Mauro Pfister

Identification and control of a miniature hovercraft

Semester Project

Automatic Control Laboratory
École polytechnique fédérale de Lausanne (EPFL)

Supervision

Professor: Prof. Colin Neil Jones
Assistant: Yingzhao Lian

July 2019

Acknowledgment

During this semester project I received help and support from several people who I would like to thank at this point.

First of all I would like to express my gratitude towards my assistant, Yingzhao Lian, who was always there to help and who kept me motivated throughout the semester.

I would also like to thank Peter Listov for the many reservations of the camera room he has made for me.

Further, I owe a debt of gratitude to Fabian Schilling from the LIS who patiently answered all my questions and gave a lot of helpful advice concerning the setup of the ROS environment in the camera room.

Finally I would like to thank Professor Colin Jones for supervising the project.

Contents

Chapter 1

Introduction

1.1 Goal of project

The aim of this semester project is to identify the physical parameters of a miniature hovercraft and to implement a real-time feedback controller that steers the hovercraft along a given trajectory. The necessary feedback is provided by the motion capture system of a camera room at EPFL. All components of hovercraft-camera-room system are implemented in Python and use the framework of the Robot Operating System (ROS) which provides a convenient way of parallelizing different processes.

Since this semester project is the first to work on the miniature hovercraft, no pre-existing system was available. Therefore, two overlapping tasks were developed simultaneously. One of them consisted of the modelling, simulation, identification and control of the hovercraft in Matlab. The other focussed on the setup of the hovercraft and ROS environment in the camera room. As soon as a working controller was found in simulation it could be implemented on the real system using the previously developed ROS framework.

1.2 Hardware overview

1.2.1 Tiny Whoover hovercraft

The basis of this project is the Tiny Whoover miniature hovercraft [1] which uses almost all the components of the Tiny Whoop [2], a small first-person-view quadcopter for indoor use. The four motors are controlled by an AlienWhoop flight controller running BetaFlight 3.4 which is powered by a 650 mAh LiHV battery. An Adafruit HUZZAH32 - ESP32 Feather Board is used to communicate with a computer over WiFi using the UDP protocol. In order to keep the battery in place, a small 3D-printed mount was installed on the hovercraft frame. Additionally, four markers for the motion capture system are glued to the top of the Tiny Whoover as shown in Figure 1.1.

1.2.2 Drone room

To track the position of the hovercraft the OptiTrack motion capture system in the drone room of the MED building at EPFL is used. The OptiTrack software Motive, running on a Windows computer, allows to adjust the tracking settings and then streams the incoming location information into the network. The processing of the tracking information as well as the control of the hovercraft take place in ROS which runs on a separate Mac mini (running Ubuntu 12). The latter

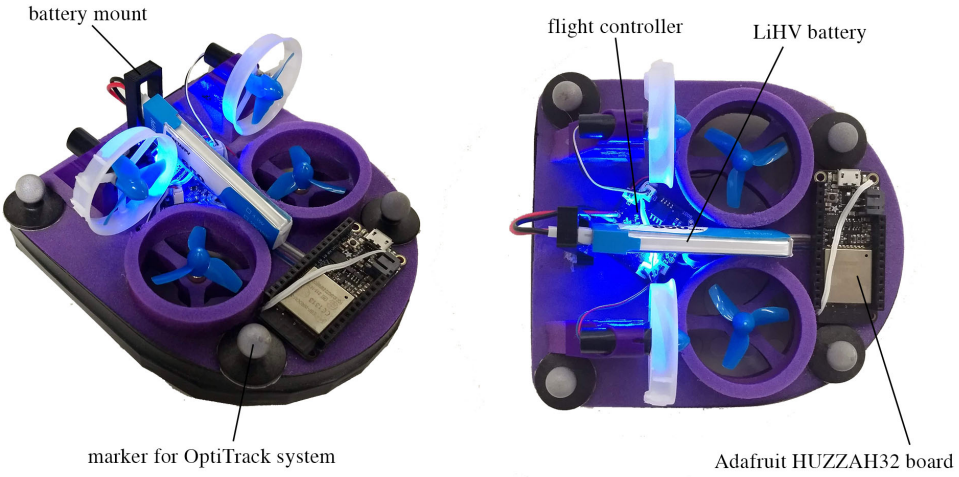


Figure 1.1: Tiny Whoover with labeled components.

communicates to the hovercraft via WiFi as indicated before. An overview of the whole system is shown in Figure 1.2.

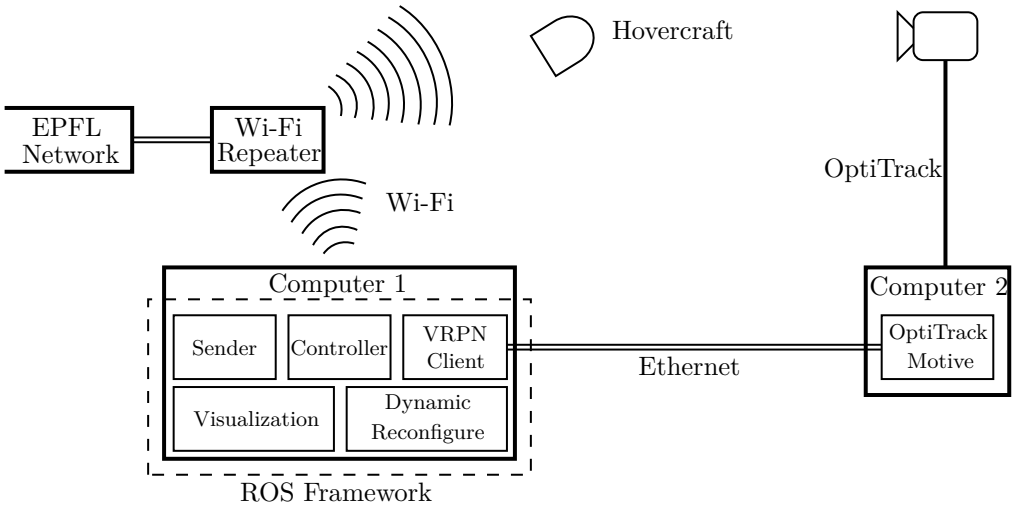


Figure 1.2: Diagram of the drone room setup. With permission from [3].

Chapter 2

System model and simulation

The goal of this chapter is to introduce the mathematical notation used throughout the report and to describe a physical model of the hovercraft. Additionally, the numerical implementation of said model in Matlab is briefly discussed.

2.1 Coordinate convention

The reference literature for ship modelling [4] usually uses two coordinate frames to describe the motion of a vessel; A world-fixed frame (X, Y, Z) and a body-fixed frame (X_b, Y_b, Z_b) whose origin is located at the center of gravity of the ship. Conventionally, both z-axes are chosen to point downwards and the x-axis of the body frame is aligned with the ships forward direction. A schematic of both reference frames is shown in Figure 2.1. Since the z-axis points downward, the yaw angle ψ is positive in clock-wise direction when viewed from above. The rotation matrix from the body frame to the world frame is called $\mathbf{R}_z(\psi)$ and defined as:

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

2.2 Surface ship model

As proposed in [4], the horizontal motion of a surface ship, of which the hovercraft is a special case, can be described using the following vectors:

$$\begin{aligned} \boldsymbol{\eta} &= \begin{bmatrix} x & y & \psi \end{bmatrix}^T, \\ \boldsymbol{\nu} &= \begin{bmatrix} u & v & r \end{bmatrix}^T, \\ \boldsymbol{u} &= \begin{bmatrix} u_L & u_R \end{bmatrix}^T, \end{aligned} \tag{2.2}$$

where $\boldsymbol{\eta}$ denotes the position and orientation of the hovercraft with respect to the world-fixed reference frame, $\boldsymbol{\nu}$ denotes the linear and angular velocities in the body-fixed reference frame and $\boldsymbol{u}$ is the input to the system. The surge and sway velocities are denoted by u and v respectively and the yaw rate is called r.

