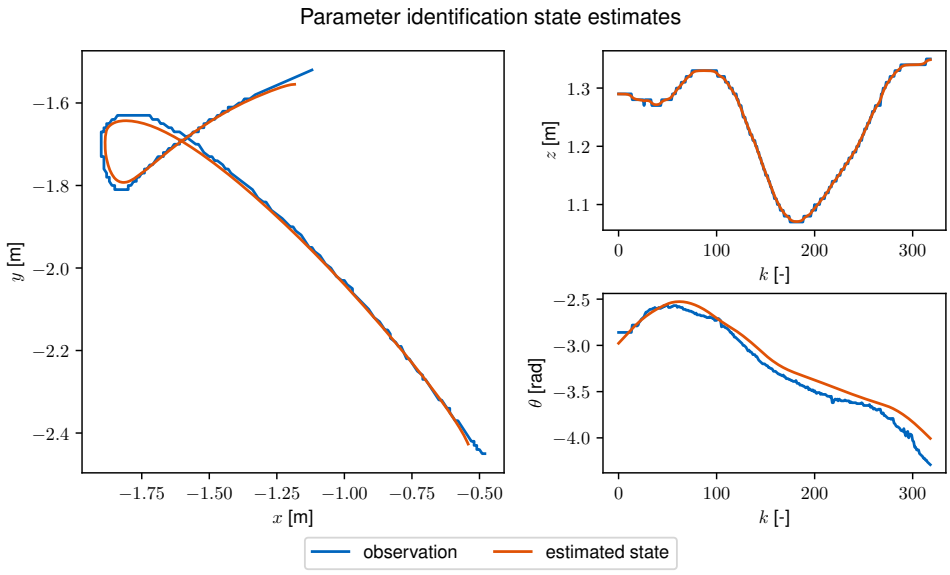


Figure 5.2: Verification plot for the parameter identification

Bootstrap. We have explained how to get the values from the estimator, but when using estimators, it is important to talk about the variance of the estimators. We use the bootstrap method [70], we repeatedly resample the 89 trajectories with repetition and estimate the values with this sample.

This allows us to get an approximation of the distribution of the parameters. With this distribution, we can create confidence intervals (CI) for our parameter estimates.

Estimated values. The values found for our model are presented in table 5.1. To outline the differences between the variances of the estimates, figure 5.3 presents a boxplot for the bootstrap values.

parameter	value	std. dev.	CI(95%) lower	CI(95%) upper
λ_h [m/s]	1.065	0.075	0.908	1.198
λ_v [m/s]	0.447	0.050	0.349	0.540
λ_a [m/s]	0.955	0.052	0.851	1.064
θ_0 [rad]	-0.171	0.150	-0.442	0.126

Table 5.1: Parameter estimate values

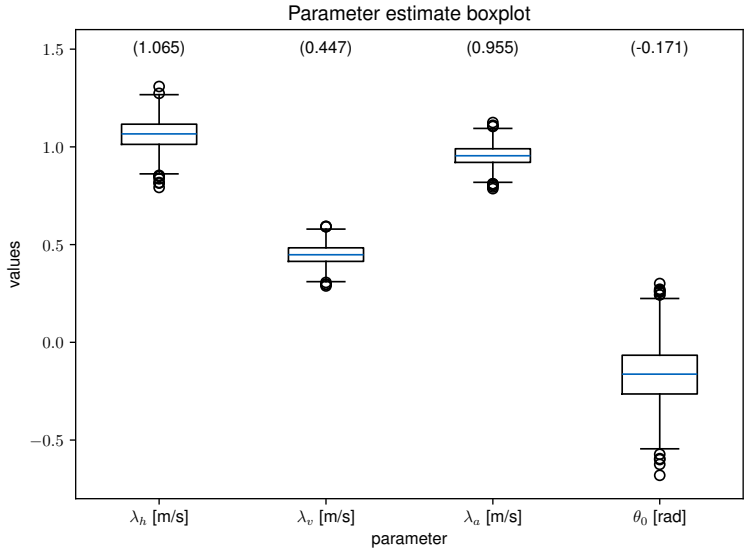


Figure 5.3: Boxplot for the bootstrap (estimated values in brackets on top)

5.2 Flight control

To evaluate the controller's performance, we are using a simple scenario; fly in a square of size 1.5 meters at the altitude of 1.0 meters or 1.3 meters.

The heading is constant during the flight, varied across trials from -0.2 rad to 0 rad. This produces 5 targets, in each corner of the square (including the starting one), that have to be met within the distance of 0.2 meters. The order of targets (whether we fly first along the x-axis or the y-axis) is also varied to demonstrate robustness. An example of a flown trajectory is in figure 5.4, reference of targets is dashed, time of flight in the brackets.

