Ecole Polytechnique Federale de Lausanne

System Dynamics Identification

Identification of System dynamics of Holohover Vehicle

Author:
Nicolaj Schmid

Supervisor:
Roland Schwan

LAB:
STI IGM LA3

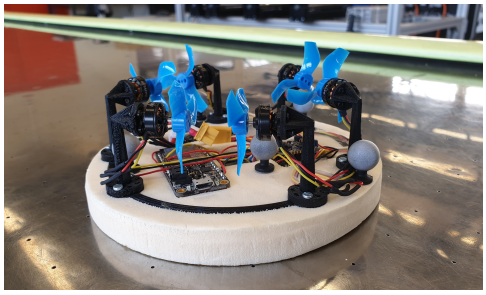
Professor
Colin Jones

Project:
Semester Project

Contact:
nicolaj.schmid@epfl.ch

Start:
20.09.2022

Finish:
06.01.2023



.

Contents

Chapter 1 Introduction

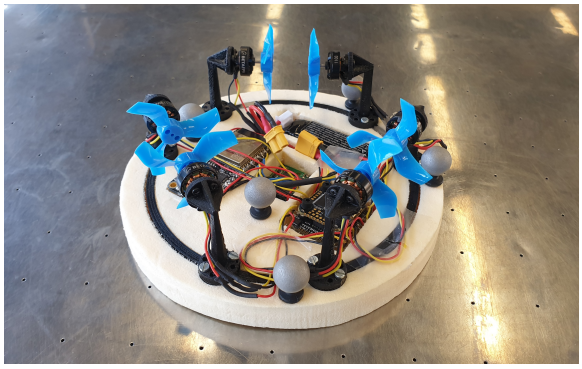


Figure 1.1: Holohover

The Holohover is an omnidirectional vehicle that moves on an air-hockey table. Normally, such tables are used to play the game *Air-Hockey*. Two players try to push a plastic puck into the opposite goal and score as many points as possible. Air is ejected through tiny holes in the table, forming a small layer between the table and the puck and reducing the puck's friction. The same principle applies to the Holohover which can float on the table because of its lightweight construction. The vehicle is omnidirectional and driven by six motors. Always two motors face each other and are placed with an offset of 120° around the circular body (see figure 1.1). Only one of the two motors is used at a time depending on in which direction the Holohover should move. The long-term goal is to design a Holohover that can play *Air Hockey* against a human or another robot.

During three previous semester projects [1] [2] [3], the Holohover was designed. Further, K. Samaha derived the system dynamics from the first principle [3]. R. Schwan implemented a *Robotic Operating System* (ROS) package aimed to control the vehicle. The software runs in a *ROS2 Humble* environment and includes the Holohover model, an LQR-controller, a simulation, the interface towards the *Motion Capture System* (MCS) from *OptiTrack*, and the connection to the Holohover via the *Micro-ROS Agent*. This semester project aims to improve the model so that more advanced controllers could be implemented, for example, based on *Model Predictive*

Control (MPC). Further, we would like to ensure that the system dynamics are stabilizable.

The chapters of this report are structured as follows: Chapter 2 explains the theory behind stabilizable system dynamics and showcases the formulation by simulating a *Damped Harmonic Oscillator* (DHO) and a *Continuous Stirred Tank Reactor* (CSTR). Even though we invested the largest part of the first half of the semester for this implementation, the stabilizable dynamics are not yet learned for the Holohover because of two main reasons: First, we have to adapt the theory as explained at the end of the chapter 2. And second and more important, we want to have good system dynamics at the end of the project even if they are not proven to be stabilizable. Therefore, we determine the signal-to-thrust curves again in chapter 3. And in chapter 4, we test different models on real data and evaluate their performances.

Please visit the *GitHub* page for more details about the implementation.

Chapter 2 Stabilizable Dynamics

2.1 Previous work

It is common to use deep neural networks to approximate dynamical systems. However, when using neural networks, it is difficult to ensure stability. A network could first be trained and then tested if the resulting dynamics are stabilizable. However, this may be a tortuous process because nothing directs the neural network to be stabilizable. Therefore, stability was encouraged by learning a Lyapunov function by adding a dedicated loss [4], [5], [6]. Still, stability can not be guaranteed using this approach. Therefore, G. Manek et al. learned the system dynamics and a Lyapunov function jointly such that stability is ensured by design for autonomous systems [7]. R. Schwan extends this idea for controlled systems [8]. This is the formulation presented in this chapter.

2.2 Theory

Equation (2.1) defines the dynamics of a continuous control affine system. This means that the control input $\mathbf{u}(t) \in \mathbb{R}^M$ has a linear mapping to the dynamics, but the relationship between the state $\mathbf{x}(t) \in \mathbb{R}^D$ and the dynamics $\dot{\mathbf{x}}(t) \in \mathbb{R}^D$ may be non-linear where $f : \mathbb{R}^D \to \mathbb{R}^D$ and $g : \mathbb{R}^D \to \mathbb{R}^{DxM}$.

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t))\mathbf{u}(t) \tag{2.1}$$

In the following, we assume the dependency of $\mathbf{x}$ and $\mathbf{u}$ on time implicitly. Learning the system dynamics from data can be achieved by approximating $f(\mathbf{x})$ and $g(\mathbf{x})$ by two neural networks $\tilde{f}(\mathbf{x})$ and $\tilde{g}(\mathbf{x})$. The approximation of the dynamics is given by equation (2.2).

$$\dot{\tilde{\mathbf{x}}} = \tilde{f}(\mathbf{x}) + \tilde{g}(\mathbf{x})\mathbf{u} \tag{2.2}$$

The neural networks $\tilde{f}(\mathbf{x})$ and $\tilde{g}(\mathbf{x})$ are learned by an iterative optimization process using equation (2.3) as loss function where N is the number of samples per batch.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} ||\dot{\tilde{\mathbf{x}}}_i - \dot{\mathbf{x}}_i||_2^2 \tag{2.3}$$

Now, we would like to ensure stability of the approximated dynamics $\dot{\tilde{\mathbf{x}}}$: We assume that our system is stabilizable for a bounded control input $\mathbf{u}$ i.e. there exists a bounded control sequence which steers the system to the equilibrium point. Then, Lyapunov stability theory ensures that a dynamical system is stable if there exists a Lyapunov function such that the following inequality is satisfied (2.4):

$$\forall \mathbf{x}, \exists \mathbf{u} \text{ s.t. } ||\mathbf{u}||_\infty \leq 1 \text{ and } \frac{\delta V(\mathbf{x}(t))}{\delta t} = \nabla V(\mathbf{x})^T (f(\mathbf{x}) + g(x)\mathbf{u}) \leq -\alpha V(\mathbf{x}) \tag{2.4}$$

Therefore, we redefine our system dynamics by equation (2.5) where f^* is the optimal approximation of $\tilde{f}$ that ensures stability i.e. f^* is the projection of $\tilde{f}$ such that it satisfies equation (2.4).

$$\dot{\tilde{\mathbf{x}}} = f^*(\mathbf{x}) + \tilde{g}(\mathbf{x})\mathbf{u} \tag{2.5}$$

where

$$\begin{aligned} f^*(\mathbf{x}) = \ & \arg\min_{f \in \mathbb{R}^D} \frac{1}{2}||f - \tilde{f}(\mathbf{x})||_2^2 \\ & \text{s.t.} \min_{||\mathbf{u}||_\infty \leq 1} \nabla \tilde{V}(\mathbf{x})^T (f(\mathbf{x}) + \tilde{g}(\mathbf{x})\mathbf{u}) + \alpha \tilde{V}(\mathbf{x}) \leq 0 \end{aligned} \tag{2.6}$$

By solving the inner minimization problem, equation (2.6) can be reformulated as follows:

$$\begin{aligned} f^*(\mathbf{x}) = \ & \arg\min_{f \in \mathbb{R}^D} \frac{1}{2}||f - f(\mathbf{x})||_2^2 \\ & \text{s.t. } c(\mathbf{x}) \leq 0 \end{aligned} \tag{2.7}$$

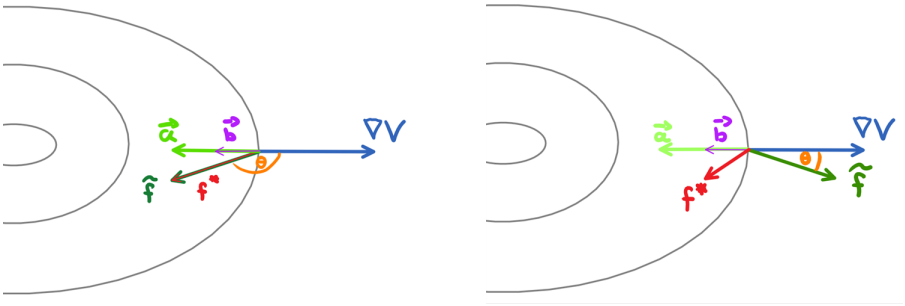
where

$$c(\mathbf{x}) = \nabla \tilde{V}(\mathbf{x})^T f(\mathbf{x}) + ||\Delta \tilde{V}(\mathbf{x})^T \tilde{g}(\mathbf{x})||_1 + \alpha \tilde{V}(\mathbf{x}) \tag{2.8}$$

Finally, since there exists only one constraint there are two *Karush Kuhn Tucker* (KKT) solutions to equation (2.7):

$$f^*(\mathbf{x}) = \begin{cases} \tilde{f}(\mathbf{x}) & \text{if } c(\mathbf{x} \leq 0 \\ \tilde{f}(\mathbf{x}) - \frac{\tilde{V}(\mathbf{x})}{||\tilde{V}(\mathbf{x})||_2^2} c(\mathbf{x}) & \text{otherwise} \end{cases} \tag{2.9}$$

For simplicity, we analyse an autonomous system for a brief moment: The two cases of equation (2.9) are represented in figure 2.1: In case *a*, the constraint is already satisfied because the dynamics drive the system towards the equilibrium point. In case *b*, the convergence is not fast enough or the system is even diverging. Then, we correct $\tilde{f}$ by two terms: $\vec{\mathbf{a}}$ brings the system back to the same gradient line of the Lyapunov function and in addition, $\vec{\mathbf{b}}$ ensures a certain rate of convergence. For controlled systems,



(a) if $\nabla V(\mathbf{x})^T f(\mathbf{x}) \leq -\alpha V(\mathbf{x})$ $\Rightarrow f^*(\mathbf{x}) = \tilde{f}(\mathbf{x})$

(b) if $\nabla V(\mathbf{x})^T f(\mathbf{x}) > -\alpha V(\mathbf{x})$ $\Rightarrow f^*(\mathbf{x}) = \tilde{f}(\mathbf{x}) + \vec{\mathbf{a}} + \vec{\mathbf{b}}$

Figure 2.1: Two cases of f^* if $\mathbf{u} = \mathbf{0}$

the concept remains the same: $\tilde{f}$ is corrected such that there exists a bounded control input for which the system converges at the given rate.

$$\vec{\mathbf{a}} = -\frac{\tilde{V}(\mathbf{x})}{||\tilde{V}(\mathbf{x})||_2} * cos(\theta)||\tilde{f}(\mathbf{x})||_2 \tag{2.10}$$

$$\vec{\mathbf{b}} = -\frac{\tilde{V}(\mathbf{x})}{||\tilde{V}(\mathbf{x})||_2} * \frac{\alpha \tilde{V}(\mathbf{x})}{||\tilde{V}(\mathbf{x})||_2} \tag{2.11}$$

where

$$cos(\theta) = \frac{\nabla V(\mathbf{x})^T f(\mathbf{x})}{||\tilde{V}(\mathbf{x})||_2 ||\tilde{f}(\mathbf{x})||_2} \tag{2.12}$$

The Lyapunov function is approximated as well by a neural network $\tilde{V}(x)$. Two criteria must be satisfied such that $\tilde{V}(x)$ is a Lyapunov function:

1. $\tilde{V}(\mathbf{x})$ should have one global optimum
2. $\tilde{V}(\mathbf{x})$ should be positive definit: $\tilde{V}(\mathbf{0}) = 0$ and $\tilde{V}(\mathbf{x}) > 0$ for $\forall \mathbf{x} \neq \mathbf{0}$

To have one global optimum, we choose an *Input Convex Neural Network* (ICNN) [9] as architecture for $\tilde{V}(\mathbf{x})$. To ensure positive definiteness, the output of the ICNN denoted as $\tilde{h}(\mathbf{x})$ is parameterized by equation (2.13) where σ is a positive convex non-decreasing function and ϵ is a small constant:

$$\tilde{V}(\mathbf{x}) = \sigma(\tilde{h}(\mathbf{x}) - \tilde{h}(\mathbf{0})) + \epsilon||\mathbf{x}||_2^2 \tag{2.13}$$

2.3 Damped harmonic oscillator

2.3.1 System

The first example is the DHO which is a linear system (2.14) (see figure 2.2):

$$\dot{\mathbf{x}}(t) = A\mathbf{x} + B\mathbf{u}(t) \tag{2.14}$$

where the state is defined by $\mathbf{x}(t) = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$ and the matrices are the following:

$$A = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\mu}{m} \end{pmatrix} \tag{2.15}$$

$$B = \begin{pmatrix} 0 \\ -\frac{1}{m} \end{pmatrix} \tag{2.16}$$

The experiment is done with the following parameters: $m = 1$, $k = 0.5$ and $\mu = 0.1$.

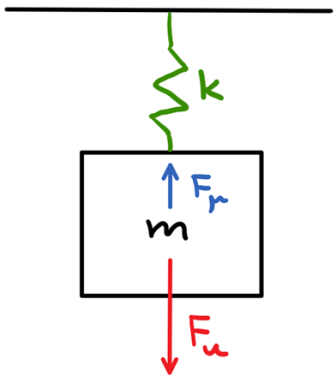


Figure 2.2: Damped harmonic oscillator (F_u: input force, F_μ: friction force, m: mass, k: spring constant)

2.3.2 Learning

These are the model parameters:

- $\tilde{f}(\mathbf{x})$: 1 linear layer
- $\tilde{h}(\mathbf{x})$: 2 hidden layers of size 60, softplus activation fct.
- $\tilde{g}(\mathbf{x})$: 1 linear layer
- Convergence rate: $\alpha = 0.1$

The following parameters are used for the training process:

- Optimizer: Adam
- Initial learning rate: 10^{-4}
- Nb. samples: 1'048'576
- Batch size: 8192
- Nb. epochs: 200

- Test share: 0.1
- Loss: mean squared error

Figure 2.3 shows that the method works well on linear systems like the DHO. The training process is really smooth and the resulting Lyapunov function has - as expected - a circular shape. Comparing the real and the learned dynamics shows the high accuracy of the model.