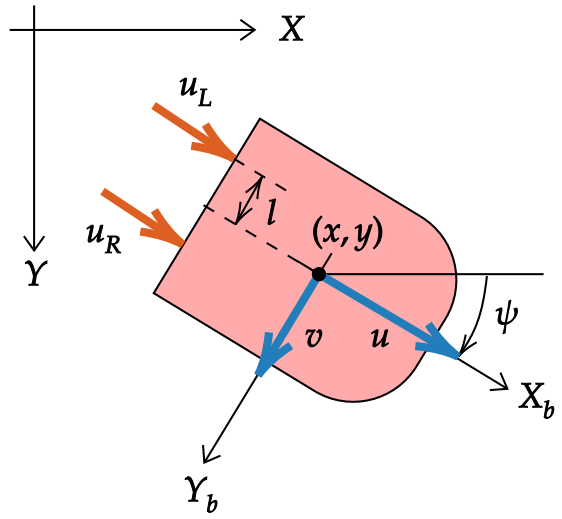


Figure 2.1: World-fixed and body-fixed reference frames.

Note that in [4] the input of the system is defined as the forces and moments generated by the thruster onto the ship. For convenience reasons we use a different notation where u_L and u_R denote a normalized (non-dimensional) force in the range $[0, 1]$ which is generated by the left and right thruster respectively. The actual force generated by a thruster can be calculated using a (thruster speed dependent) coefficient $k(\boldsymbol{u})$ which can only be found by experiments (see Chapter 3).

Using the notation defined above, the kinematics of the hovercraft can then be expressed as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{\nu}) = \mathbf{R}_z(\psi)\boldsymbol{\nu}. \tag{2.3}$$

The dynamics are given by the following non-linear differential equation:

$$\dot{\boldsymbol{\nu}} = \boldsymbol{f}_{\boldsymbol{\nu}}(\boldsymbol{\nu}, \boldsymbol{u}) = \mathbf{M}^{-1}(\mathbf{B}(\boldsymbol{u})\boldsymbol{u} - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}\boldsymbol{\nu}) \tag{2.4}$$

with

$$\mathbf{M} = \operatorname{diag}(m, m, I_z), \quad \mathbf{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix}$$

$$\mathbf{D} = \operatorname{diag}(X_u, Y_v, N_r), \quad \mathbf{B}(\boldsymbol{u}) = k(\boldsymbol{u}) \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ l & -l \end{bmatrix}$$

where

$\mathbf{M}$: is the mass matrix with mass m and moment of inertia I_z.
$\mathbf{D}$: is the damping matrix with surge, sway and yaw damping coefficients X_u, Y_v and N_r.
$\mathbf{C}(\boldsymbol{\nu})$: is the coriolis matrix.
$\mathbf{B}(\boldsymbol{u})$: is the input matrix with conversion coefficient $k(\boldsymbol{u})$ and lever arm of the thrusters l.

2.3 Simulation

In order to analyze the behaviour of the hovercraft dynamics, equations (2.3) and (2.4) are numerically integrated in time using the fourth-order Runge-Kutta (RK4) method. We will use the following notation for a simulation over N time steps throughout the report:

$$\begin{cases} \boldsymbol{\eta}_{sim}(t_{k+1}) & = \text{RK4}\left(\boldsymbol{f}_{\boldsymbol{\eta}}, \boldsymbol{\nu}_{sim}(t_k),\right) \\ \boldsymbol{\nu}_{sim}(t_{k+1}) & = \text{RK4}\left(\boldsymbol{f}_{\boldsymbol{\nu}}, \boldsymbol{\nu}_{sim}(t_k), \boldsymbol{u}(t_k)\right) \\ t_{k+1} - t_k & = h \end{cases} \qquad \forall k = 1, \ldots, N-1 \tag{2.5}$$

To select an appropriate step-size h, the following input is applied to the system

$$\boldsymbol{u} = \begin{bmatrix} \sin(2t) \\ \sin(2t+2) \end{bmatrix} \tag{2.6}$$

and the RK4 integration is run with different step-sizes. Figure 2.2 shows the resulting trajectories. Clearly, $h = 0.1$ is not small enough since its trajectory deviates a lot from the ones generated by $h = 0.01$, $h = 0.005$ and $h = 0.001$. As a compromise between accuracy and speed, a final step-size of $h = 0.005$ is selected for all further simulations in this report.

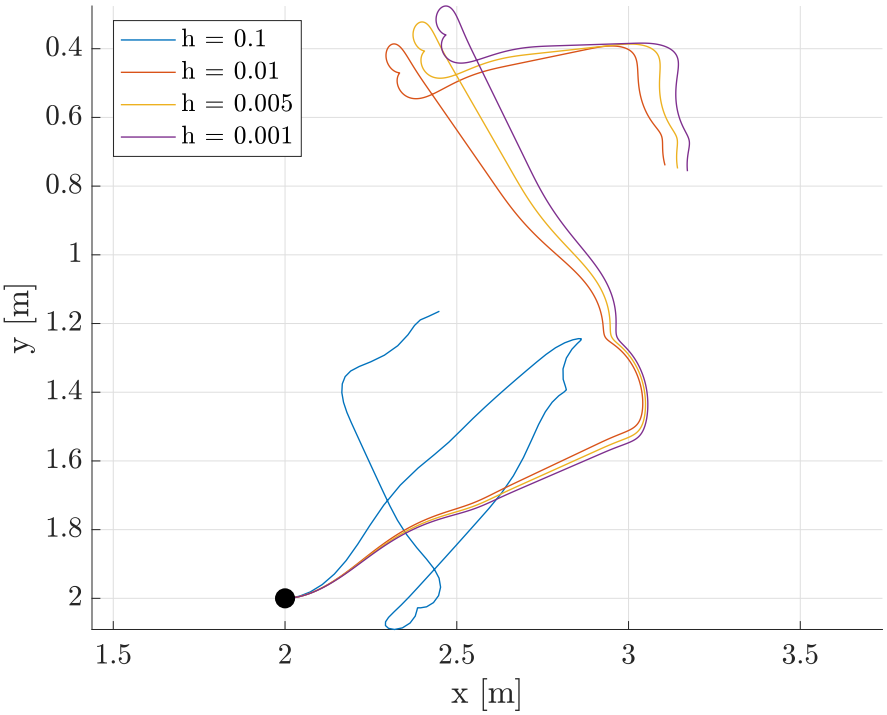


Figure 2.2: Integrated trajectories using RK4 and different step-sizes h.

Chapter 3

System identification

This chapter is dedicated to the identification of the physical parameters of the Tiny Whoover hovercraft. The knowledge of these values is crucial for the subsequent controller design.

3.1 System parameters

The differential equation (2.4) governing the velocities contains seven unknown parameters:

- m : Mass of hovercraft
- l : Length of lever arm of thrusters
- I_z : Moment of inertia around z-axis
- X_u : Surge damping coefficient
- Y_v : Sway damping coefficient
- N_r : Yaw damping coefficient
- k : Thrust coefficient

Out of those seven parameters, the mass (58.3 g) as well as the lever arm length (3.25 cm) can be easily measured and the thrust coefficient can be determined approximately by measurements as described hereafter. The remaining four parameters are difficult to measure and thus a parameter estimation algorithm is employed.

3.2 Thrust coefficient measurement

The thrust coefficient k determines how much force is generated by one thruster for a given input to that thruster. A priori this relationship is not expected to be linear and thus k is a function of the input u_t, i.e.

$$k = k(u_t) \tag{3.1}$$

Using the measurement technique described in [3], the thruster force is determined for an input u_t varying between 0 and 1. The measurements in Figure 3.1 clearly show the same linear relationship between input and thrust coefficient for the right and left side. Thus the use of a simple constant k instead of a function is justified. Linear regression on the measurements gives a coefficient of $k_L = 0.103\,\mathrm{N}$ and $k_R = 0.100\,\mathrm{N}$ for the left and right thruster respectively. Note that one reason

for the linear behaviour might be that the AlienWhoop flight controller already accounts for the non-linearity of the motor and the propeller.

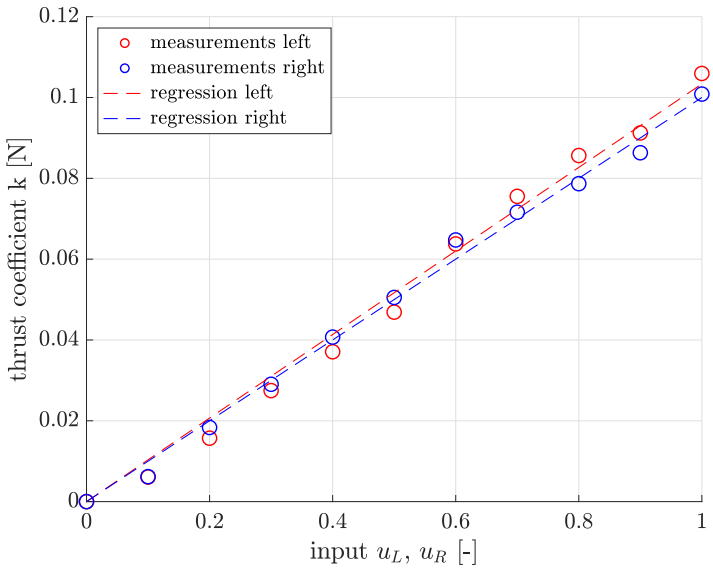


Figure 3.1: Thrust coefficient measurements and regression for left and right thruster.