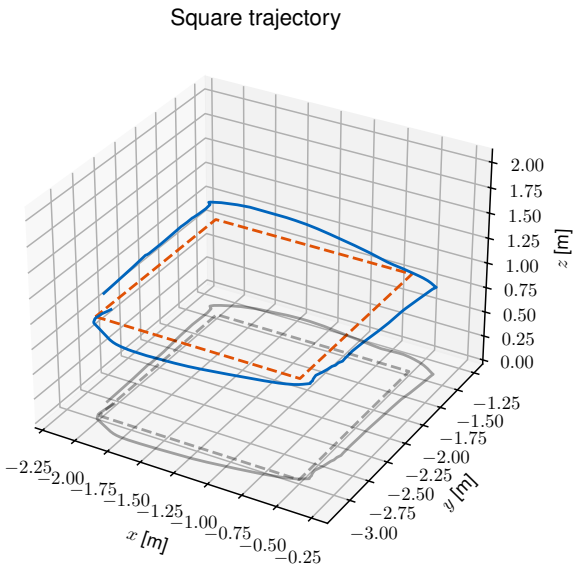


Figure 5.4: Square trajectory (14.8 seconds)

Although the trajectory might seem simple, it requires the drone to repeatedly accelerate towards the target and then decelerate to meet the target, which is not an elementary task for a system with such significant input delay and should provide ample information regarding the controller's behavior.

A trajectory starts when the state estimate is within the required distance of the first target and ends when the state estimate is within the required distance of the last target, or is ended due to external conditions. External condition preventing the drone from completing the trajectory include low battery, the drone is not responding to commands (loss of connection), and manual control is used to prevent the drone from crashing.

During the tuning of the controller, we recorded every trajectory. The analysis is done for a filtered subset that are deemed valid. For a considered trajectory to be valid, the drone flew throughout the whole trajectory (met all targets) and did not deviate from the trajectory in the horizontal direction (only x-axis

and y-axis) for more than 0.3 meters. The filtered subset of valid trajectories totals 29 (about 550 seconds of flight time).

5.2.1 Control loop configuration

The final control loop configuration has the loop frequency of 50 Hz (time-step Δt is 20 milliseconds) and an estimated input delay of 30 time-step. The computation distribution is 6 threads for MPC, 1 thread for MHE and 1 thread for each communication handler. The horizon length is set to 13 time-steps for MPC and 30 for MHE, which is about the maximum possible for the used computer with Inter Core i5-8500U CPU (at 1.6 GHz) and 8 GB of RAM. Full table of the tuned coefficients is in appendix D, the most notable is the adjustment of the heading offset angle, which is further explained in discussion 6.2.1.

5.2.2 Prediction inconsistency

As mentioned in section 4.3.10, the model prediction is used to compensate the input delay of the model, but could become unreliable when the model does not match the real system. This assumption has been proven correct, during the flight the prediction seems to follow the trajectory reasonably well, but when lowering the input (on corners of the square), the real system stops, while the prediction is drifting beyond the position. Figure 5.5 shows the state, its prediction, and cumulative input to illustrate this effect. Analysis of this effect is in discussion 6.1.2 and 6.2.3.

5.2.3 Controller performance

From the 29 valid trajectories, the median time to finish is 18.2 seconds, the 25% quantile is 14.8 seconds, and the 75% quantile is 22.2 seconds. The median speed for valid trajectories is 0.35 m/s, comparison is done in discussion 6.2.4. The median horizontal distance to trajectory is 0.14 m,[1] the median input norm is 0.48 (inputs are scaled target velocities).

[1] We are using the horizontal distance (x and y-axis) as the main metric, because the drone has its internal altitude control.

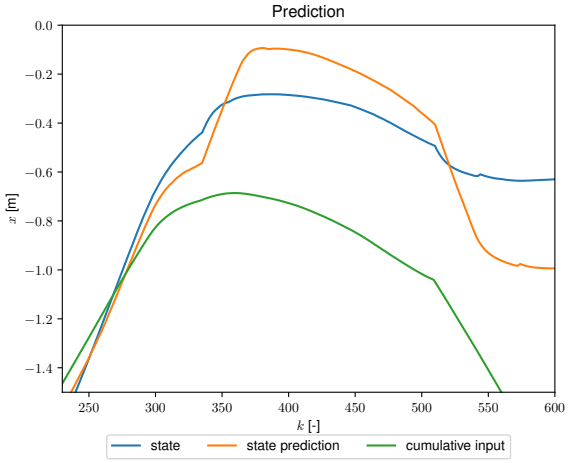


Figure 5.5: Prediction inconsistency plot

Trajectory progression matching. To further analyze the dynamic behavior of the controller, we are going to match each time-step to the closest point on the trajectory. This will allow us to compare results across trajectories regardless of time, because they will be matched to the same progression of the trajectory. Figure 5.6 shows the Gaussian kernel density estimate (KDE) for the progression. In the sections with higher density the drone spent more time.

