

# PATH-FOLLOWING MODEL PREDICTIVE CONTROL FOR MICRO-SCALE CARS

Path-Following using Adaptive and Stochastic MPC  
Schemes and Machine Learning Tools

Adi Vardi

Master Project Thesis

Professor: Colin Jones

Supervisor: Sanket Diwale

June 23, 2017

Section de Microtechnique - Micro-Engineering

École Polytechnique Fédérale de Lausanne

Switzerland



**Mr. Adi VARDI**

Candidate for the microengineer  
title

## **Master Project**

### **Airbone Wind Energy**

Wind resources tend to be significantly stronger and more consistent with increasing altitude. Furthermore, conventional wind turbines quickly reach their limit in accessing these higher winds due to structural limitations. This creates a potential for improved power generation with Airborne Wind Energy systems. A frequent design for such systems includes a flying airfoil tethered to a ground station. The station is equipped with a power generator and the aerodynamic force generated by the flying wing is used to drive this generator as the tether reels out under the aerodynamic force. Once the tether reaches its maximum length, the wing enters a low force generating gliding trajectory, and the tether is reeled back in with low power consumption, thus generating a net positive power generation cycle.

The execution of such motion however requires overcoming critical control challenges and perform control and optimization for improved power generation in presence of system and environmental (wind) uncertainties. The proposed project will provide opportunities to students to work on different aspects of the system and has several subprojects as listed below:

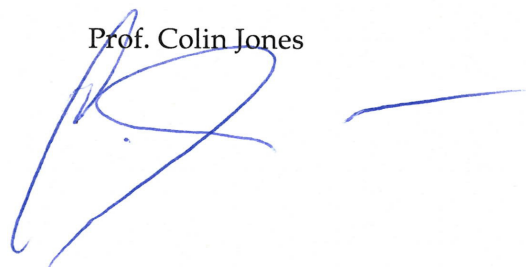
1. System modeling and nonlinear control: Will explore methods for modeling the system for better control and will explore the tools of robust and adaptive nonlinear control theory to implement controllers for the existing experimental platforms.
2. Path following model predictive control: Will implement and test model predictive control (MPC) schemes for tracking of power optimal trajectories in the experimental system.
3. Data based modeling and control of experimental systems: We will explore the use of Gaussian Process Machine Learning tools to model the dynamics and performance characteristics of the system to find the optimal control inputs for the system. The learning based models further provide an improved prediction performance crucial for the model based control methods explored above.

Realistic high fidelity simulators and working experimental platforms are ready and will be available to bench test the controllers and learning algorithms on the real system.

Lausanne, April 2017

**Co-supervisor :**  
Sanket DIWALE

Prof. Colin Jones





# Path-Following Model Predictive Control for Micro-Scale Cars

*Adi Vardi*

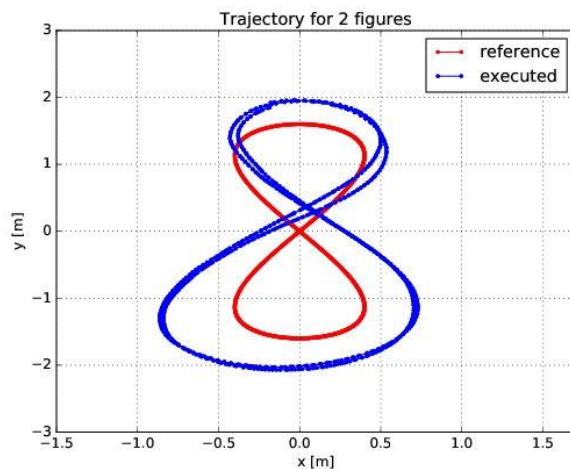
*Section Microtechnique*

*Supervisor: Sanket Diwale*

*Professor: Colin Jones*

## Abstract

This thesis presents different approaches and schemes for non-linear model predictive control (MPC) for path-following. Previous work on path-following MPC for the Airborne Wind Energy system are extended using different methods, implemented and tested both on simulation and on an experimental setup. The experiments are done using micro-scale radio controlled car. The experimental setup is presented along with necessary tools. The optimal control problem and the control law are formulated such that an MPC controller solving it would produce path following.

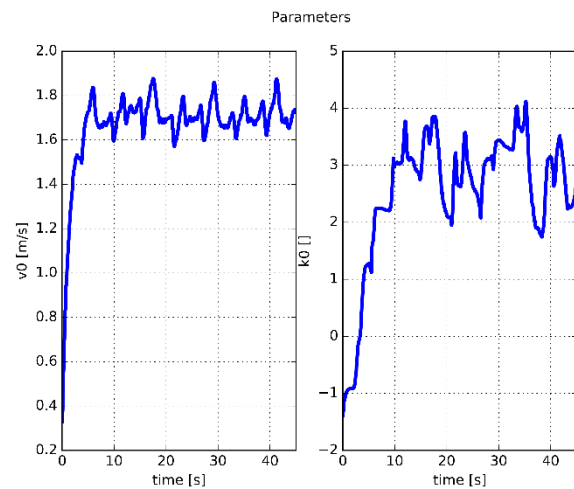


## Experimental results using Nominal MPFC

A nominal nonlinear constrained model predictive path-following (MPFC) scheme is first implemented and tested on the experimental setup. It is shown that additional regulation terms are necessary for a real-world application. The results of the obtained MPC controller are presented, as well as an investigation of the effects of parameter tuning.

Adaptive MPFC is used in order to improve the tracking performance. The model's parameters are updated continuously using an adaptation algorithm. An adaptation algorithm based on recursive least squares is

derived for several representations of the system, as well as a convergence condition. The adaptation algorithm is then combined with the MPFC controller and its performance is presented and analyzed.



## Parameters evolution in Adaptive MPFC

A second adaptation algorithm is established using Gaussian processes. A comparison of different types of Gaussian processes fitted to the car's dynamics is presented. The corresponding adaptive MPFC scheme is proposed and its performance is presented.

Stochastic MPFC is explored, using Gaussian processes to create a stochastic model of the system's dynamics. A stochastic MPFC scheme is devised by replacing the system's deterministic model with a Gaussian process. The tracking performance as well as the uncertainty of the controller are presented and investigated.

Uncertainty propagation tools for Gaussian processes are derived and implemented using both exact moment matching for Gaussian kernels and approximate moment matching for any kernel function.

*Keywords: nonlinear model predictive control, path following, adaptive control, stochastic control, recursive least squares, Gaussian process, moment matching*



## Abstract

This thesis presents different approaches and schemes for non-linear model predictive control (MPC) for path-following. Previous work on path-following MPC for the Airborne Wind Energy system are extended using different methods, implemented and tested both on simulation and on an experimental setup. The experiments are done using micro-scale radio controlled car. The experimental setup is presented along with necessary tools. The optimal control problem and the control law are formulated such that an MPC controller solving it would produce path following.

A nominal nonlinear constrained model predictive path-following (MPFC) scheme is first implemented and tested on the experimental setup. It is shown that additional regularization terms are necessary for a real-world application. The results of the obtained MPC controller are presented, as well as an investigation of the effects of parameter tuning.

Adaptive MPFC is used in order to improve the tracking performance. The model's parameters are updated continuously using an adaptation algorithm. An adaptation algorithm based on recursive least squares is derived for several representations of the system, as well as a convergence condition. The adaptation algorithm is then combined with the MPFC controller and its performance is presented and analyzed.

A second adaptation algorithm is established using Gaussian processes. A comparison of different types of Gaussian processes fitted to the car's dynamics is presented. The corresponding adaptive MPFC scheme is proposed and its performance is presented.

Stochastic MPFC is explored, using Gaussian processes to create a stochastic model of the system's dynamics. A stochastic MPFC scheme is devised by replacing the system's deterministic model with a Gaussian process. The tracking performance as well as the uncertainty of the controller are presented and investigated.

Uncertainty propagation tools for Gaussian processes are derived and implemented using both exact moment matching for Gaussian kernels and approximate moment matching for any kernel function.

*Keywords:* nonlinear model predictive control, path following, adaptive control, stochastic control, recursive least squares, Gaussian process, moment matching

## Acknowledgments

This project would not have been possible without the support of many people.

Many thanks to Professor Colin Jones who was in charge of this project and helped define new directions and alleys to explore. Second, to my supervisor Sanket Diwale who helped to bring this work to term. I would also like to thank Timm Fullwasser who is the external expert for this work. Many thanks to members of the Automatic Laboratory Harsh Shukla, Petr Listov and Christophe Salzmann for their help in various subjects along the way. Finally, to all the members of LIS and BioRob for sharing the experiments room with me and for their help during the many hours I spent in there.





# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>1</b>  |
| <b>2</b> | <b>Preliminaries</b>   | <b>3</b>  |
| 2.1      | Experimental setup . . . . .                                     | 3         |
| 2.1.1    | Steering commands . . . . .                                      | 3         |
| 2.2      | State Observer . . . . .   | 4         |
| 2.2.1    | Velocity Estimation . . . . .                                    | 4         |
| 2.2.2    | Velocity Direction Estimation . . . . .                          | 4         |
| 2.3      | Delay Estimation . . . . .                                       | 5         |
| <b>3</b> | <b>Low Level Tracking</b>  | <b>7</b>  |
| <b>4</b> | <b>Problem Statement and Dynamics</b>                            | <b>9</b>  |
| 4.1      | Car Model . . . . .  | 9         |
| 4.2      | Reference Path . . . . .   | 9         |
| 4.3      | Augmented System . . . . .                                       | 10        |
| 4.4      | Tracking Error Metric . . . . .                                  | 10        |
| <b>5</b> | <b>Nominal Model Predictive Path-Following Control</b>           | <b>11</b> |
| 5.1      | The Model Predictive Path-Following Control Scheme . . . . .     | 11        |
| 5.2      | Additional Constraints and Cost . . . . .                        | 11        |
| 5.3      | Experimental Results . . . . .                                   | 13        |
| 5.3.1    | Improving Performance by Parameters Tuning . . . . .             | 14        |
| 5.4      | Summary . . . . .  | 15        |
| <b>6</b> | <b>Adaptation algorithm - Recursive Least Squares</b>            | <b>17</b> |
| 6.1      | Representation I . . . . .                                       | 17        |
| 6.2      | Representation II . . . . .                                      | 18        |
| 6.2.1    | Sufficient Condition for Convergence . . . . .                   | 19        |
| 6.3      | Filter RLS . . . . .   | 20        |
| 6.4      | Summary . . . . .  | 21        |
| <b>7</b> | <b>Adaptive MPFC using Recursive Least Squares</b>               | <b>23</b> |
| 7.1      | Motivation . . . . .   | 23        |
| 7.2      | Recursive Least Squares for the Car's Dynamical System . . . . . | 23        |
| 7.2.1    | Representation I . . . . .                                       | 24        |
| 7.2.2    | Representation II . . . . .                                      | 24        |
| 7.2.3    | Replacing the Low Level Tracker . . . . .                        | 24        |
| 7.3      | Simulation Results . . . . .                                     | 25        |
| 7.3.1    | Simulation Setup . . . . .                                       | 25        |
| 7.3.2    | Model $\dot{\gamma} = k \cdot u_\gamma$ . . . . .                | 25        |
| 7.3.3    | Model $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ . . . . .        | 26        |
| 7.3.4    | Using a Low Level Tracker . . . . .                              | 27        |
| 7.4      | Experimental Results . . . . .                                   | 28        |
| 7.4.1    | Model $\dot{\gamma} = k \cdot u_\gamma$ . . . . .                | 28        |
| 7.4.2    | Model $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ . . . . .        | 29        |
| 7.4.3    | Small Reference Path . . . . .                                   | 30        |
| 7.5      | Summary . . . . .  | 31        |

|           |  |           |
|-----------|--|-----------|
| <b>8</b>  | <b>Gaussian Processes Modeling for Adaptive MPFC</b>     | <b>33</b> |
| 8.1       | Gaussian Processes . . . . .                             | 33        |
| 8.2       | System Modeling using Gaussian Processes . . . . .       | 33        |
| 8.3       | Comparison of Fitting Methods . . . . .                  | 35        |
| 8.4       | Adaptive MPFC using Gaussian Processes . . . . .         | 35        |
| 8.5       | Summary . . . . .  | 37        |
| <b>9</b>  | <b>Stochastic MPFC</b>                                   | <b>39</b> |
| 9.1       | The Stochastic MPFC Scheme . . . . .                     | 39        |
| 9.1.1     | Integration of Distributions . . . . .                   | 40        |
| 9.2       | Simulation Results . . . . .                             | 41        |
| 9.3       | Experimental Results . . . . .                           | 42        |
| 9.4       | Summary . . . . .  | 44        |
| <b>10</b> | <b>Uncertainty Propagation for Gaussian Processes</b>    | <b>45</b> |
| 10.1      | Testing System . . . . .                                 | 45        |
| 10.2      | Exact Moment Matching . . . . .                          | 45        |
| 10.3      | Approximate Moment Matching . . . . .                    | 46        |
| <b>11</b> | <b>Conclusions and Future Work</b>                       | <b>49</b> |
| <b>A</b>  | <b>Numerical Values of Parameters and Results</b>        | <b>51</b> |
| <b>B</b>  | <b>Affine transformations of Normal random variables</b> | <b>52</b> |
| <b>C</b>  | <b>Approximate Moment Matching - Derivation</b>          | <b>53</b> |
| <b>D</b>  | <b>Implementation Guide</b>                              | <b>56</b> |
|           | <b>References</b>  | <b>59</b> |

## List of Figures

|      |  |    |
|------|--|----|
| 2.1  | The RC car used in the experiments . . . . .   | 3  |
| 2.2  | The experimental setup . . . . .   | 3  |
| 2.3  | Velocity estimation using uniform finite differences . . . . .   | 4  |
| 2.4  | Comparison of different velocity estimation methods . . . . .  | 5  |
| 2.5  | Comparison of different $\gamma$ estimations. Pure Kalman filter (green) uses a single filter while Wrapped Kalman filter (red) uses two separate filters. . . . . | 5  |
| 2.6  | Delay Estimation . . . . .   | 6  |
| 3.1  | Low Level Tracking results for a step input . . . . .  | 7  |
| 5.1  | $u_\gamma$ and $\gamma$ for nominal MPFC (experimental) . . . . .  | 12 |
| 5.2  | Nominal MPFC with regularization (experimental) . . . . .  | 13 |
| 5.3  | Velocity profile and low level tracking for nominal MPFC (experimental) . . . . .  | 13 |
| 5.4  | Trajectories for different values of $Q_E$ (experimental) . . . . .  | 14 |
| 5.5  | Trajectories for different values of $Q$ (experimental) . . . . .  | 15 |
| 7.1  | Adaptive control scheme . . . . .  | 23 |
| 7.2  | Adaptive MPFC with $\mu_k = 1$ for $\dot{\gamma} = k \cdot u_\gamma$ (simulation) . . . . .  | 25 |
| 7.3  | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k \cdot u_\gamma$ (simulation) . . . . .   | 25 |
| 7.4  | Adaptive MPFC with $\mu_k = 2$ for $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ (simulation) . . . . .  | 26 |
| 7.5  | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ (simulation) . . . . .   | 27 |
| 7.6  | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ - better result (simulation) . . . . .   | 27 |
| 7.7  | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k \cdot u_\gamma$ with low level tracker (simulation) . . . . .  | 28 |
| 7.8  | Adaptive MPFC with $\mu_k = 1$ for $\dot{\gamma} = k \cdot u_\gamma$ (experimental) . . . . .  | 29 |
| 7.9  | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k \cdot u_\gamma$ (experimental) . . . . .   | 29 |
| 7.10 | Adaptive MPFC with $\mu_k = 1$ for $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ (experimental) . . . . .  | 30 |
| 7.11 | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ (experimental) . . . . .   | 30 |
| 7.12 | Adaptive MPFC with $\mu_k = 10$ for $\dot{\gamma} = k \cdot u_\gamma$ for small reference path (experimental) . . . . .  | 31 |
| 8.1  | Gaussian process fitting using the differential wheels explicit basis . . . . .  | 34 |
| 8.2  | Gaussian process fitting for simulation using (a)full, (b)linear basis, (c)squared exponential kernel . . . . .  | 35 |
| 8.3  | Gaussian process fitting for experiment using (a)full, (b)linear basis, (c)squared exponential kernel . . . . .  | 36 |
| 8.4  | Adaptive MPFC using Gaussian processes fitting (simulation) . . . . .  | 36 |
| 8.5  | Adaptive MPFC using Gaussian processes fitting (experimental) . . . . .  | 37 |
| 9.1  | Stochastic MPFC (simulation) . . . . .   | 41 |
| 9.2  | Prediction of the state over the horizon (simulation) . . . . .  | 42 |
| 9.3  | Stochastic MPFC at normal velocity (experimental) . . . . .  | 42 |
| 9.4  | Prediction of the state over the horizon (experimental) . . . . .  | 43 |
| 9.5  | Velocity profile for the normal velocity stochastic MPFC experiment . . . . .  | 43 |
| 9.6  | Stochastic MPFC at high speed (experimental) . . . . .   | 44 |
| 10.1 | Gaussian process fitting to the true function using squared exponential kernel and linear explicit basis . . . . .   | 45 |
| 10.2 | Uncertainty propagation using Exact Moment Matching . . . . .  | 46 |
| 10.3 | Uncertainty propagation using Approximate and Exact Moment Matching . . . . .  | 47 |



# 1 Introduction

Following a predefined path has been a challenge for automatic systems for many years. As human beings, path following is as natural to us as can be. When it comes to automation and robotics, however, following a path is a task not easily accomplished. Many factors contribute to this challenge such as a lack of certainty in the localization and positioning, actuation limits rendering certain movements impossible and sensitivity to noise and disturbances. Unfortunately, following a path is a common task in many applications. One can come up with a huge number of examples, ranging from a factory-floor machine screwing bolts, a robot gripping a tool or an autonomous car trying to stay on course.

A key component in a path following system is the controller. A controller's main purpose is to find the optimal input to apply to the system in order to generate the desired behavior. This work concentrates on a certain type of control - Model Predictive Control (MPC) applied in a receding horizon fashion. An MPC controller is based on the idea that the system's behavior can be predicted if a model of its dynamics is available. Naturally, predicting an infinitely long horizon and taking into account all possible cases is impossible. As such, an MPC controller uses the information currently available to it to predict the system's state over a finite horizon. It can then find the optimal sequence of inputs to apply in order to best accomplish the goal while respecting the constraints of the system. Once an optimal input found, the first input is applied to the system and the optimization problem is solved again for the new state of the system.