3.3 Parameter estimation algorithm

As mentioned before, the three damping coefficients as well as the moment of inertia are hard to measure and thus we try to estimate them using data collected on the real system. It was decided to include the thrust coefficient into the parameter identification as well in order to confirm or correct the results obtained by measurement. Hence, the vector of parameters to estimate is given by:

$$\boldsymbol{\theta} = \begin{bmatrix} I_z & X_u & Y_v & N_r & k \end{bmatrix} \tag{3.2}$$

The idea of the parameter estimation algorithm used here is to minimize the absolute value of the difference between a simulated velocity profile and the actual recorded velocity profile for a given input to the system. The reason for only taking the velocities into account is based on the observation that the hovercraft trajectory behaves very differently when applying the same input several times. This can partly be explained by the uneven and rough floor of the drone room. Since the hovercraft has very little friction with the ground, a slight perturbation of its rotation can cause large deviations in the $x - y$ trajectory even if the velocity profile is the same.

Note that from now on the mass m and the thruster lever length l are fixed at the values given in Section 3.1.

3.3.1 Mathematical formulation

Given a set of identification data recorded with a given sampling time h_{id}

$$\begin{cases} \boldsymbol{\nu}_{meas}(t_k) \\ \boldsymbol{u}(t_k) \end{cases} \qquad \forall k = 1, \ldots, N \tag{3.3}$$

the identification procedure can formally be described as the following optimization problem:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{k=1}^{N} \left| \boldsymbol{w} \cdot (\boldsymbol{\nu}_{sim}(t_k, \boldsymbol{\theta}) - \boldsymbol{\nu}_{meas}(t_k) \right| \tag{3.4}$$

$$\boldsymbol{\nu}_{sim}(t_{k+1}, \boldsymbol{\theta}) = \text{RK4}\left(\boldsymbol{f}_{\boldsymbol{\nu}}, \boldsymbol{\nu}_{sim}(t_k, \boldsymbol{\theta}), \boldsymbol{u}(t_k)\right) \quad \forall k = 1, \ldots, N-1$$

Here $\boldsymbol{w}$ is a weighting vector defined as

$$\boldsymbol{w} = \begin{bmatrix} \frac{1}{\sigma_u} & \frac{1}{\sigma_v} & \frac{1}{\sigma_r} \end{bmatrix}^T$$

where σ_u, σ_v and σ_r are the empirical standard deviations of the three measured velocities $\boldsymbol{\nu}_{meas}$. These weighting factors were introduced because the range of the three velocities is not the same. Without weighting this could lead to an potential overfitting of the component with the largest range.

Note that the simulated velocity profile is calculated using the same RK4 integration as described in Chapter 2. However, there might be a discrepancy between the sampling time h_{id} (usually between 0.01 and 0.1 s) of the identification data and the step-size $h = 0.005\,\text{s}$ that is required for simulation. For this reason, the recorded input $\boldsymbol{u}$ is up-sampled using "previous" interpolation before calculating the simulated velocities. Similarly, the output of the simulation has to be down-sampled to the time stamps of the measured velocities which is done by "nearest" interpolation.

Since the optimization problem defined in (3.4) is non-convex the non-linear programming solver `fmincon` from Matlab is used for minimization. However, due to the non-convexity of the problem a globally optimal solution cannot be expected if the initial parameter guess $\boldsymbol{\theta}_0$ is too far from the unknown real parameters.

A brief pseudo code of the whole parameter estimation procedure is shown in Algorithm 3.1.

Algorithm 3.1 Parameter estimation procedure

```
function ObjFun(ν, u, θ)
    u_up ← interp(u, h_id, h, previous)        ▷ up-sample u from sampling time h_id to h
    ν_up ← SimHovercraft(u_up, h)              ▷ simulate hovercraft using up-sampled input
    ν_sim ← interp(ν_up, h, h_id, nearest)     ▷ down-sample ν from sampling time h to h_id
    return CostFun(ν_sim, ν)                   ▷ cost function defined in equation (3.4)
end function

function ParamID(ν, u, θ_0)
    θ* ←Minimization(ObjFun, ν, u, θ_0)        ▷ find optimal θ that minimizes ObjFun
    return θ*
end function
```

3.3.2 Validation

Before applying the identification procedure to real data, it is important to verify that the estimated parameters correspond to the true ones. To do so, an artificial identification data set is generated in simulation with a manually chosen set of parameters $\boldsymbol{\theta}_{true}$. The input applied to the hovercraft is a PRBS signal in the range $\{0, 1\}$ with a frequency of 20 Hz, which means that it is piecewise constant over periods of at least 0.05 s. In order to account for noisy real world data, a random gaussian noise term is added to the simulated velocities as follows:

$$\boldsymbol{\nu}_{meas}(t_k) = \boldsymbol{\nu}_{sim}(t_k) + e(k) \begin{bmatrix} \frac{\epsilon}{\sigma_u} & \frac{\epsilon}{\sigma_v} & \frac{\epsilon}{\sigma_r} \end{bmatrix}^T \quad \forall k = 1, \ldots, N, \tag{3.5}$$

where

$$e(k) \sim \mathcal{N}(0, 1).$$

The amplitude of the noise is controlled by the factor ϵ and weighted inversely proportional to the standard deviation of the simulated data.

The parameter estimation algorithm is then run on the first 4 seconds of this artificial data set and ideally the estimated parameters correspond to $\boldsymbol{\theta}_{true}$. Indeed, for the noise-less data the exact parameters can be found as shown in Table 3.1. Moreover, the identification produces good results up to a noise level of around 0.3 times the standard deviation of the data. Thus, the chosen identification approach is validated and expected to produce reasonable results on measurements from the real system.

Table 3.1: Parameter estimation on artificial data with different noise levels ϵ.

data	I_z [kg m^2]	X_u [N s/m]	Y_v [N s/m]	N_r [N m s]	k [N]	max error [%]
true	10×10^{-4}	5.00×10^{-2}	5.00×10^{-2}	10×10^{-4}	0.1	0.0
data, $\epsilon = 0.0$	9.9×10^{-4}	4.97×10^{-2}	4.97×10^{-2}	9.94×10^{-4}	0.099	0.5
data, $\epsilon = 0.1$	9.6×10^{-4}	4.73×10^{-2}	5.00×10^{-2}	9.80×10^{-4}	0.097	0.5
data, $\epsilon = 0.3$	9.6×10^{-4}	5.01×10^{-2}	5.19×10^{-2}	9.96×10^{-4}	0.101	0.8
data, $\epsilon = 0.5$	12.7×10^{-4}	7.56×10^{-2}	4.83×10^{-2}	13.0×10^{-4}	0.129	51.2

Initial parameter guess: $\boldsymbol{\theta}_0 = \begin{bmatrix}0.2 & 0.2 & 0.2 & 0.2 & 0.2\end{bmatrix}$.

3.3.3 Experimental results

For the parameter identification of the real hovercraft, several PRBS input signals with different ranges and frequencies were applied and the resulting trajectories and velocities were recorded. All experiments were done while hovering with maximum lift (minimum friction). The velocities are calculated using finite differences and are filtered with a median filter to eliminate spikes.

In a first step PRBS signals with ranges {0.0, 0.4}, {0.2, 0.6}, {0.4, 0.8}, {0.6, 1.0}, {0.2, 1.0} and a frequency of 10 Hz were tested. The trajectories and velocities were recorded at 10 Hz as well. This approach did not result in good identification results, i.e. the value of the objective function (3.4) did not decrease significantly and the estimated parameters were not consistent over different runs. This might be due to the low sampling frequency and an insufficient excitation of the system.