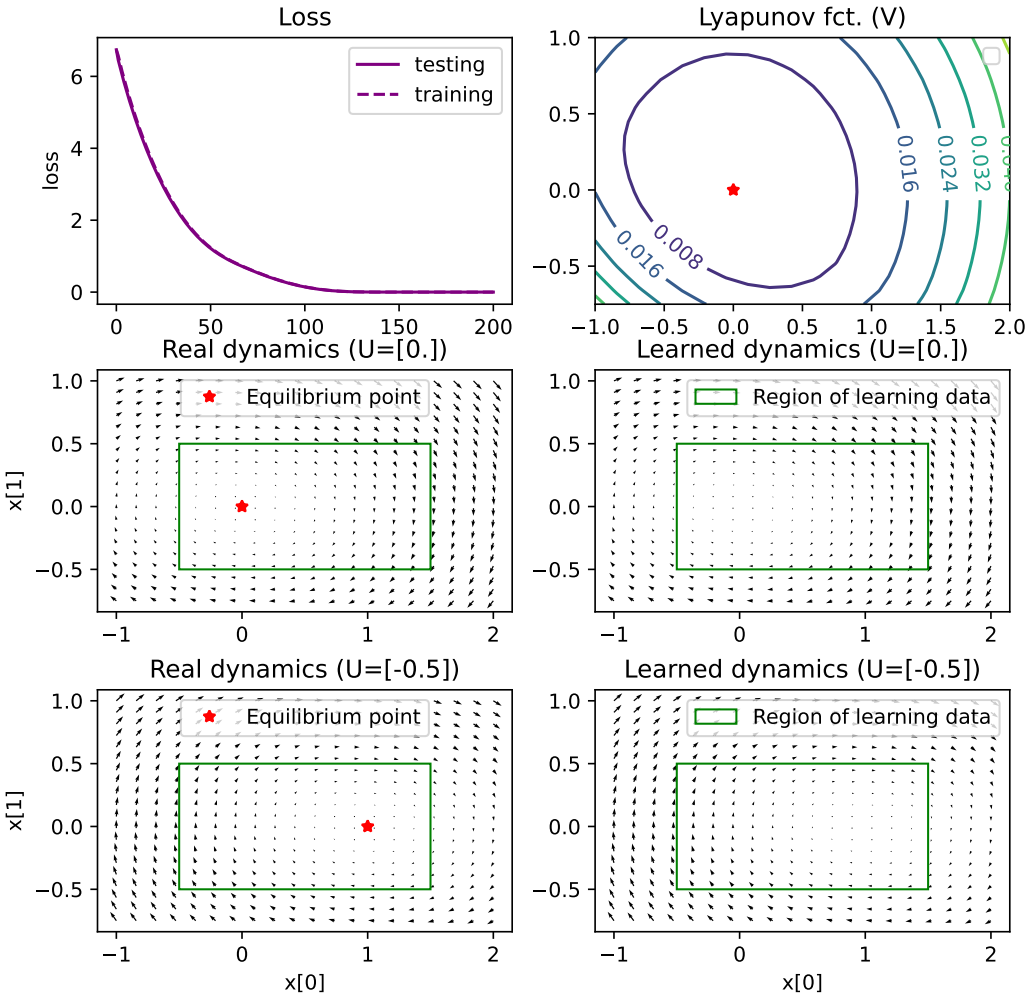


Figure 2.3: DHO, top left: training loss in function of epochs, top right: learned Lyapunov function, center/bottom: real and learned dynamics for two different control inputs

2.4 Continuous stirred tank reactor

2.4.1 System

The CSTR is control affine, but the system is not anymore linear. It is the same system that was used by E. Maddalena et al. [10] (caution: they do not use the same constants for their simulation as mentioned in the paper!). The state is defined by $\mathbf{x}(t) = \begin{pmatrix} c_A(t) \\ c_B(t) \end{pmatrix}$ where c_A and c_B denote respectively the concentrations of cyclopentadiene and cyclopentenol. The dynamics are given by the following differential equations:

$$\dot{c}_A(t) = -\frac{1}{3600} * (\rho_1 c_A(t) + \rho_3 c_A(t)^2 + u(t)(c_A(t) - c_{A0})) \tag{2.17}$$

$$\dot{c}_B(t) = \frac{1}{3600} * (\rho_1 c_A(t) - \rho_2 c_B(t)^2 - u(t) c_B(t)) \tag{2.18}$$

The experiment is done with the following parameters: $\phi_1 = \phi_2 = 14.736 h^{-1}$, $\phi_3 = 2.284 h^{-1}$ and $c_{A0} = 5.1 \frac{mol}{l}$.

2.4.2 Control input mapping

The variables and the control input are sampled in the range of $\begin{pmatrix} 1 \\ 0.5 \end{pmatrix} \leq \begin{pmatrix} c_A(t) \\ c_B(t) \end{pmatrix} \leq \begin{pmatrix} 3 \\ 2 \end{pmatrix}$ and $3 \leq u \leq 35$ respectively. Equation (2.4) bounds the control input between -1 and 1. To be able to use the same formulation, we map $u \in [3, 35]$ to $\hat{u} \in [-1, 1]$. More generally, this can be done for any arbitrary control input (2.19):

$$\hat{f}(\mathbf{x}) + \hat{g}(\mathbf{x})\hat{\mathbf{u}} = \hat{f}(\mathbf{x}) + \hat{g}(\mathbf{x})(A\mathbf{u} + B) = (\hat{f}(\mathbf{x}) + \hat{g}(\mathbf{x})B) + \hat{g}(\mathbf{x})A\mathbf{u} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} \tag{2.19}$$

where A is diagonal matrix with the entries $a_{ii} = \frac{2}{max(u_i) - min(u_i)}$ and B is a vector with the entries $b_i = -\frac{max(u_i) + min(u_i)}{max(u_i) - min(u_i)}$ for $i = 1, 2, ..., M$.

2.4.3 Learning

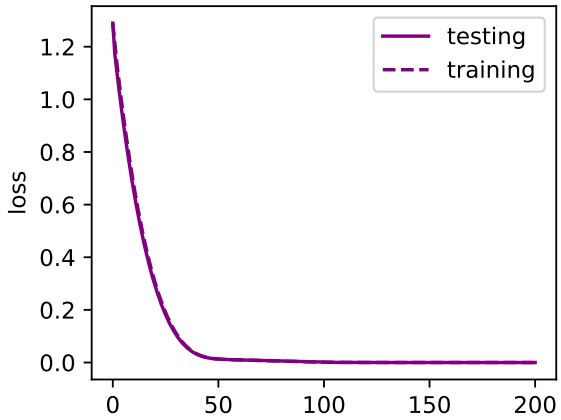


Figure 2.4: CSTR pre-training without Lyapunov correction, (loss in fct. of epochs)

These are the model parameters:

- $\tilde{f}(\mathbf{x})$: 2 hidden layers of sizes 80-200, tangent hyperbolicus (tanh) activation fct.
- $\tilde{h}(\mathbf{x})$: 3 hidden layers of sizes 60-60-30, softplus activation fct.
- $\tilde{g}(\mathbf{x})$: 1 linear layer
- Convergence rate: $\alpha = 0.05$

For better convergence, we pre-trained the model without applying the Lyapunov correction term (setting $f^*(\mathbf{x}) = \tilde{f}(\mathbf{x})$, see figure 2.4) and then continue to train as usual (see figure 2.5). When using this pre-training approach, the training process is much faster and more reliable. The following parameters are used for training:

- Optimizer: Adam
- Initial learning rate: 10^{-5} (10^{-4} when pre-training)
- Nb. samples: 1'048'576
- Batch size: 8192
- Nb. epochs: 50 (+200 pre-training)
- Test share: 0.1
- Loss: mean squared error

Figure 2.5 shows that, as for the DHO, the learned dynamics are very close to the real ones. Here, the main contribution is already done during pre-training. The Lyapunov function is less circular than the one from DHO and it is not clear how it behaves further away from the training region. However, it should still assure the desired convergence because of its design.

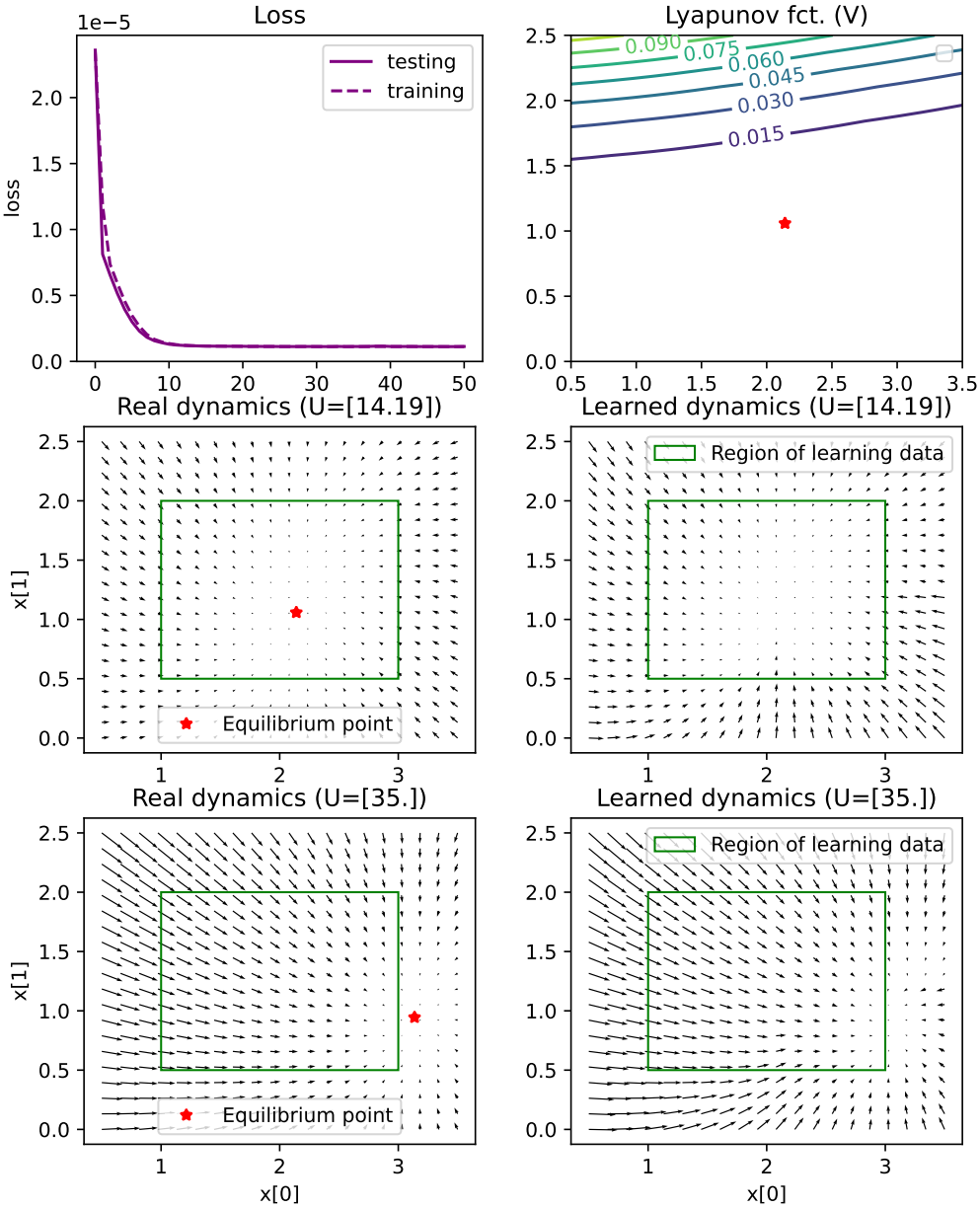


Figure 2.5: CSTR, top left: training loss in function of epochs, top right: learned Lyapunov function, centre/bottom: real and learned dynamics for two different control inputs

2.5 Safety filter

The idea of a safety filter is to find the optimal control input $\mathbf{u}$ such that the system converges towards a given equilibrium point defined by the Lyapunov function. This optimization problem is described by equation (2.20):

$$\begin{aligned} \mathbf{u}^*(\mathbf{x}) = \arg \min_{||\mathbf{u}||_{\inf} \leq 1} & \frac{1}{2}||u - \tilde{u}(\mathbf{x})||_2^2 \\ \text{s.t. } & \Delta V(\mathbf{x})^T (f^*(\mathbf{x}) + \tilde{g}(\mathbf{x})\mathbf{u}) \leq -\alpha \tilde{V}(\mathbf{x}) \end{aligned} \tag{2.20}$$

When doing some simulations using the safety filter, we add some slack such that the optimization is always feasible. However, the simulation does not produce the desired results. First of all, the system diverges even though the Lyapunov function should prevent this. And second, the control input seems to be saturated often while the system becomes unstable: This means $\mathbf{u}$ is either equal to $\mathbf{1}$ or $-\mathbf{1}$ and switches fast between the two boundary values. This is why we are forced to change the optimization problem of equation (2.6). The new formulation is still in development. Please contact R. Schwan (roland.schwan@epfl.ch) for more information.

Chapter 3 Thrust Curves

3.1 Goal

When collecting data to learn the stabilizable dynamics for the Holohover, we observed that there is a large error between the model that was developed during the previous semester project [3] and the real data. One possible reason is that the signal-to-thrust mapping is not sufficiently accurate. This function is approximated by a third-order polynomial function. In the previous experimental set-up, a single motor was mounted on top of a scale which estimated the force indirectly by measuring the weight [3]. This method has multiple weaknesses: The force is measured indirectly by a scale and not by a dedicated force sensor. The different motors are not distinguished even though they may have different thrust curves. The experiment is not done with the real system but with an isolated motor. Therefore, it neglects any Holohover-specific effects, for example, the influence of aerodynamics. Now that the lab is in possession of a high-precision force sensor, we would like to repeat this experiment.

3.2 Set-up

We are using the force sensor from *Bota Systems* [11] which is an extremely sensitive force-torque sensor. The sensor has a noise-free resolution of 0.2N defined as 6σ of a signal over 1 second of measurements in stable environmental conditions. Note that we would like to measure signals below 0.2N. Therefore, we need to average the measurements to get rid of the noise. The Holohover is fixed on a 3D printed mount that is screwed on top of the force cell 3.1. The force sensor is connected via Ethernet to the computer and a dedicated ROS node publishes the measurements. To avoid any effect of battery discharging, the Holohover is connected to a power supply. *control_signal_node.cpp* applies 16 increasing control inputs in the range of $(0.0, 1.0]$ for every motor. Each signal is applied for 4s with a 1s pause in between. At the same time, the force in the x and the y directions as well as the control input are logged.