One application in particular has inspired the work on this project, as a part of the Airborne Wind Energy (AWE) project. AWE systems involve an airborne vehicle capable of generating energy from the strong winds in high altitude. A common design is composed of a kite tethered to a power generator on the ground. Using the generated lift, energy is produced while the tether reels out. Once the maximum length is reached, the kite follows a gliding trajectory with a lower lift, enabling to reel the tether back using only a small amount of energy. As the net energy produced over one cycle is positive, repetitively following the whole trajectory would produce green energy to power humanity's growing demand. (For more details about Airborne Wind Energy see [1, 2])

An optimal trajectory for generating maximum power exists and can be found using numerical control optimizers ([3, 4, 5]). However, tracking this trajectory using a kite is not an easy task. In addition to the challenges confronting any path-following application, kite systems are especially limited as their main propulsion is provided by the wind. As such, a kite can only follow trajectories that are coherent with the current wind conditions.

Previous works have approached this challenge in different ways. In [6] the time parameterized trajectory is tracked using time-varying sets of Lyapunov functions as terminal regions for a constrained non-linear MPC. In [7] the path following scheme is based on linearizing the AWE system. This however localizes the region of attraction and leads to sub-optimal control. [8] presents a non-linear MPFC scheme for constrained path-following problems using a virtual time system to track a geometric path. They provide sufficient convergence conditions and explore the geometric nature of path-following problems.

In this work, we propose a model predictive path following control (MPFC) similar to [9]. Instead of tracking a time-dependent trajectory, the reference is a geometric path. By removing the time dependency, the system can track the path at its own speed, defined by the wind. To create an MPFC controller, the system is augmented by adding a virtual system. The virtual system includes a virtual time which controls the position of a virtual point on the reference path. The virtual point serves as the tracking reference for the controller at any given time and can be moved along the path by controlling the virtual time. The MPC controller searches for the optimal inputs to both the real and virtual system and is applied in a receding horizon fashion in order to follow the path as best possible. The terminal constraints used are inspired from vector field schemes. The investigation of convergence and recursive feasibility are presented in [9].

In order to test the MPFC schemes presented in this work, both simulation and an experimental setup are used. To facilitate testing and concentrate the effort on the controller, the kite system is replaced by a micro-scale RC car. The car's throttle is fixed as to imitate a gliding kite system.

In order to explore the performance of such system and to improve the tracking, we propose several different MPFC schemes. We derive the problem statement and the controller using a standard nominal MPFC. We then propose an adaptive MPFC scheme using an adaptation algorithm to better

estimate the system's parameters. Two adaptation algorithms are presented. First, several versions of recursive least squares. Second, a recursive modeling of the system using Gaussian processes. Both algorithms are coupled with the MPFC controller to constitute an adaptive MPFC. Finally, we explore the usage of Gaussian processes in a stochastic MPFC controller, where the system's model is replaced by a Gaussian process.

All proposed schemes are tested on simulation and on the experimental setup and their performances are presented and analyzed.

This thesis is structured as follows. In section 2 the experimental setup is presented in details, along with estimation tools necessary to its functioning and the delay in the system is characterized. In section 3 the low level tracker used to track the controller's plan is presented. Section 4 presents the problem statement as well as the system's dynamics, the virtual and augmented systems, the reference path and a tracking error metric. Section 5 discusses the nominal MPFC and the required modifications needed for a real world application. The experimental results are presented as well as an investigation on the effects of tuning the controller's parameters. In section 6 the recursive least squares algorithms and a sufficient convergence condition are derived. In section 7 an adaptive MPFC scheme using RLS and its performance are presented. Section 8 presents Gaussian processes and compares fitting of different types of Gaussian processes. It then presents an adaptive MPFC controller using Gaussian processes fitting. Section 9 discusses a stochastic MPFC scheme in which the deterministic system is replaced by a Gaussian process and presents its performance. Section 10 presents the derivation and implementation results of uncertainty propagation using Exact and Approximate Moment Matching for Gaussian processes. Final conclusions and possible future work are presented in section 11.

## 2 Preliminaries

### 2.1 Experimental setup

In order to test the algorithms developed in this thesis, an experimental setup is needed. We have chosen to use a micro scale radio controlled car, shown in fig. 2.1.

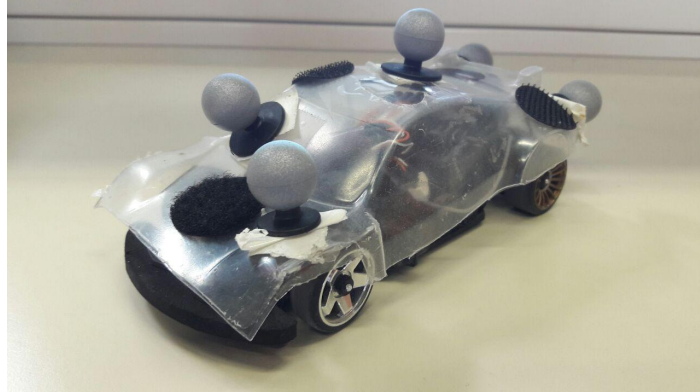


Figure 2.1: The RC car used in the experiments

The car's position and orientation are measured using a camera tracking system [10] which tracks markers on the car. The car is controlled using a DX6i transmitter RC controller [11], which receives commands from the computer using a PCTx cable [12] and sends them to the car. Fig. 2.2 illustrates the experimental setup.

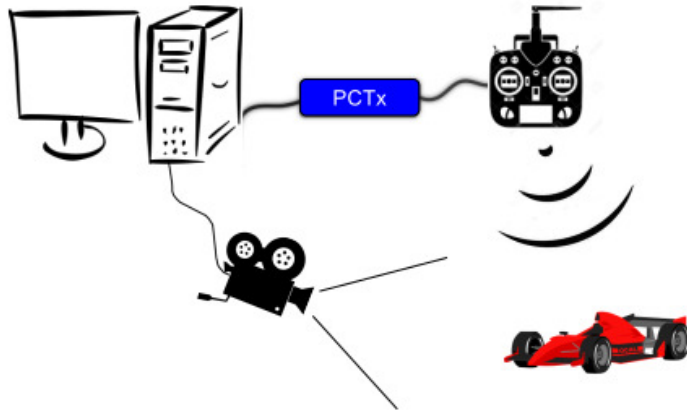


Figure 2.2: The experimental setup

Similarly to a kite system where we don't have control over the kite's speed, the throttle of the car is set to a fixed value.

The implementation of the system is done using ROS. Different scripts run simultaneously and communicate between them using ROS messages.

#### 2.1.1 Steering commands

As described, the car is controlled by sending steering commands to an RC controller. The steering command  $w$  must be a number between 0 and 1023 ( $2^{10}$  values) where a straight motion corresponds to a bias value. Values below 100 or above 900 exceed the physical limits of the steering motor and thus have no effect. The steering commands can be modeled as

$$w = bias + u \tag{2.1}$$

where  $u \in [-u_{max}, u_{max}]$  in order to keep the possible values symmetric to both directions.

For our particular car, the numerical values are  $bias = 485$ ,  $w \in [100, 870]$ ,  $u_{max} = 385$ . The possible values of  $w$  correspond to possible steering angles in  $[-25^\circ, 29^\circ]$ .

## 2.2 State Observer

The camera system provides the car’s current position, orientation (in quaternions form) and timestamp. At the origin, the local coordinates frame of the car is identical to the global frame. From the provided data, the car’s velocity and the yaw, pitch and roll angles must be estimated.

### 2.2.1 Velocity Estimation

Several methods exist to compute the car’s velocity based on its position. One could use a Kalman filter [13] or Finite Differences (FD) [14]. The main issue with both methods is the non-uniformity in time of the position measurements. This non-uniformity limits the FD’s performance and creates extra noise, as can be seen in fig. 2.3. In order to cancel this noise, a non-uniform FD algorithm can be used and even combined with a filter. Fig. 2.4 shows the results of velocity estimation using several different methods. As can be seen, the best result is given by using non-uniform FD combined with a Kalman filter.

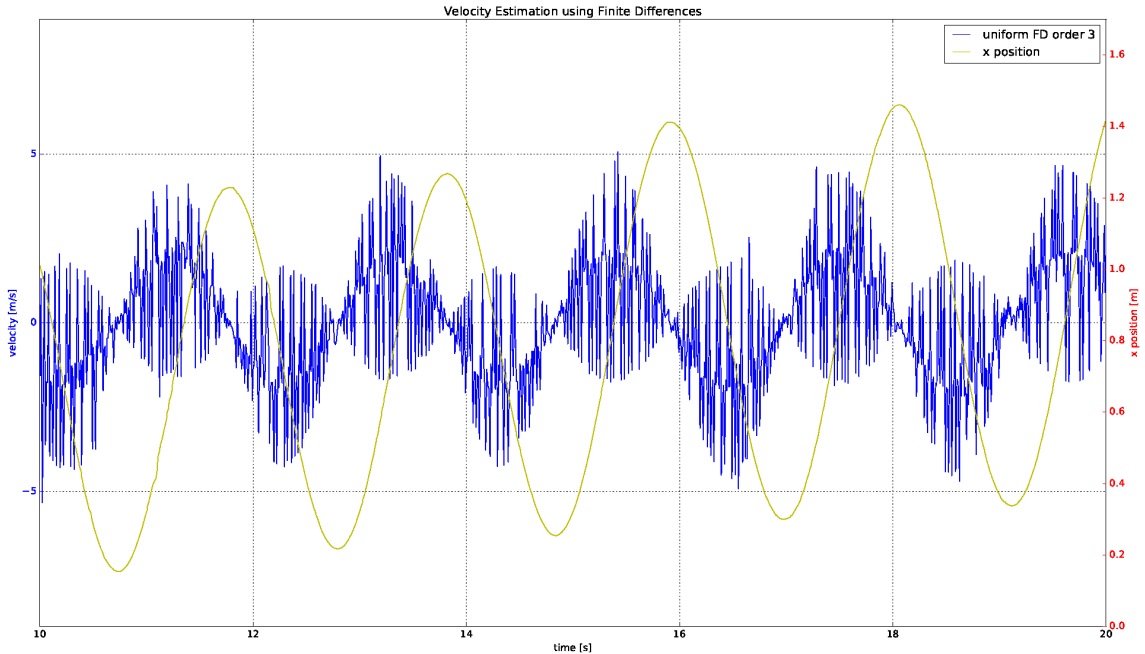


Figure 2.3: Velocity estimation using uniform finite differences

### 2.2.2 Velocity Direction Estimation

In addition to the velocity, we need to estimate the velocity direction, i.e.

$$\gamma = \text{atan2}(v_y, v_x) \quad (2.2)$$

where  $\text{atan2}(\cdot, \cdot)$  is the four quadrant inverse tangent defined over  $]-\pi, \pi]$ .

As the car is limited to planar motion, this quantity can also be approximated by the yaw angle. This method removes the dependency on the estimated velocity, for which the measurement is noisy and delayed (estimation delay due to the use of a Kalman filter). The yaw of the car, as well as the pitch and roll, can be calculated from the quaternions using a function from the ROS Transformation library.

In order to compare the yaw to the actual velocity direction, we also estimate  $\gamma$ . This is done using equ. 2.2, based on  $v_x$  and  $v_y$  obtained as described in the previous subsection. However, as an angle



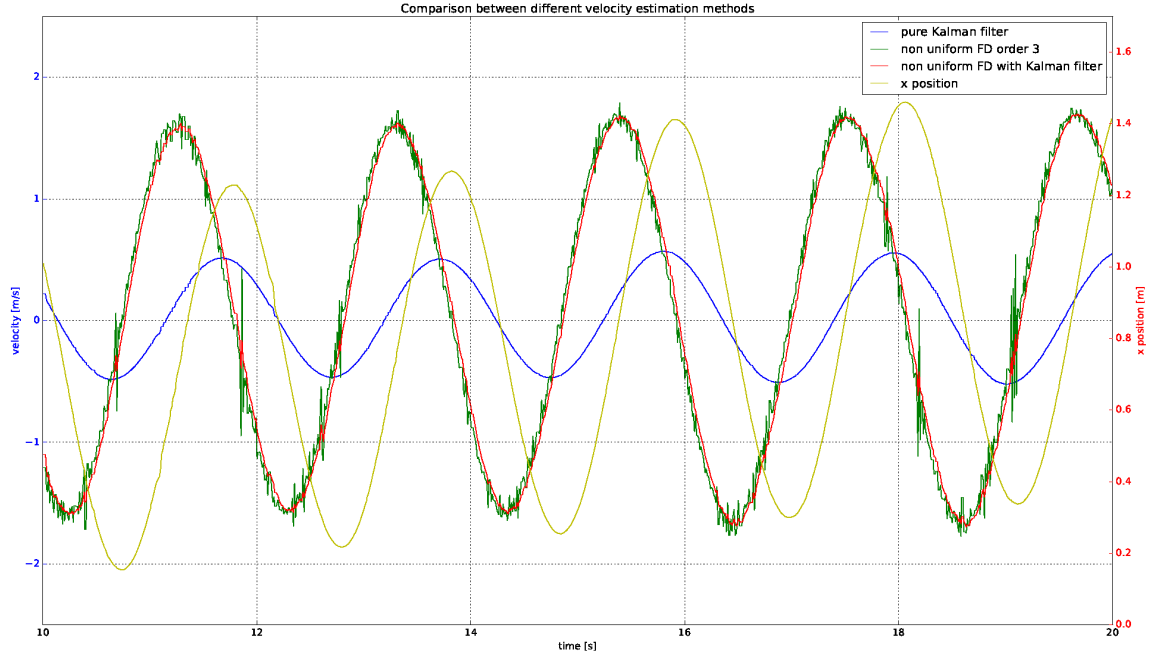


Figure 2.4: Comparison of different velocity estimation methods

$\gamma$  is limited to  $[-\pi, \pi]$ . In order to negate the wrapping effect on the estimation, we use two Kalman filters, for  $\cos(\gamma)$  and  $\sin(\gamma)$ . Fig. 2.5 shows a comparison between different estimations of  $\gamma$ . As can be seen, there is a small delay between the yaw estimation and the  $\gamma$  estimations. The effect of wrapping in the pure Kalman filter can also be seen.

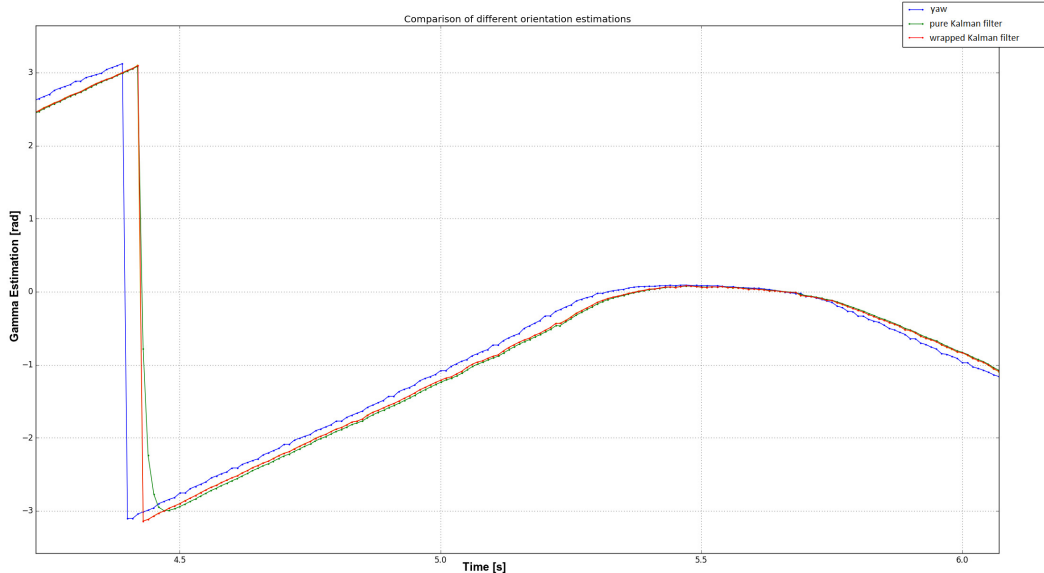


Figure 2.5: Comparison of different  $\gamma$  estimations. Pure Kalman filter (green) uses a single filter while Wrapped Kalman filter (red) uses two separate filters.

### 2.3 Delay Estimation

The first step in the project was to measure the delay between the transmitter and the receiver. As we cannot directly record the commands received by the car's receiver, a second receiver connected to the computer is used. Thus, every command sent by the computer to the car is received by both the car and the second receiver connected to the computer by a USB cable.

The delay in the system can come from several sources. First, the PCTx itself can create delays. Second, the USB buffer between the PCTx and the computer. Finally, another USB buffer is present in the connection between the receiver and the PC. Any delay caused by the latter buffer will only affect the recordings, but not the vehicle.

In order to measure the delay, we run a basic controller commanding the car's velocity while simultaneously logging both the sent commands and the commands received by the receiver. Fig. 2.6a shows the sent and received commands. Fig. 2.6b shows a closer zoom into one of the steps. Note that the sent and received commands use different scales to represent the same commands.

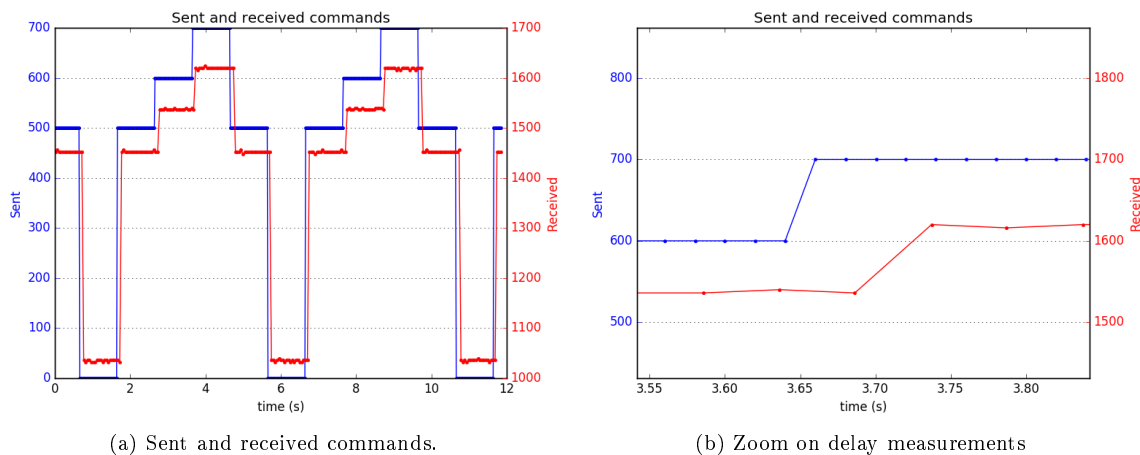


Figure 2.6: Delay Estimation

Measuring the delay at different steps results in a delay between  $80ms$  to  $120ms$ , with an average around  $100ms$ . We estimate that this delay should not be too disruptive to our control scheme.