Therefore, a second batch of data was recorded at 50 Hz with a PRBS signal varying in {0.0, 1.0} and with frequencies in {10 Hz, 15 Hz, 20 Hz, 25 Hz, 50 Hz}. For each frequency at least 5 runs were used for identification and the following observations were made:

- The identification procedure only produces consistent results over a time span of 2-4 s. This is most likely due to the rough and uneven floor which respresents an unmodelled random disturbance which the model cannot fit. Over a short time period the influence of this disturbance is small enough not to affect the fitting of the model, but over longer durations it introduces a significant error.
- Good results are only obtained when taking samples from the beginning of the data, i.e. when the hovercraft starts from a stationary pose.
- The PRBS signal with a frequency of 15 Hz results in the most consistent parameter estimates over different runs. It is possible, that the system is indeed best excited at that frequency but for a sound analysis more data would be required.

Based on these observation the first 4 seconds of the 15 Hz PRBS experiments were chosen for the final identification. Table 3.2 presents the estimated parameters over different runs. Out of 10 experiments only the results of 9 are shown. One run was removed because it gave unrealistically large parameters compared to the 9 others. It is also worth mentioning that for each experiment the hovercraft was placed at slightly different initial positions and orientations in the drone room. Since the floor of the room is not uniform this could contribute to parameter variations across different runs.

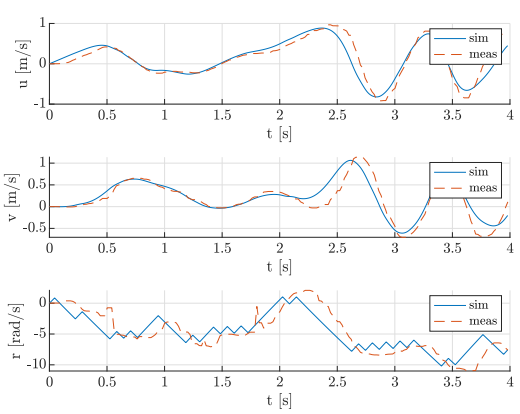
The estimates of moment of inertia I_z, yaw damping ration N_r and thrust coefficient k are all within the same order of magnitude as their average over all runs. This indicates a reliable estimation of these parameters or potentially a systematic bias. However, since the measured and simulated trajectories as depicted in Figures 3.2 to 3.5 show similar features, a systematic bias is unlikely.

Multiple zero estimates for the linear damping coefficients, particularly in surge, indicate a less reliable identification of those parameters. Nevertheless, the fact that both surge and sway damping are in the same order of magnitude suggests that they physically make sense. Even though they are estimated separately in the identification procedure, the same value might be used for both during controller design.

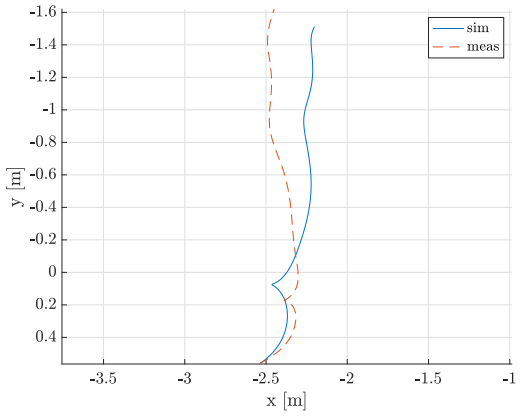
Table 3.2: Parameter estimation based on a 15 Hz PRBS input signal in the range $\{0, 1\}$.

data	I_z [kg m^2]	X_u [N s/m]	Y_v [N s/m]	N_r [N m s]	k [N]	obj minimum [-]
run 1	5.23×10^{-5}	2.64×10^{-2}	3.81×10^{-2}	3.21×10^{-5}	4.9×10^{-2}	1.52
run 2	8.36×10^{-5}	0.00	2.64×10^{-2}	5.12×10^{-5}	5.8×10^{-2}	1.34
run 3	12.0×10^{-5}	0.00	3.31×10^{-2}	2.20×10^{-5}	7.0×10^{-2}	0.97
run 4	9.89×10^{-5}	0.00	4.72×10^{-2}	2.35×10^{-5}	6.7×10^{-2}	1.00
run 5	7.79×10^{-5}	0.54×10^{-2}	1.47×10^{-2}	2.20×10^{-5}	5.6×10^{-2}	0.88
run 6	7.45×10^{-5}	1.80×10^{-2}	0.00	6.76×10^{-5}	5.9×10^{-2}	1.13
run 7	10.9×10^{-5}	0.00	2.78×10^{-2}	2.38×10^{-5}	6.4×10^{-2}	0.83
run 8	6.76×10^{-5}	0.00	2.94×10^{-2}	2.86×10^{-5}	6.5×10^{-2}	0.93
run 9	10.5×10^{-5}	0.00	2.68×10^{-2}	4.62×10^{-5}	6.8×10^{-2}	1.24
average	8.8×10^{-5}	0.55×10^{-2}	2.71×10^{-2}	4.62×10^{-5}	6.2×10^{-2}	-

Initial parameter guess: $\boldsymbol{\theta}_0 = [0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2]$.

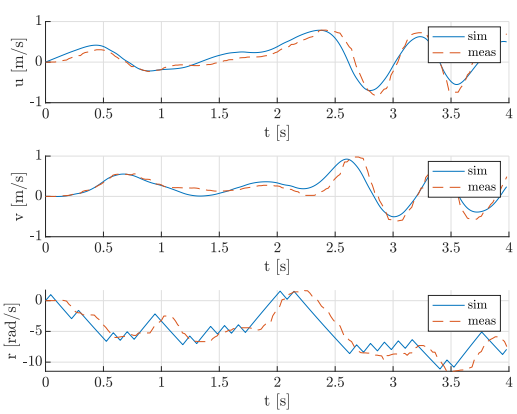


(a) Measured and simulated velocities

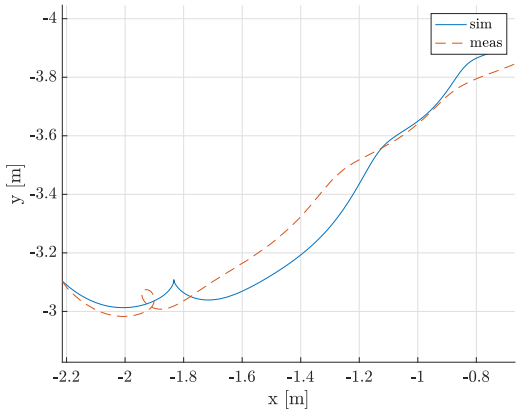


(b) Measured and simulated trajectory

Figure 3.2: Run 3 of 15 Hz PRBS identification.

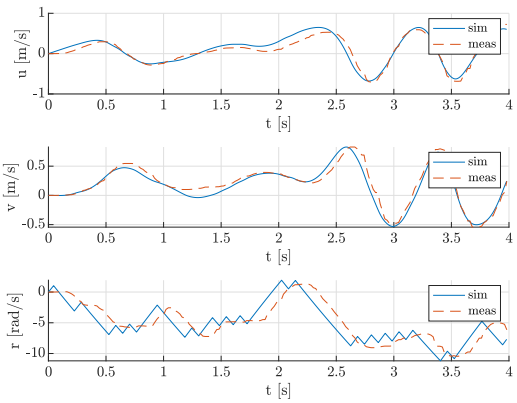


(a) Measured and simulated velocities

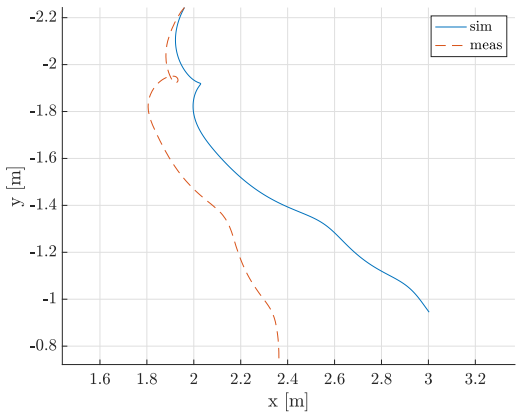


(b) Measured and simulated trajectory

Figure 3.3: Run 4 of 15 Hz PRBS identification.