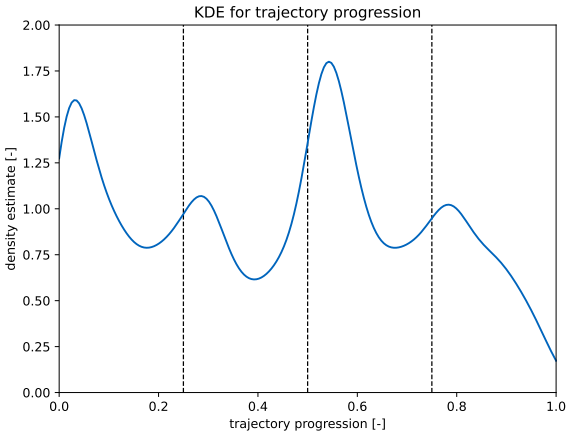


Figure 5.6: KDE for trajectory progression

Figure 5.7 shows a heatmap (estimated density), median, and 95% quantile interval plot for the horizontal distance across trajectory progression, figure 5.8 shows the same information for the input norm, analysis of these plots is in discussion 6.2.5.

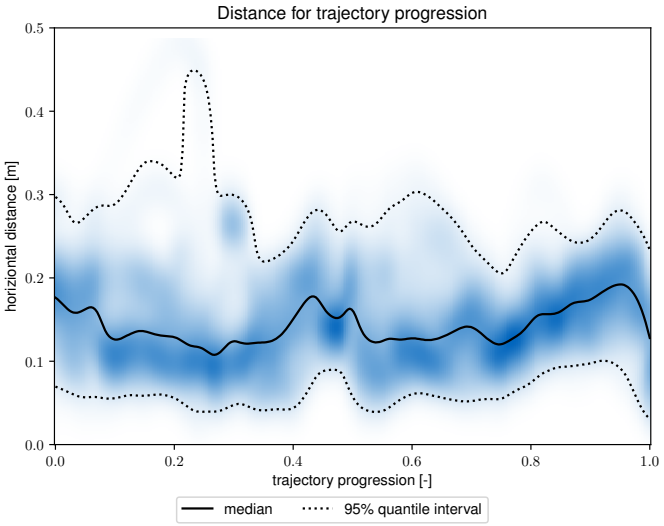


Figure 5.7: Horizontal distance for trajectory progression

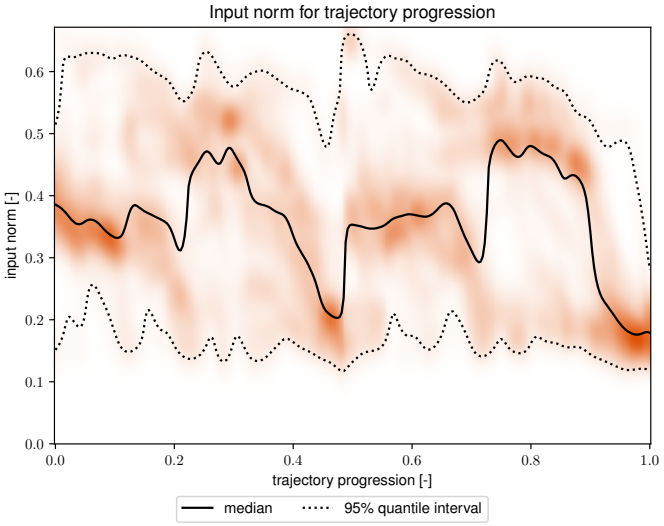


Figure 5.8: Input norm for trajectory progression

5.2.4 Additional trajectories

As a proof of concept for the generality of our controller, figures 5.9 and 5.10 present 2 trajectories flying around the laboratory (reference is dashed), the trajectories are bigger, and the target distance required is set to 0.3 m.

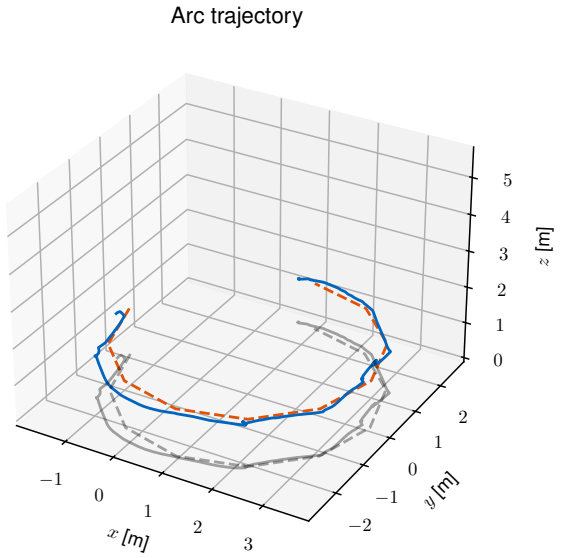


Figure 5.9: Arc trajectory (67.8 seconds)