### 3 Low Level Tracking

Before being able to use an MPC a low level tracker must be implemented. This tracker would command the vehicle using the RC in order to track a reference velocity direction ( $\gamma$ ) given by the MPC.

The tracking is done using a PID controller. However, as  $\gamma$  is an angle, the wrapping problem must be avoided when calculating the error. Thus, instead of using the common formulation of

$$e = \gamma_{current} - \gamma_{ref} \quad (3.1)$$

we use a formulation based on the dot product between the vectors defined by  $\gamma_{current}$  and  $\gamma_{ref}$ , defined as:

$$s = \text{sign}(\sin(\gamma_{current} - \gamma_{ref}))$$

$$e = s * \arccos(\text{clip}(\cos(\gamma_{current}) \cos(\gamma_{ref}) + \sin(\gamma_{current}) \sin(\gamma_{ref}), -1, 1))$$

The dot product is clipped between -1 and 1 to avoid any numerical error. The final error is the angle, or the *arccos* of the dot product.

The standard PID formulation is then used:

$$u = P + T_i I + T_d D$$

$$u_{clipped} = \text{clip}(u, -u_{max}, u_{max})$$

where

$$P = K_p * e, \quad I = \text{clip}\left(\sum e - \frac{\text{controlDiff}}{2}, -30, 30\right), \quad D = e - e_{previous} \quad (3.2)$$

$$\text{controlDiff} = u_{clipped} - u \quad (3.3)$$

Thus, the steering command  $w$  is given by

$$w = u_{clipped} + \text{bias} \quad (3.4)$$

$K_p, T_i, T_d, u_{max}$  and  $\text{bias}$  depend on the platform used. For the car a pure P controller is used. Numerical values can be found in appendix A.

Fig. 3.1 shows the results of using the tracker for a step input.

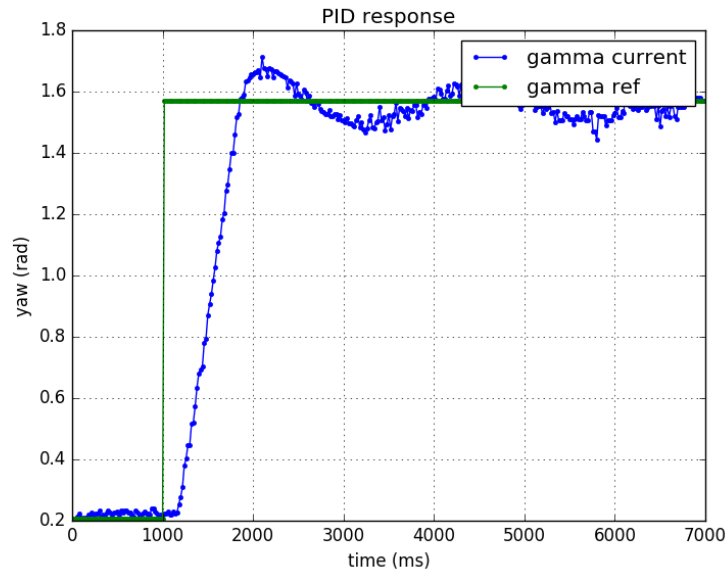


Figure 3.1: Low Level Tracking results for a step input



## 4 Problem Statement and Dynamics

In order to create a path following behavior, a Model Predictive Path-Following Control (MPFC) scheme is used. The main idea of MPFC is to replace the time-parametrized reference trajectory by a geometric reference path. A virtual system is added including a virtual time variable  $\tau$  and a virtual reference point. At any given time, the reference point acts as the tracking reference for the car and can be moved along the path by changing  $\tau$ . The inputs to the real system and the virtual system are computed by receding horizon optimization such that the reference path is followed with minimum tracking error.

We use a system similar to the one described in [9]. In order to define the path following problem for the car, the models for the car system and the virtual system are defined. A tracking error metric is also defined to enable comparison between different MPFC schemes.

### 4.1 Car Model

The car's position is described in a fixed absolute coordinates system of  $(x, y, z)$ . As the car's altitude is constant, the  $z$  component is dropped from the state and is replaced by the direction of the car's velocity, noted  $\gamma$ . By simple geometric considerations,  $\gamma$  can be related to  $x$  and  $y$  by

$$\gamma = \text{atan2}(\dot{y}, \dot{x}) \quad (4.1)$$

where  $\text{atan2}(\cdot, \cdot)$  is the four quadrant inverse tangent defined over  $]-\pi, \pi]$ . The state of the car at time  $t$  and the system dynamics can be described as,

$$q(t) := \begin{pmatrix} x(t) \\ y(t) \\ \gamma(t) \end{pmatrix} \quad (4.2)$$

$$\dot{q} = \begin{pmatrix} v \cos \gamma \\ v \sin \gamma \\ k \cdot u_\gamma \end{pmatrix} \quad (4.3)$$

where  $q \in \mathcal{Q}$ ,  $v$  is the speed of the car and  $u_\gamma$  is an input affecting the steering of the system, as  $\dot{\gamma} = k \cdot u_\gamma$ . In order to imitate a kite in an Airborne Wind Energy system whose velocity cannot be controlled, the same constraint is applied on the car's velocity. The throttle is thus set to a fixed value and isn't controlled by the controller. The car's velocity may only change due to system dynamics, without any external input.

### 4.2 Reference Path

A suitable reference path for the system is a twice continuously differentiable periodic function  $q_{ref}(\tau) : \mathbb{R} \rightarrow \mathcal{Q}$  which verifies the following assumptions:

*Assumption 1:* (Nonholonomicity Constraint)  $\gamma_{ref}$  verifies the nonholonomicity constraint of  $\gamma_{ref} = \text{atan2}(\partial_\tau y_{ref}, \partial_\tau x_{ref})$ .

*Assumption 2:* (Regular Curve) The path is a regular curve such that the reference point doesn't remain stationary while  $\tau$  changes,  $\|q_{ref}(\tau)\| \neq 0 \forall \tau$ .

*Assumption 3:* (Feasibility) The curvature of the reference path is limited such that tracking is feasible under the input constraints given the car's speed.

The chosen reference path is a figure-8 defined as:

$$x_{ref}(\tau) = h + a \sin(2\tau) \quad y_{ref}(\tau) = 4a \cos(\tau) \quad (4.4)$$

and  $\gamma_{ref}$  is computed based on assumption 1.

In order to track the reference path, a virtual system is created where a virtual point is moved along the path by varying  $\tau$  using the input  $u_\tau$ . The virtual system dynamics is

$$\dot{\tau} = u_\tau \quad (4.5)$$

where  $u_\tau \in \mathbb{R}^+$  such that the virtual point can only move in a single direction along the path.

### 4.3 Augmented System

The augmented system is obtained by combining the physical and virtual systems

$$x(t) := (q(t), \tau(t)) \quad (4.6)$$

whose dynamics are given by equ. 4.3 and 4.5. The system's input is given by

$$u(t) := (u_\gamma, u_\tau) \quad (4.7)$$

We also define the path following error at time  $t$  as

$$e(t) = q(t) - q_{ref}(\tau(t)) \quad (4.8)$$

### 4.4 Tracking Error Metric

In order to measure the tracking error and compare different MPFC schemes, a Root Mean Square Error (RMSE) measurement is used. Over  $N = 2$  loops of the reference path, the error between the observed state and the reference are recorded. As the reference point is defined by  $\tau$  and is  $2\pi$ -periodic, we take an interval where  $\tau$  varies by  $2\pi N$ . In order to exclude any transitory effect in the beginning of the test, we start measuring the error at  $\tau = 2\pi$ . The final tracking error is the RMSE value of these errors, such that

$$e_{tracking}(t) = \|q(t) - q_{ref}(\tau(t))\| \quad (4.9)$$

$$E_T = RMSE(e_{tracking}(\tau)) = \sqrt{\mathbb{E}[e(\tau)]} \quad \forall \tau \in [2\pi, 2\pi(N+1)] \quad (4.10)$$

where  $\mathbb{E}[\ ]$  notes the expected value.

## 5 Nominal Model Predictive Path-Following Control

### 5.1 The Model Predictive Path-Following Control Scheme

The first scheme is a nominal MPFC scheme. The Optimal Control Problem (OCP) to be solved is based on the following OCP:

$$\min_{x(\cdot), u(\cdot)} \int_0^T \frac{1}{2} \|e(t)\|_Q^2 + \frac{1}{2} \|u(t)\|_{R_u}^2 ds + \frac{1}{2} \|e(T)\|_{Q_f}^2 \quad (5.1)$$

$$\text{subject to} \quad (5.2)$$

$$\text{dynamics with } x(0) = \hat{x}(t) \quad (5.3)$$

$$u(s) \in \mathcal{U}, x(s) \in \mathcal{X} \quad \forall s \in [0, T] \quad (5.4)$$

$$\frac{1}{2} \|e(T)\|_Q^2 + e(T)^T Q_f \dot{q}(T) \leq 0 \quad (5.5)$$

$$e(T)^T Q_f \dot{q}_{ref}(\tau(T)) \geq 0 \quad (5.6)$$

where  $\mathcal{X} := \mathcal{Q} \times \mathbb{R}$ ,  $\mathcal{U} \in \mathbb{R} \times \mathbb{R}^+$  and  $\hat{x}(t)$  is the observed system state at time  $t$  under the MPFC closed loop controller. Equ. 5.5 and 5.6 are the terminal constraints of the system, used to guarantee convergence. The terminal constraints are derived based on vector field controllers.

The OCP is solved repetitively in a receding horizon fashion at current time  $t$ . As described previously, the low level tracker receives the optimal  $\gamma$  from the MPFC controller and tracks it. In other words, the actual reference to the tracker is

$$\hat{\gamma}(t) = \gamma^*(0) \quad (5.7)$$

This reference is sent to the low level tracker after each successful optimization. The OCP is then solved again using the new measured state as the initial condition. The virtual system evolves according to its dynamics using the optimal  $u_\tau^*$  obtained.

As the implementation cannot use continuous time, the OCP is transformed into a discrete time system. A horizon length of  $N$  is chosen, so the horizon is divided into  $N$  steps of length  $\delta$ . Each step is indexed by  $k = 0, \dots, N$ , such that  $t_k = k\delta$  and  $T = N\delta$ . To predict the next state,  $x_k$  is integrated using a standard order 4 Runge-Kutta (RK4) method (see [15]), assuming a constant input during the integration. Between steps, multiple shooting continuity constraints are applied, such that  $x_{k+1} = RK4(x_k)$ .

**Remark 1:** As the velocity isn't controlled but is part of the dynamics, constant velocity is assumed over the full horizon. The velocity of the car is measured once before solving the OCP.

**Remark 2:** For the nominal MPFC, we assume  $k = 1$  in equ. 4.3.

### 5.2 Additional Constraints and Cost

While theoretically sound, the standard OCP formulated in equ. 5.1 is not enough to work on a real world problem. The experimental setup has many imperfections compared to the simulation. Such imperfections can be noise from sensors and ambient conditions, limitations of the hardware (such as motors and speed limits), inaccuracies, control and estimation delays, communication desynchronization and difficulty to maintain real-time. Due to these imperfections, experiments present more difficulties and require corrections compared to the simulation. The main problem encountered during experiments is the Non-Linear Problem (NLP) solver jumping significantly between different local solutions on different runs. As all MPC formulations assume the solution does not jump from one iteration to another, the problem requires regularization. Fig. 5.1 shows an example of nominal MPFC running on the car. As can be seen in fig. 5.1a,  $u_\gamma$  is extremely jittery, which results in unsmooth  $\gamma$ . As  $\gamma$  is then sent to the low level tracker the result is a unsmooth trajectory with many direction changes (fig. 5.1b)

In order to reduce this effect and improve the performance of the real car, several regularization schemes are added as additional constraints and costs. These are described in the following paragraphs.

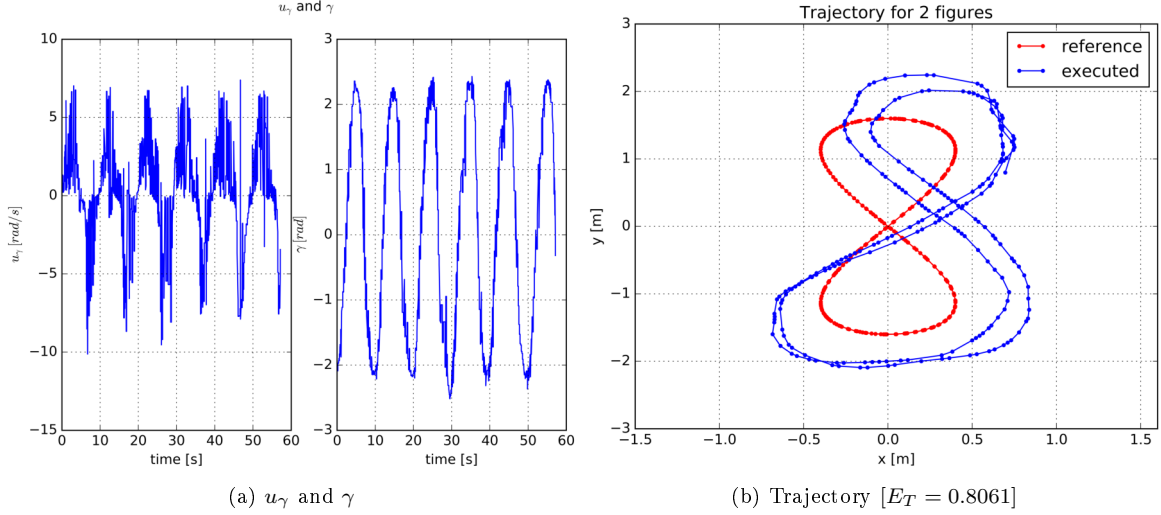


Figure 5.1:  $u_\gamma$  and  $\gamma$  for nominal MPFC (experimental)

### Input constraints

The following input constraints are added to the system:

$$\begin{cases} u_k - u_{k-1} - E_k < \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ u_k - u_{k-1} + E_k > \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{cases} \quad (5.8)$$

where  $E_k$  are slack variables and  $k$  is the step index over the horizon.

By also adding a penalty term on  $E_k$  to the objective function, variations between consecutive inputs over the horizon are penalized. This has the effect of reducing rapid variation in  $u$  and smoothing the control. The penalty term for each slack variable is  $\|E_k\|_{Q_E}^2$ , so the overall cost is increased by  $\sum_k \|E_k\|_{Q_E}^2$ .

### Objective function

In order to avoid a wrapping problem in the error on  $\gamma$  we split the error penalty in equ. 5.1 to three terms, as followed:

$$\|e(t)\|_Q^2 = \left\| \begin{pmatrix} e_x(t) \\ e_y(t) \end{pmatrix} \right\|_{Q_{11,22}}^2 + \|e_{\sin \gamma}(t)\|_{Q_{33}}^2 + \|e_{\cos \gamma}(t)\|_{Q_{33}}^2 \quad (5.9)$$

where  $e_{\sin \gamma}(t) = \sin(\gamma(t)) - \sin(\gamma_{ref}(t))$  and similarly for  $e_{\cos \gamma}$ . A similar modification is applied to  $\|e(T)\|_{Q_f}^2$ .

In addition, the formulation in equ. 5.1 penalizes  $u(t)$  if it is different than 0. In addition,  $u$  could be regularized towards the input found in the previous solution of the MPFC problem, by penalizing the difference between the current input and the previous one. Thus, the term  $\frac{1}{2} \|u_k\|_{R_u}^2$  is replaced by

$$\frac{1}{2} \|u_k\|_{R_u}^2 + \frac{1}{2} \|u_k - u_{reg,k}\|_R^2 \quad (5.10)$$

where  $u_{reg}$  is the control input sequence found in the last solution of the OCP, shifted in time to match the difference in the initial times of the two control sequences.

The numerical values of all the controller's parameters can be found in appendix A.



### 5.3 Experimental Results

Two experiments were performed using the described MPFC scheme. Using the tracking error measurement from equ. 4.9 and 4.10, the corresponding tracking errors are 0.9025 and 0.8605. Fig. 5.2 shows the results of one experiment. Fig. 5.2a shows the executed trajectory and reference path while fig. 5.2b shows the tracking error. As can be seen, the tracking is not perfect and there is quite a big overshoot.

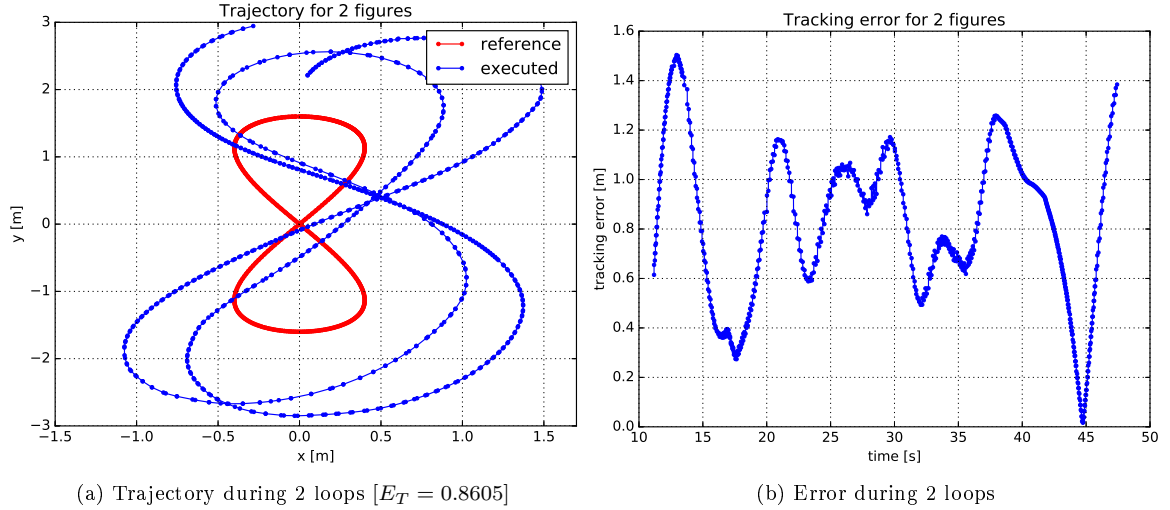


Figure 5.2: Nominal MPFC with regularization (experimental)

As a control, our assumption of constant velocity can be examined. Fig. 5.3a shows the observed velocity of the car during the run. As can be seen, even for a fixed throttle the actual velocity is not constant and the assumption of constant velocity over the horizon is not perfectly verified.

Another point to examine is the performance of the low level tracker. If the tracking is bad, the car would not follow the optimal path planned by the MPFC. Fig. 5.3b shows the low level tracking performance. It can be seen that while the tracking is decent, it is not perfect and there is a slight delay between the reference and the observed  $\gamma$ .