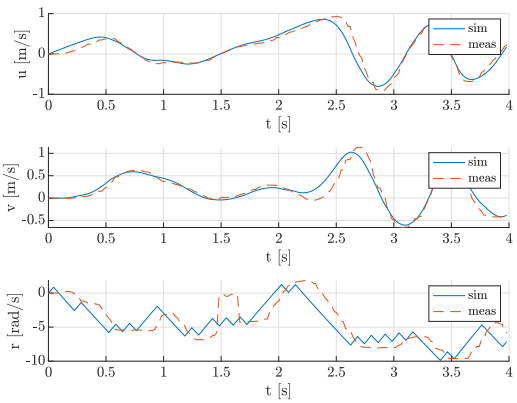


(a) Measured and simulated velocities

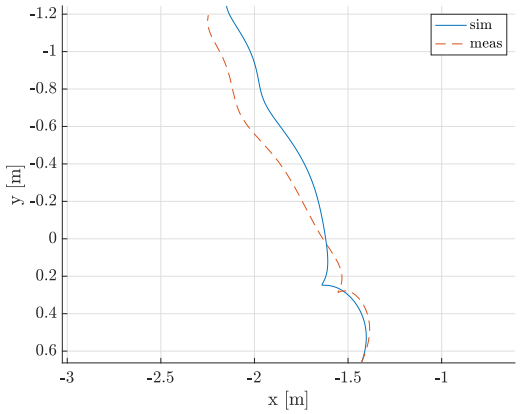


(b) Measured and simulated trajectory

Figure 3.4: Run 5 of 15 Hz PRBS identification.



(a) Measured and simulated velocities



(b) Measured and simulated trajectory

Figure 3.5: Run 7 of 15 Hz PRBS identification.

Chapter 4

Trajectory tracking controller

The following chapter presents a non-linear controller to track a given reference trajectory with the hovercraft. While the emphasis is not on the mathematical development, sufficient details are given to understand the implementation of the controller. Before employing the regulator on the real system, simulations in Matlab were performed. Both simulation results as well as experiments recorded in the drone room are given at the end of the chapter.

4.1 Reference literature

The position tracking controller for an underactuated hovercraft proposed by [5] is based on an iterative Lyapunov-based technique, in particular, integrator backstepping. For a sufficiently smooth reference trajectory it "yields global stability and exponential convergence of the position tracking error to a neighborhood of the origin that can be made arbitrarily small" [5]. Experiments with the Caltech MVWT (Multi-Vehicle Wireless Testbed), a roughly 5 kg heavy hovercraft, show the ability of the control algorithm to successfully track a circular trajectory. In a previous paper [6] by the same authors a more general approach is presented that can be applied to 3d-vehicles as well. The notation in both works is similar but for the sake of clarity [5] is used as the main reference.

4.2 Control law

As already mentioned, it is out of the scope of this report to discuss the mathematical derivation of the trajectory tracking controller in details and the interested reader is referred to [5]. However, all necessary equations for the full implementation of the control algorithm are given below.

To stay close to the notation in the reference paper, the hovercraft's position and the linear velocity in the body frame are denoted by

$$\begin{aligned} \boldsymbol{p} &= \begin{bmatrix} \eta_1 & \eta_2 \end{bmatrix}^T = \begin{bmatrix} x & y \end{bmatrix}^T \\ \boldsymbol{v} &= \begin{bmatrix} \nu_1 & \nu_2 \end{bmatrix}^T = \begin{bmatrix} u & v \end{bmatrix}^T . \end{aligned} \tag{4.1}$$

Moreover the damping coefficients in surge and sway are assumed to be equal, i.e.

$$X_u = Y_v = d_v. \tag{4.2}$$

The reference trajectory is defined as the three times continuously differentiable (in time) function

$$\boldsymbol{p}_d : [0, \infty) \to \mathbb{R}^2. \tag{4.3}$$

To formulate the control law, the tracking error is defined in the body frame as

$$\boldsymbol{e} = \mathbf{R}(\psi)^T(\boldsymbol{p} - \boldsymbol{p}_d). \tag{4.4}$$

The goal of the controller is to reduce this error to a neighborhood of the origin which is achieved by the following control law:

$$\begin{aligned} u_1 &= \begin{bmatrix} 1 & 0 \end{bmatrix} \boldsymbol{\alpha} \\ u_2 &= -m\mathbf{B}_b^T \boldsymbol{\varphi} + N_r \begin{bmatrix} 0 & 1 \end{bmatrix} \boldsymbol{\alpha} + \begin{bmatrix} 0 & I_z \end{bmatrix} \dot{\boldsymbol{\alpha}} - k_z z_2 \end{aligned} \tag{4.5}$$

with the expressions

$$\begin{aligned} z_2 &= r - \begin{bmatrix} 0 & 1 \end{bmatrix} \boldsymbol{\alpha} \\ \\ \boldsymbol{\alpha} &= -\mathbf{B}^{-1} \left(\boldsymbol{h} - \mathbf{D}_v \boldsymbol{\delta} + \frac{1}{m}\boldsymbol{e} + \frac{1}{m}\mathbf{K}_\varphi \boldsymbol{\varphi} \right) = \begin{bmatrix} \alpha_1 & \alpha_2 \end{bmatrix}^T \\ \dot{\boldsymbol{\alpha}} &= -\mathbf{B}^{-1} \left(\dot{\boldsymbol{h}} + \frac{1}{m}\boldsymbol{e} + \frac{1}{m}\mathbf{K}_\varphi \dot{\boldsymbol{\varphi}} \right) \\ \\ \boldsymbol{h} &= -\mathbf{D}_v \mathbf{R}(\psi)^T \dot{\boldsymbol{p}}_d + \frac{k_e}{m}\mathbf{D}_v \boldsymbol{e} - m\mathbf{R}(\psi)^T \ddot{\boldsymbol{p}}_d + k_e \boldsymbol{z}_1 - \frac{k_e^2}{m}\boldsymbol{e} \\ \dot{\boldsymbol{h}} &= -\mathbf{D}_v \left(-\mathbf{S}(r)\mathbf{R}(\psi)^T \dot{\boldsymbol{p}}_d + \mathbf{R}(\psi)^T \ddot{\boldsymbol{p}}_d + \frac{k_e}{m}\dot{\boldsymbol{e}} \right) \\ &\quad - m \left(\mathbf{S}(r)\mathbf{R}(\psi)^T \ddot{\boldsymbol{p}}_d + \mathbf{R}(\psi)^T \dddot{\boldsymbol{p}}_d \right) + k_e \dot{\boldsymbol{z}}_1 - \frac{k_e^2}{m}\dot{\boldsymbol{e}} \\ \\ \boldsymbol{\varphi} &= \boldsymbol{z}_1 - \boldsymbol{\delta} \\ \dot{\boldsymbol{\varphi}} &= \dot{\boldsymbol{z}}_1 \\ \\ \boldsymbol{z}_1 &= \boldsymbol{v} - \mathbf{R}(\psi)^T \dot{\boldsymbol{p}}_d + \frac{k_e}{m}\boldsymbol{e} \\ \boldsymbol{z}_1 &= \dot{\boldsymbol{v}} + \mathbf{S}(r)\mathbf{R}(\psi)^T \dot{\boldsymbol{p}}_d - \mathbf{R}(\psi)^T \ddot{\boldsymbol{p}}_d + \frac{k_e}{m}\dot{\boldsymbol{e}} \\ \\ \boldsymbol{e} &= \mathbf{R}(\psi)^T(\boldsymbol{p} - \boldsymbol{p}_d) \\ \dot{\boldsymbol{e}} &= -\mathbf{S}(r)\boldsymbol{e} + \boldsymbol{v} - \mathbf{R}(\psi)^T \dot{\boldsymbol{p}}_d \end{aligned} \tag{4.6}$$

$$\mathbf{S}(r) = \begin{bmatrix} 0 & -r \\ r & 0 \end{bmatrix}, \quad \mathbf{D}_v = \begin{bmatrix} d_v & 0 \\ 0 & d_v \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & m\delta_2 \\ 0 & -m\delta_1 \end{bmatrix}, \quad \mathbf{B}_b = \mathbf{B}(:,2) = \begin{bmatrix} m\delta_2 \\ -m\delta_1 \end{bmatrix}$$

where ψ is the yaw angle, r the yaw rate and m the mass of the hovercraft.

The tunable parameters of the controller are

$$k_e \in \mathbb{R}, \quad k_z \in \mathbb{R}, \quad \boldsymbol{\delta} = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \in \mathbb{R}^2, \quad \mathbf{K}_\varphi \in \mathbb{R}^{2\times 2} \text{ and SPD} \tag{4.7}$$

where SPD means symmetric positive definite. Unfortunately the authors of the reference paper do not mention any physical intuition behind the parameters nor do they give a clear explanation on how to find appropriate values for them.