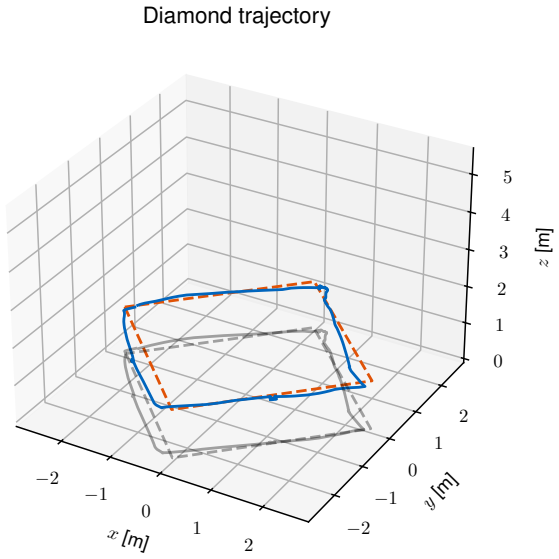


Figure 5.10: Diamond trajectory (67.5 seconds)

Chapter 6

Discussion

This section will discuss the results presented in the previous chapter as well as findings noted during the tuning of the methods.

6.1 System identification

6.1.1 Input delay

When testing the controller while ignoring the input delay, the closed-loop system was oscillating (circling) around the target and an erroneous orientation θ reading could completely make it unstable, the drone would fly off and had to be stopped manually. The value of 30 time-steps for the input delay is chosen conservatively, a longer delay means farther prediction, making it more prone to model misspecification.

6.1.2 Parameter values

The estimated parameter values (table 5.1) proved to be concurrent with the initial elementary analysis of step-input measurements. However, due to the non-linearity of the control around zero input, shown in figure 5.5,

probably caused by the internal regulator which is tuned for a human pilot, making easier to manually control the drone around zero input, the model was not working well in this range. It should be noted that the estimated values presented in [4] are smaller (for the horizontal coefficient by 45%), this could be explained by the fact that the author only used one trajectory or a different communicating procedure that scales the inputs for the drone.

6.1.3 Parameter variance

The values of the heading offset are only relevant for a specific Vicon configuration, however, looking at the values in table 5.1 or boxplot in figure 5.3, we can see that the variance of heading estimation is quite large, 95% confidence interval ranging from -0.442 rad to 0.126 rad (range of 32.5 degrees).

This finding is important as it pointed us in the right direction when tuning the controller, the orientation measurements are unreliable, and we need to rely on the horizontal displacements to estimate our heading. Because of this, the state and observation weighing coefficients are higher for the MHE in the x and y-axis, compared to the orientation coefficients.

6.2 Flight control

6.2.1 Heading adjustment

As seen from some of the valid trajectories, the drone often flies slightly off-target, only to be corrected right before reaching the target. This effect is due to the unreliable estimation of orientation θ and the heading offset θ_0, as seen from the variance of the heading offset estimation. To combat this, we manually adjust the heading offset θ_0 for the MPC computation by an additional -0.3 rad, forcing the drone to shift the trajectory to the target, and improving the controller's performance.

6.2.2 Computation feasibility

The real-time optimization of a non-linear MPC problem proved to be the limiting factor for the speed of the control loop. The Jacobian of the MPC problem is much denser than for the MHE. Because of this, the computation of the MPC problem takes the majority of the computation power (6 threads, the rest of the tasks have one each).

Still, the `Ceres` solver proved to be effective for this real-time task. The solution was precise enough to be used as a control strategy. Only during a fast target change, it took the optimization algorithm a few cycles of the control loop to catch up and converge to the optimal strategy, as it had limited computation time so as not to lag behind the current state. This can be seen from figure 5.6, the drone "lingers" after the corner before speeding up.

Limiting the distance of the target proved to be a good extension, allowing for a smooth flight along the trajectory, making the computation more stable.

6.2.3 Prediction inconsistency

Due to the imperfections of our model around the zero input, the prediction was moving off-target during changes in flight direction. This, coupled with significant delay meant that the distance requirement of 0.2 meters was about as low as possible without having to wait a significant amount of time for the drone to converge to the target.

An additional effect of altitude (z-axis) decrease during horizontal acceleration is also sometimes visible for the drone, it is usually quickly corrected by the drone's internal altitude control.

6.2.4 Controller performance

When evaluating the performance of the controller, it should be noted that we have kept the trajectories even for the untuned parameters of the controller, so a realistic expectation of the performance for a tuned controller is around the 25% quantile of 14.8 seconds to finish the square trajectory; this would

correspond to a speed of 0.41 m/s. For comparison with other works, the same drone is flown with a speed of 0.1 m/s in [4], and the maximum speed is about 0.2 m/s in [5].

6.2.5 Trajectory progression

From figure 5.7 we can see that the median distance does not rise above the target required distance of 0.2 meters, stays roughly consistent throughout the trajectory progression. This is a good finding, if there was some significant increases in some parts of the trajectory that would mean either overshooting or significant deviance from the expected behavior.