Both the velocity variation and the delay in the low level tracking may contribute to a lower performance of the path-following scheme.

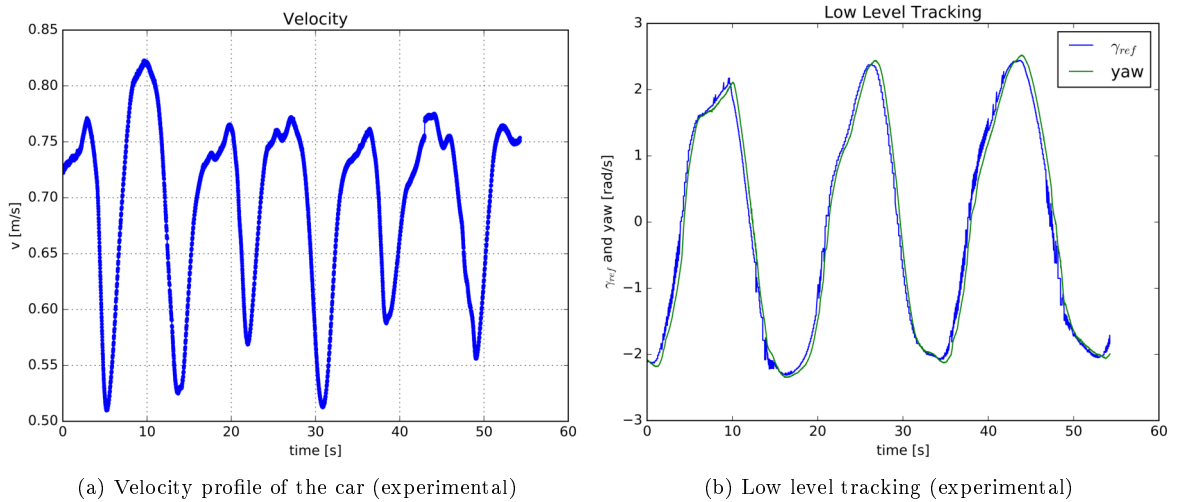


Figure 5.3: Velocity profile and low level tracking for nominal MPFC (experimental)

**Remark:** These results use the numerical values found in appendix A, with the exception of  $Q_E =$

1000. The value presented in the appendix is used throughout the rest of this thesis (see next subsection).

### 5.3.1 Improving Performance by Parameters Tuning

The performance of the nominal MPFC can be improved by tuning the controller's parameters.

#### Slack Variables Cost

As the slack variables  $E_k$  are not part of the theoretical proof on which the algorithm is based ([9]), we start by reducing  $Q_E$ . Fig. 5.4 shows the tracking results for different values of  $Q_E$ . As can be seen, the tracking gets better the smaller  $Q_E$  is and the tracking error decreases. Logically, one would aspire to reduce  $Q_E$  as much as possible. However, setting  $Q_E$  to 0, i.e. ignoring the slack variables, results in a jittery  $\gamma$  as was described previously (and can be seen in fig. 5.1, where the tracking error was 0.8061).

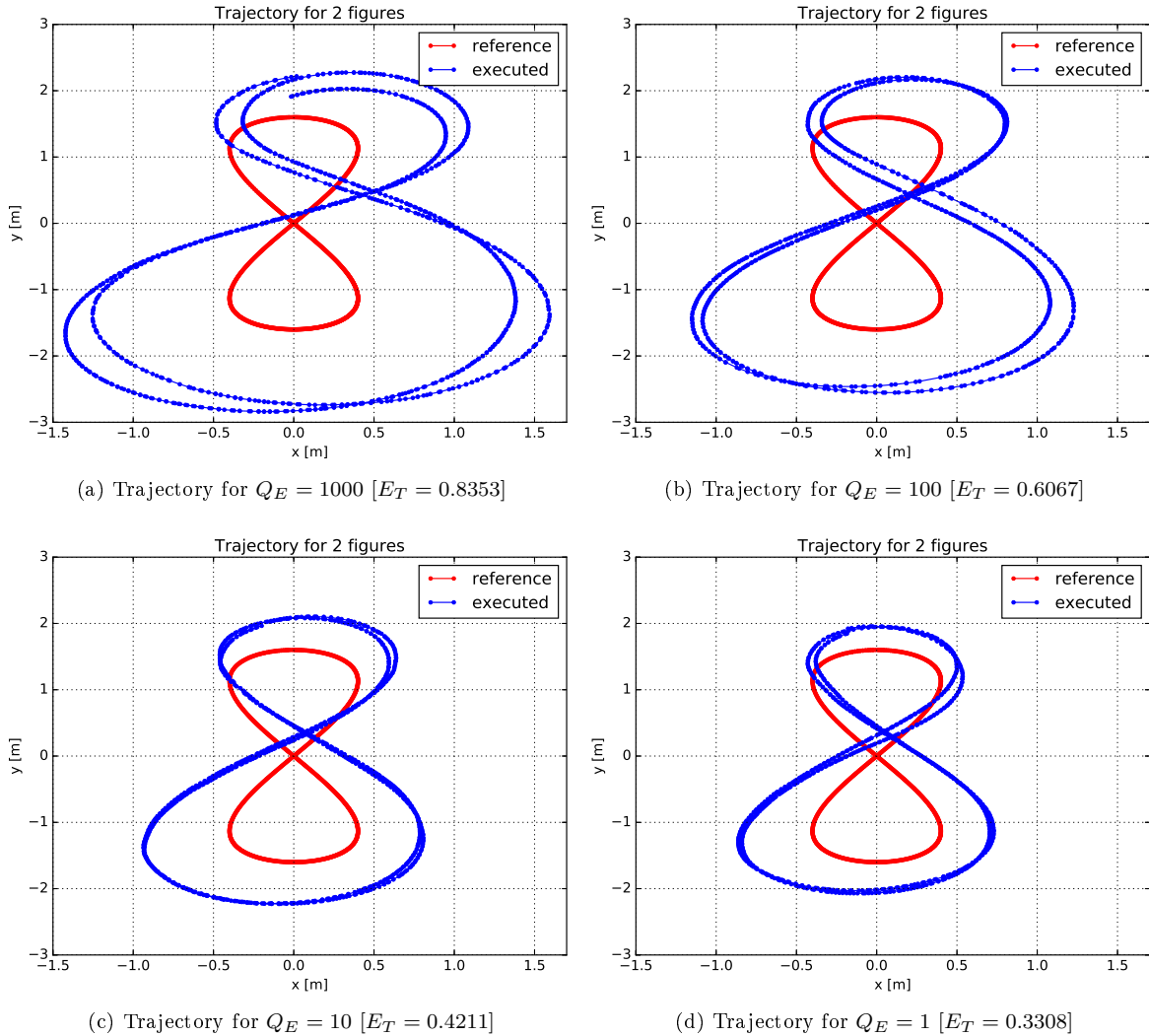


Figure 5.4: Trajectories for different values of  $Q_E$  (experimental)

These results were also confirmed using simulation, with the main difference in the case of  $Q_E = 0$ . In simulation, where noise and imperfections don't exist,  $\gamma$  isn't jittery and thus the slack variables can be removed.

## Tracking Error Cost

Another parameter which can be tuned is the error cost matrix  $Q$ , which penalizes the error between the car's position and the reference path. We are mainly interested in reducing the error in  $x$  and  $y$ , so solely the values of  $Q_{11}$  and  $Q_{22}$  are varied, leaving  $Q_{33}$  to its default value of 1000. Fig. 5.5 shows the results of tracking for different values of  $Q_{11} = Q_{22}$ . For the shown cases  $Q_E$  was fixed to 1. As can be seen, while the highest penalty results in the lowest tracking error, the effect is minor. The original value of 1000 is thus chosen for the rest of this thesis.

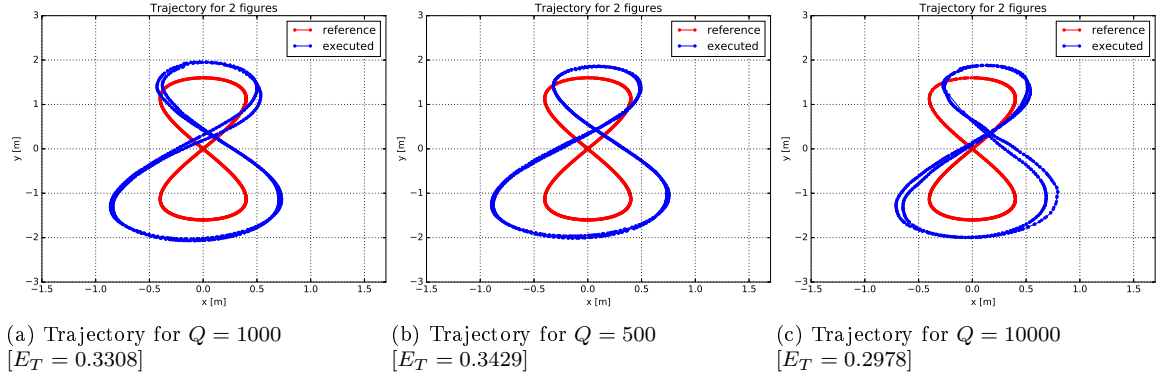


Figure 5.5: Trajectories for different values of  $Q$  (experimental)

## 5.4 Summary

In this section, we have shown the first of several model predictive path-following control schemes presented in this thesis. The OCP used in an MPFC to create path-following was formulated and explained. Central to this section are the challenges confronting us when working on a real world problem, compared to computer simulations. Additional terms and constraints must be added in order to solve these challenges. Especially essential are the regularization methods without which the MPC solver tends to jumps between local optimal solutions, creating an unsmooth trajectory.

We then presented the first results using the proposed nominal MPFC scheme. In addition to tracking results, we have shown the evolution of the error during the experiment. Two imperfections in the system were then presented. First, the velocity profile of the car was examined, proved to vary during the experiment even when using a fixed throttle value. This effect is not taken into account in the MPC controller, as the velocity over the horizon is assumed to be constant. Second, the performance of the low level tracker was investigated. While its performance was decent, a small delay was noticed. Both these imperfections can harm the path-following.

Finally, we explored the effect of tuning the controller's parameters. Two parameters were tuned - the cost matrix penalizing the slack variables in the regularization term and the cost matrix penalizing the tracking error. The slack variables cost was proven to have a huge effect on the tracking of the reference path. In fact, reducing this cost and thus reducing the regularization improved the tracking profoundly. However, completely removing the slack variables and discarding the regularization caused the issue of local optimal solutions to reappear. The second parameters tuned, the penalty on the tracking error, was proven to have only a very minor effect on the tracking.



## 6 Adaptation algorithm - Recursive Least Squares

In order to improve the MPFC tracking results, we need a method to better estimate the parameters of the system's dynamics. Using a suitable adaptation algorithm, an adaptive MPFC can be created, where the parameters' estimates are constantly updated.

For the adaptation algorithm, we have chosen to use Recursive Least Squares (RLS) - a recursive algorithm based on the Robbins-Monro algorithm [16, 17], where the parameters' update rule is defined based on a chosen auxiliary function  $\Phi(\hat{p}, y)$ , as

$$\hat{p}_{k+1} = \hat{p}_k - \mu_k \Phi(\hat{p}_k, y_k) \quad (6.1)$$

As our goal is to minimize the estimation error,  $\Phi$  is chosen following the gradient descent method. Thus  $\Phi(\hat{p}, y) = \frac{\partial J(\hat{p}, y)}{\partial \hat{p}}$ , where  $J$  is a cost function based on the error between the predicted and observed systems. Using such cost function would cause the algorithm to adapt the parameters' estimates in order to reduce the error between the predicted state and the observed state. The resulting cost function and update rule are

$$J(\hat{p}, y_k) = \frac{1}{2} \|y_{k+1} - \hat{y}_{k+1}\|^2 \quad (6.2)$$

$$\hat{p}_{k+1} = \hat{p}_k - \mu_k \left. \frac{\partial J(\hat{p}, y)}{\partial \hat{p}} \right|_{\hat{p}_k, y_k} \quad (6.3)$$

where  $y$  is the observed state,  $\hat{y}$  is the predicted state,  $\hat{p}$  contains the estimated parameters and  $\mu_k$  is the learning rate. Notice that  $J$  is a scalar.

There are two possible representations for the state, both of which can be used in the RLS derivation. In the following sections, we derive the update rule for both representations for a general state  $y$ . Chapter 7 will detail the algorithm for the car's system.

**Remark 1:** The following derivations use the Numerator convention for matrix calculus.

**Remark 2:** The following derivations use the notation  $p_k$  to represent the true parameters at iteration  $k$ . In the case where the parameters are constant  $p_k = p$ .

### 6.1 Representation I

The first representation for  $y$  is:

$$y_{k+1} = f(y_k) + g(y_k)p \quad (6.4)$$

where  $y \in \mathbb{R}^{n \times 1}$ ,  $p \in \mathbb{R}^{m \times 1}$ ,  $g(y) \in \mathbb{R}^{n \times m}$ . By the Certainty Equivalence (C.E.), the predicted state is

$$\hat{y}_{k+1} = f(y_k) + g(y_k)\hat{p}_k \quad (6.5)$$

where  $\hat{p}$  contains the estimated parameters.

**Proposition 1** (Representation I update rule). *The update rule for the RLS algorithm (equ. 6.3) using the representation in 6.4 is given by*

$$\hat{p}_{k+1} = \hat{p}_k + \mu_k \left( (y_{k+1} - \hat{y}_{k+1})^T \left. \frac{\partial \hat{y}}{\partial \hat{p}} \right|_{\hat{p}_k, y_k} \right)^T \quad (6.6)$$

*Proof.* For brevity, we note  $g(y)$  as  $g$  in the following.

The cost function defined in equ. 6.2 becomes

$$J(\hat{p}_k, y_k) = \frac{1}{2} \|y_{k+1} - \hat{y}_{k+1}\|^2 = \frac{1}{2} \|y_{k+1} - f(y_k) - g_k \hat{p}_k\|^2 \quad (6.7)$$

We derive  $J(\hat{p}_k, y_k)$  w.r.t  $\hat{p}_k$ :

$$\begin{aligned}
\frac{\partial J(\hat{p}_k, y_k)}{\partial \hat{p}_k} &= (y_{k+1} - f(y_k) - g_k \hat{p}_k)^T (-g_k) \\
&= - (y_{k+1}^T - (f(y_k) - g_k \hat{p}_k)^T) g_k \\
&= -(y_{k+1}^T - \hat{y}_{k+1}^T) g_k \\
&= -(y_{k+1} - \hat{y}_{k+1})^T \frac{\partial \hat{y}}{\partial \hat{p}} \Big|_{\hat{p}_k, y_k}
\end{aligned} \tag{6.8}$$

Due to the use of the numerator convention, the resulting gradient is a row vector. Thus its transpose is used in the update rule defined in equ. 6.3:

$$\hat{p}_{k+1} = \hat{p}_k - \mu_k \frac{\partial J(\hat{p}, y)}{\partial \hat{p}} \Big|_{\hat{p}_k, y_k}^T = \hat{p}_k + \mu_k \left( (y_{k+1} - \hat{y}_{k+1})^T \frac{\partial \hat{y}}{\partial \hat{p}} \Big|_{\hat{p}_k, y_k} \right)^T \quad \blacksquare$$

## 6.2 Representation II

The previous representation doesn't allow us to derive a convergence condition. For this reason, we use a second representation, in which  $p$  is a matrix whose diagonal elements are the estimated parameters, such that:

$$y_{k+1} = f(y_k) + pg(y_k) \tag{6.9}$$

where  $y \in \mathbb{R}^{n \times 1}$ ,  $p \in \mathbb{R}^{n \times m}$ ,  $g(y) \in \mathbb{R}^{m \times 1}$ . By the Certainty Equivalence (C.E.),

$$\hat{y}_{k+1} = f(y_k) + \hat{p}_k g(y_k) \tag{6.10}$$

where  $\hat{p}$  contains the estimated parameters.

**Proposition 2** (Representation II update rule). *For the representation in 6.9, the RLS update rule is given by*

$$\hat{p}_{k+1} = \hat{p}_k + \mu_k (g_k (y_{k+1} - \hat{y}_{k+1})^T)^T \tag{6.11}$$

*Proof.* Again, we note  $g(y)$  as  $g$ . The cost function defined in equ. 6.2 becomes

$$\begin{aligned}
J(\hat{p}_k, y_k) &= \frac{1}{2} \|y_{k+1} - \hat{y}_{k+1}\|^2 = \frac{1}{2} (y_{k+1}^T - \hat{y}_{k+1}^T) (y_{k+1} - \hat{y}_{k+1}) \\
&= \frac{1}{2} (g_k^T (p_k - \hat{p}_k)^T) ((p_k - \hat{p}_k) g_k) \\
&= \frac{1}{2} g_k^T \tilde{p}_k^T \tilde{p}_k g_k
\end{aligned} \tag{6.12}$$

using the differential notation  $\tilde{y}_k = y_k - \hat{y}_k$ ,  $\tilde{p}_k = p_k - \hat{p}_k$ . As  $J(\hat{p}_k, y_k) \in \mathbb{R}$ , the trace operator can be applied:

$$J(\hat{p}_k, y_k) = \frac{1}{2} \text{tr}(g_k^T \tilde{p}_k^T \tilde{p}_k g_k) = \frac{1}{2} \text{tr}(\tilde{p}_k^T \tilde{p}_k g_k g_k^T) \tag{6.13}$$

We can then use the formula for trace differentiation  $\frac{\partial \text{tr}(AXBX^T C)}{\partial X} = BX^T C A + B^T X^T A^T C$ , with  $A = C = I$ ,  $B = B^T = g_k g_k^T$ :

$$\frac{\partial J(\hat{p}_k, y_k)}{\partial \tilde{p}_k} = \frac{1}{2} (g_k g_k^T \tilde{p}_k^T + g_k g_k^T \tilde{p}_k^T) = g_k g_k^T \tilde{p}_k^T = g_k (\tilde{p}_k g_k)^T = g_k (y_{k+1} - \hat{y}_{k+1})^T \tag{6.14}$$

$$\frac{\partial J}{\partial \hat{p}} = \frac{\partial J}{\partial \tilde{p}} \frac{\partial \tilde{p}}{\partial \hat{p}} = -\frac{\partial J}{\partial \tilde{p}} \tag{6.15}$$

Combining equ. 6.14 and 6.15 and transposing due to the numerator convention, we get the update rule:

$$\hat{p}_{k+1} = \hat{p}_k - \mu_k \frac{\partial J(\hat{p}, y)}{\partial \hat{p}} \Big|_{\hat{p}_k, y_k}^T = \hat{p}_k + \mu_k (g_k (y_{k+1} - \hat{y}_{k+1})^T)^T \quad \blacksquare$$

### 6.2.1 Sufficient Condition for Convergence

In order to guarantee the convergence of the estimated parameters using the RLS algorithm, the convergence of the adaptation system must be proven. A sufficient condition for convergence can be then obtained.