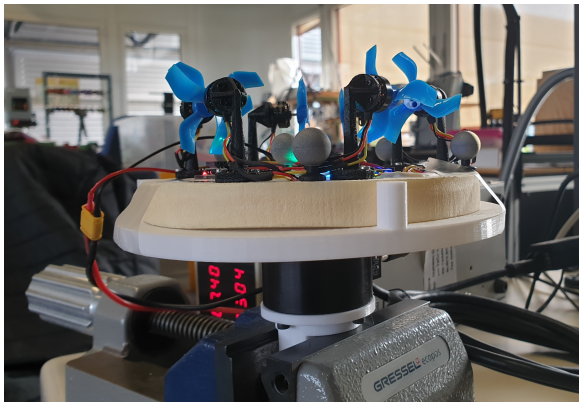


Figure 3.1: Force sensor measurement set-up

3.3 Data preparation

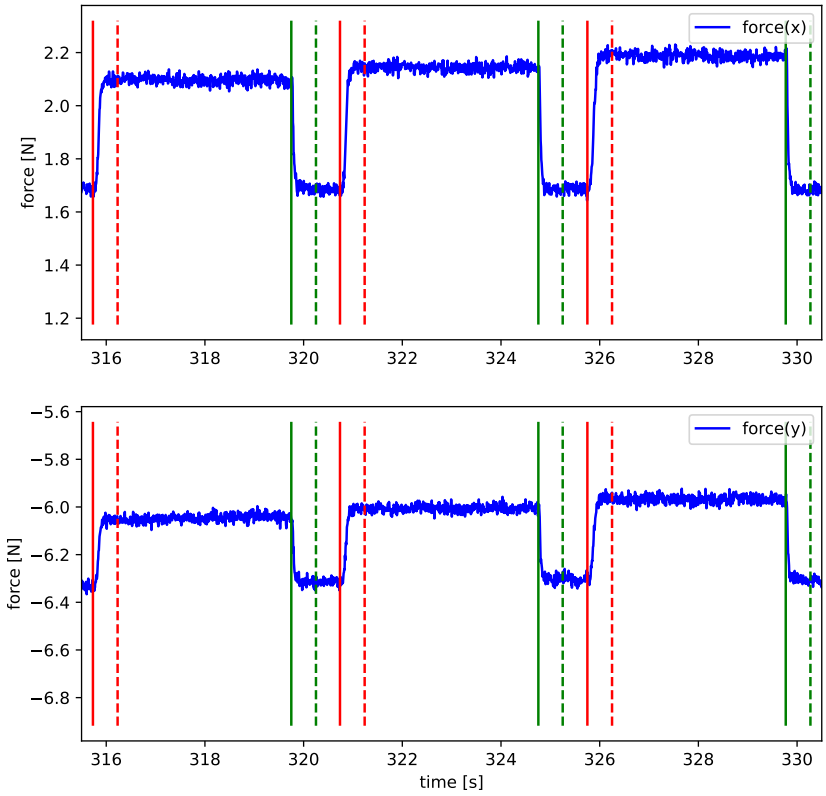


Figure 3.2: Localise signal (solid red line: start sending a signal, solid green line: stop sending a signal, dashed red/green line: delayed by 0.5s to avoid transition zone)

Before analysing the measurements the data is processed as follows:

1. Convert time stamp to seconds (take the first measurement as reference zero)
2. Crop data in the beginning when no signal is applied
3. Interpolate control input such that it matches the force measurements in time (similar to point 3 in chapter 4.3)
4. Localise signal: The localisation of the signal is shown in figure 3.2: The solid lines mark the moment when starting (in red) and stopping (in green) to send the signal. However, to avoid any transition zone the measurements are analysed only after a delay of 0.5s (dashed lines).
5. Remove force biases: As seen in figure 3.2, the force measurements are biased and we subtract the mean of the background from the signal: $F_x = M_x - b_x$ and $F_y = M_y - b_y$. The bias drifts over the course of the measurement. Therefore, we remove the bias piece-wise by subtracting $b_{x/y}$ (the mean between the dashed green and the solid red line) from the local signal $M_{x/y}$ (the region between two solid green lines, see figure 3.2)
6. Remove noise bias: The thrust of the motors is the norm of the forces: $N_{biased} = \sqrt{F_x^2 + F_y^2}$. Using Pythagoras' formula on the noisy force measurements leads again to biased data because the squared noise is strictly positive. We assume that the noise is independent of the measurement i.e. it is orthogonal to the force. Therefore, we remove the squared noise as follows: $N = \sqrt{N_{biased}^2 - b_{noise}^2}$

These steps lead to the force measurements seen in figure 3.3.

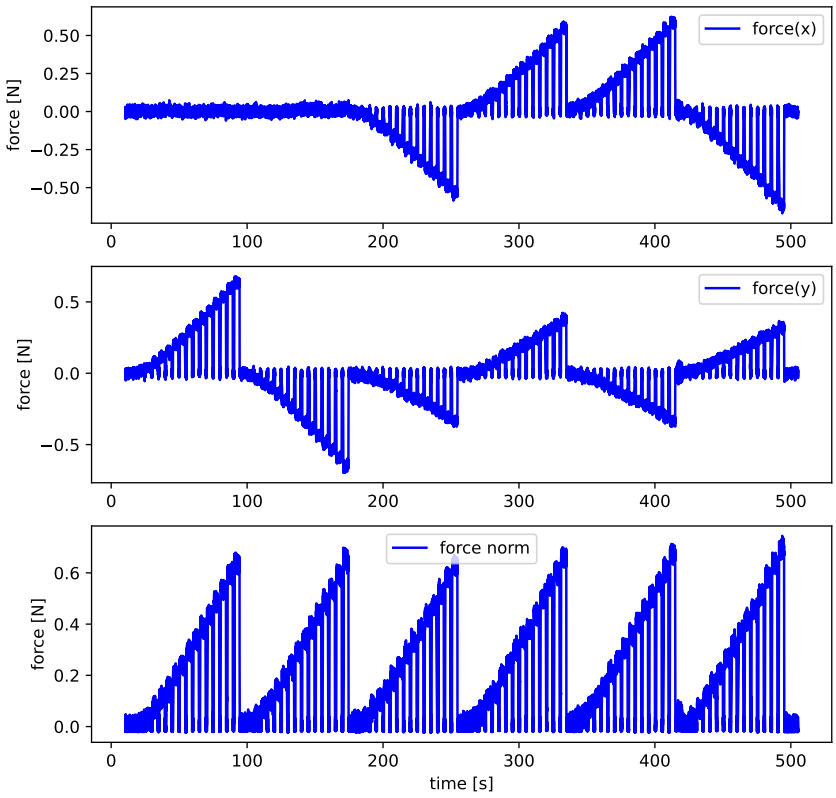


Figure 3.3: Unbiased force measurement and norm of thrust

3.4 First measurements

3.4.1 Results

Figure 3.4 shows the response of 16 increasing signals in the range of (0.0,1.0] for motor 1. In violet, there is the thrust measurement when the signal is on (4s) and in blue when it is off (1s). In green, we fit an exponential function to the measurement. This function has two degrees of freedom: τ is the time constant and δ is the delay before the function starts rising/falling. We identify two zones: In grey, the "dead zone" between zero and δ. Here, the signal is already set/cleared but the thrust does not yet respond. Then there is the "exponential zone" in brown where the thrust rises/falls to the steady state. We define the end of this zone when the approximation reaches 95% of the final value ($trans = -\tau * ln(1 - 0.95)$. To avoid fitting the noise, only those signals are used that have a mean thrust that is twice as high as the peak-to-peak noise of the background (between green dashes and red solid lines in figure 3.2). Hence, normally the first 4 to 5 signals between 0.0625 and 0.25/0.3125 are not used for further calculations.

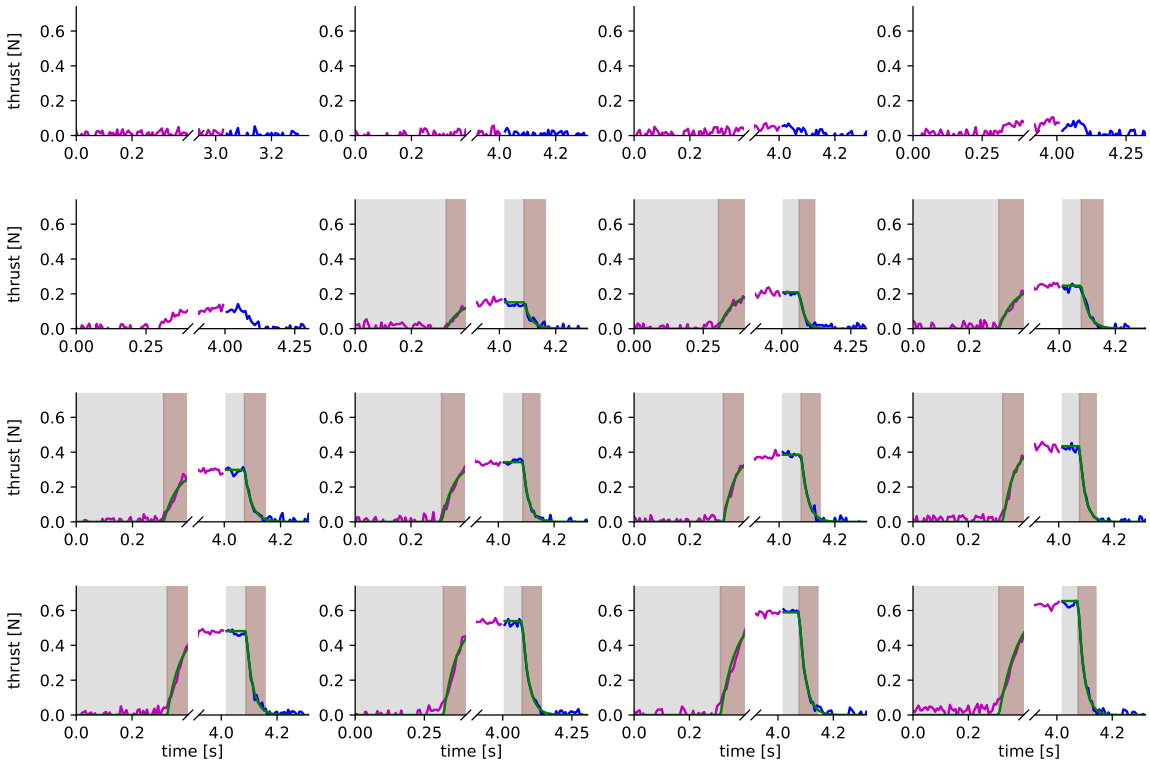


Figure 3.4: Motor 1 thrust measurements of 16 increasing signals: 0.0625, 0.125, ..., 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

3.4.2 Discussion

It makes sense that there exists a delay δ because when the signal is set/-cleared, first it is sent to the Holohover, then it is processed by Betaflight and applied by the motors and finally, the thrust is measured by the force sensor and the measurement is sent back to the computer. However, there is no reason why this delay is longer when the thrust is rising compared to when it is falling (see figure 3.4). The average δ_{down} for all motors is 67ms which is in the range of the expected delay. On the other hand, the average δ_{up} is 313ms which is more than 4.5 times longer. The average transition time is equal to $trans_{up} = 135ms$ and $trans_{down} = 65ms$. We identify two major problems:

1. Start-up delay: δ_{up} is at minimum 246ms longer as the expected delay.
2. Transition time is not negligible and should be modelled.

3.4.3 Idle thrust

The motors of the Holohover are normally used for drones and they are always turned on during usage. Therefore, we think that we can reduce the start-up delay δ_{up} significantly by slightly rotating the rotors in idle state. We choose the idle signal equal to 0.03 such that it is high enough to turn the motors continuously but as small as possible.

3.4.4 Motor transition approximation

To solve the second problem we approximate the thrust transition with a first-order model (3.1). By applying the Laplace transform, we get (3.2), where $x(t)$ is a step function, y_{final} the desired thrust and $y(t)$ the measured/real thrust. Further, we determine the differential equation (3.3) and make an Euler integration (3.4) to calculate the real thrust recursively in discrete time. The step function x^i of size y_{final} becomes the desired thrust $f(u^i)$ where u^i is the control input at time step i, f is the mapping from signal-to-thrust and T is the periode.

$$y(t) = (1 - e^{-t/\tau}) * y_{final} \tag{3.1}$$

$$Y(s) = \frac{y_{final}}{s * (1 + \tau * s)} = \frac{y_{final}}{1 + \tau * s} * X(s) \tag{3.2}$$

$$\frac{\delta y(t)}{\delta t} = \frac{1}{\tau} * (y_{final} * x(t) - y(t)) \tag{3.3}$$

$$y^{(i+1)} = y^{(i)} + T * \frac{f(u^{(i)}) - y^{(i)}}{\tau} = y^{(i)} * (1 - \frac{T}{\tau}) + f(u^{(i)}) * \frac{T}{\tau} \tag{3.4}$$

This model does not include any time delay (as for example δ_{up}) and remains time invariant i.e. the states form a Markov chain where each state depends exclusively on the previous one. However, the state must be augmented to capture the information about the real thrust. Equation (3.5) describes the system dynamics before augmentation where f is the signal-to-thrust mapping, M is the mapping from thrust to acceleration in the world frame and A and B are the matrices of a linear control affine system. Equations (3.6) and (3.7) describe the augmented system:

$$\dot{\mathbf{x}} = A\mathbf{x} + BMf(\mathbf{u}) = A \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} + BM(\theta) f(\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}) \tag{3.5}$$

$$\dot{\mathbf{y}} = C\mathbf{y} + Df(\mathbf{u}) = (1 - \frac{T}{\tau})I \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} + \frac{T}{\tau} If(\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}) \tag{3.6}$$

$$\dot{\tilde{\mathbf{x}}} = \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{y}} \end{bmatrix} = \begin{bmatrix} A & BM(\theta) \\ (0) & C \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} (0) \\ D \end{bmatrix} f(\mathbf{u}) \tag{3.7}$$

Note that in the simulation code of this semester project θ and $\dot{\theta}$ are at positions 2 and 5 of the state vector, whereas in the c++ ROS implementation these are at positions 4 and 5.

3.5 Second measurements

3.5.1 Results

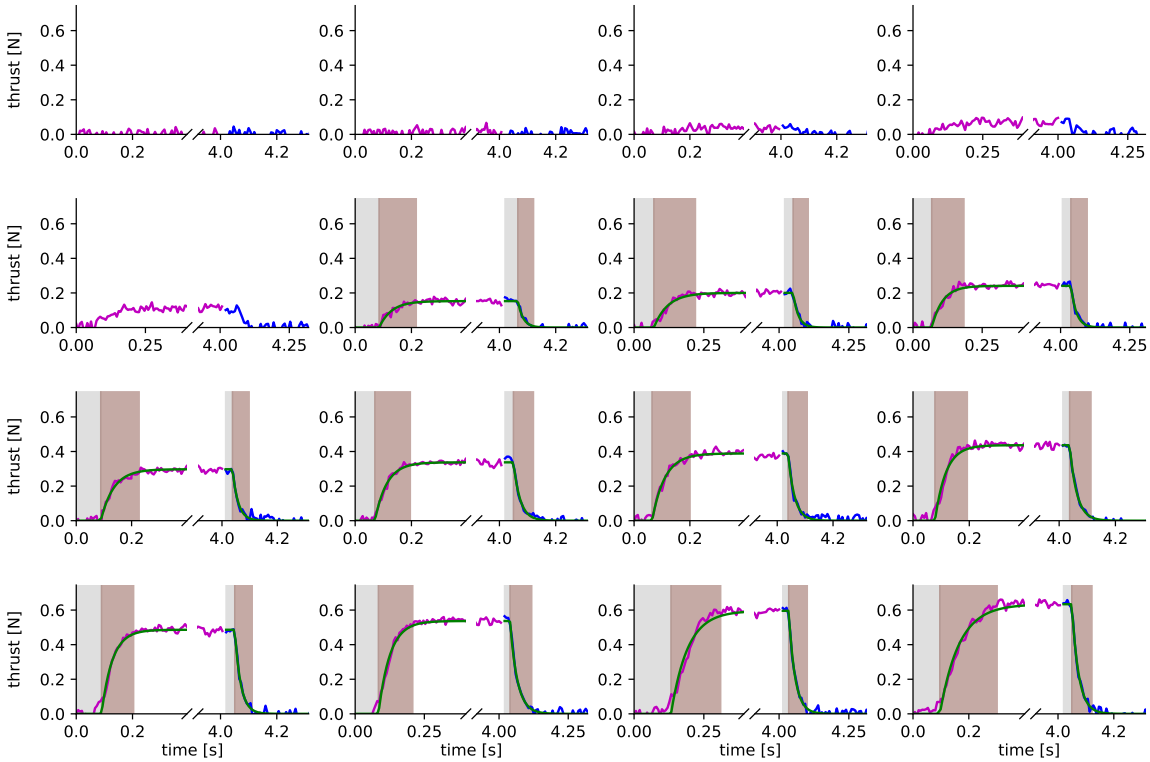


Figure 3.5: Motor 1 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

In chapter 3.4, we discussed two major problems: the start-up delay δ_{up} and the thrust transition. We implemented an idle thrust to reduce δ_{up} and a first-order model to approximate the thrust transition. The thrust curves of motor 1 are shown in figure 3.5. Motors 2 to 6 have similar results and are represented in figures 6.4 to 6.8 in the annexe. Figure 3.6 summarizes the delays and time constants for all motors.