From figure 5.8 we can see an expected behavior of accelerating towards the target and then a slight drop-off when we are getting closer to the target (4 drops corresponding to 4 targets). One finding that is a little bit unexpected is the fact that in the 1st and 3rd quarter both the input values and the drop-off are relatively smaller for the median values than for the 2nd and 4th quarter of the trajectory, suggesting some unexplained effects, however the range of the confidence interval stays almost the same so it could be just some bias from repeated dependent experiments.

Chapter 7

Conclusion

Assignement completion. The assignment guidelines were completed fully, with the slight reservation regarding the requirement for different trajectories, as while multiple trajectories were present, we only collected enough data for a proper evaluation of the control performance for the square trajectory.

Real-time control. The main conclusion from this thesis is the viability of a fully non-linear MPC controller for real-time flight control of robotic helicopter with significant input delay. Most of the previous works used simpler methods of control or used the MPC either with a linearized system or outside the loop as a guide-line to which the drone was controlled with some other method.

MAP estimate for parameters. The values from the derived MAP estimate of model parameters are successfully used to control the dynamic system. The implemented method can be applied to other dynamic systems just by creating a template of the system state-space model.

State prediction for input delay. For a system with such significant input delay (over 0.6 seconds), the state prediction using previous inputs is a crucial part of making the controller viable.

Vicon limitation. The Vicon tracking system is pushed to its limits when tracking such a small object, and the markers are often obscured. The developed filter and tuned MHE make Vicon a viable measurement in the control loop. However, the use of a better marker configuration, or additional measurements such as odometry or camera tracking to correct the measurements is preferred.

Ceres library. The open-source library `Ceres` is proven to be capable of handling real-time tasks like the MPC and the MHE, as well as handling large models used for off-line parameter estimation. For example our model for parameter estimation has over 100 000 variables, and over 200 000 residuals and the computation finished in a matter of minutes on a personal computer.

Appendix A

Bibliography

[1] V. Pritzl, P. Stepan, and M. Saska, "Autonomous flying into buildings in a firefighting scenario," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 239–245.

[2] M. Bekhzod, "Ryze tello drone tracking," 2022, bachelor thesis, Department of Cybernetics, FEE CTU Prague. [Online]. Available: https://dspace.cvut.cz/handle/10467/101276

[3] J. Ševic, "Ryze tello drone flying through an obstacle," 2021, bachelor thesis, Department of Cybernetics, FEE CTU Prague. [Online]. Available: https://dspace.cvut.cz/handle/10467/101326

[4] D. Pařil, "Autonomous control of drone ryze tello," 2021, bachelor thesis, Department of Cybernetics, FEE CTU Prague. [Online]. Available: https://dspace.cvut.cz/handle/10467/94756

[5] J. Gärtner, "Uav formation keeping based on mutual distance measurement," 2021, master thesis, Department of Cybernetics, FEE CTU Prague. [Online]. Available: https://dspace.cvut.cz/handle/10467/95269

[6] T. Saini, "Manoeuvring drone (tello ans tello edu) using body poses or gestures," 2021, master thesis, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Universitat Politècnica de Catalunya. [Online]. Available: https://dspace.cvut.cz/handle/10467/95269

[7] K. Hulek, M. Pawlicki, A. Ostrowski, and J. Mozaryn, "Implementation and analysis of ryze tello drone vision-based positioning using apriltags," *MMAR 2023 - 27th International Conference on Methods and Models in Automation and Robotics*, 05 2023.

[8] T. Báča, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," 11 2018.

[9] G. SONUGÜR, "A review of quadrotor uav: Control and slam methodologies ranging from conventional to innovative approaches," *Robotics and Autonomous Systems*, vol. 161, p. 104342, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889022002317

[10] J. J. Castillo-Zamora, K. A. Camarillo-GóMez, G. I. PéRez-Soto, and J. RodríGuez-ReséNdiz, "Comparison of pd, pid and sliding-mode position controllers for v–tail quadcopter stability," *IEEE Access*, vol. 6, pp. 38 086–38 096, 2018.

[11] A. G. Varghese and D. Sreekala, "Modeling and design of uav with lqg and h-inf controllers," *International Journal of Engineering Research & Technology (IJERT)*, vol. 8, no. 5, pp. 446–450, 2019.

[12] N. S. Özbek, M. Önkol, and M. Ö. Efe, "Feedback control strategies for quadrotor-type aerial robots: a survey," *Transactions of the Institute of Measurement and Control*, vol. 38, no. 5, pp. 529–554, 2016.

[13] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter," 2006, Conference paper, p. 3255 – 3260. [Online]. Available: https://ieeexplore.ieee.org/document/4058900

[14] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5, pp. 1327–1349, 2021.

[15] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 2, feb 2019. [Online]. Available: https://doi.org/10.1145/3301273

[16] M. Petrlik, P. Petracek, V. Kratky, T. Musil, Y. Stasinchuk, M. Vrba, T. Baca, D. Hert, M. Pecka, T. Svoboda, and M. Saska, "UAVs Beneath the Surface: Cooperative Autonomy for Subterranean Search and Rescue in DARPA SubT," *Field Robotics*, vol. 3, pp. 1–68, January 2023.