**Theorem 1** (Sufficient condition for convergence). *Consider the system representation in 6.9 and an adaptation algorithm given by the update rule in 6.11. Suppose the true value of the parameter  $p$  is constant. Suppose a learning rate  $\mu_k$  satisfying*

$$\mu_k < \frac{2}{g_k^T g_k} \quad (6.16)$$

*The parameters' estimates given by the adaptation algorithm are Lyapunov stable and thus converge to a fixed value.*

*Proof.* In order to guarantee stability (and thus convergence), a Lyapunov function for the system must be found. Consider the following function

$$V(\hat{p}_k) = \frac{1}{2} \text{tr} \left( (p_k - \hat{p}_k)(p_k - \hat{p}_k)^T \right) = \frac{1}{2} \text{tr} \left( \tilde{p}_k \tilde{p}_k^T \right) \quad (6.17)$$

In order to show that this function is indeed a Lyapunov function, the following conditions must be verified:

- $\|\hat{p}\| \rightarrow \infty \Rightarrow V(\hat{p}) \rightarrow \infty$
- $V(0) = 0$  and  $V(\hat{p}) > 0 \forall \hat{p} \neq 0$
- $V(\hat{p}_{k+1}) - V(\hat{p}_k) < 0 \forall \hat{p} \neq 0$

The first two conditions are easily verified based on the definition of  $V$ . For the third, the following proof applies

$$V(\hat{p}_{k+1}) - V(\hat{p}_k) = \frac{1}{2} \text{tr} \left( \tilde{p}_{k+1} \tilde{p}_{k+1}^T \right) - \frac{1}{2} \text{tr} \left( \tilde{p}_k \tilde{p}_k^T \right) \quad (6.18)$$

$$\begin{aligned} \tilde{p} = p - \hat{p} &\Rightarrow \tilde{p}_{k+1} = p_{k+1} - \hat{p}_{k+1} = p_{k+1} - \left( \hat{p}_k + \mu_k (g_k \tilde{y}_{k+1}^T)^T \right) \\ &\Leftrightarrow \tilde{p}_{k+1} = \tilde{p}_k - \mu_k \tilde{y}_{k+1} g_k^T \end{aligned} \quad (6.19)$$

Combining equ. 6.18 and 6.19

$$\begin{aligned} V(\hat{p}_{k+1}) - V(\hat{p}_k) &= \frac{1}{2} \text{tr} \left( (\tilde{p}_k - \mu_k \tilde{y}_{k+1} g_k^T) (\tilde{p}_k - \mu_k \tilde{y}_{k+1} g_k^T)^T \right) - \frac{1}{2} \text{tr} \left( \tilde{p}_k \tilde{p}_k^T \right) \\ &= \frac{1}{2} \text{tr} \left( -2\mu_k \tilde{p}_k g_k \tilde{y}_{k+1}^T + \mu_k^2 \tilde{y}_{k+1} g_k^T g_k \tilde{y}_{k+1}^T \right) \\ &= \text{tr} \left( \mu_k \left( -\tilde{y}_{k+1} \tilde{y}_{k+1}^T + \frac{\mu_k}{2} \tilde{y}_{k+1} g_k^T g_k \tilde{y}_{k+1}^T \right) \right) \\ &= \mu_k \text{tr} \left( \tilde{y}_{k+1} \left( -I + \frac{\mu_k}{2} g_k^T g_k \right) \tilde{y}_{k+1}^T \right) \end{aligned} \quad (6.20)$$

Notice that the inner term belongs to  $\mathbb{R}$ . Thus, we can use the trace property and get

$$V(\hat{p}_{k+1}) - V(\hat{p}_k) = \mu_k \text{tr} \left( \tilde{y}_{k+1}^T \left( -I + \frac{\mu_k}{2} g_k^T g_k \right) \tilde{y}_{k+1} \right) \quad (6.21)$$

and finally, as the resulting expression is a real value,

$$V(\hat{p}_{k+1}) - V(\hat{p}_k) = \mu_k \tilde{y}_{k+1}^T \left( -1 + \frac{\mu_k}{2} g_k^T g_k \right) \tilde{y}_{k+1} \quad (6.22)$$

The third condition thus becomes

$$V(\hat{p}_{k+1}) - V(\hat{p}_k) < 0 \Leftrightarrow -1 + \frac{\mu_k}{2} g_k^T g_k < 0 \Leftrightarrow \mu_k < \frac{2}{g_k^T g_k} \quad (6.23)$$

A sufficient condition for convergence for the RLS algorithm is thus

$$\mu_k < \frac{2}{g_k^T g_k} \quad \blacksquare$$

### 6.3 Filter RLS

In order to improve the memoryless RLS algorithm, a filter RLS can be derived, which that takes into account the previous measurements over a window of a certain length  $N$ . We derive this algorithm for representation II, but a similar method would produce the formulation for representation I.

**Proposition 3** (Filter RLS update rule). *The update rule for the system representation 6.9 for a filter RLS is given by*

$$\hat{p}_{k+1} = \hat{p}_k + \mu_k [R_k + g_k (y_{k+1}^T - f(y_k)^T) - (M_k + g_k g_k^T) \hat{p}_k]^T \quad (6.24)$$

where the recursive update rules for  $R$  and  $M$  are

$$R_{k+1} = R_k + g_k (y_{k+1}^T - f(y_k)^T), \quad M_{k+1} = M_k + g_k g_k^T \quad (6.25)$$

*Proof.* Recall that for the memoryless RLS the cost function was given by

$$J(\hat{p}_k, y_k) = \frac{1}{2} \|y_{k+1} - \hat{y}_{k+1}\|^2 = \frac{1}{2} \|y_{k+1} - f(y_k) + \hat{p}_k g(y_k)\|^2 \quad (6.26)$$

For the filter RLS a similar cost function is used, but over a window of previous states, such that

$$J(\hat{p}_k, y) = \frac{1}{2} \sum_{i=k-N}^k \|y_{i+1} - f(y_i) + \hat{p}_k g(y_i)\|^2 \quad (6.27)$$

Similarly to the memoryless RLS proof,

$$\begin{aligned} J(\hat{p}_k, y) &= \frac{1}{2} \sum_{i=k-N}^k \|y_{i+1} - f(y_i) + \hat{p}_k g_i\|^2 \\ &= \frac{1}{2} \sum_{i=k-N}^k (g_i^T (p_i - \hat{p}_k)^T) ((p_i - \hat{p}_k) g_i) \\ &= \frac{1}{2} \sum_{i=k-N}^k g_i^T \tilde{p}_{ik}^T \tilde{p}_{ik} g_i \end{aligned} \quad (6.28)$$

using the differential notation  $\tilde{p}_{ik} = p_i - \hat{p}_k$ . As  $J(\hat{p}_k, y_k) \in \mathbb{R}$ , the trace operator can be used

$$J(\hat{p}_k, y_k) = \frac{1}{2} \sum_{i=k-N}^k \text{tr}(\tilde{p}_{ik}^T \tilde{p}_{ik} g_i g_i^T) \quad (6.29)$$

$$\frac{\partial J(\hat{p}_k, y)}{\partial \tilde{p}_{ik}} = \sum_{i=k-N}^k g_i g_i^T \tilde{p}_{ik}^T = \sum_{i=k-N}^k g_k (\tilde{p}_{ik} g_i)^T = \sum_{i=k-N}^k g_i (y_{i+1} - f(y_i) - \hat{p}_k g_i)^T \quad (6.30)$$

$$\frac{\partial J}{\partial \tilde{p}} = \frac{\partial J}{\partial \tilde{p}} \frac{\partial \tilde{p}}{\partial \hat{p}} = - \frac{\partial J}{\partial \hat{p}} \quad (6.31)$$

Combining equ. 6.30 and 6.31 and transposing due to the numerator convention, the update rule is obtained

$$\begin{aligned} \hat{p}_{k+1} &= \hat{p}_k - \mu_k \left. \frac{\partial J(\hat{p}, y)}{\partial \hat{p}} \right|_{\hat{p}_k, y}^T \\ &= \hat{p}_k + \mu_k \left[ \sum_{i=k-N}^k (g_i (y_{i+1}^T - f(y_i)^T) - g_i g_i^T \hat{p}_k^T) \right]^T \\ &= \hat{p}_k + \mu_k \left[ \sum_{i=k-N}^{k-1} g_i (y_{i+1}^T - f(y_i)^T) + g_k (y_{k+1}^T - f(y_k)^T) - \left( \sum_{i=k-N}^{k-1} g_i g_i^T + g_k g_k^T \right) \hat{p}_k^T \right]^T \end{aligned} \quad (6.32)$$



where the terms of the past iterations and the term of the current iteration have been separated. This results in a recursive algorithm, where

$$\hat{p}_{k+1} = \hat{p}_k + \mu_k [R_k + g_k (y_{k+1}^T - f(y_k)^T) - (M_k + g_k g_k^T) \hat{p}_k]^T \quad (6.33)$$

with

$$R_k = \sum_{i=k-N}^{k-1} g_i (y_{i+1}^T - f(y_i)^T), \quad M_k = \sum_{i=k-N}^{k-1} g_i g_i^T \quad (6.34)$$

and their corresponding update rules

$$R_{k+1} = R_k + g_k (y_{k+1}^T - f(y_k)^T), \quad M_{k+1} = M_k + g_k g_k^T \quad \blacksquare$$

## 6.4 Summary

This section presented the derivation of the recursive least squared algorithm. The basic idea and principles of the algorithm were presented. The algorithm was then derived for two different representations of the state and the corresponding update rules were obtained. For representation II, a stability theorem was proven, concluding in a sufficient condition on the learning rate to guarantee the stability of the parameters' estimates. Lastly, a filter RLS algorithm was derived, utilizing a window of past measurements in order to better estimate the parameters.



## 7 Adaptive MPFC using Recursive Least Squares

### 7.1 Motivation

Using the algorithms derived in section 6, an adaptive MPFC scheme can be implemented. In adaptive control, the controller can be viewed as an adjustable controller, coupled with an adaptation mechanism. The adaptation mechanism updates the estimates the system’s parameters in real time. When using direct adaptive control the adaptation mechanism updates the parameters based on the system’s output and a reference model. Using RLS, the reference model is replaced by an adjustable predictor which predicts the system’s output using the current parameters’ estimates. The error between the prediction and the real system’s output is fed into the adaptation algorithm. This scheme is a particular form of direct adaptive control, sometimes called “implicit model reference adaptive control”. By using an adaptation mechanism, the parameters’ estimates are improved. This reduces plant-model mismatch and should improve the controller’s performance.

More details about adaptive control can be found in [18] (in particular chapter 1.3.4).

In the implementation, the previously derived nominal MPFC controller is an adjustable controller. An adaptation script performs both the prediction and the adaptation. It receives the necessary information - applied input, observed state and state’s derivative and recursively updates the parameters’ estimations starting from an initial guess. It then sends the estimated parameters to the MPC controller, which uses them for solving the OCP problem described in equ. 5.1 - 5.6. The corresponding control scheme is illustrated in fig. 7.1.

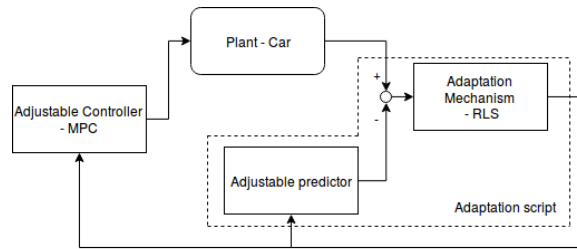


Figure 7.1: Adaptive control scheme

Apart from improving the tracking, a big advantage of the adaptive scheme is the ability to learn the relation between the dynamics and the input. Previously, the input  $u_\gamma$  was chosen by the solver based on  $\dot{\gamma} = 1 \cdot u_\gamma$ , without any control on the nature of  $u_\gamma$ . Using the adaptive scheme, we can now feed in any type of input and learn the correct correlation between it and the system’s dynamics. For example, we can impose  $u_\gamma = \textit{steering command}$  (output of the low level tracker) and learn the parameter  $k$  in the model  $\dot{\gamma} = k \cdot u_\gamma$  (or any other model). Done correctly, the MPC controller would find the optimal steering command to apply and thus the low level tracker would become obsolete.

In addition to these advantages, this section is also a real-world test to the results presented in [19], where the same type of adaptive MPFC scheme was derived and tested in simulation. We use this opportunity to test their results applied to our system on both simulation and hardware.

### 7.2 Recursive Least Squares for the Car’s Dynamical System

In section 6 the RLS algorithm was derived for general dynamics. In order to apply it to the car, the general terms must be replaced by the system’s dynamics.

For both possible representations, the state used in the RLS algorithm (previously  $y$ ) is the system’s dynamics  $\dot{q}$  (as defined in equ. 4.3). In other words, by using a model for  $\dot{q}$ , the next  $\dot{q}$  can be predicted based on the estimated parameters and can then be compared to the next observed  $\dot{q}$ . Thus, in all the equations from section 6,  $y$  is replaced by  $\dot{q}$  and  $\hat{y}$  by  $\hat{\dot{q}}$ . In addition, as described in equ. 4.3,  $\dot{q}$  doesn’t depend on itself. The term  $f(y_k)$  in the RLS equation is thus equal to zero.

The parameters in the car’s system are the velocity  $v$  and the factor  $k$  between  $\dot{\gamma}$  and  $u_\gamma$ .

The two remaining variables to define,  $p$  and  $g(y)$ , depend on the representation and are described in the following subsections.

### 7.2.1 Representation I

Recall that representation I defines  $y$  as

$$y_{k+1} = f(y_k) + g(y_k)p \quad (7.1)$$

As described previously, this representation becomes

$$\dot{q}_{k+1} = g(\dot{q}_k)p \quad (7.2)$$

Comparing this equation with equ. 4.3, we get

$$p = \begin{pmatrix} v \\ k \end{pmatrix} \quad g = \begin{pmatrix} \cos \gamma & 0 \\ \sin \gamma & 0 \\ 0 & u_\gamma \end{pmatrix} \quad (7.3)$$

### 7.2.2 Representation II

Representation II defines  $p$  as a matrix instead of a column vector.  $y$  is thus defined as

$$y_{k+1} = f(y_k) + pg(y_k) \quad (7.4)$$

which becomes

$$\dot{q}_{k+1} = pg(\dot{q}_k) \quad (7.5)$$

By identifying  $g$  and  $p$  based on the dynamics, we find

$$p = \begin{pmatrix} v & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & k \end{pmatrix} \quad g = \begin{pmatrix} \cos \gamma \\ \sin \gamma \\ u_\gamma \end{pmatrix} \quad (7.6)$$

Using equ. 6.16, the sufficient condition for convergence is  $\mu_k < \frac{2}{1+u_\gamma^2} \forall u_\gamma^2 \Rightarrow \mu_k < \frac{2}{1+\max(u_\gamma^2)} = \frac{2}{1+1} = 1$  (see next subsection for  $\max(u_\gamma^2)$ ).

**Remark:** This representation uses  $p \in \mathbb{R}^{3 \times 3}$  while there are only two parameters and thus over-parametrizes the system. One could, if wished to, estimate different velocities for  $\dot{x}$  and  $\dot{y}$ , such that

$$\dot{q} = \begin{pmatrix} v_x \cos \gamma \\ v_y \sin \gamma \\ k \cdot u_\gamma \end{pmatrix} \quad (7.7)$$

However, this is out of the scope of this project and we impose  $v_x = v_y$ .

**Notation:**  $v$  is also noted as  $v_0$ .

### 7.2.3 Replacing the Low Level Tracker

In order to use the adaptive MPFC to replace the PID controller, one more detail must be added. As described before, the steering command from the PID is used to train the adaptation algorithm. However, the steering command is a number between 100 and 870 where a bias term of 485 corresponds to zero steering (see section 2.1.1). Using the steering command directly for the adaptation would require a very low learning rate  $\mu_k$  in order to satisfy the sufficient convergence condition (equ. 6.16). Thus, an affine transformation is used to normalize the input to  $[-1, 1]$ , such that

$$u_\gamma = \frac{w - \bar{w}}{\sigma_w} \quad (7.8)$$

where  $w$  is the steering command,  $\bar{w}$  is the mean value of the commands and is equal to the bias, and  $\sigma_w$  is equal to half the possible interval of  $w$ . The resulting  $u_\gamma$  is then used in the adaptation script, as well as in the MPC controller. Once an optimal  $u$  is found by the MPC, it is transformed to a steering command using the inverse transformation

$$w = u_\gamma \sigma_w + \bar{w} \quad (7.9)$$

## 7.3 Simulation Results

### 7.3.1 Simulation Setup

As representation II has a sufficient condition for convergence, it is used for the implementation of the RLS algorithm. The adaptive MPFC scheme is first tested in simulation. The simulation uses a known system model in order to move the car at every time step. These system dynamics are as described previously in equ. 4.3, but the values of  $v$  and  $k$  are fixed to known “true” values. For all following results, we set  $v = 0.5 \text{ m/s}$  and  $k = 9.6$ .

### 7.3.2 Model $\dot{\gamma} = k \cdot u_\gamma$

The simulation results for the adaptive MPFC are shown in fig. 7.3 and 7.2. These results were obtained without a PID controller, the steering commands being generated following equ. 7.9. Two learning rates were tested -  $\mu_k = 1$  (fig. 7.2) which verifies the sufficient condition and  $\mu_k = 10$  (fig. 7.3).