3.5.2 Discussion

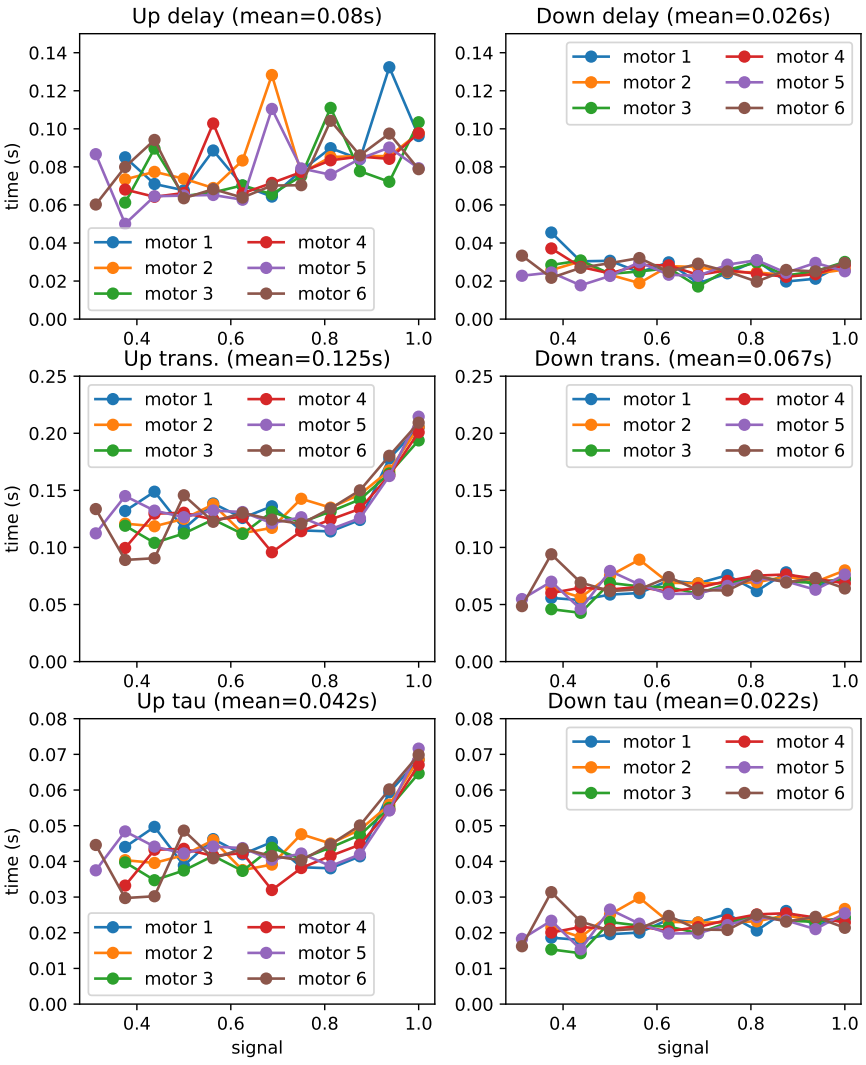


Figure 3.6: δ: duration of "dead zone" where the signal is already set/cleared, but the thrust is not yet reacting; *trans*: duration of transition zone; τ: time constant of exponential transition function (for $trans_{up}$ and tau_{up} the last two signals are not considered to calculate the mean)

The average delay δ_{up} is equal to 80ms and much smaller than before (313ms). Therefore, we conclude that adding an idle thrust helps to reduce this delay. Interestingly, the average $\delta_{down} = 26ms$ is also smaller than before (67ms). δ_{up} is still about 3 times larger than δ_{down}. Hence, about 54ms of δ_{up} cannot be explained by the delay in the measurement loop. This part is also not approximated by the first-order model. Looking at the thrust transition, we see that the time constant $\tau_{up} = 42ms$ is almost twice $\tau_{down} = 22ms$. Therefore, we use two different time constants depending on if the desired thrust is larger or smaller than the current thrust. This means the system switches between two modes where the only difference is a constant scaling of the matrices C and D from equation (3.6).

Due to time constraints, the augmented system is not yet implemented in the Holohover ROS node but only for data analysis. One important difference is that when analysing the data the approximation acts on the control input and not on the thrust itself (as described in chapter 3.4.4) because this is much simpler to implement. The τ determined in the previous section is used to approximate the thrust. However, for the implementations of this semester project, $\tau_{signal-space}$ is used which describes how the transition of the signal evolves in function of time: First, the thrust measured with the force sensor is converted to a signal using the thrust-to-signal mapping (which is discussed in chapter 3.6) and then, $\tau_{signal-space}$ is determined in signal space (see figure 6.2). The results in signal space ($\tau_{up} = 38ms$ and $\tau_{down} = 28ms$, see figure 6.1) are similar to the one in thrust space ($\tau_{up} = 42ms$ and $\tau_{down} = 22ms$, see figure 3.6).

3.6 Signal-to-Thrust curves

3.6.1 Results

Using the results from chapter 3.5, the signal-to-thrust mapping is calculated. Figure 3.7 shows that the standard deviation of the 16 measurements is relatively small. We explore polynomial fitting for degrees between 2 and 5. The polynomial functions have no constant term because for a control input equal to zero, the thrust is zero as well. Similarly, the thrust-to-signal mapping is determined by the inverse function (see figure 6.3).

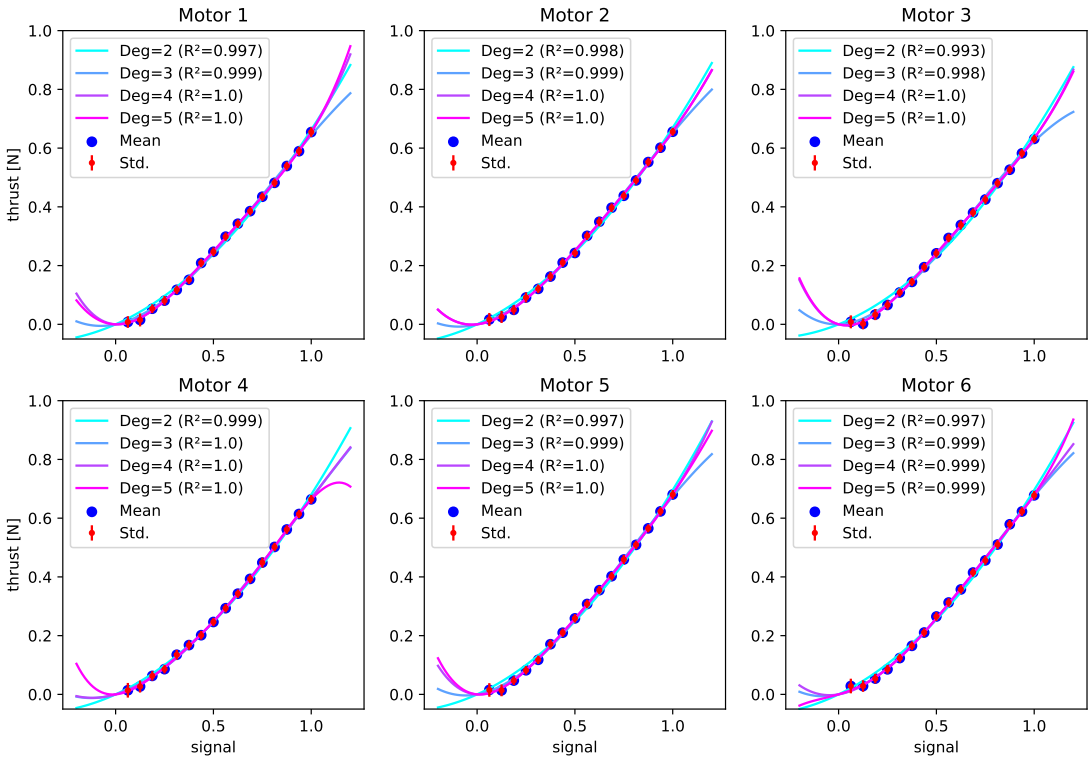


Figure 3.7: Signal to thrust mapping for all motors (cyan blue to pink: polynomial fits of various degrees, blue points: mean of measurements, red bars: standard deviations of measurements)

3.6.2 Discussion

Figure 3.7 shows that the signal-to-thrust mapping is quasi-linear in the range of [0.3, 1.0]. The R^2 value is almost equal to one for all approximations. For the white box model, we choose a polynomial of degree 3 because degree 2 has a larger error for signals below 0.3 and polynomials of order 3 and higher are very similar in the range of [0,1]. As figure 3.8 shows, the polynomial functions are not exactly the same for the different motors. Therefore, each motor uses its own coefficients.

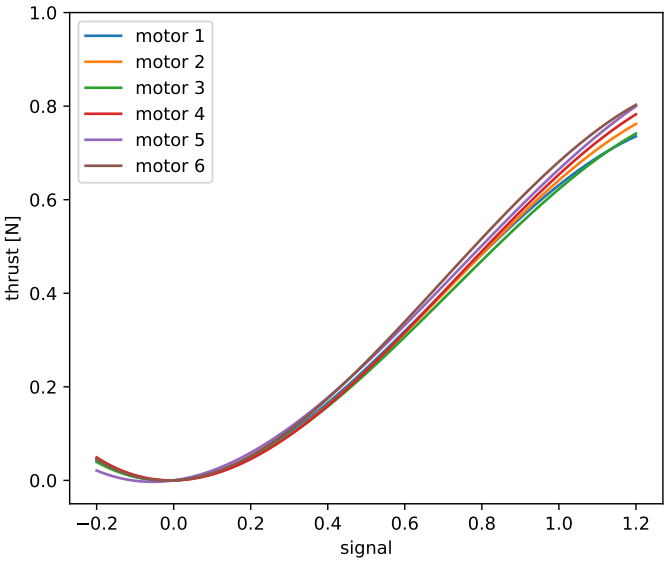


Figure 3.8: Signal to thrust mapping for all motors using a third order polynomial (without constant term)

Chapter 4 System Dynamics

4.1 Goal

In chapter 3, we determined the signal-to-thrust curves for all motors (see chapter 3.6). This mapping is very essential for many control systems. Further, we addressed two basic issues: The start-up delay is reduced by introducing an idle thrust and the transition zone is approximated by a first-order model (see chapter 3.4.2).

Now, the goal is to identify the system dynamics: First, data must be collected through some experiments. Second, we want to evaluate/learn three different models using this data: the white, grey and black box model.

4.2 Experiment

4.2.1 Set-up

The Holohover floats on top of the air hockey table. It receives commands sent by the *Micro Agent* node via WLAN from the main computer. At the same time, the MCS detects the position of the Holohover using 5 markers and the dedicated software running on a separate computer. Then, the position is sent to the main computer, which runs the controller node, via WLAN.

4.2.2 Controller

The LQR controller that was previously implemented proves to be really unstable because of two reasons: First, it uses a model that was never validated and flips the sign of θ for any transition from $+\pi$ to $-\pi$ which causes issues. Hence, it happens often that the controller overreacts, turns too much in one direction and starts to spin crazy. A second issue is that the Kalman filter is not well-tuned. In order to have a better state estimation, the covariance of the IMU and the mouse sensor are set to very high values such that they are ignored completely. In theory, these sensors would enhance the state estimation, but before using them, the sensors would need to be calibrated and the measurement noise would need to be estimated. Also, the MCS is sufficient

for our experiment because it has millimetre accuracy and broadcasts the position with 240Hz.

4.2.3 Design

We would like to collect data over the entire signal range [0,1]. In the best case, the control input, velocity, and acceleration would be equally distributed. This would facilitate any evaluation/learning process afterwards. However, there are two major constraints when designing the experiment:

1. The Holohover should not collide with any wall.
2. Any transition from $\theta = -\pi$ to $\theta = +\pi$ should be avoided due to the LQR controller

The first idea is to drive the Holohover towards random positions on the air hockey table. However, the dynamics do not depend on the position and therefore, it makes no sense to ensure random position while the control input, the velocity and the acceleration are given by the LQR-controller and most probably not equally distributed.

In a second attempt, we move the Holohover with a random velocity in the x and y directions. As soon it crosses an imaginary boundary, the velocity is reflected on the border to prevent colliding with any wall. In simulation, we are able to show that this approach leads to equally distributed position and velocity (see figure 4.1). Also, we hope to get a balanced acceleration and control input by alternating the velocities frequently. However, the LQR controller has conflicting interests: On the one hand, it should track the random velocity and on the other hand, it calculates a velocity by minimizing its cost function. We believe that this is the reason why this approach fails when testing.

In the final attempt, we use the LQR controller as a save guard that drives the Holohover back to an equilibrium point. Every 100ms, there is a new random noise in the range of [-0.6, 0.6] added to this control input. Note that only one noise is added to each pair of motors preventing two motors facing each other to be used at the same time:

```
1    // determine to which motor random signal is added
2    float u_pairA = u_signal(0) - u_signal(1);
3    if(u_pairA + u_randA > 0) {
4        u_signal(0) = u_pairA + u_randA;
5        u_signal(1) = 0;
6    } else {
7        u_signal(0) = 0;
8        u_signal(1) = -(u_pairA + u_randA);
9    }
10
11   // clip between 0.03 and 1
12   u_signal = u_signal.cwiseMax(IDLE_SIGNAL).cwiseMin(1);
```

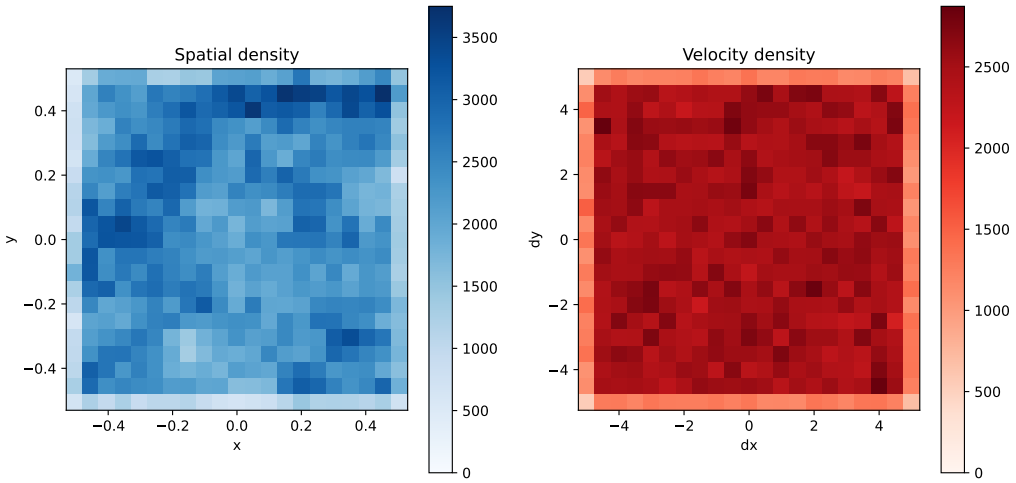


Figure 4.1: Experiment simulation with random velocity changed every 10ms and reflected at a border (borders: $-0.5m$ and $0.5m$, i.i.d. random velocity in the range $[-5\frac{m}{s}, 5\frac{m}{s}]$, simulation time: $10^3 s$, simulation frequency: $10^3 Hz$)

This approach does not provide an equally distributed state or control input, but it is working in practice. Also, the data could be sampled during training/evaluation if balanced distributions are desired.

Figure 3.5 shows that the average transition time is equal to $trans_{up} = 125ms$ and $trans_{down} = 67ms$. This means if we change every 100ms the added noise, the motors are most times in the transition zone. This is exactly what we want because air hockey is a really dynamic game and the system must be identified for fast-changing control inputs. It may even be impossible for higher signals, to reach the motor's steady state without crashing into a wall because the air hockey table is relatively small.