Note that the control signals u_1 and u_2 correspond to the force and torque generated by the thrusters. Since the system defined in Chapter 2 takes input signals u_L, u_R for each thruster separately, the control signals defined above have to be converted using

$$\begin{aligned} u_L &= \frac{1}{2k}\left(u_1 + \frac{u_2}{l}\right) \\ u_R &= \frac{1}{2k}\left(u_1 - \frac{u_2}{l}\right). \end{aligned} \tag{4.8}$$

This relationship can be derived by inverting the definition of the force and torque generated by the thrusters:

$$\begin{aligned} u_1 &= k(u_L + u_R) \\ u_2 &= kl(u_L - u_R) \end{aligned} \tag{4.9}$$

4.3 Simulation

The controller is implemented in the Matlab simulation according to equations (4.5), (4.6) and (4.8). The control signals are updated at a frequency of 50 Hz, i.e. close to the 60 Hz used during the experiments in [5]. A saturation block limits the input to the system to a predefined constant in order to account for the physical limitations of the hovercraft. Most importantly, it prevents a negative input which is not feasible since the motors can only turn in one direction.

Two different hovercrafts were simulated in closed loop; the MVWT from the reference paper and the Tiny Whoover. The physical parameters of both systems are given Table 4.1.

Table 4.1: Parameters of MVWT and Tiny Whoover hovercrafts.

hovercraft	m [k]	l [m]	I_z [kg m^2]	d_v [N s/m]	N_r [N m s]	k [N]	max input
MVWT [5]	5.15	0.123	0.047	4.5	0.41	1.0	4.5 N
Tiny Whoover*	0.0583	0.0325	8.8×10^{-5}	0.0271	4.62×10^{-5}	0.062	1.0

* Parameters are average of identification (see Table 3.2)

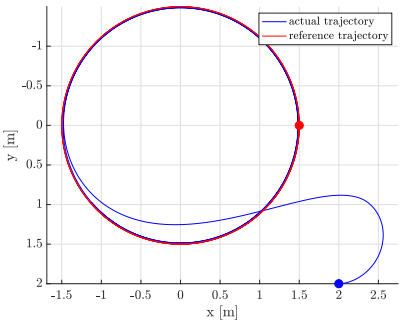
4.3.1 MVWT hovercraft

To verify the correctness of the controller implementation, the MVWT hovercraft is simulated in closed loop using the following control gains as suggested in [5]:

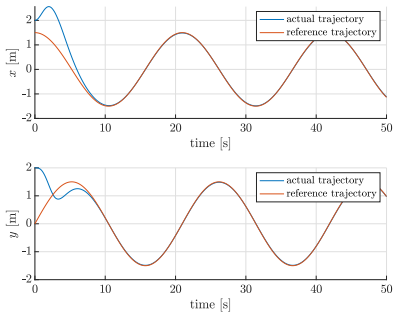
$$k_e = 4.0, \quad k_z = 0.6, \quad \mathbf{K}_\varphi = \mathrm{diag}(1.5, 1.5), \quad \boldsymbol{\delta} = \begin{bmatrix} -0.01 & 0 \end{bmatrix}^T \tag{4.10}$$

Figure 4.1 shows the simulation results of tracking a circular trajectory. Initially the controller saturates while trying to converge to the reference position but as soon as the latter is reached the control signal reduces to a value below the maximal input. The same can be observed when tracking a sinusoidal path as depicted in Figure 4.2. It is worth noting that without the saturation block, the controller outputs infeasibly large values at the beginning that lead to an unstable system.

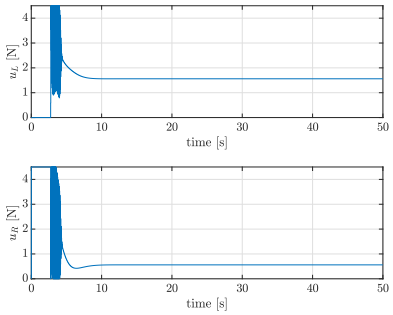
Nevertheless, since two reference trajectories can be successfully tracked successfully, the implementation is deemed correct.



(a) Actual and reference trajectory

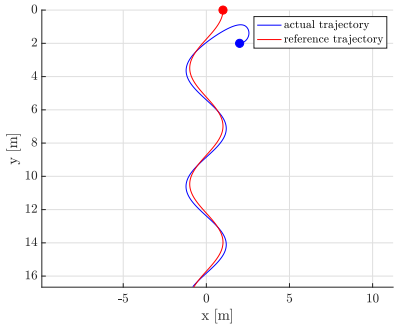


(b) Actual and reference x and y position

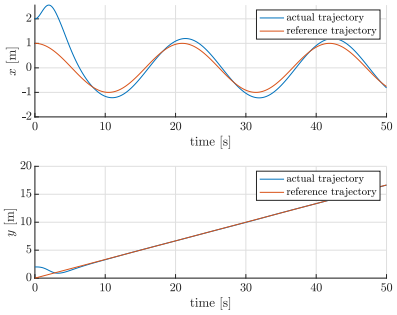


(c) Input to left and right thruster

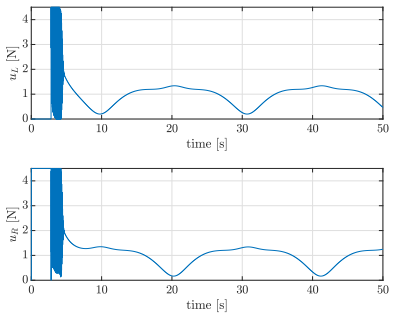
Figure 4.1: Closed loop simulation of MVWT. Tracking of circular trajectory with radius 1.5 m and angular velocity 0.3 rad/s.



(a) Actual and reference trajectory



(b) Actual and reference x and y position



(c) Input to left and right thruster

Figure 4.2: Closed loop simulation of MVWT. Tracking of sinusoidal trajectory.

4.3.2 Tiny Whoover hovercraft

After the verification of the controller, several closed loop simulations of the Tiny Whoover hovercraft were performed using the following controller parameters:

$$k_e = 4.0, \quad k_z = 0.0003, \quad \mathbf{K}_\varphi = \mathrm{diag}(0.005, 0.005), \quad \boldsymbol{\delta} = \begin{bmatrix} -0.01 & 0 \end{bmatrix}^T \tag{4.11}$$

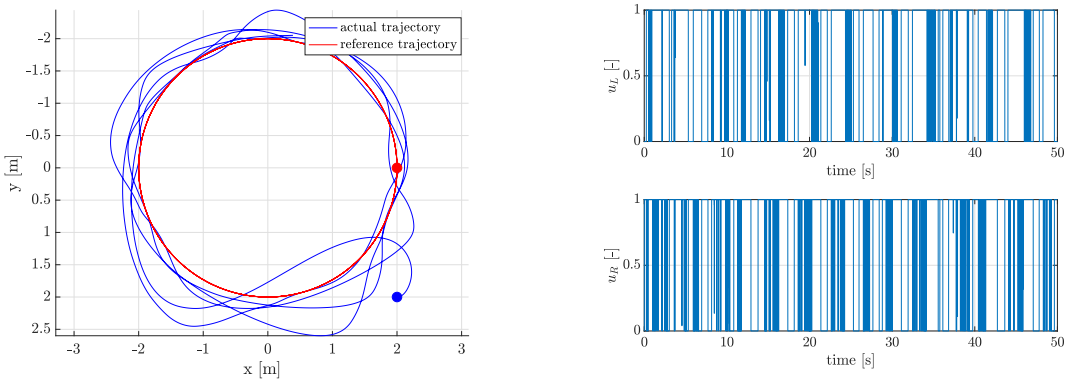
These parameters were found by extensive trial and error and are by no means optimal. While the hovercraft can track all imposed trajectories, there are often large oscillations around the latter as can be seen in Figures 4.3, 4.4 and 4.5. Moreover, the control signal saturates most of the time at 0 or 1. Unfortunately no combination of controller parameters could be found that reduced the amplitude of the control output to the desired range.