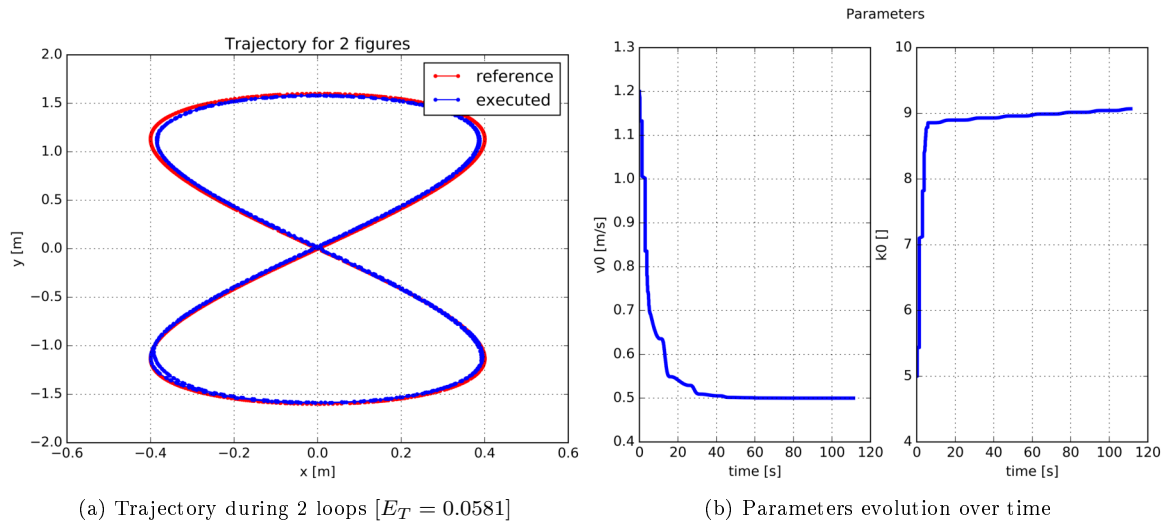


Figure 7.2: Adaptive MPFC with  $\mu_k = 1$  for  $\dot{\gamma} = k \cdot u_\gamma$  (simulation)

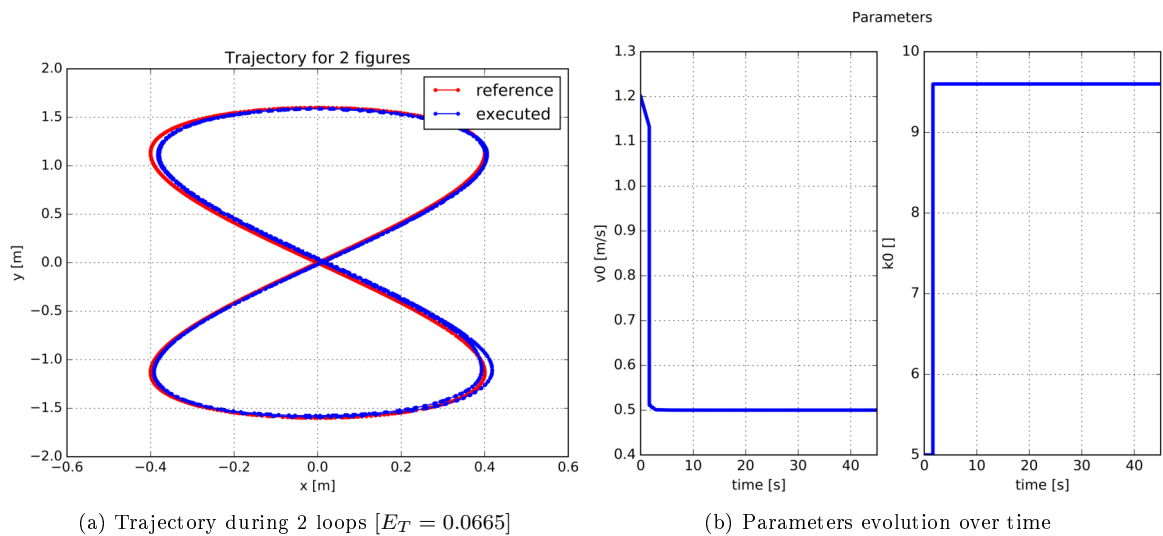


Figure 7.3: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k \cdot u_\gamma$  (simulation)

As can be seen, the parameters estimates converge to the true values of the parameters quickly, even though the convergence for  $\mu_k = 10$  is not guaranteed. With good estimates, the tracking is very good and the tracking errors are very low. Note that the error is measured only after the initial transitory period, so the convergence speed doesn't affect the total tracking error.

### 7.3.3 Model $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$

As the real car's dynamics are unknown, it is interesting to explore another type of model for  $\dot{\gamma}$ . In this model, a bias term is added to  $\dot{\gamma}$ . The adaptation script thus uses the following model

$$\dot{\gamma} = k_0 + k_1 \cdot u_\gamma \quad (7.10)$$

Note that the real system is not affected by the choice to model  $\dot{\gamma}$  differently. Thus, when testing on simulation, the car is still simulated as the same dynamic system described in section 7.3.1.

For this model and using representation II, the algorithm variables are

$$p = \begin{pmatrix} v & 0 & 0 & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & k_0 & k_1 \end{pmatrix} \quad g = \begin{pmatrix} \cos \gamma \\ \sin \gamma \\ 1 \\ u_\gamma \end{pmatrix} \quad (7.11)$$

The algorithm is tested again using two learning rates -  $\mu_k = 2$  (fig. 7.4) and  $\mu_k = 10$  (fig. 7.5).

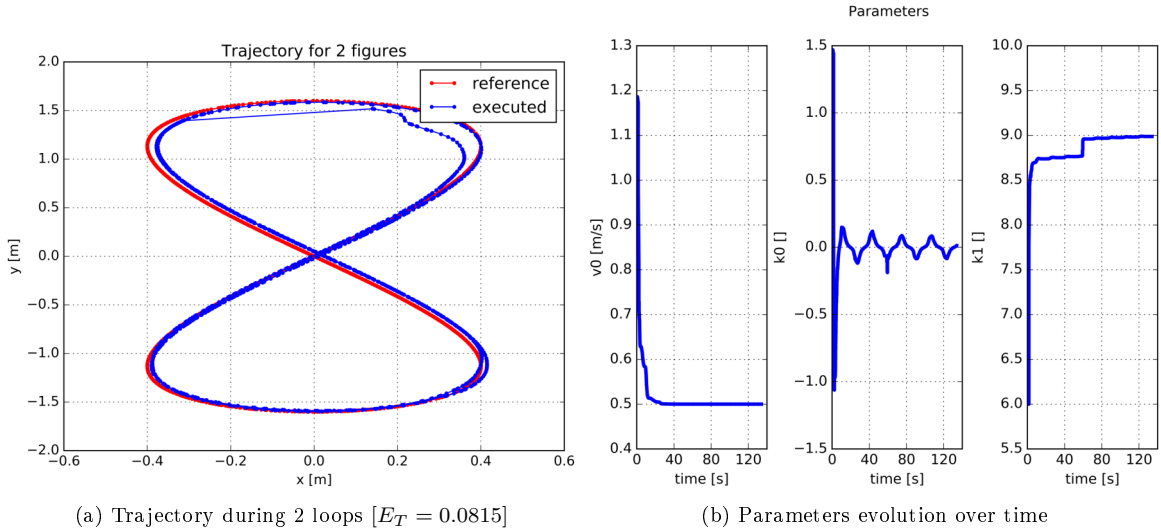


Figure 7.4: Adaptive MPFC with  $\mu_k = 2$  for  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$  (simulation)

As can be seen, both results show the parameters approaching their true values, with faster convergence using  $\mu_k = 10$ .

However, a certain phenomenon can be seen in these results. Both estimates of  $k_1$ , even when using a high learning rate, show a cutoff of the convergence around 9.0. While the estimation continues to increase slowly, this increase is much slower than initially. Meanwhile, while  $k_0$  decreases towards 0, it doesn't converge to it but oscillates around, with peaks over 0.1. In fact, when the estimation of  $k_1$  is too low, its effect in the dynamics must be modeled somehow as the RLS algorithm aspires to reduce the prediction error towards 0. In this case,  $k_0$  takes over and is adjusted at every turn of the car to compensate for the low  $k_1$ . Thus, the predicted  $\dot{\gamma}$  matches the observed  $\hat{\gamma}$ , even though the parameters estimations are incorrect.

A closer look reveals the extent of this effect. Inspecting fig. 7.4b, we see a positive jump in the estimation of  $k_1$  around  $t = 60s$ . At the same time, there is a short negative jump of  $k_0$ , followed by a decrease in the amplitude of its oscillations. A similar effect can be seen in fig. 7.5b. After an initial rapid increase, the estimation of  $k_1$  slows down to a very slow rise. Meanwhile, the oscillations

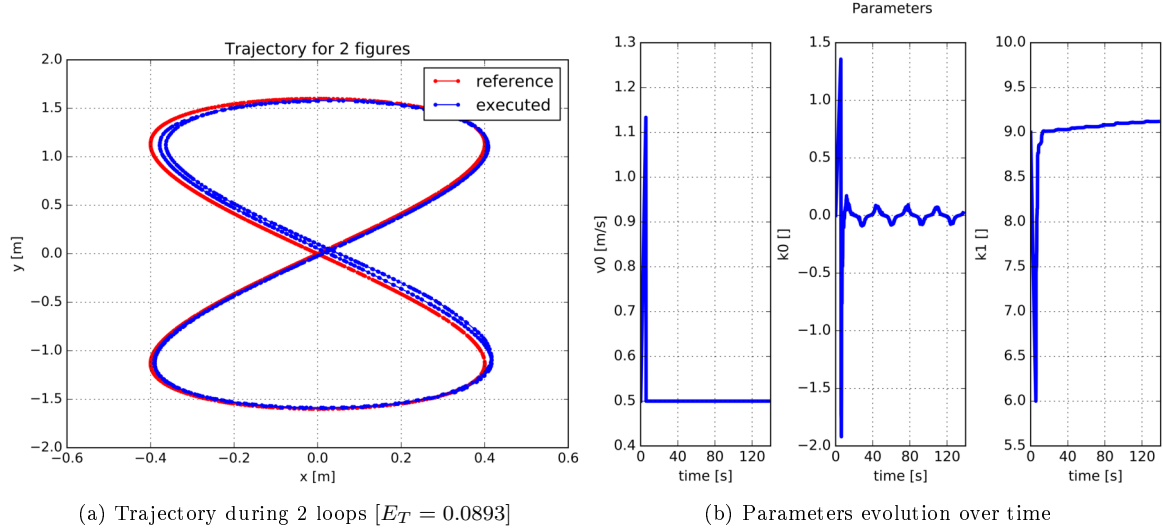


Figure 7.5: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$  (simulation)

of  $k_0$  slowly decrease in amplitude. Both these effects confirm the phenomenon. When  $k_1$  is low, its effect in the dynamics is taken by  $k_0$ . As  $k_1$  increases and its value approaches its true value of 9.6,  $k_0$  needs to be adjusted less in order to correctly predict the system's behavior.

Naturally, while adjusting  $k_0$  compensates the incorrect  $k_1$  estimation in the RLS algorithm, it doesn't match the real system. Thus, the MPC controller is not supplied with a correct model of the system, which harms the tracking performance. Comparing these results to those obtained in section 7.3.2 confirms this statement.

Another test, shown in fig. 7.6, does show convergence and results in a much better tracking.

These results lead us to conclude that while RLS does work well for parameters estimation, when given additional degrees of freedom that do not exist in the real system, it might associate a certain effect to the wrong parameter.

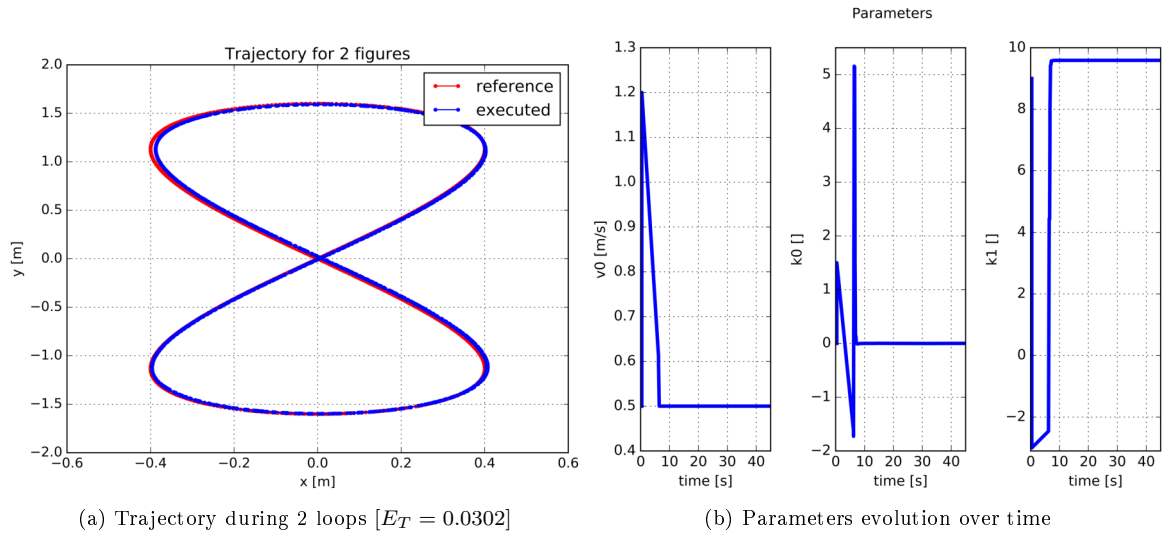


Figure 7.6: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$  - better result (simulation)

### 7.3.4 Using a Low Level Tracker

For comparison, one result using the PID controller to track the optimal plan, with the  $\dot{\gamma} = k \cdot u_\gamma$  model, is shown in fig. 7.7. We see similar results to those obtained without the low level tracker.

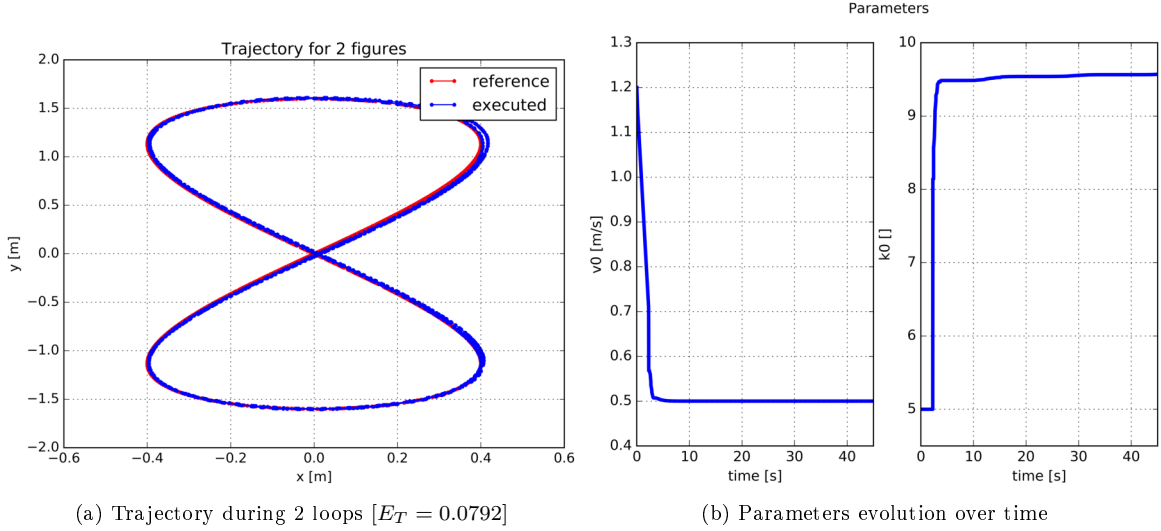


Figure 7.7: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k \cdot u_\gamma$  with low level tracker (simulation)

## 7.4 Experimental Results

After running the algorithm on simulation, the adaptive MPFC can be tested on the experimental setup.

In the simulation, the coefficient  $k$  between the  $u_\gamma$  and  $\dot{\gamma}$  was set to a relatively high value. This enabled the simulated car to make sharp turns. However, this is not the case for the real car. Preliminary tests showed that the true value of  $k$  is approximately 2.5. This is due to the fact that the car's turning radius is quite big, especially when the velocity is increased. Thus, if  $u$  is limited to  $[-1, 1]$ , the MPC solver cannot plan a trajectory satisfying the constraints, resulting in an infeasible trajectory. Following a bigger path with broader turns can help resolve the problem, as slower turning rate is needed. The reference path is thus changed to  $x_{ref}(\tau) = h + 3a \sin(2\tau)$ ,  $y_{ref}(\tau) = 8a \cos(\tau)$ .

However, in many cases this change is not sufficient to make the problem feasible at all time. For this reason, we decided to not constraint  $u$  to  $[-1, 1]$  for the experiments. However, as  $u$  isn't constrained, the steering commands based on the planned input cannot be directly used. We thus continue to use the low level tracker used in the previous sections to track the planned optimal  $\gamma$ . This enables the MPC solver to plan an optimal path which would be then tracked as best possible by the low level tracker.

The following subsections present the results on the experimental setup for the two  $\dot{\gamma}$  models, exactly as done for the simulation in the previous section.

### 7.4.1 Model $\dot{\gamma} = k \cdot u_\gamma$

Fig. 7.8 and 7.9 show the experimental results for the model  $\dot{\gamma} = k \cdot u_\gamma$  for low and high learning rates respectively. As can be seen, both give comparable results, with relatively good tracking error. The estimation of  $v_0$  stabilizes relatively fast but does not converge to a constant value as in the simulation. This is understandable, since the car's velocity is affected by its dynamics and characteristics (e.g. slowing while turning, center of gravity shifting due to rolling and squatting, etc.). Thus, even for a constant throttle value the car's velocity is not constant. A higher learning rate adapts faster to quick changes in the car's velocity but risks overshooting due to too big jumps of the estimate. A low learning rate can act as a filter to reduce fast variations of the estimate. The  $k_0$  estimate shows a similar behavior, but we can note that for  $\mu_k = 1$  the estimate has not reached a stable estimation in the given time and is still increasing. For  $u_k = 10$  we see a fast increase in the estimation followed by oscillations around  $k \approx 3$ . The oscillations may be due to variations in the system's real dynamics, similarly to the velocity variations.