4.3 Data preparation

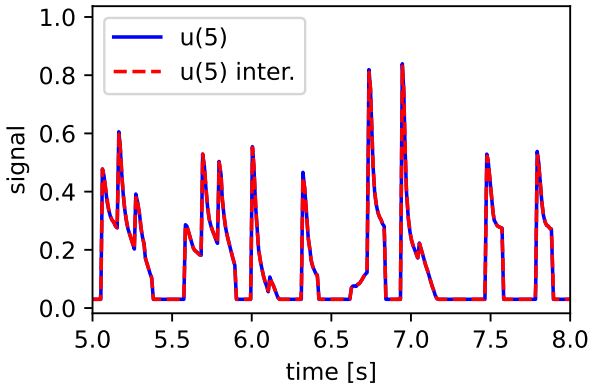


Figure 4.2: Interpolated control input such that it matches the Optitrack data in time (u(5): control input of motor 5)

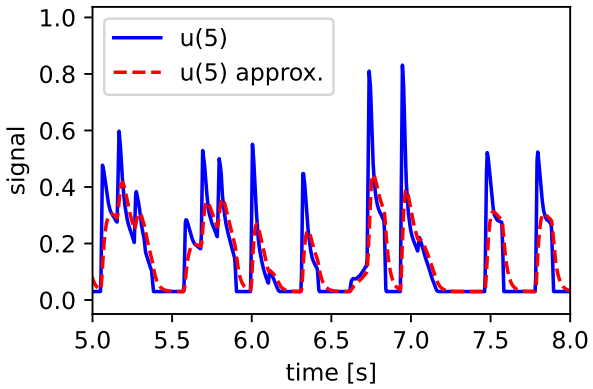


Figure 4.3: Approximation of the control input by first-order model (u(5): control input of motor 5)

Before analysing the measurements the data is processed as follows:

1. Convert time stamp to seconds (take the first measurement as reference zero)
2. Crop data in the beginning when no signal is applied and at the end if an angle of $\pm 3 rad$ is surpassed.
3. Interpolate control input such that it matches the Optitrack data in time using the 1D interpolation from *scipy* [12] (see figure 4.2).

4. Apply first-order approximation on the control input (not the thrust). In signal space, τ has slightly different values than in thrust space, but the concept remains the same (see chapter 3.5.2). Figure 4.3 shows the approximated control input and how the first-order model acts as a low-pass filter.

5. The MCS only provides positional information. These measurements must be integrated twice to get the acceleration. We use the *Savitzky Golay* differentiation of *pysindy* [13], [14] of $order = 3$ and $neighbourhood = 0.05$ to the left and right. We apply this smoothed differentiation because differentiating the measurements without smoothing leads to really steep changes in acceleration due to noise. Figure 4.4 shows on the left the velocity and the acceleration that is derived from the position. On the right, the velocity and the acceleration are integrated back using Euler integration and the error is calculated w.r.t. the original position. During 50s of one experiment, the maximum error is only about $\pm 15cm$ for x and y and $0.4rad$ for θ.

6. There is a delay between the control input and the Optitrack data: The control input must first be sent via WLAN to the Holohover, processed by the motors and only then the position is tracked by the MCS and sent via WLAN to the main computer. Therefore, we calculate the cross-correlation between the Optitrack and the control input acceleration using *signal.correlate* from *scipy* [12] where the Optitrack acceleration is the differentiated Optitrack position and the control input acceleration is the acceleration estimated using the white box model. Then, the position of the maximum cross-correlation is evaluated and the signals are shifted to reduce the error. The white box model depends on the angle θ that is shifted afterwards. Hence, the cross-correlation and the shift of the signals are recalculated in an iterative process until the shift becomes zero. For our data, the optimum is already reached after one step and is equal to 20 shifts corresponding to 83ms. This is a reasonable value because it includes two transmissions via WLAN and all the processing of the Holohover and the MCS. The maximum delay when performing the force sensor measurements is approximately 26ms (see $delay_{down}$ in figure 3.6). However, there the force sensor is connected via Ethernet to the computer which is expected to be faster than WLAN. Also, probably the MCS does more calculations in the background than the force sensor. The difference between the original and the shifted data is shown in figure 4.5.

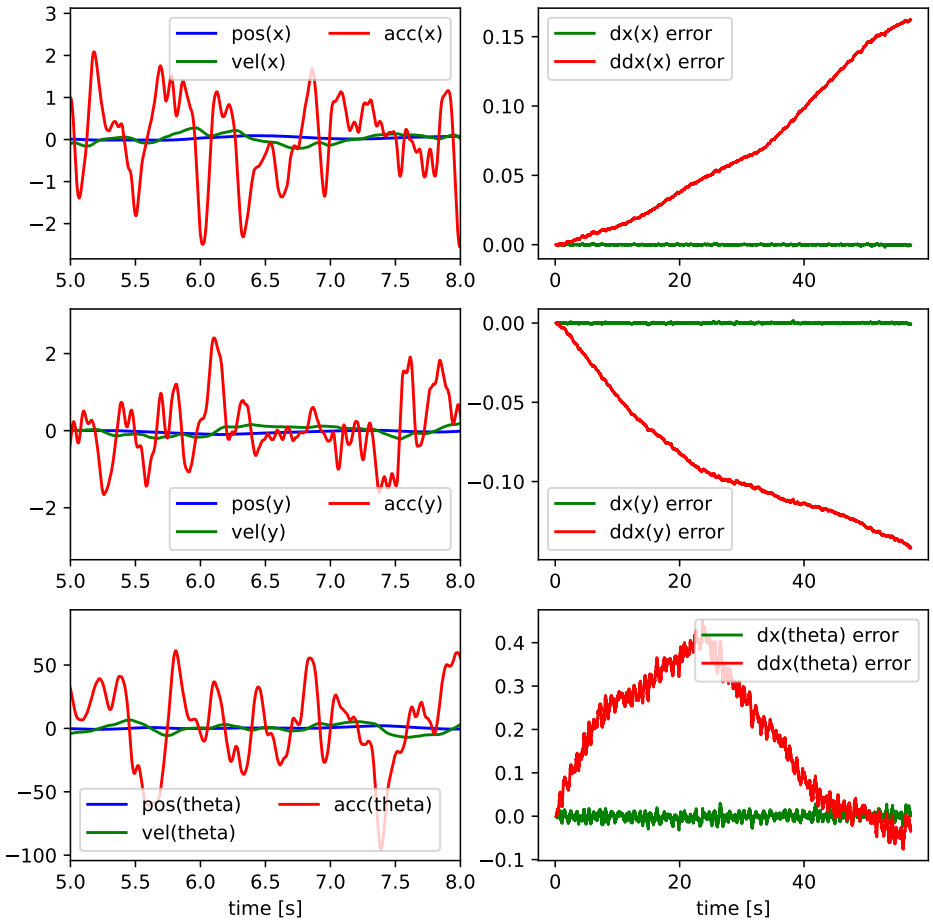


Figure 4.4: Left: differentiation of the Optitrack position data, right: error estimation by integrating back the differentiated values

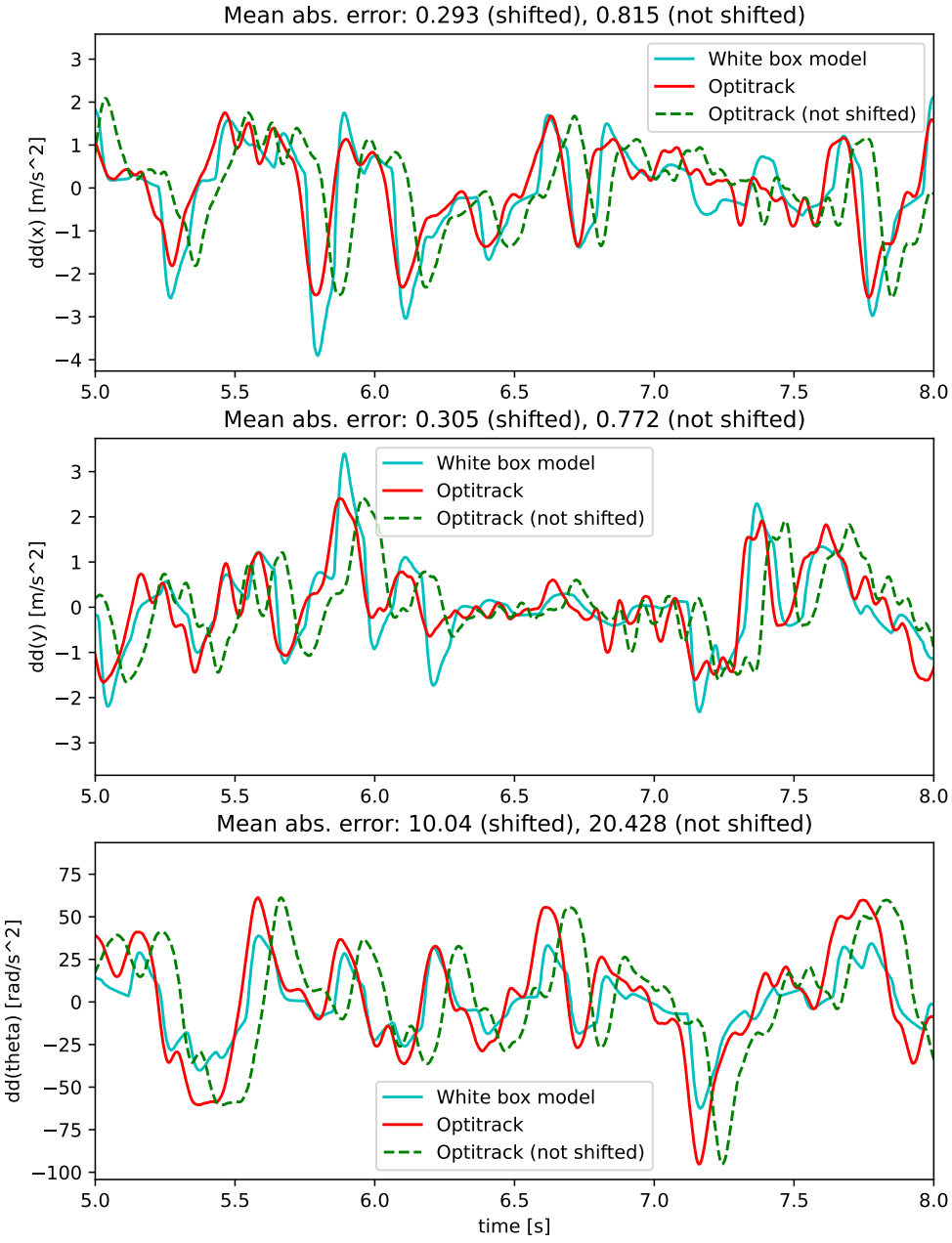


Figure 4.5: Shifting Optitrack data with respect to control input data (cyan blue: acceleration calculated with control input and white box model, red: shifted Optitrack acceleration, green: not shifted Optitrack acceleration)

4.4 Models

4.4.1 White box model

$$\dot{\mathbf{x}} = A\mathbf{x}+BM(\theta)f(\mathbf{u}) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} M(\theta)f(\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}) \tag{4.1}$$

The dynamics of the white box model are described by the control affine system of equation (4.1). The two unknown entities are the function f and the matrix M: f is the signal-to-thrust mapping that is approximated by a third-order polynomial in chapter 3.6. $M \in \mathbb{R}^{3x6}$ is the transformation from the thrust vector that contains the norm of all motor thrusts to the acceleration in the world frame. The transformation M is split into several more comprehensive steps:

1. Map norm of thrusts $f(\mathbf{u})$ to thrust vectors in body-frame using the geometry of the Holohover (4.2) (where $\odot$ is the element-wise matrix multiplication)
2. Sum forces over all dimensions (4.3).
3. Convert sum of forces from body to world frame (4.4).
4. Calculate sum of moments where $\mathbf{r}_i$ is the vector to motor i (4.5).
5. Divide sum of forces by mass and sum of moments by inertia and calculate acceleration (4.6).

$$\begin{bmatrix} \mathbf{F}_1^T \\ \mathbf{F}_2^T \\ \mathbf{F}_3^T \\ \mathbf{F}_4^T \\ \mathbf{F}_5^T \\ \mathbf{F}_6^T \end{bmatrix} = \begin{bmatrix} F_{x1} & F_{y1} & F_{z1} \\ F_{x2} & F_{y2} & F_{z2} \\ F_{x3} & F_{y3} & F_{z3} \\ F_{x4} & F_{y4} & F_{z4} \\ F_{x5} & F_{y5} & F_{z5} \\ F_{x6} & F_{y6} & F_{z6} \end{bmatrix} = \begin{bmatrix} -0.000 & 1.0 & 0.0 \\ 0.000 & -1.0 & 0.0 \\ -0.866 & -0.5 & 0.0 \\ 0.866 & 0.5 & 0.0 \\ 0.866 & -0.5 & 0.0 \\ -0.866 & 0.5 & 0.0 \end{bmatrix} \odot \begin{bmatrix} f(\mathbf{u}) & f(\mathbf{u}) & f(\mathbf{u}) \end{bmatrix} \tag{4.2}$$

$$\mathbf{F}_{body} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \sum_{i=1}^{6} \mathbf{F}_i \tag{4.3}$$

$$\mathbf{F}_{world} = R^{bw}\mathbf{F}_{body} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{F}_{body} \tag{4.4}$$

$$\mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \sum_{i=1}^{6} \mathbf{r}_i \times \mathbf{F}_i \tag{4.5}$$

$$\mathbf{a} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{F_x}{m} \\ \frac{F_y}{m} \\ \frac{M_z}{I_z} \end{bmatrix} \tag{4.6}$$

The geometrical parameters are estimated from the design. The mass is measured using a scale and the inertia is approximated by the formula for a homogeneous disc.

4.4.2 Grey box model

The grey box model uses exactly the same equations as the white box model (see chapter 4.4.1). However, in the grey box model the parameters are learned from data. We allow the system to learn the following parameters:

- Motor position and orientation relative to the body-frame
- Center of mass relative to the body-frame
- Inertia around z-axes
- Signal-to-thrust coefficients

The mass is not learned because it is measured with a scale that has a resolution of less than $1mg$. The following parameters are used for the training process:

- Optimizer: Adam
- Initial learning rate: $lr_{CoM} = 10^{-6}$, $lr_{inertia} = 10^{-7}$, $lr_{sig2thrust} = 10^{-4}$, $lr_{motor-pos} = 10^{-5}$, $lr_{motor-vec} = 10^{-5}$
- Nb. samples: 159'041
- Batch size: 8192
- Nb. epochs: 500
- Test share: 0.1
- Loss: mean squared error

We add two regularization terms to the loss. The first one prevents the optimizer to deviate much from the initial motor positions. This is necessary because a measurement error of the motor positions in the order of mm is possible but not in the order of cm. The second term ensures that the motor vectors remain unit vectors. If this is not the case, then the motor vectors

would start to scale the thrusts instead of just estimating their directions. The final loss is the following (4.7):

$$L = \sum_{i=1}^{6} \frac{(\dot{x}_i^r - \dot{x}_i^l)^2}{\sigma_i} + 10\sqrt{\sum_{i=1}^{6}\sum_{j=1}^{3}(p_{ij}^r - p_{ij}^l)^2} + 0.1\sum_{i=1}^{6} |||v^l||_2 - 1| \quad (4.7)$$

where r stands for real and l for learned, σ_i is the standard deviation of the squared data for dimension i and used to scale the error, i loops over the number of motors and j over the number of dimensions, p_{ij} indicate the jth dimension of the position from motor i and v_i is the vector of motor i pointing in the direction of the thrust.