During parameter tuning the following qualitative observations were made:

- Parameters k_e and $\mathbf{K}_\varphi$ do not seem to have a large effect on the quality of the tracking performance.
- Increasing k_z while keeping the other parameters as given in (4.11) makes the trajectory smoother but introduces a lag with respect to the desired position. Decreasing k_z makes the trajectory more "jaggy".
- The vector $\boldsymbol{\delta}$ seems to be a kind tolerance for the tracking error. Values that deviate more than a magnitude from the value given in (4.11) result in poor tracking.

It is also interesting to observe how the controller handles an overshoot of the desired trajectory. Since the motors of the hovercraft are unidirectional there is no direct way of braking. Instead, the hovercraft has to spin around by 180 degrees and apply thrust in the opposite direction. This is likely the reason for the "jagged" trajectory in Figure 4.4.

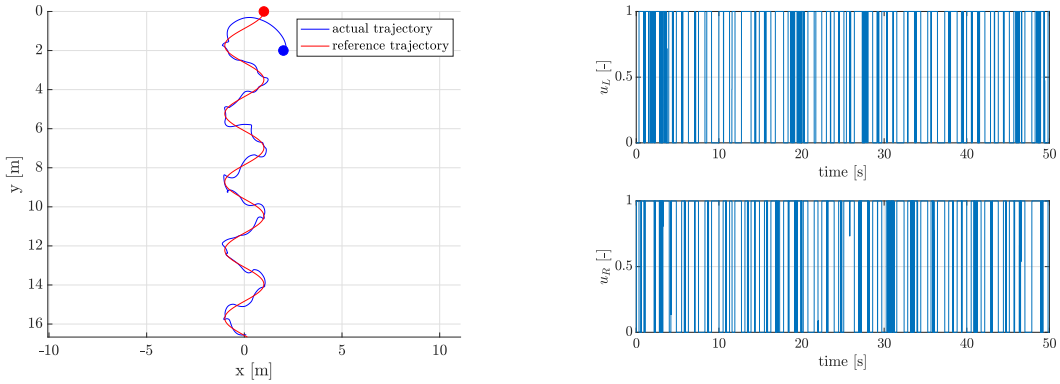
The fact that the controller produced very good results on the large MVWT system but showed poor performance and saturation of the input on the Tiny Whoover hovercraft indicates that the non-linear control law might not scale well to smaller systems. Unfortunately, due to time constraints this hypothesis could not be further investigated.



(a) Actual and reference trajectory

(b) Input to left and right thruster

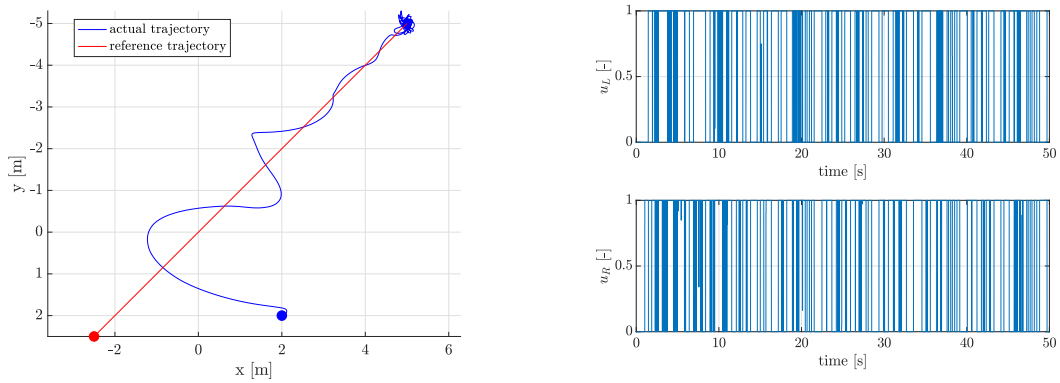
Figure 4.3: Closed loop simulation of Tiny Whoover. Tracking of circular trajectory with radius 2.0 m and angular velocity 0.6 rad/s.



(a) Actual and reference trajectory

(b) Input to left and right thruster

Figure 4.4: Closed loop simulation of Tiny Whoover. Tracking of sinusoidal trajectory.



(a) Actual and reference trajectory

(b) Input to left and right thruster

Figure 4.5: Closed loop simulation of Tiny Whoover. Tracking of line segment with a speed of 0.707 m/s and stopping at endpoint.

4.4 Experiments

After evaluating the trajectory controller in simulation it was implemented and tested on the ROS framework. The following section only presents the outcome of these experiments. For details about the implementation the reader is referred to Chapter 5.

For the tracking of a circle and a line segment, control parameters around the following values were used:

$$k_e = 0.5, \quad k_z = 0.0004, \quad \mathbf{K}_\varphi = \mathrm{diag}(0.05, 0.05), \quad \boldsymbol{\delta} = \begin{bmatrix} -0.01 & 0 \end{bmatrix}^T \tag{4.12}$$

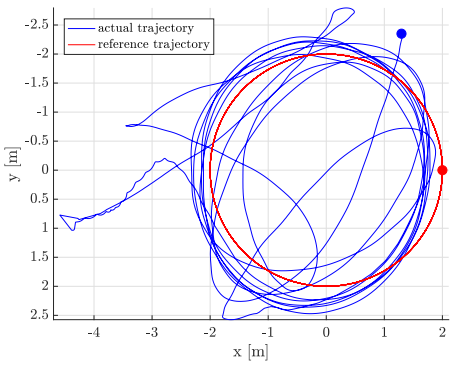
During experiments it was found that the force generated by the thrusters decreases over time, most likely due to battery depletion. This results in a reduced lift and hence more friction on the ground as well as less propulsion power. To account for this, the control parameters often have to be slightly adjusted.

It was also observed that at full battery charge, i.e. full lift power, the trajectory controller is very unstable. Often the hovercraft cannot recover from a deviation from the reference trajectory and starts spinning on the spot. There are several plausible reasons for that:

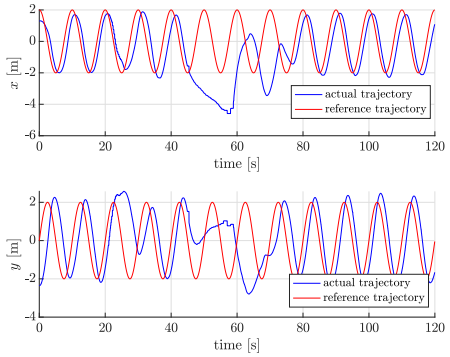
- When overshooting the trajectory the hovercraft often comes close to the borders of the tracking area of the drone room. In these situations the motion capture system sometimes looses one of the trackers which results in poor estimates of the position and velocities.
- The controller is very sensitive to the parameter k_z. A "good" value can only be achieved by extensive tuning.
- The controller constantly operates in saturation while the non-linear control law is designed without this limitation in mind.

Despite these issues a circle of radius 2 m can be tracked as shown in Figure 4.6. As already observed in simulation, the control signal is constantly saturating. The values of the latter are in the order of 10^4 to 10^6 before saturation (not shown in Figure) and thus very far from the desired range of 0 to 1. Note that during this experiment, the controller was able to recover from the large deviation at around 50 s. However, as mentioned before this is not always the case.

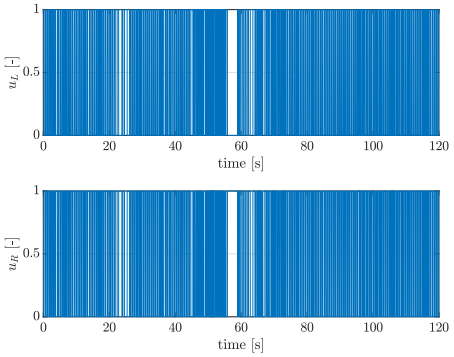
In a second experiment a linear trajectory from a point A to a point B was tracked. Due to space limitation in the drone room, the line is not long enough to allow the hovercraft to converge to it. However, it is possible to converge to a neighborhood of the endpoint B as shown in Figure 4.7. Note that the behavior of the hovercraft looks similar to the line tracking in simulation but with larger oscillations around the endpoint.