[17] P. Antsaklis and A. Michel, *A Linear Systems Primer*, 01 2007.

[18] F. Brown, *Engineering System Dynamics: A Unified Graph-Centered Approach, Second Edition*, ser. Control engineering. Taylor & Francis, 2006. [Online]. Available: https://books.google.cz/books?id=UzqX4j9VZWcC

[19] J. Krška, "Přednášky z B3M35LSY," 2018, lecture notes from FEE CTU Prague.

[20] Z. Chen and E. N. Brown, "State space model," *Scholarpedia*, vol. 8, no. 3, p. 30868, 2013, revision #189565.

[21] J. Peters, J. Mooij, D. Janzing, and B. Schölkopf, "Causal discovery with continuous additive noise models," 2014.

[22] Z. Hurák, "Introduction to numerical simulation," 2020, lecture at FEE CTU Prague.

[23] J. Švrčina, "Adaptive control using neural networks," 2020, bachelor thesis, Department of Cybernetics, FEE CTU Prague. [Online]. Available: https://dspace.cvut.cz/handle/10467/89834

[24] G. F. Franklin, J. D. Powell, A. Emami-Naeini, and H. S. Sanjay, *Feedback control of dynamic systems*, seventh, global ed. Boston: Pearson, 2015.

[25] F. Fisher, *The Identification Problem in Econometrics*. R. E. Krieger Publishing Company, 1976. [Online]. Available: https://books.google.cz/books?id=rFsrnQEACAAJ

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning; an Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: http://www.incompleteideas.net/book/the-book.html

[27] O. N. Bjørnstad and B. T. Grenfell, "Noisy clockwork: Time series analysis of population fluctuations in animals," *Science*, vol. 293, no. 5530, pp. 638–643, 2001. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1062226

[28] C. Breto and E. Ionides, "Compound Markov counting processes and their applications to modeling infinitesimally over-dispersed systems," *Stochastic Processes and their Applications*, vol. 121, 02 2010.

[29] G. Grimmett and D. Stirzaker, *Probability and random processes*. Oxford; New York: Oxford University Press, 2001. [Online]. Available: http://www.worldcat.org/search?qt=worldcat_org_all&q=9780198572220

[30] J. Bouda, "Markov processes," 2009, lecture at Masaryk university in Brno. [Online]. Available: https://www.fi.muni.cz/~xbouda1/teaching/2009/IV111/lecture5.pdf

[31] P. Poupart, *Partially Observable Markov Decision Processes*. Boston, MA: Springer US, 2010, pp. 754–760. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_629

[32] J. M. Ondřej Drbohlav, "Parameter estimation: Maximum likelihood (ml), maximum a posteriori (map), and bayesian," 2022, lecture at CTU in Prague. [Online]. Available: https://cw.fel.cvut.cz/b231/_media/courses/be5b33rpz/lectures/pr_03_parameter_estimation_2022.pdf

[33] C. V. Rao, J. B. Rawlings, and D. Q. Mayne, "Constrained state estimation for nonlinear discrete-time systems: stability and moving horizon approximations," *IEEE Transactions on Automatic Control*, vol. 48, no. 2, pp. 246–258, 2003.

[34] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[35] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[36] Grune, J. P. Pannek, Jurgen, and Lars, *Nonlinear Model Predictive Control: Theory and Algorithms*, 2nd ed. Cham: Springer, 2016;2017;.

[37] S. Lucia, A. Tatulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, p. 51–62, 03 2017. [Online]. Available: https://www.do-mpc.com/

[38] K. H. Ang, G. Chong, and Y. Li, "PID control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

[39] F. Lewis, *Optimal Control*. John Wiley & Sons Australia, Limited, 2005. [Online]. Available: https://books.google.cz/books?id=FGGIPwAACAAJ

[40] N. Lehtomaki, N. Sandell, and M. Athans, "Robustness results in linear-quadratic gaussian based multivariable control designs," *IEEE Transactions on Automatic Control*, vol. 26, no. 1, pp. 75–93, 1981.

[41] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. [Online]. Available: https://books.google.cz/books?id=VbHYoSyelFcC

[42] D. Arrowsmith and C. Place, *Dynamical Systems: Differential Equations, Maps, and Chaotic Behaviour*, ser. Chapman Hall/CRC Mathematics Series. Taylor & Francis, 1992. [Online]. Available: https://books.google.cz/books?id=8qCcP7KNaZ0C

[43] A. Galántai, "The theory of newton's method," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 25–44, 2000, numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377042700004350

[44] J. Velebil, *Abstraktní a konkrétní lineární algebra*. Departement of mathematics, FEE CTU in Prague, 2023. [Online]. Available: https://math.fel.cvut.cz/en/people/velebil/files/akla/akla_2023_09_15.pdf

[45] K. Levenberg, "A method for the solution of certain non – linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[46] D. W. Marquardt, "An algorithm for least-squares estimation of non-linear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[47] K. Madsen, H. Nielsen, and O. Tingleff, *Methods for Non-Linear Least Squares Problems (2nd ed.)*, 01 2004.