The training process and a comparison between the grey box and the white box models are represented in figure 4.6. The reason why the error in the y direction is lower than the one in the x direction could be that in the body frame, one of the motor pairs is oriented completely in the y direction and the Holohover is most often oriented around $\theta = 0$ during the experiment. This means that in the y direction, the model has more freedom to adapt and may reduce the error more significantly.

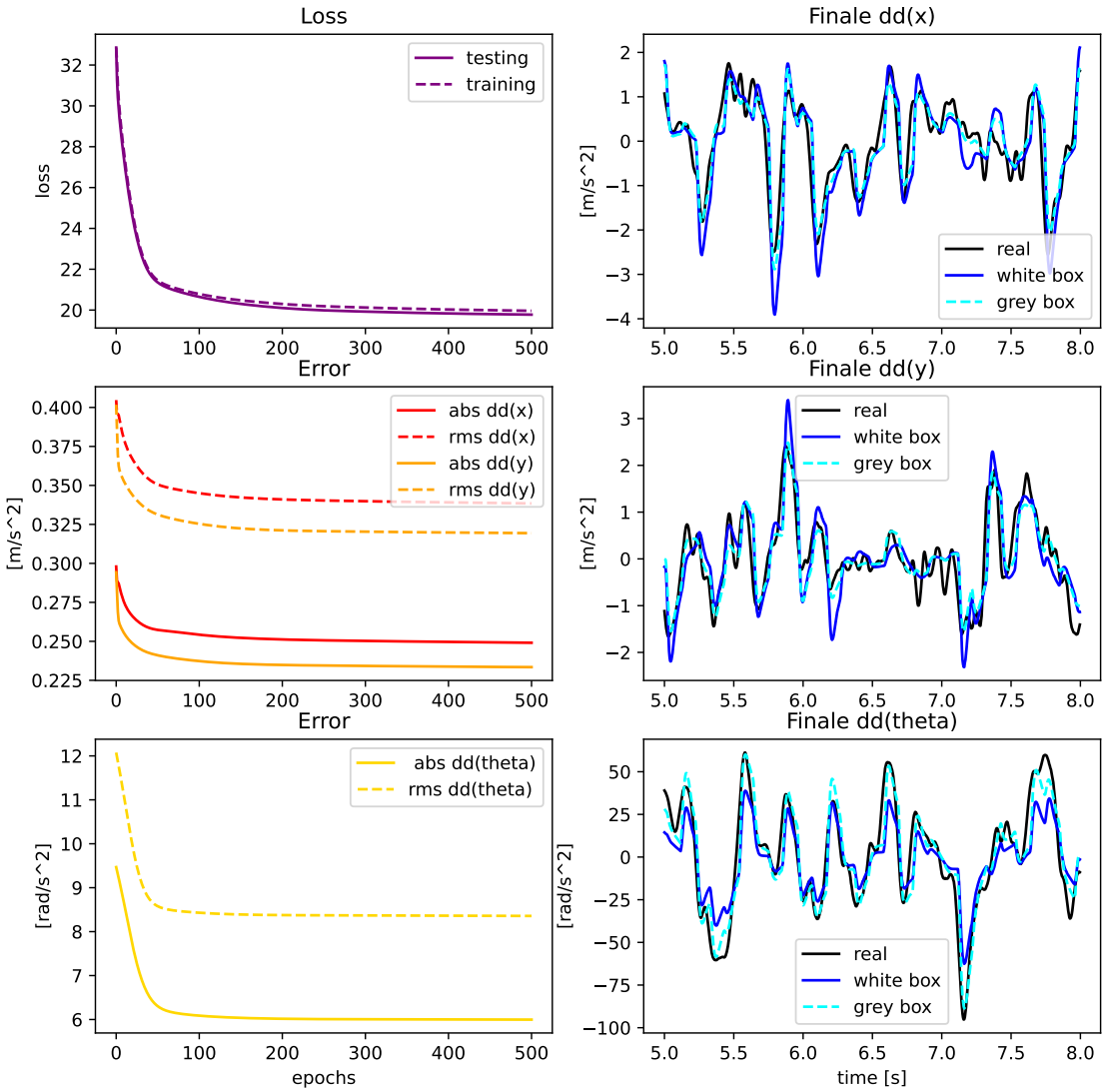


Figure 4.6: Training process of the grey box model and comparison with the white box model (error is evaluated on test data).

4.4.3 Black box model

The black box model does the entire mapping from the control input to the acceleration in the world frame using a neural network. Then, the same linear transformation as for the white box model is applied to get the system dynamics using matrices A and B (4.2). The control input to acceleration mapping is done separably for $\ddot{x}$, $\ddot{y}$ and $\ddot{\theta}$ using three fully connected neural networks. The sub-models for the three dimensions have the same architecture:

- 5 hidden layers of size 64
- Activation function: tanh

Weight decay is added to prevent overfitting. The following parameters are used for the training process.

- Optimizer: Adam
- Initial learning rate: 10^{-4}
- Weight decay: 10^{-6}
- Nb. samples: 159'041
- Batch size: 8192
- Nb. epochs: 1000
- Test share: 0.1
- Loss: mean squared error

The results are shown in figure 4.7. Before comparing the results, we would like to make two general remarks: First, the model is not entirely a black box because it uses some information about the system to facilitate the approximation: The mapping from acceleration in the world frame to state dynamics is done after the pass forward through the neural network. Also, the sub-models of the three dimensions $\ddot{x}$, $\ddot{y}$ and $\ddot{\theta}$ are separated to prevent any interference which would make the training process more difficult and which would not be possible when implementing the stabilizable dynamics as discussed in chapter 2. Second, the black box model is not very optimized: a parameter search should be performed and different architectures tested to improve its performance further.

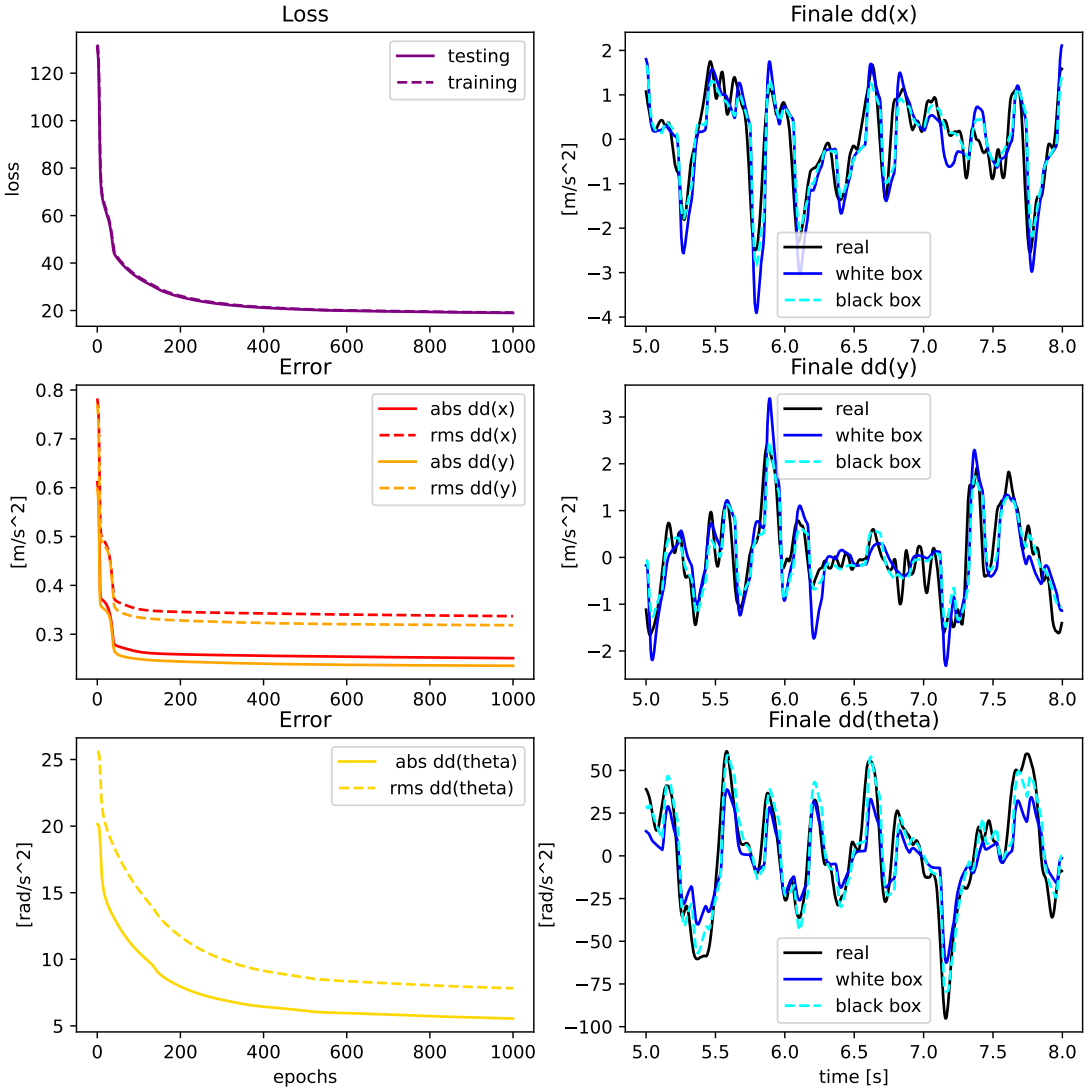


Figure 4.7: Training process of the black box model and comparison with the white box model (error is evaluated on test data).

4.5 Model Comparison

4.5.1 Results

In this section, we compare the error of the different models and show how much each step contributed to the error reduction. The final results are shown in table 4.1:

- The first row shows the error before adding the idle thrust as well as before approximating the motor transition. This is the initial error

before starting this project. Note that the evaluation of the error without idle thrust is done on the *holohover_20221130* data-set compared to all other evaluations which are done using the *holohover_20221208* data-set.

- In the second row, there is the white box model error without approximating the thrust using the first-order model.
- The third row shows the white box model error, except that the signal-to-thrust mapping is done using the coefficients found during previous work [3] and not using the ones determined during this semester project (see chapter 3.6).
- The fourth row is the white box model error. This means the newly determined signal-to-thrust mapping is used, but no parameters are learned.
- The fifth row shows the grey box model error after learning the parameters as described in chapter 4.4.2.
- The last row shows the black box model error.

	Mean abs. error			RMS error		
Model	$\ddot{x}$	$\ddot{y}$	$\ddot{\theta}$	$\ddot{x}$	$\ddot{y}$	$\ddot{\theta}$
No idle thrust	1.112	1.106	20.21	1.543	1.526	27.49
No motor transition	0.6043	0.5793	11.15	0.9530	0.9229	15.48
Old sig2thrust curve	0.4211	0.4096	16.12	0.5431	0.5327	20.57
White box	0.2979	0.2943	9.468	0.4047	0.4017	12.08
Grey box	0.2491	0.2335	5.997	0.3385	0.3194	8.357
Black box	0.2510	0.2355	5.550	0.3370	0.3184	7.823

Table 4.1: Mean absolute and RMS error

4.5.2 Discussion

The largest improvement is achieved by adding the idle thrust. This makes sense because without idle thrust, the motors are not reactive and have a significant delay as discussed in chapter 3.4. Modelling the transition zone of the motors divides the error for $\ddot{x}$ and $\ddot{y}$ approximately by two (compare rows 2 and 4 in table 4.1). This means that it is definitely worth it to add the additional complexity of the augmented state (see chapter 3.4.4). The influence of the transition approximation on $\ddot{\theta}$ is not as pronounced as for $\ddot{x}$ and $\ddot{y}$. Comparing the results of the new signal-to-thrust coefficients and the old ones [3] shows that it was a good idea to re-determine this mapping: The

error is improved by a factor of about 1.5 (compare rows 3 and 4 in table 4.1). For $\ddot{\theta}$ the difference is even higher. Notice that the RMS error for $\ddot{x}$ and $\ddot{y}$ changes less than the abs error. This means that the improvement is less important for extreme error values. The white box model has good results for $\ddot{x}$ and $\ddot{y}$ with a mean absolute error of less than $0.3\frac{m}{s^2}$. However, the mean absolute error for $\ddot{\theta}$ is equal to $9.5\frac{rad}{s^2}$ and seems to be relatively high. Here, the grey box model helps to improve the error of $\ddot{\theta}$ by approximately a factor of 1.5. The black box model achieves a similar performance to the grey box model: The absolute error in the $\ddot{x}$ and $\ddot{y}$ directions is slightly worse but for the rotational acceleration $0.44\frac{rad}{s^2}$ better. This shows that the physical relationships used for the grey box model are accurate. Otherwise, we would expect the black box model to outperform the grey box model by adapting better to the underlying physics.

4.6 Parameter Comparison

4.6.1 Results

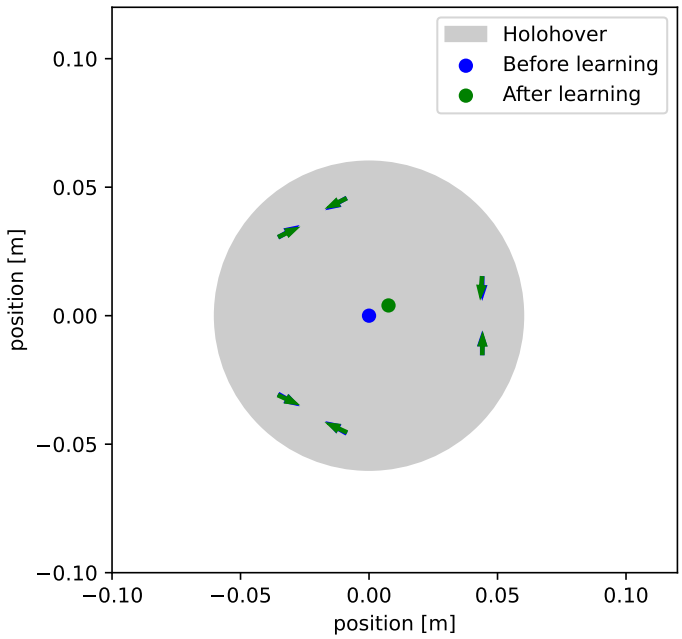


Figure 4.8: Dots: centre of masses before (in blue) and after (in green) learning; arrows: point in the directions of the motor thrust before (in blue) and after (in green) learning.

The inertia changes drastically during the learning process: The initial inertia from the white box model is $3.599 * 10^{-4} kgm^2$ and the learned inertia is $1.928 * 10^{-4} kgm^2$. Further, the learned offset of the centre of mass is $8.563mm$ and it is well visible in figure 4.8. The position of the motors and the direction of their thrust stays almost the same. Figure 4.9 shows the signal-to-thrust mapping before and after the training. In general, the curves are less steep and no longer linear for signals above 0.5. Also, motors 1 and 5 reach their maximum at 0.85 and decrease afterwards.