(a) Actual and reference trajectory

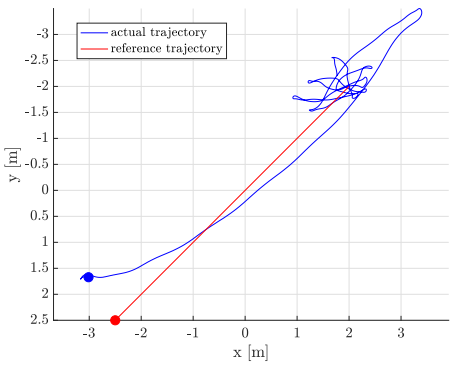


(b) Actual and reference x and y position

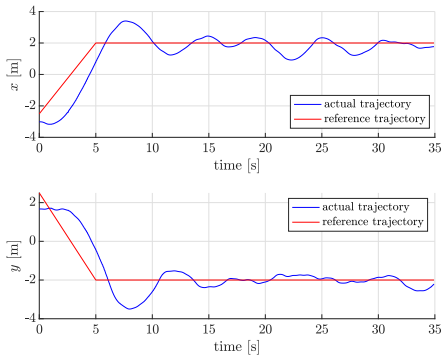


(c) Input to left and right thruster

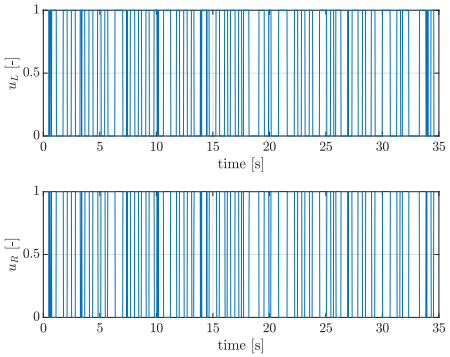
Figure 4.6: Closed loop experiment with Tiny Whoover. Tracking of circular trajectory with radius 2.0 m and angular velocity 0.63 rad/s.



(a) Actual and reference trajectory



(b) Actual and reference x and y position



(c) Input to left and right thruster

Figure 4.7: Closed loop experiment with Tiny Whoover. Tracking of line segment with a speed of 1.27 m/s and stopping at endpoint.

Chapter 5

Drone room setup

This chapter provides insight into the drone room setup that was used to record identification data and to evaluate the tracking controller on the Tiny Whoover. The goal is to provide the reader with an overview of the overall system and in particular data acquisition and hovercraft control. For the sake of brevity, it is assumed that the reader is familiar with the basics of the Robot Operating System (ROS). Moreover, for specific details concerning the implementation of various modules the reader is referred to the source code.

5.1 Motion capture system

The drone room in the MED building at EPFL is equipped with a OptiTrack motion capture system consisting of 26 cameras. If correctly calibrated, the tracking reaches millimeter precision. OptiTrack's software platform allows the user to set up a rigid body and to stream the rigid body data via a VRPN streaming engine into the network. This information can then be read by any VRPN client on the same network.

5.2 ROS environment

Everything related to the identification and control of the hovercraft in the drone room is currently implemented in Python 2.7 in a ROS package called *ground*. Python 2.7 was chosen due to compatibility reasons. Note that this package is still in a very early state and by no means production ready. Nevertheless, particular emphasis was put on a structured and clean code base to facilitate further development in future projects. The *ground* package itself makes use of many standard ROS packages such as the *vrpn_client_ros* package to receive rigid body information from the OptiTrack system or the *dynamic_reconfigure* package to adjust controller parameters in real time. Another standalone package called *rqt_multiplot* is used to visualize the position and velocities of the hovercraft.

In the following sections the content of the *ground* package is described in a bit more detail.

5.2.1 Sender

The sender node establishes a connection with the hovercraft using the UDP protocol. It subscribes to a topic called *controls* to which the control signal $\boldsymbol{u} = \begin{bmatrix} u_L & u_R \end{bmatrix}^T$ as well as the lift are published. Whenever a new message is published on the topic, the sender node converts and clips the control

signal and the lift to the range $[1000, 2000]$ which is the input range of the flight controller. The converted data is then sent via WiFi to the hovercraft.

Whenever the node is shut down it will try to send a disarming command to the hovercraft which sets all motor inputs to zero. This provides a convenient way of shutting down the whole ROS setup without having to manually turn off the hovercraft.

5.2.2 Controller

Most controllers need some common functions such as requesting feedback from the OptiTrack system or executing some calculations in each iteration. For this reason a controller skeleton is implemented in a base class called `Controller`. Any specific controller will then inherit from this class and implement any other required functionality.

The two main methods of the base class are called `_get_state` and `_publish_state`. The first one receives and converts the position and velocities from the OptiTrack system to the coordinate frames described in Chapter 2 while the latter publishes this information to different topics. Since all topics are logged and saved in a bagfile (logfile of ROS) this information can for example be used in identification. Note that in the future the acquisition of position and velocities could be handled in a separate process to strictly separate controller and feedback modules. This would also allow to run the data acquisition at a higher frequency than the controller.

The actual control action is implemented in the `iteration` method. This method should also publish the control signals to the *controls* topic. Since the control law is different for every controller the `iteration` method has to be implemented in each child controller separately.

The two available controllers, an open loop controller and a trajectory tracking controller are described below.

Open loop controller

The open loop controller called `OpenLoopPRBS` reads a PRBS signal from a csv file and applies its content to the hovercraft. It was used to generate the data used for system identification in Chapter 3.

Trajectory tracking controller

The class `TrajectoryTrackingController` implements the non-linear trajectory following controller covered in Chapter 4. It is possible to select from three different reference trajectories, namely a circle, a line or a rhodonea curve. The parameters of the controller can be adjusted in real time using the *dynamic_reconfigure* package in ROS which facilitates the controller tuning.

Chapter 6

Conclusion

This semester project presents the identification and control of a miniature hovercraft called Tiny Whoover. In the first phase of the project a mathematical model of the hovercraft was implemented in Matlab and an identification procedure was developed to estimate the physical parameters of the vehicle. This parameter estimation algorithm was successfully validated using artificial identification data generated in simulation. To perform system identification on the real system, the tracking data from a motion capture system installed in the EPFL drone room was recorded using a custom ROS package developed by the author. The most consistent identification results were obtained by applying a PRBS signal of 15 Hz to the thrusters and averaging over nine different runs. It is important to note that particularly the linear damping coefficients vary considerably over different runs. This can most likely be attributed to the rough and uneven floor of the drone room.

In a second phase a non-linear trajectory controller based on the Lyapunov theory was implemented in Matlab. The correctness of the implementation could be confirmed by replicating the experiments of the reference paper in simulation. It could be shown that the roughly 5 kg heavy reference hovercraft can precisely track a circle without saturation effects of the controller. Using the previously estimated parameters, the controller was then evaluated on the Tiny Whoover system in simulation. Through trial and error a set of controller gains could be found that allows the tracking of several different trajectories. However, the resulting control signal by far exceeds the admissible input range and thus the controller operates constantly in saturation.

Finally, the trajectory controller was implemented in ROS and tested on the Tiny Whoover in the drone room. The experimental results show similar effects as observed in simulation. While a circular trajectory can be tracked, the controller saturates permanently.

The fact that the considered tracking controller successfully operates on a large hovercraft but shows poor performance on the very light Tiny Whoover leads to the hypothesis that the non-linear control law does not scale well to small systems. In future projects a grid search over all control parameters could be performed and the resulting controllers should be evaluated with respect to tracking accuracy and amplitude of the control signal. Further, it might be worth considering a non-linear model predictive controller. Intuitively, the predictive capabilities of the latter could help with the limitations imposed by the unidirectional motors. Overshooting the reference trajectory could be reduced by braking earlier on the trajectory instead of at the moment of deviation.

Bibliography

[1] T. Whoop. (2019) Tiny whoover. [Accessed 2019-06-25]. [Online]. Available: https://www.tinywhoop.com/products/tiny-whoover-micro-drifter-hovercraft-kit-v2

[2] ——. (2019) Tiny whoop. [Accessed 2019-06-25]. [Online]. Available: https://www.tinywhoop.com

[3] K. Abdelaziz, "Modelling, Simulation and Control of a Miniature Hovercraft," Semester project, EPFL, Laboratoire Automatique, July 2019.

[4] T. I. Fossen, *Guidance and control of ocean vehicles*. John Wiley & Sons Inc, 1994.

[5] A. P. Aguiar, L. Cremean, and J. P. Hespanha, "Position tracking for a nonlinear underactuated hovercraft: controller design and experimental results," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, vol. 4, Dec 2003, pp. 3858–3863 vol.4.

[6] A. P. Aguiar and J. P. Hespanha, "Position tracking of underactuated vehicles," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 3, June 2003, pp. 1988–1993 vol.3.