[48] J. Hadamard, *Mémoire sur le problème d'analyse relatif a l'équilibre des plaques élastiques encastrèes*, ser. Académie des sciences. Mémoires. Imprimerie nationale, 1908. [Online]. Available: https://books.google.cz/books?id=8wSUmAEACAAJ

[49] Y. Bard, *Nonlinear Parameter Estimation*. Academic Press, 1974. [Online]. Available: https://books.google.cz/books?id=fNo6MR1iS1UC

[50] S. Agarwal, "Trust region methods," 2023, Ceres Solver Documentation. [Online]. Available: http://ceres-solver.org/nnls_solving.html#trust-region-methods

[51] S. Agarwal, K. Mierle, and The Ceres Solver Team, "Ceres solver," 2023, A large scale non-linear optimization library. [Online]. Available: http://ceres-solver.org

[52] J. Gram, "Ueber die entwickelung reeller functionen in reihen mittelst der methode der kleinsten quadrate." *Journal für die reine und angewandte Mathematik*, vol. 1883, no. 94, pp. 41–73, 1883. [Online]. Available: https://doi.org/10.1515/crll.1883.94.41

[53] E. Schmidt, "Zur theorie der linearen und nichtlinearen integralgleichungen. i. teil: Entwicklung willkürlicher funktionen nach systemen vorgeschriebener," *Mathematische Annalen*, vol. 63, pp. 433–476, 1907. [Online]. Available: http://eudml.org/doc/158296

[54] A.-L. Cholesky, "Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés a un système d'équations linéaires en nombre inférieur a celui des inconnues. — application de la méthode a la résolution d'un système defini d'équations linéaires," *Notices Scientifiques*, 1924. [Online]. Available: https://doi.org/10.1007/BF03031308

[55] M. Lira, R. Iyer, A. Trindade, and V. Howle, "Qr versus cholesky: A probabilistic analysis," *International Journal of Numerical Analysis and Modeling*, vol. 13, no. 1, pp. 114–121, 2016.

[56] G. Fasshauer, "Least squares problems," 2007, lecture at Illinois Institute of Technology. [Online]. Available: http://www.math.iit.edu/~fass/477577_Chapter_5.pdf

[57] T. Davis, *Direct Methods for Sparse Linear Systems*, ser. Fundamentals of Algorithms. SIAM, Society for Industrial and Applied Mathematics, 2006. [Online]. Available: https://books.google.cz/books?id=aTLYrafw3vUC

[58] Ryze, "Tello user manual," 2023. [Online]. Available: https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20User%20Manual%20v1.4.pdf

[59] R. B. Heywood, *The Works of the Mind ... Edited ... by R.B. Heywood, Etc.* University of Chicago Press, 1947. [Online]. Available: https://books.google.cz/books?id=w4wjMwEACAAJ

[60] B. Stroustrup, *The C++ Programming Language, Third Edition*, 3rd ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[61] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[62] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[63] B. Crow, I. Widjaja, J. Kim, and P. Sakai, "Ieee 802.11 wireless local area networks," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[64] Ryze, "Tello SDK 2.0 User Guide," 2018. [Online]. Available: https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf

[65] J. Postel, "Rfc0768: User datagram protocol," USA, 1980.

[66] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974.

[67] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[68] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[69] K. Alligood, T. Sauer, and J. Yorke, *Chaos: An Introduction to Dynamical Systems*, ser. Textbooks in Mathematical Sciences. Springer New York, 2000. [Online]. Available: https://books.google.cz/books?id=48YHnbHGZAgC

[70] B. Efron, "Bootstrap methods: Another look at the jackknife," *Annals of Statistics*, vol. 7, pp. 1–26, 1979. [Online]. Available: https://api.semanticscholar.org/CorpusID:227312712

Appendix B

Repository

The code source repository is available from `https://github.com/svrc-jan/master-thesis/`, including instructions for building and running the programs in `README.md`.

```
repository
├─ config
│  ├─ ident_boostrap.json
│  ├─ ident_real.json
│  ├─ mhe_real.json
│  ├─ mpc_real.json
│  ├─ io_real.json
│  ├─ tar_square.json
│  └─ ...
├─ logs
│  └─ ...
├─ logs_square
│  └─ ...
└─ logs_ident
   └─ ...
```

```
repository
├── src
│   ├── filter
│   │   ├── vicon_filter.cpp
│   │   └── vicon_filter.hpp
│   ├── model
│   │   ├── drone_model.hpp
│   │   └── model_sim.hpp
│   ├── optim
│   │   ├── mhe.hpp
│   │   ├── model_ident.hpp
│   │   └── mpc.hpp
│   ├── tello
│   │   ├── tello.cc (supervisor)
│   │   └── tello.h (supervisor)
│   ├── utils
│   │   ├── keyboard_handler.cpp
│   │   ├── keyboard_handler.hpp
│   │   ├── logger.hpp
│   │   └── parser.hpp
│   ├── vicon
│   │   ├── vicon_client.cc (supervisor)
│   │   ├── vicon_client.h (supervisor)
│   │   ├── vicon_handler.cpp
│   │   └── vicon_handler.hpp
│   ├── run_model_ident.cpp
│   └── mpc_control.cpp
└── ...
```

Appendix C

Configuration file specifications

All of the configuration files are in the JSON format.