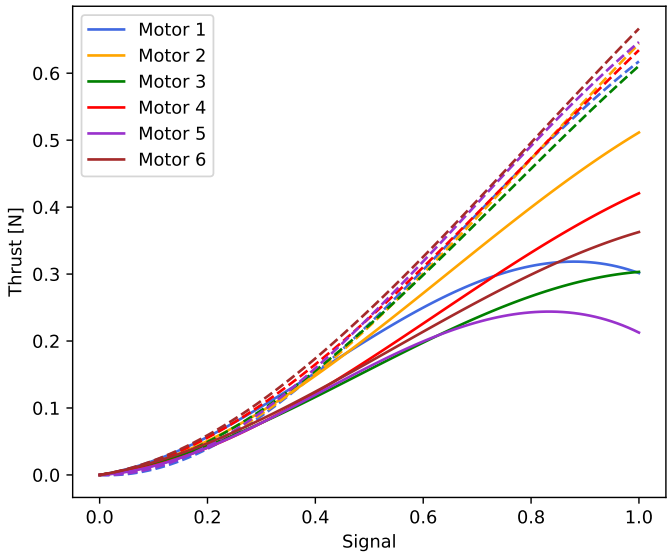


Figure 4.9: Signal-to-thrust curves before (dashed lines) and after (solid lines) training

4.6.2 Discussion

Calculating the inertia by assuming that the Holohover is a homogeneous disc is a rough approximation. Therefore, it makes sense that the inertia varies a lot. If the inertia decreases, this means that $\ddot{\theta}$ increases. This effect is clearly visible in figure 4.6 when comparing $\ddot{\theta}$ from the white and the grey box model. The offset of the centre of mass towards the right top (see dots in figure 4.8) is reasonable because the battery and a lot of the electronics/cables are placed in this direction. The position of the motors and the direction of their thrust is well defined by the design and therefore, it makes sense that it stays almost the same. It is possible that the signal-to-thrust curves are in practice a little bit lower than when doing the force sensor measurements (see figure 4.9). However, for signals above 0.5, the difference becomes really large and there seems to be no reason why the

curves decrease again at the end for motors 1 and 5. We believe that this is due to the distribution of the training data that is shown in figure 4.10. Note that the top left plot shows the distribution of a motor pair and not a single motor because one of the two motors outputs by design the idle thrust. The control input follows a *Poisson* like distribution and the accelerations a *Gaussian* one. Hence, there exist much more samples with low control inputs and accelerations. Therefore, the training adapts the signal-to-thrust curves for low signals, but not very well for higher ones. One could try to sample the data to get a more balanced data set. However, the control input and the acceleration are not independent which makes the sampling more difficult. Also, this would reduce the data set significantly and more data would need to be collected. We suggest using the learned signal-to-thrust coefficients only if the Holohover is controlled with small control inputs. Otherwise, it is safer to take the coefficients from the white box model that perform a little bit weaker for low signals but that seem to be more reasonable for higher ones.

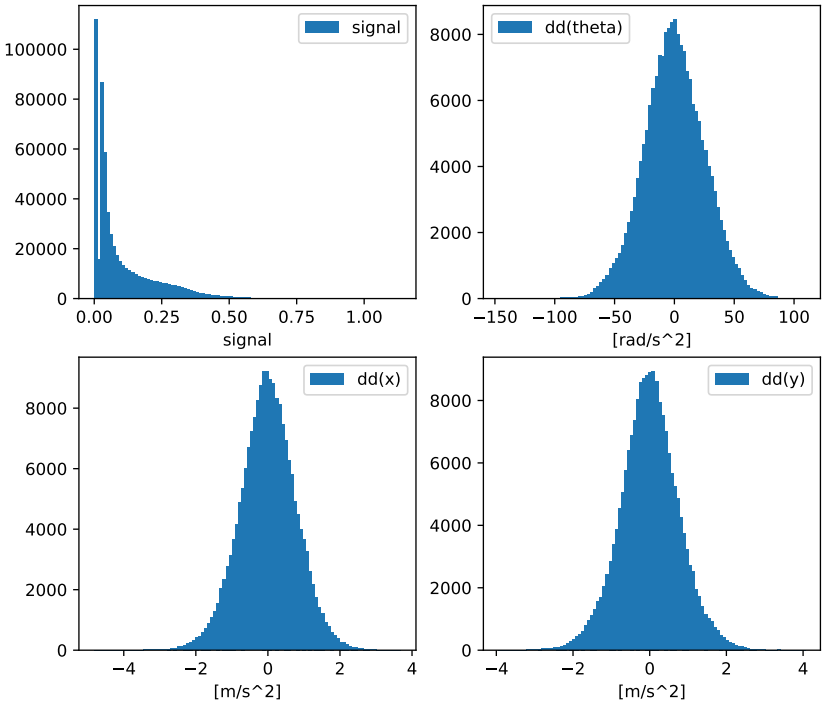


Figure 4.10: Histogram of learning data (top left shows the distribution of a motor pair and not a single motor)

Chapter 5 Conclusion

The main focus of this project is to improve the model of the system dynamics from the Holohover vehicle. First, the concept of learning stabilizable system dynamics is explored in chapter 2. The theory is showcased by the simulation of the *Damped Harmonic Oscillator* and the *Continuous Stirred Tank Reactor*. Even though the stabilizable system dynamics are not learned for the Holohover due to time constraints, everything is ready for doing so: Enough data is collected for the learning process and the training implementation is complete, except that the formulation must be adapted to address the issues discussed in chapter 2.5. Second, the white box model is improved by three measurements:

- Reduce motor start-up delay by adding an idle thrust and by keeping the rotors always moving. This step is extremely important because the Holohover becomes much more reactive and the mean absolute error is cut approximately in half.
- Approximating the motor transition zone by a first-order model reduces the mean absolute error of the acceleration in the x and y directions by a factor of about two. Also, the motors are expected to be a large part of their operation time in the transition zone because the vehicle is lightweight and *Air Hockey* is a really dynamic game.
- The signal-to-thrust mapping is repeated using the new force sensor from *Bota Systems* [11] for every motor individually. Repeating this experiment with the dedicated measurement tool is effective and divides the mean absolute error by a factor of 1.5.

Third, the motor positions and orientations, the centre of mass, the inertia and the signal-to-thrust coefficients are learned by the grey box model. The motor position and orientations remain almost unchanged. For the centre of mass and the inertia, the difference between the initial and the learned parameters is more important. Especially, the reduction of inertia helps to reduce the mean absolute error in angular acceleration which is more than 1.5 times smaller for the grey box model compared to the white box model. The learned coefficients of the signal-to-thrust curves should be taken with a grain of salt because as discussed in chapter 4.6.2, the training data set is not

well balanced with respect to the control input. Therefore, the curve adapts well to low signals and accepts larger errors for higher ones. When using a black box approach as discussed in chapter 4.4.3, the results are comparable to the one from the grey box model. Therefore, it is not worth implementing it for the real-time system. However, it is a good way to compare the grey box model and it shows that it has a relatively good performance.

One could argue that even when moving fast, the occurrence of high thrusts is much less likely than lower ones and therefore, it does not matter that low signals are over-represented in the training and evaluation data set. Nevertheless, high-thrust manoeuvres are probably the most difficult ones when controlling the Holohover. Therefore, we believe that it is important to collect a more balanced data set as discussed in chapter 4.6.2. One other crucial point is to tune the Kalman filter. First of all, this would improve the performance of all controllers using a state estimation. And second of all, it could provide a better ground truth for the acceleration. Deriving the Optitrack data twice like in this project is not the best solution. We chose the *Savitzky Golay* derivation from *pysindy* [13], [14] because the resulting acceleration looks reasonable. However, there is no proof that this is in fact the case. We considered trying different differentiation methods and comparing the model performance using this data. However, it would be very difficult to say if the derived acceleration is really closer to the real one or if we are just overfitting the model.

Nonetheless, this semester project helped to reduce the mean absolute error by a factor of about 4.5 for the acceleration in the x and y directions and by a factor of 3.3 for the angular acceleration. This is a satisfying result. We believe that it may be very useful in the future when implementing more advanced controllers for example using *Model predictive control*.

Chapter 6 Annex

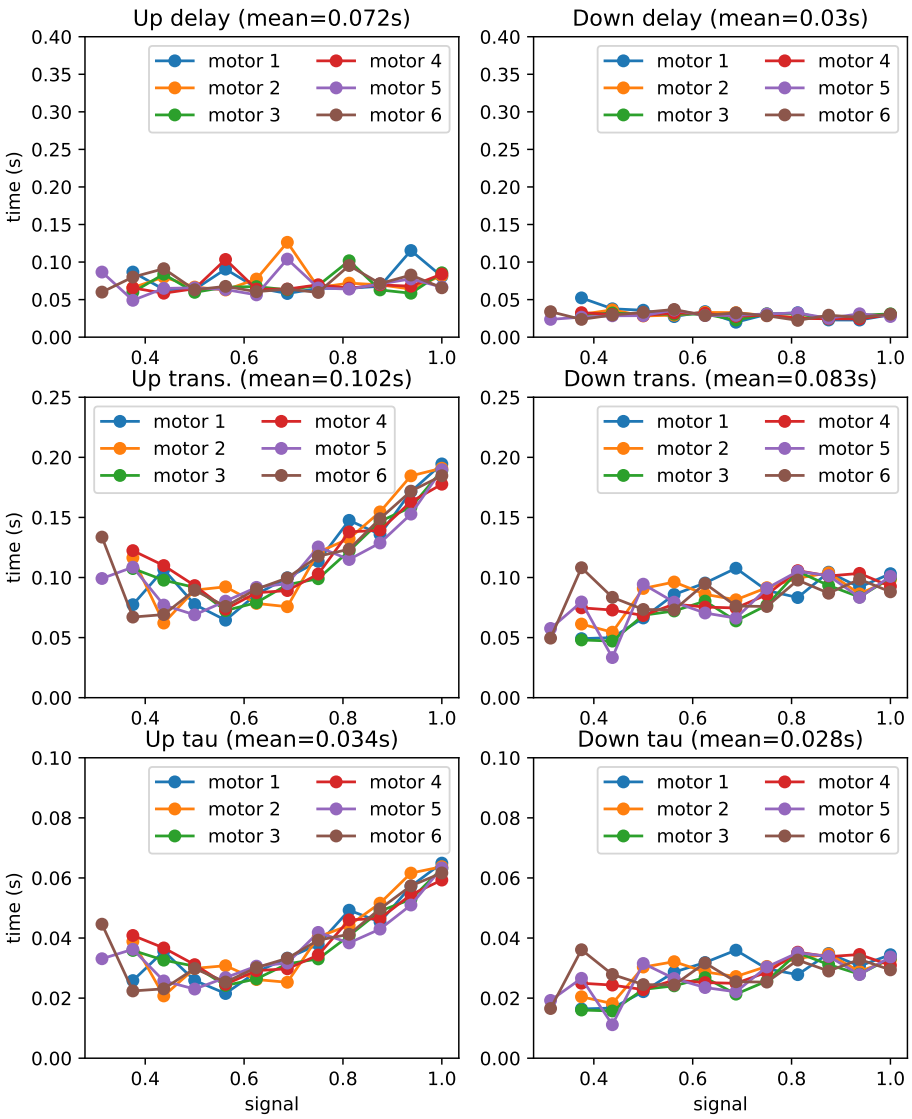


Figure 6.1: Shows delay (δ), transition time and time constant τ for up and down thrusts in function of signal. Note that this is the same plot than figure 3.6, but calculated in signal space and not using the thrust.

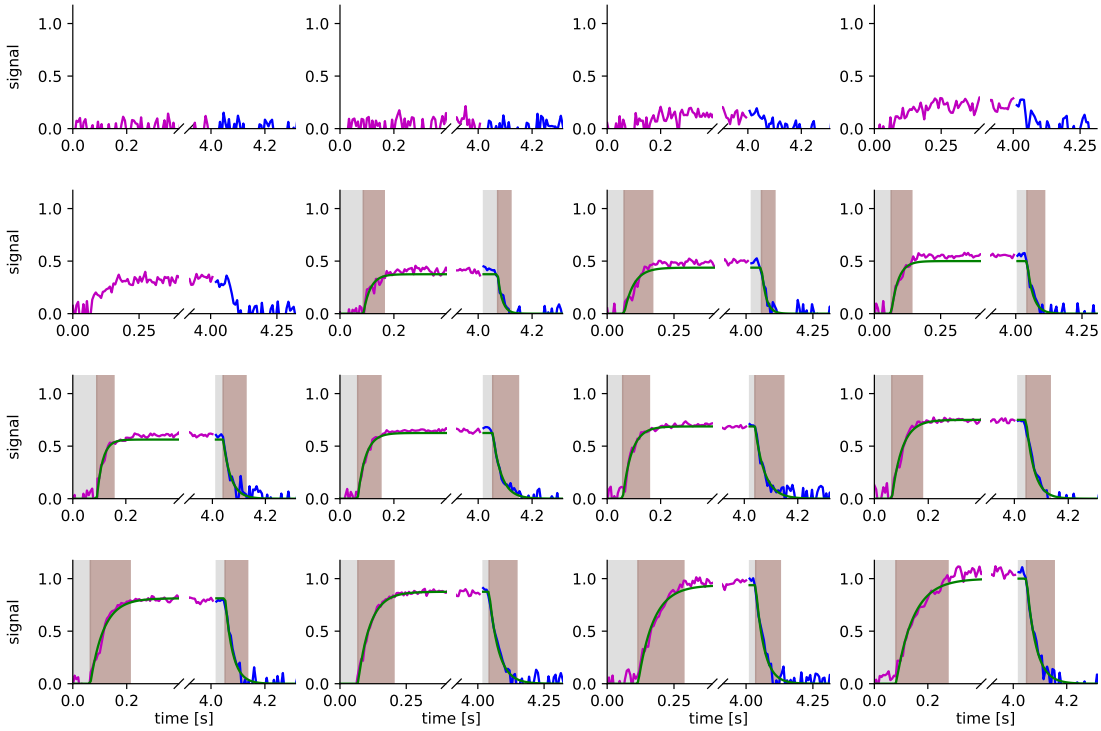


Figure 6.2: Motor 1 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone). This is the same figure than 3.5, but the thrust is converted to a signal.