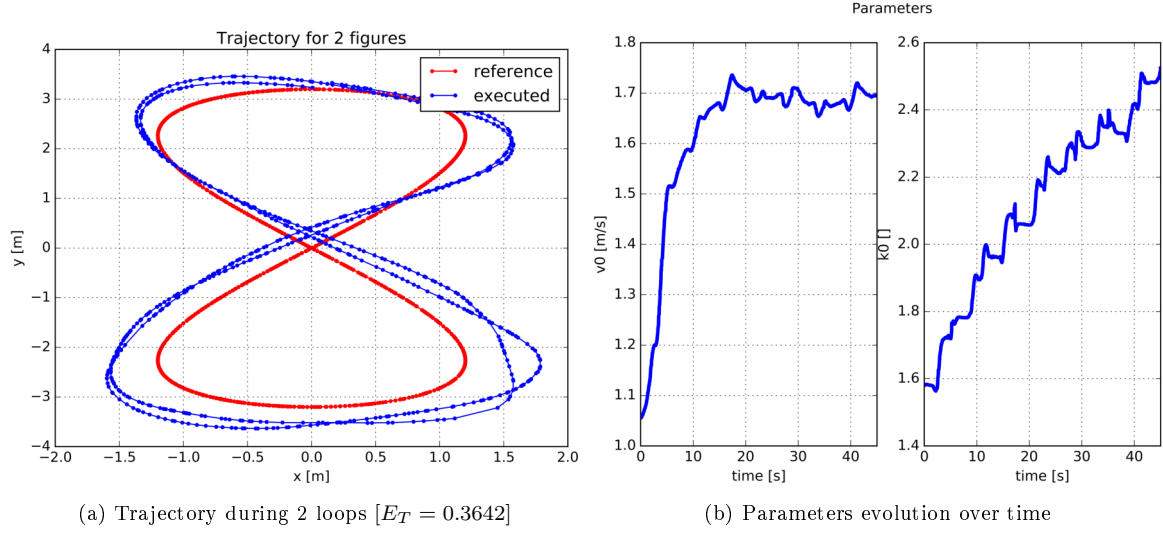


Figure 7.8: Adaptive MPFC with  $\mu_k = 1$  for  $\dot{\gamma} = k \cdot u_\gamma$  (experimental)

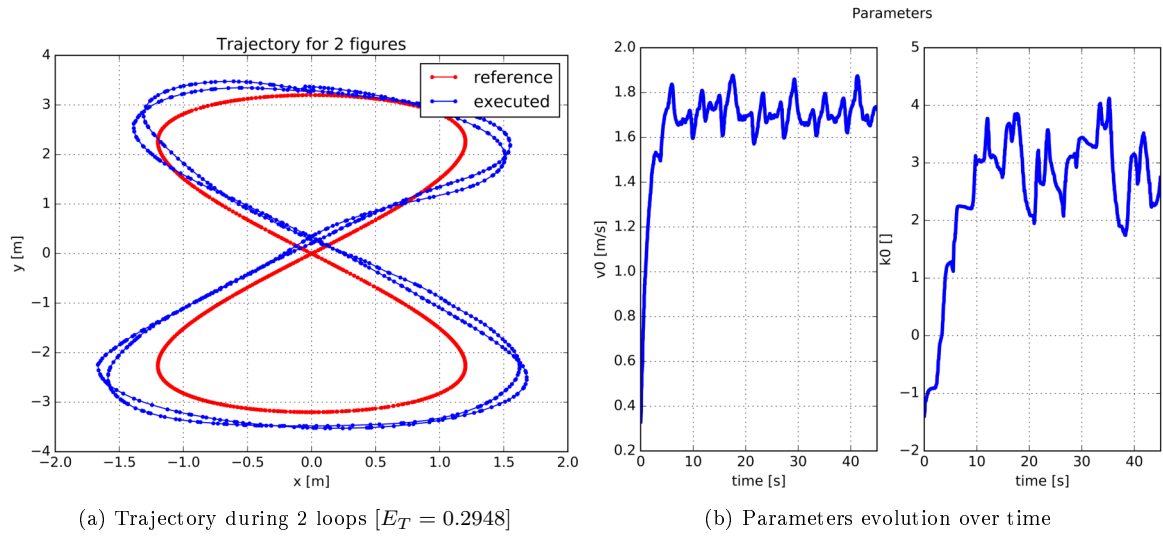


Figure 7.9: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k \cdot u_\gamma$  (experimental)

#### 7.4.2 Model $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$

Fig. 7.10 and 7.11 show the experimental results for this model. We can directly remark a decrease in performance. These experimental results show a similar issue as in simulation, where the additional degree of freedom given to the system causes one parameter to take the role of another. In this case  $k_0$  oscillates around 0, while the  $k_1$  estimate doesn't converge towards a fixed value. Using a high learning rate aggravates the problem as  $k_0$  changes faster and takes bigger values, which then destabilizes the  $k_1$  estimation and the tracking. A low learning rate keeps  $k_0$  around its mean value of 0, and thus enables a more stable estimation of  $k_1$  and better tracking.

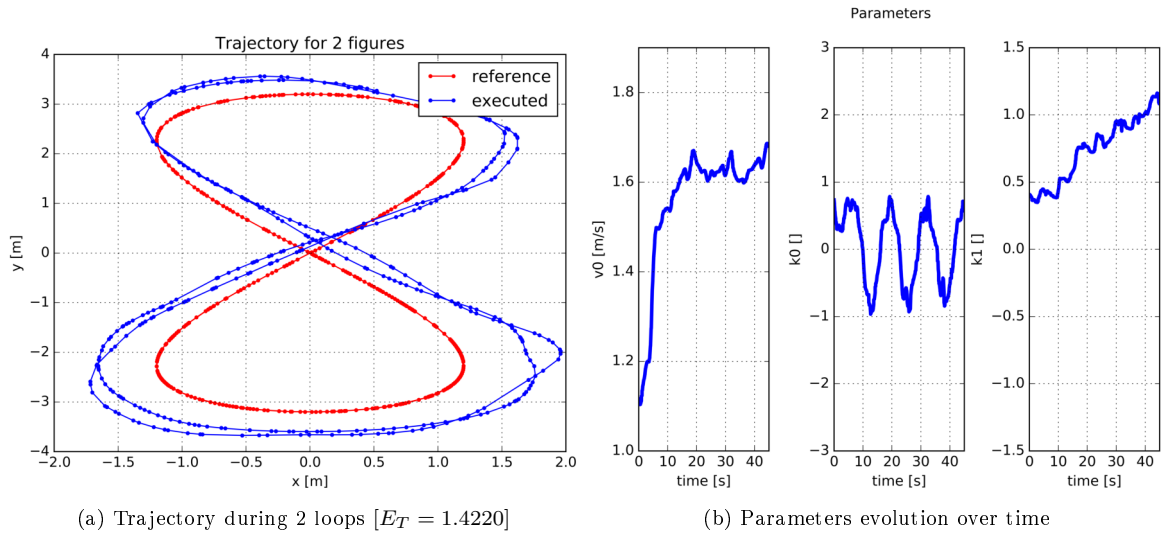


Figure 7.10: Adaptive MPFC with  $\mu_k = 1$  for  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$  (experimental)

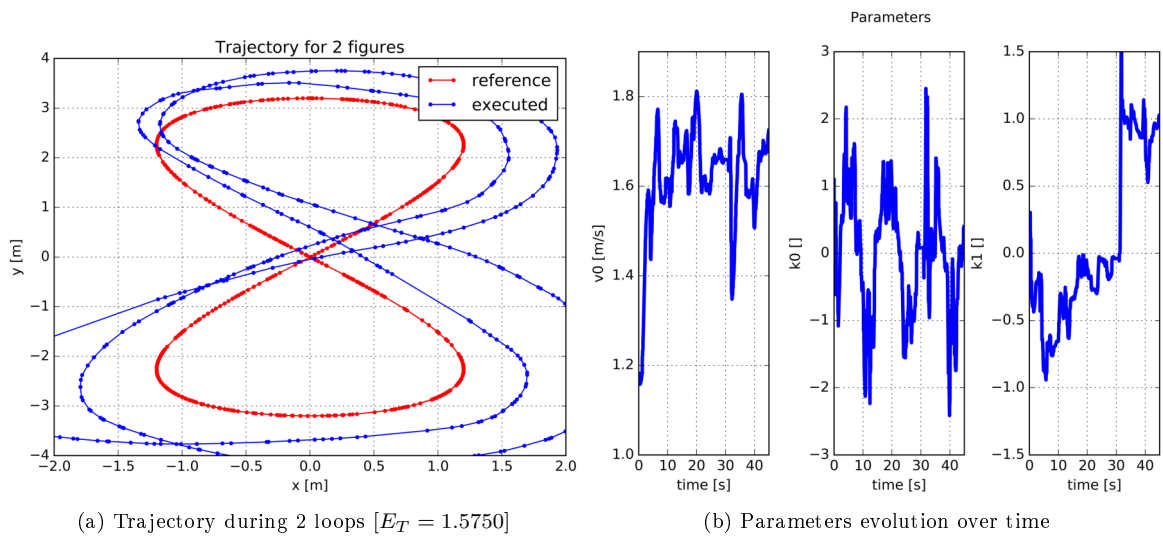


Figure 7.11: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$  (experimental)

### 7.4.3 Small Reference Path

For comparison, we present here one result when tracking a small reference path (equ. 4.4). Fig. 7.12 shows the results of adaptive MPFC using the  $\dot{\gamma} = k \cdot u_\gamma$  model, low level tracker and a high learning rate. As can be seen, the adaptation algorithm works well, but the tracking isn't perfect, as the car struggles to track the tight turns.

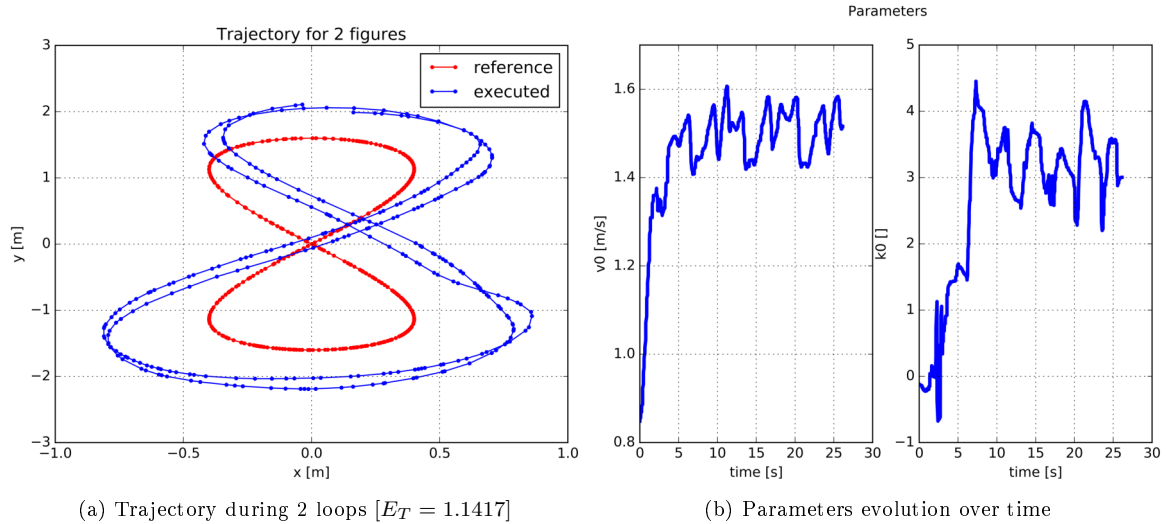


Figure 7.12: Adaptive MPFC with  $\mu_k = 10$  for  $\dot{\gamma} = k \cdot u_\gamma$  for small reference path (experimental)

## 7.5 Summary

This section presented the use of the recursive least squares algorithm in an adaptive MPFC scheme. In order to improve the system's model used by the MPFC controller, the controller was supplemented with an adaptation script which provided it with updated estimations of the system's parameters. The adaptation mechanism received the required measurements and used the RLS algorithm to update the parameters' estimates. The resulting adaptive control scheme was presented.

Before using the adaptive MPFC, the RLS algorithm had to be adapted to the car's system by replacing the general terms used in its derivation by the system dynamics. The final algorithm was derived for both possible representations of the system. In addition, by re-parameterizing the system, the parameters learned by the adaptation algorithm can correspond to any type of input. Using this method, the input was chosen as the normalized steering commands, allowing the removal of the low level tracker.

The adaptive MPFC scheme was tested for two different system models using two learning rates. The first model modeled the angular velocity as a multiple of the steering input. Simulation results showed a convergence of the parameters to their true values at a speed proportional to the learning rate. The second model added a bias term, thus adding a degree of freedom to the model. It was shown that the RLS algorithm does not handle well such case, as it tended to attribute the effect of one parameter to another. While this effect reduced the prediction error and the penalty used in the RLS algorithm, it resulted in a mismatch between parameters' estimates and their true values which harmed the tracking performance.

The simulation results were confirmed on the experimental setup. The parameters' estimates converged to a neighborhood of a fixed value, due to small variations of the car's real dynamics. The phenomenon of associating an effect to the wrong parameter was noticed again when using the extended  $\dot{\gamma}$  model. However, using the experimental setup, the car's turning rate is very low and the need for a bigger reference path was remarked in order to reduce the needed turning rate. Even so, under the limitations of the hardware, the problem may not be feasible at all times. The low level tracker was thus reintroduced. A final test using tracking of the small reference path was presented and demonstrated the limits of car's dynamic capabilities, even when given good estimates for the parameters.



## 8 Gaussian Processes Modeling for Adaptive MPFC

### 8.1 Gaussian Processes

Adaptive MPC can employ many adaptation algorithms for parameters estimation. In the previous sections, we showed the derivation of the recursive least squares algorithm and the results of adaptive MPFC using it. In this section, we present a different adaptation algorithm that utilizes stochastic Machine Learning tools. We use Gaussian processes (GP), a stochastic tool based on a Gaussian distribution of functions. Using Gaussian processes, the parameters are modeled as a distribution. This will also serve as a doorway to stochastic MPC, where the system's dynamics are stochastic instead of deterministic.

Strictly speaking, a Gaussian process is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution. A Gaussian process is characterized by its mean and covariance functions,  $m(x)$  and  $k(x_i, x_j)$  respectively. A GP defines a distribution of the relation between the inputs  $x$  and the outputs  $y$ . In other words, when sampling a GP, the result is a function  $f : x \mapsto y$ , where  $x$  and  $y$  can be multidimensional. The covariance function used in this work is the Squared Exponential, defined as:

$$k(x_i, x_j) = \text{cov}(f(x_i), f(x_j)) = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right) + \sigma_n^2 \delta_{ij} \quad (8.1)$$

where  $\delta_{ij}$  is the Kronecker's delta ( $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$ ). The hyperparameters are the signal variance  $\sigma_f^2$ , the length-scale  $l$  and the noise variance  $\sigma_n^2$ .

A Gaussian process is trained using training data. Initially, the GP is described by a prior distribution. Using Bayes law, the training data is incorporated into the prior, resulting in a posterior distribution. The posterior can then be sampled to get a function  $f$  or queried at different query points to get predictions of the outputs.

Finally, a Gaussian process can be decomposed by specifying an explicit basis for the mean. The GP would thus be a sum of a zero mean GP and an explicit basis, i.e.

$$\begin{aligned} f &\sim \beta^T \Phi(x) + g \\ g &\sim GP(0, k(x_i, x_j)) \end{aligned} \quad (8.2)$$

where  $\beta$  is a vector of coefficients and  $\Phi(x)$  is a set of fixed basis functions. This form will be used throughout this work.

**Notation:**  $X$  denotes the inputs of a GP,  $y$  denotes the outputs.

More details about Gaussian processes can be found in [20, 21].

### 8.2 System Modeling using Gaussian Processes

Recall that the deterministic system was defined as

$$\dot{q} = \begin{pmatrix} v \cos \gamma \\ v \sin \gamma \\ k_0 + k_1 \cdot u_\gamma \end{pmatrix} \quad (8.3)$$

Instead of using this deterministic system, the relationship between  $\dot{q}$ ,  $\gamma$  and  $u_\gamma$  can be modeled as a Gaussian process, such that

$$X = \begin{pmatrix} \gamma \\ u_\gamma \end{pmatrix}, \quad y = \dot{q} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\gamma} \end{pmatrix} \quad (8.4)$$

However, the deterministic dynamics contain the cosine and sine of  $\gamma$ . In order to handle this, we can either change the inputs or choose an appropriate explicit basis. By including the sine and cosine in

the explicit basis, the GP can get a better understanding of the data structure. Thus, the explicit basis is given by the Differential Wheels Basis

$$\Phi(X) = \begin{pmatrix} 1 \\ \cos X_1 \\ \sin X_1 \\ X_2 \end{pmatrix} \quad (8.5)$$

where  $X_k$  is the  $k$ -th element of  $X$ .

For the first test, only an explicit basis is used. Thus, the GP model can be described by the coefficients vector  $\beta$ , such that

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\gamma} \end{pmatrix} = \beta^T \Phi(X) \quad (8.6)$$

If we compare equ. 8.3 and 8.6, we find that  $\beta$  is given by

$$\beta^T = \begin{pmatrix} 0 & v_x & 0 & 0 \\ 0 & 0 & v_y & 0 \\ k_0 & 0 & 0 & k_1 \end{pmatrix} \quad (8.7)$$

**Remark 1:** As both the linear basis and the differential wheels basis include the basis function  $\phi(x) = 1$ , it is natural to use the more general model  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ .

**Remark 2:** To lighten the plots, the legend is emitted from the plots showing a fitting of a Gaussian process. In all, the blue dots represent data points, the solid red line is the mean of the GP, and the two dashed red lines delimit the area of 95% confidence region (pointwise mean plus and minus two standard deviations).

Fig. 8.1 shows the result of fitting a Gaussian process using this basis, for the simulation and for the car.

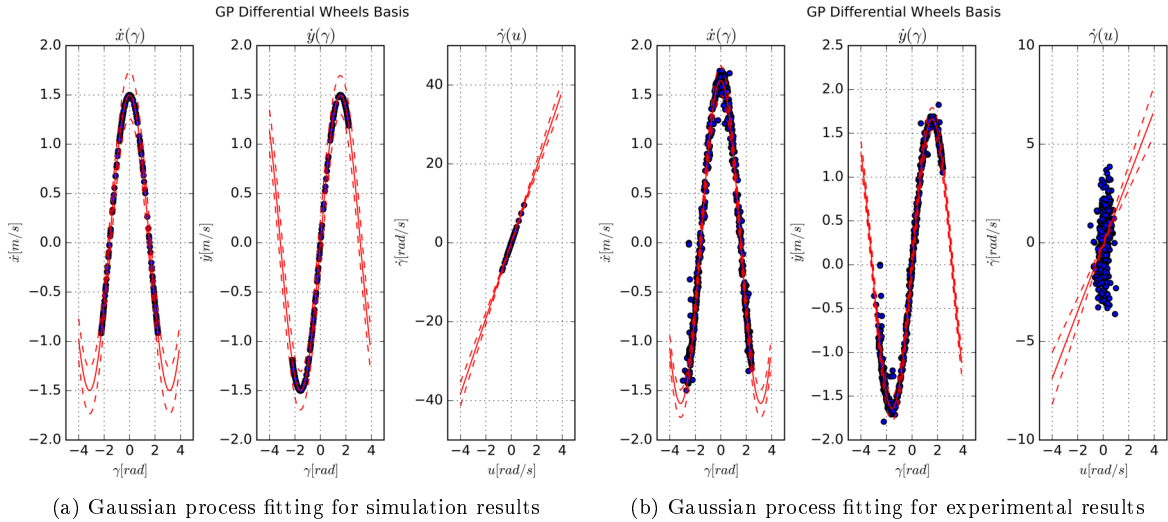


Figure 8.1: Gaussian process fitting using the differential wheels explicit basis

The resulting  $\beta$  can be found in appendix A. Using the presented form of  $\beta^T$ , the resulting parameters are

$$\text{Simulation} : v_x = v_y = 1.499, k_0 \approx 0, k_1 = 9.599 \quad (8.8)$$

$$\text{Experiment} : v_x = 1.635, v_y = 1.649, k_0 = -0.15, k_1 = 1.885 \quad (8.9)$$

(Recall that in simulation the true values of the parameters are  $v_x = v_y = 1.5$ ,  $k_0 = 0$ ,  $k_1 = 9.6$ )

The figures and the parameters estimations show us a good fit of the results by the Gaussian processes. Thus, this method of generating a GP will be used for the adaptation algorithm for an adaptive MPFC script (see section 8.4).