C.1 System identification

- `dt`: discretization step
- `u_delay`: input delay
- `C_o`: weighing coefficients for observations (size n_o)
- `C_s`: weighing coefficients for state transitions (size n_s)
- `C_p`: weighing coefficients for parameter priors (size n_p)
- `p_prior`: prior values for parameters (size n_p)
- `p_lb`: lower bound values for parameters (size n_p)
- `p_ub`: upper bound values for parameters (size n_p)
- `obs_loss_s`: 1 to use Tukey loss for observations, 0 for normal loss
- `state_loss_s`: 1 to use Tukey loss for state transitions, 0 for normal loss
- `max_models`: maximum number of logs to load
- `log_dir`: folder from where to load the trajectory logs

- `clear_log_est_dir`: boolean, if true delete contents of the estimation folder
- `solver_tol`: solver relative functional tolerance
- `solver_threads`: number of threads to use for optimization
- `solver_linear_solver_type`: what factorization the solver uses (example `sparse_cholesky`)
- `solver_stdout`: boolean, if true, solver prints optimization progress

C.2 Moving horizon estimate

Configuration parameters are the same as for system identifications (appendix C.1), with the addition of

- `h`: length of the horizon
- `solver_max_time`: maximum time for solving in seconds

and without `max_models`, `log_dir` and `clear_log_est_dir`.

C.3 Model predictive control

- `dt`: discretization step
- `h`: length of the horizon
- `u_delay`: input delay
- `max_target_distance`: distance to which the target is clipped
- `C_s`: weighing coefficients for stage state distance (size n_s)
- `C_s_end`: weighing coefficients for terminal state distance (size n_s)
- `C_u`: weighing coefficients for inputs (size n_u)
- `u_lb`: lower bound values for inputs (size n_u)

- `u_ub`: upper bound values for inputs (size n_u)
- `use_u_diff`: boolean, true for using input difference term, false for using absolute value
- `solver_tol`: solver relative functional tolerance
- `solver_max_time`: maximum time for solving in seconds
- `solver_threads`: number of threads to use for optimization
- `solver_linear_solver_type`: what factorization the solver uses (example `sparse_cholesky`)
- `solver_stdout`: boolean, if true, solver prints optimization progress

C.4 Control program

- `input_c`: constant for manual control
- `u_delay`: input delay D
- `filter_horizontal_threshold`: horizontal threshold for the Vicon filter
- `filter_vertical_threshold`: horizontal threshold for the Vicon filter
- `filter_angle_threshold`: angular threshold for the Vicon filter
- `keyboard_device`: device for user input
- `vicon_ip`: IP address for Vicon communication
- `tello_port`: port for Tello communication
- `tello_net_interface`: network interface string for Tello communication
- `mpc_config`: path to the MPC config file
- `mhe_config`: path to the MHE config file
- `log_dir`: path to the directory where the logs will be saved

Appendix D

Used coefficients

It should be noted that in the code, weighing vectors $\mathbf{c}_o$, $\mathbf{c}_s$, $\mathbf{c}_p$, $\mathbf{c}_u$ and $\mathbf{c}_t$ are applied to residual, not their squared norm, thus the values here are squared values of what is in the configuration files.

	x	y	z	θ
$\mathbf{c}_o$	9	9	4	0.09
$\mathbf{c}_s$	1	1	0.01	0.09
	λ_h	λ_v	λ_a	θ_0
$\mathbf{c}_p$	1	1	1	1
$\mathbf{p}_0$	0.9	0.6	0.9	0

Table D.1: Coefficients used for system identification

	x	y	z	θ
$\mathbf{c}_o$	9	9	4	0.25
$\mathbf{c}_s$	0.16	0.16	0.16	0.04
	λ_h	λ_v	λ_a	θ_0
$\mathbf{c}_p$	400	400	400	10000
$\mathbf{p}_0$	1.061	0.447	0.955	-0.171

Table D.2: Coefficients used for MHE

	$roll$	$pitch$	yaw	$throttle$
$\mathbf{c}_u$	9	9	4	0.25
	x	y	z	θ
$\mathbf{c}_s$	2.25	2.25	1	1
$\mathbf{c}_t$	225	225	100	100
	λ_h	λ_v	λ_a	θ_0
$\mathbf{p}_{adjust}$	0	0	0	-0.3

Table D.3: Coefficients used for MPC