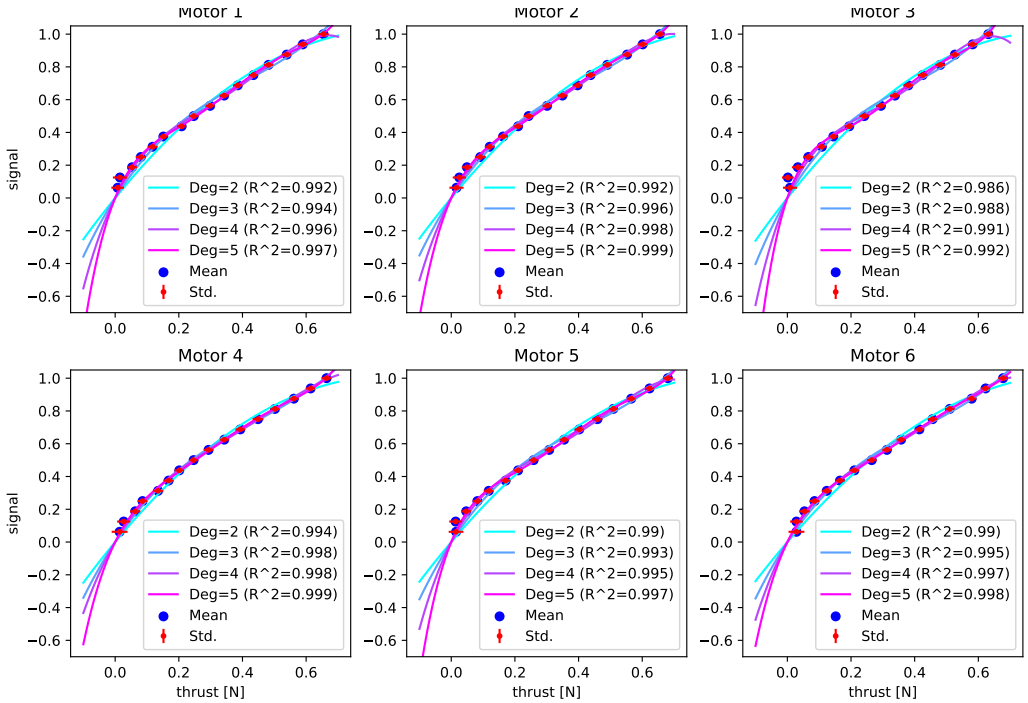


Figure 6.3: Trust to signal mapping for all motors (cyan blue to pink: polynomial fits of various degrees, blue point: mean of measurement, red bar: standard deviation of measurement

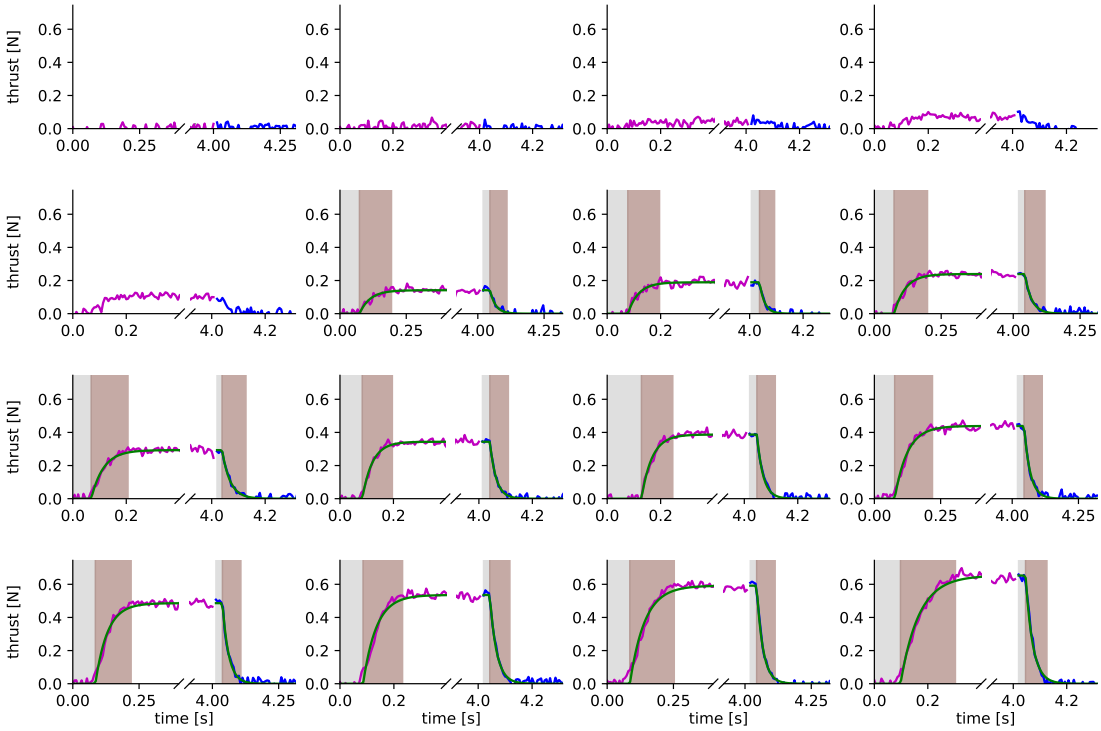


Figure 6.4: Motor 2 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

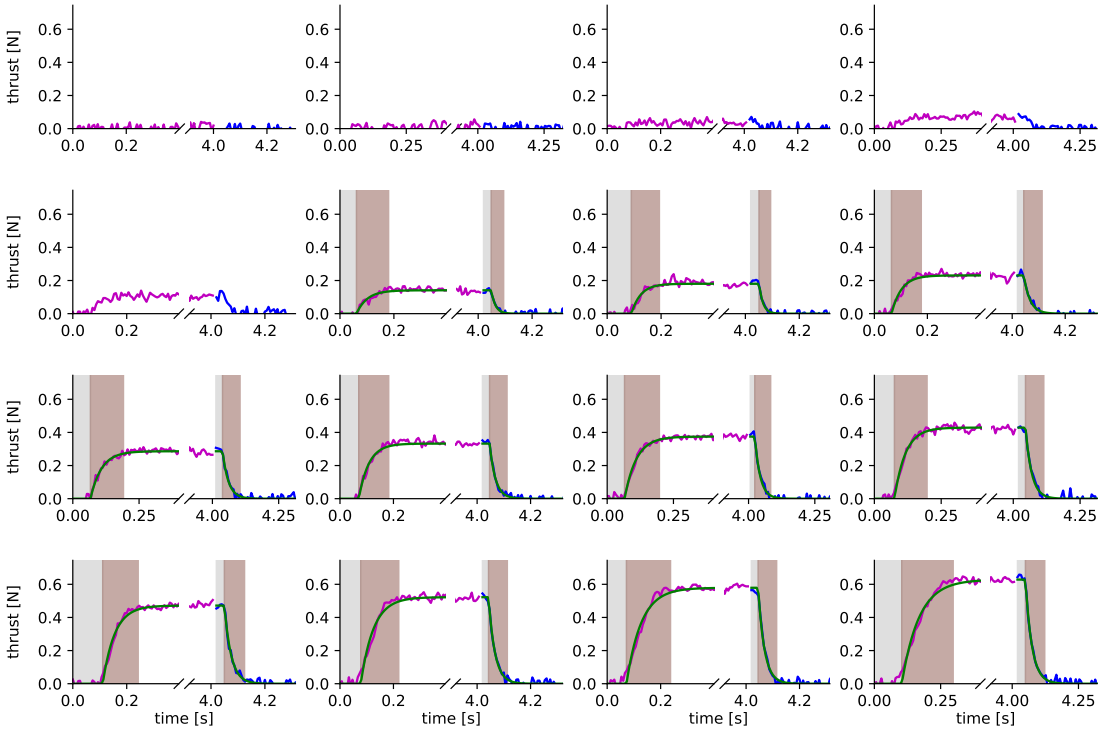


Figure 6.5: Motor 3 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

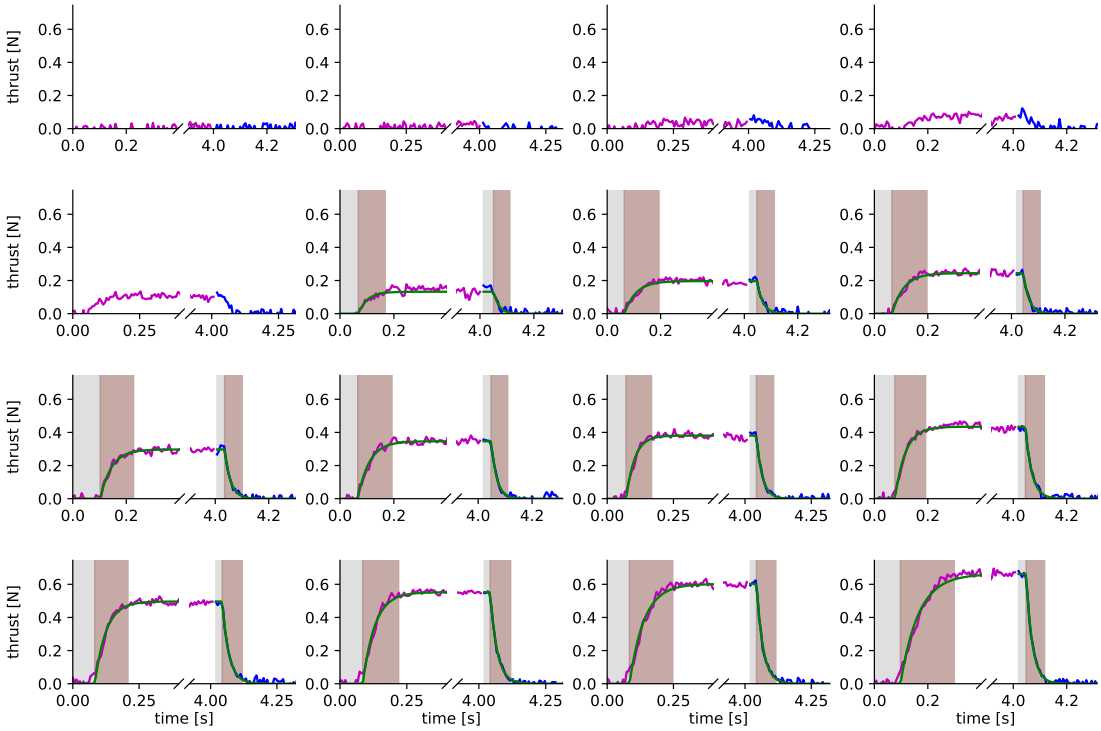


Figure 6.6: Motor 4 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

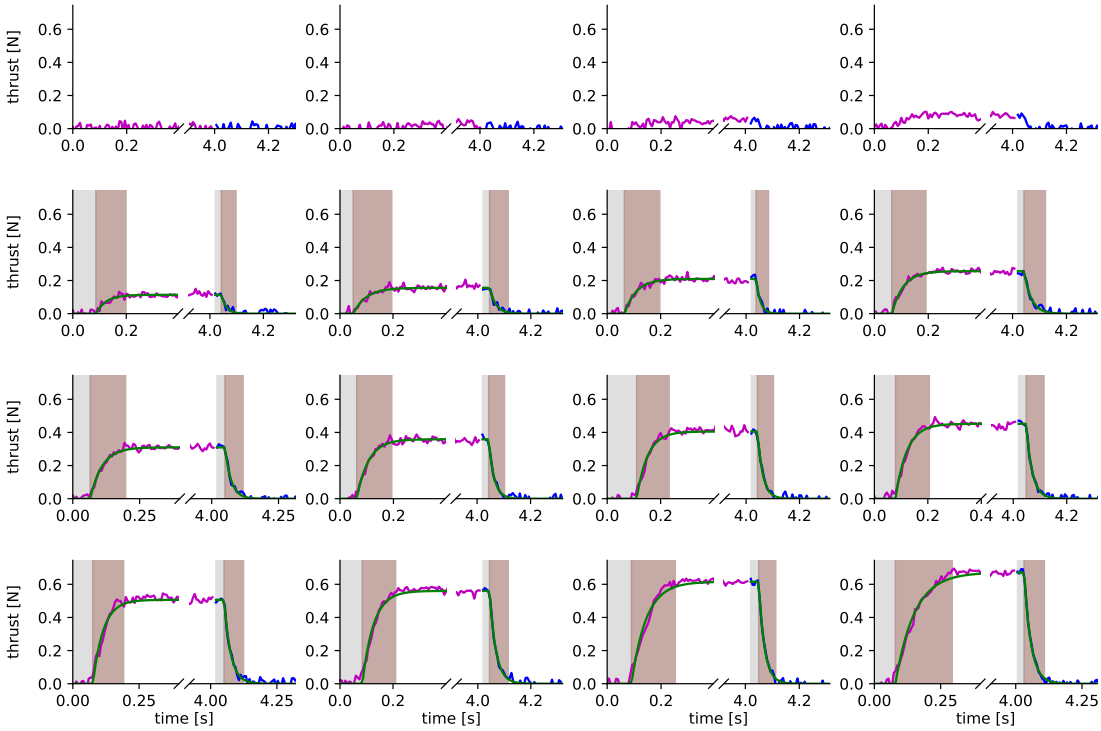


Figure 6.7: Motor 5 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

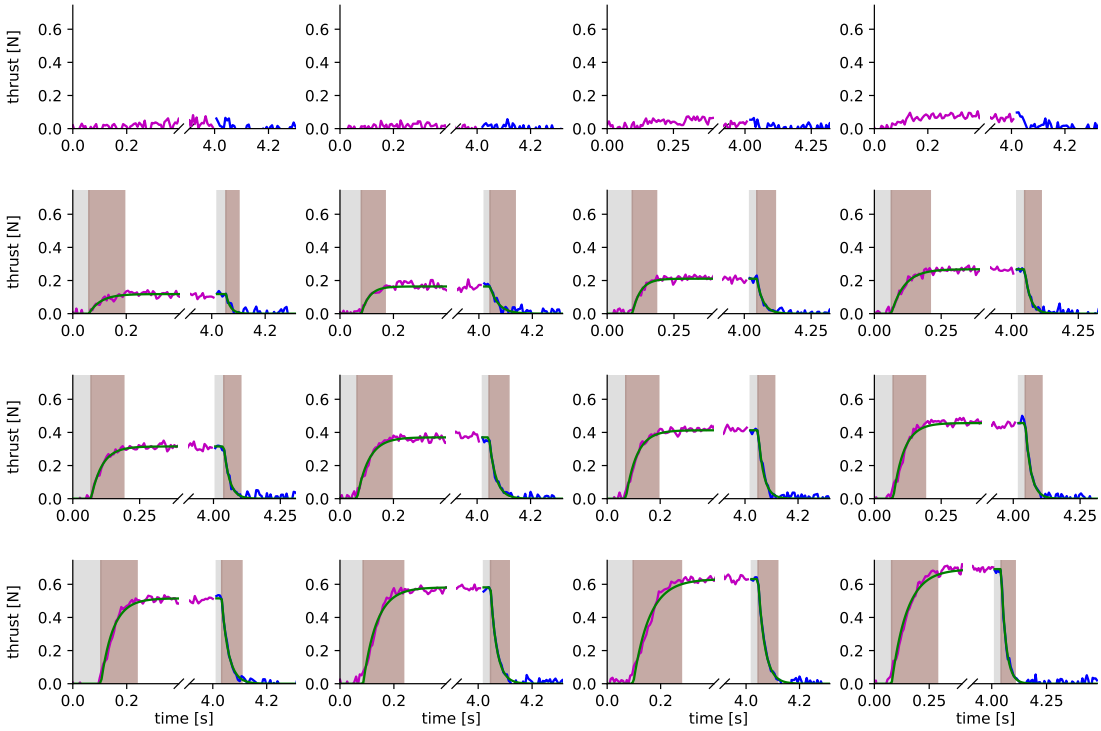


Figure 6.8: Motor 6 thrust measurements of 16 increasing signals: 0.0625, 0.125, ... 0.9375, 1.0 (violet: signal on (4s), blue: signal off (1s), green: signal approximation, grey: dead zone, brown: transition zone)

Bibliography

[1] M. Pfister, "Identification and control of a miniature hovercraft. semester project," 2019.

[2] A. Dutta, "Control and identification of hovercraft. semester project," 2020.

[3] K. Samaha, "Design and control of a miniature hovercraft. semester project," 2022.

[4] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[5] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamic systems," *CoRR*, vol. abs/1808.00924, 2018.

[6] A. J. Taylor, V. D. Dorobantu, H. M. Le, Y. Yue, and A. D. Ames, "Episodic learning with control lyapunov functions for uncertain robotic systems," *CoRR*, vol. abs/1903.01577, 2019.

[7] G. Manek and J. Z. Kolter, "Learning stable deep dynamics models," *CoRR*, vol. abs/2001.06116, 2020.

[8] C. J. Roland Schwan, "Learning stabilizable models," 2022. not published yet.

[9] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," *CoRR*, vol. abs/1609.07152, 2016.

[10] E. T. Maddalena, P. Scharnhorst, Y. Jiang, and C. N. Jones, "Kpc: Learning-based model predictive control with deterministic guarantees," 2020.

[11] B. Systems, "Force-torque-sensor of bota systems." https://www.botasys.com/force-torque-sensors/medusa. Accessed on 2021-12-27.

[12] Scipy, "scipy library." https://docs.scipy.org/doc/scipy/reference/.

[13] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton, "Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data," *Journal of Open Source Software*, vol. 5, no. 49, p. 2104, 2020.

[14] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, A. J. Goldschmidt, J. Callaham, C. B. Delahunt, Z. G. Nicolaou, K. Champion, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, "Pysindy: A comprehensive python package for robust sparse system identification," *Journal of Open Source Software*, vol. 7, no. 69, p. 3994, 2022.