### 8.3 Comparison of Fitting Methods

While the previous results are very good, our final goal is to use the same Gaussian process inside a stochastic MPFC scheme (see section 9). Due to implementation limitations, the stochastic MPFC can only handle linear basis and one-dimensional data. Thus, the inputs and outputs must be decomposed into 1D vectors and a linear basis must be used. In addition, as a linear basis is used, the  $\cos$  and  $\sin$  functions in the dynamics cannot be modeled directly. Thus, instead of using  $\gamma$  as input,  $\cos \gamma$  and  $\sin \gamma$  must be used. The input to the Gaussian process becomes

$$X = \begin{pmatrix} \cos \gamma \\ \sin \gamma \\ u_\gamma \end{pmatrix} \quad (8.10)$$

and the explicit basis is given by

$$\Phi(X) = \begin{pmatrix} 1 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} \quad (8.11)$$

For these conditions, several types of GPs can be used: a full GP (as described by equ. 8.2), only a linear explicit basis or only a squared exponential kernel. Fig. 8.2 and 8.3 show comparison between these different types for simulation and experimental results.

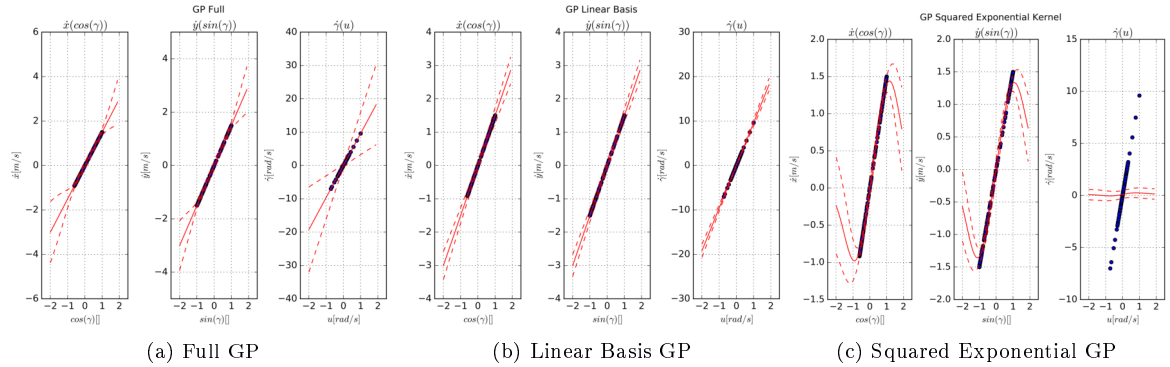


Figure 8.2: Gaussian process fitting for simulation using (a)full, (b)linear basis, (c)squared exponential kernel

As can be seen, both for simulation and experiment, the linear basis gives the best results for our system. In addition, the linear basis GP is convenient as the model parameters can be extracted easily, as was done for the differential wheels basis. Thus, this type of GP will be used for the stochastic MPFC scheme (section 9).

The numerical values of the kernel's parameters  $l$  and  $\sigma_f$  are presented in appendix A. The noise variance  $\sigma_n$  is estimated during the training of the GP.

### 8.4 Adaptive MPFC using Gaussian Processes

Using the described method for Gaussian processes fitting, an online adaptation algorithm can be implemented. At every time step, new measurements of  $X = \begin{pmatrix} \gamma \\ u_\gamma \end{pmatrix}$  and  $y = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\gamma} \end{pmatrix}$  are taken. Using the new measurements, a GP is fitted to the collected data using the differential wheels basis. As the

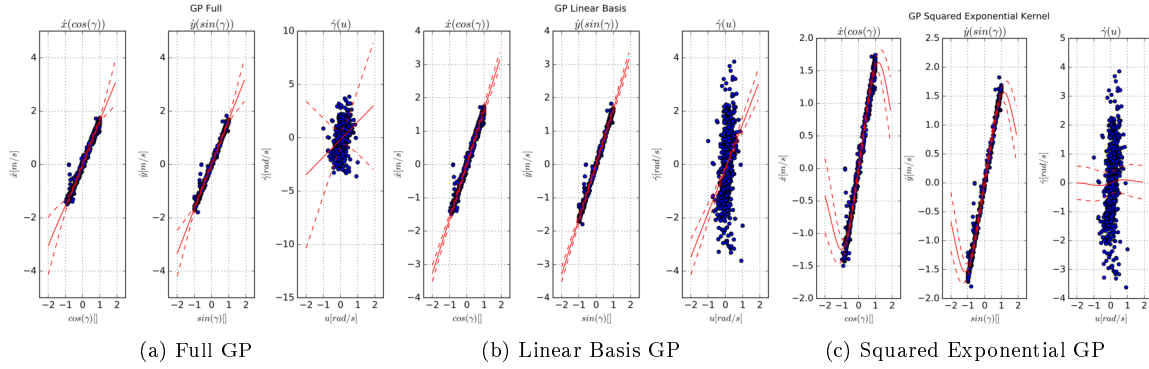


Figure 8.3: Gaussian process fitting for experiment using (a)full, (b)linear basis, (c)squared exponential kernel

computation time of fitting a GP increases rapidly with the number of training cases, the number of cases is limited to 150. In addition, the quality of the fit depends not on the number of training cases, but on their quality. In other words, in order to get a good fit the measurements should be spread over the full interval of possible values. For this end, instead of keeping every new measurement, the data selection is randomized by setting the probability to keep a new measurement to 0.1, such that the selected training cases are taken from a bigger time-span.

Once the GP is trained, the parameters  $v$ ,  $k_0$  and  $k_1$  are extracted from the resulting model and sent to the MPFC controller, similarly to the RLS adaptation script. As done previously, the bigger reference path and the low level controller are used.

Fig. 8.4 shows the results of the adaptive MPFC using GP in simulation. As can be seen, the parameters converge to their true values in around 5 seconds. During the transitory period, the phenomenon of attributing the effect of  $k_1$  to  $k_0$  can be seen but is corrected rapidly. We can see the system tracks well the reference path.

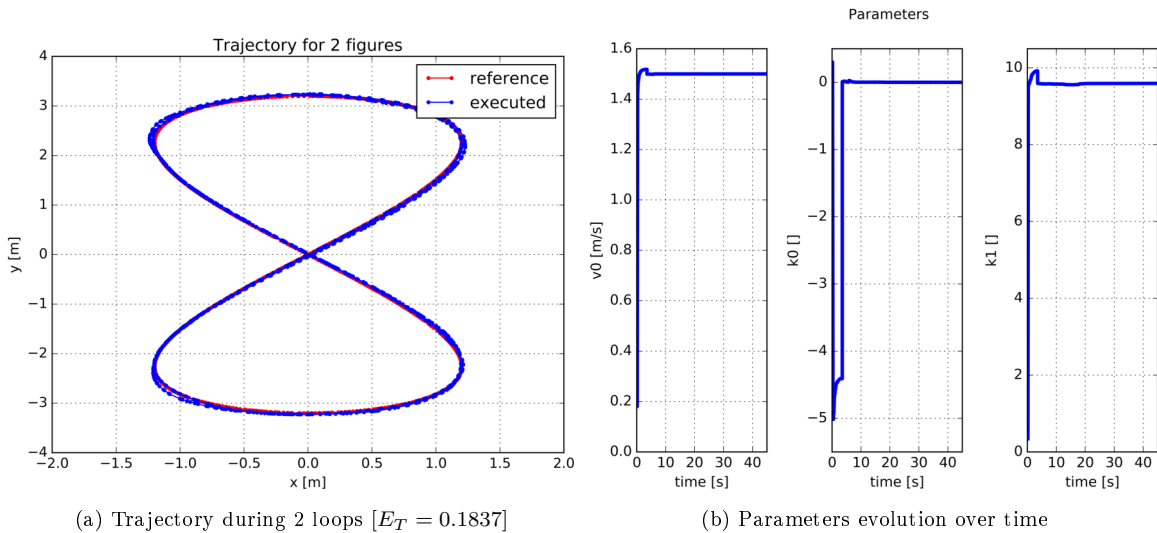


Figure 8.4: Adaptive MPFC using Gaussian processes fitting (simulation)

The experimental results can be seen in fig. 8.5. While the tracking is not perfect, it is similar to the results using RLS. The parameters can be seen converging to a neighborhood of a fixed value (notice the small scale of the  $k_0$  plot).  $v_0$  converge to around 1.65 m/s,  $k_0$  decreases to almost 0 and  $k_1$  slowly converges towards 2.5, all reasonable values based on the results of the previous experiments.



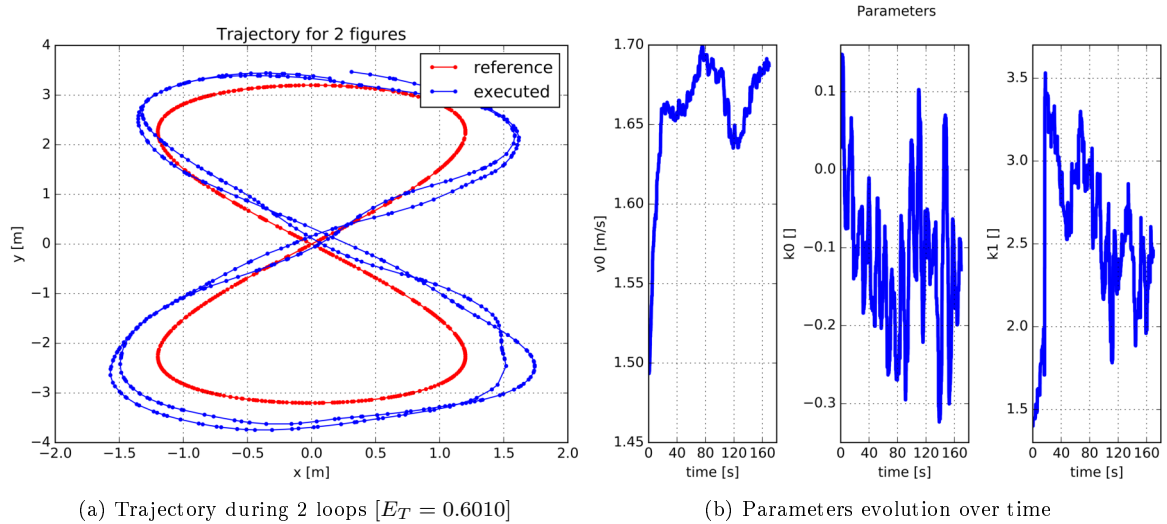


Figure 8.5: Adaptive MPFC using Gaussian processes fitting (experimental)

## 8.5 Summary

In this section, we have presented Gaussian processes, their training and their usage in adaptive model predictive path-following control.

First, Gaussian processes were introduced. A Gaussian process is a collection of random variables whose any finite number of which have a joint Gaussian distribution. By using a GP, the relationship between inputs and outputs can be modeled, usually using a kernel and an explicit basis.

We then presented a method to model the car's dynamic system using a Gaussian process. A new explicit basis, the Differential Wheels basis, was introduced. The results of fitting the measurements using a GP were presented, showing a good estimation of the system's parameters.

A comparison of fitting different types of Gaussian processes was presented. Due to implementation limitations, we concentrated on a linear explicit basis and a squared exponential kernel. The best fit was shown to be achieved using only an explicit basis.

An adaptive MPFC scheme using Gaussian processes was presented, similar to the adaptive MPFC using RLS. Its performance on both simulation and experiments was presented. The parameters were shown to converge to either a fixed value or a neighborhood of a value.



## 9 Stochastic MPFC

After using Gaussian processes to model the car's dynamics and to adapt the system's parameters online, the next step is to incorporate the Gaussian processes into the MPC controller. By replacing the deterministic system dynamics by a stochastic model, a stochastic MPFC scheme is created. In stochastic MPFC, the state is represented by a random variable whose distribution is defined by its mean and covariance. The evolution of the system is described by a stochastic system. In the case of the car, the system dynamics can be modeled using a Gaussian process (as shown in the previous section). Thus, this Gaussian process can be used to predict the next state of the system based on its current state and the applied input.

By using a stochastic MPFC scheme, the MPC controller can take into account the uncertainty of the system evolution. This is beneficial in case of disturbances, noise or system uncertainties, as the controller is more robust against uncertainties and can guarantee stability and feasibility even when faced with unknown disturbances.

This thesis presents a basic stochastic MPFC scheme using a simple control policy. The constraints of the real system are only applied to the mean trajectory and the covariance is only constrained to a bound interval. This is the first step towards a more sophisticated scheme. One example is [22] which presents a stochastic MPC framework that constraints the average violation of state constraints over time. [23] presents the equivalence between two common control policies - linear state feedback and disturbance feedback. Using this equivalence, a convex optimization problem can be solved and an optimal state feedback control policy can be found. This result enables MPC schemes for systems subject to unknown disturbances.

### 9.1 The Stochastic MPFC Scheme

The system state is now defined by a Gaussian distribution, such that

$$q \sim \mathcal{N}(\mu_q, \Sigma_q) \quad (9.1)$$

where  $\mu$  is the mean and  $\Sigma$  is the variance. As only the real system can be fitted with a GP, the virtual system remains deterministic. In addition, the current implementation limits the use of multi-input – multi-output GPs, so we are forced to model each dimension individually, without any coupling between dimensions. The full system can be described by

$$q \sim \begin{pmatrix} \mathcal{N}(\mu_x, \Sigma_x) \\ \mathcal{N}(\mu_y, \Sigma_y) \\ \mathcal{N}(\mu_\gamma, \Sigma_\gamma) \end{pmatrix} \quad x = \begin{pmatrix} q \\ \tau \end{pmatrix} \quad (9.2)$$

The state is thus represented as a 7 dimension vector ( $3 \mu_i$ ,  $3 \Sigma_i$  and  $\tau$ ). After fitting each dimension as a GP, the new system dynamics are given by

$$\dot{x} = \begin{pmatrix} \dot{\mu}_x, \dot{\Sigma}_x \\ \dot{\mu}_y, \dot{\Sigma}_y \\ \dot{\mu}_\gamma, \dot{\Sigma}_\gamma \\ \dot{\tau} \end{pmatrix} = \begin{pmatrix} gp_x(\cos \mu_\gamma, \Sigma_\gamma) \\ gp_y(\sin \mu_\gamma, \Sigma_\gamma) \\ gp_\gamma(u_\gamma, \Sigma_{u_\gamma}) \\ u_\tau \end{pmatrix} \quad (9.3)$$

where  $gp_i()$ ,  $i = x, y, \gamma$  denotes the uncertainty propagation of the mean and covariance using the GP corresponding to dimension  $i$ . The uncertainty propagation is done using exact moment matching for squared exponential kernel and linear basis (see section 10.2). As  $u_\gamma$  is deterministic, its variance is fixed to a low value, e.g.  $\Sigma_{u_\gamma} = 10^{-1}$ . The three GPs  $gp_x$ ,  $gp_y$ ,  $gp_\gamma$  were trained using measurements from either simulation or experiments using GPs containing a linear explicit basis, as described in section 8.3.

**Remark:** As stated previously, due to implementation limitations the uncertainty propagation can only handle a linear explicit basis. For this reason, the inputs to the  $gp_x$  and  $gp_y$  are  $\cos \gamma$  and  $\sin \gamma$  respectively. However, we do not have a good method to compute the covariance of a trigonometric function. One could use a Taylor expansion to approximate the covariance. In this case  $\Sigma_f = \frac{\partial f}{\partial x} \Sigma_x$ , which becomes  $\Sigma_f = \sin(\mu_x) \Sigma_x$  for  $f(x) = \cos(x)$ . However, as  $\sin(\mu_x)$  can become zero, this method can create numerical problems. For this reason, we decide to use  $\Sigma_\gamma$  directly for the propagation.

While this is not completely correct, it still provides the propagation with information about the uncertainty of the system and doesn't introduce numerical errors.

The basic optimal control problem defined in equ. 5.1 - 5.6 is still used. Solely the following changes are applied, due to the new stochastic nature of the system:

- As the state is a distribution, the tracking error also becomes a distribution ( $e \sim \mathcal{N}(\mu_e, \Sigma_q)$ ). The properties of affine transformations on distributions (see appendix B) imply

$$e(t) = q(t) - q_{ref}(\tau(t)) \Rightarrow \begin{cases} \mu_e = \mu_q - q_{ref}(\tau(t)) \\ \Sigma_e = \Sigma_q \end{cases} \quad (9.4)$$

where  $\mu_q = (\mu_x, \mu_y, \mu_\gamma)^T$ ,  $\Sigma_q = (\Sigma_x, \Sigma_y, \Sigma_\gamma)^T$ .

- The objective function remains the same, apart from the term penalizing the tracking error. As the error is a distribution, the term  $\frac{1}{2} \|e(t)\|_Q^2$  is split into two terms for the mean and covariance, such that

$$Obj_e = \frac{1}{2} \|\mu_e(t)\|_Q^2 + trace\left(\|\Sigma_e(t)\|^2\right) \quad (9.5)$$

- The state box constraints presented in section 5 become the box constraints on  $\mu$ , while additional constraints are applied on  $\Sigma$  such that  $\Sigma \in \mathcal{S}$ . Thus  $x \in \mathcal{Q} \times \mathcal{S} \times \mathbb{R}$ .
- The terminal constraints remain the same, but are only applied on the mean. Equ. 5.5 and 5.6 are thus replaced by

$$\begin{aligned} \frac{1}{2} \|\mu_e(T)\|_Q^2 + \mu_e(T)^T Q_f \mu_q(T) &\leq 0 \\ \mu_e(T)^T Q_f \dot{q}_{ref}(\tau(T)) &\geq 0 \end{aligned}$$

While this is not completely supported by the theory, experiments show that adding these terminal constraints improve the performance. The stochastic MPFC would still work well without them.

- Shooting continuity constraints are relaxed, such that instead of equality constraints we now allow a tolerance. Thus, at each step in the horizon

$$\begin{aligned} \mu_{y,k} - \mu_{x,k+1} &\in [-\varepsilon, \varepsilon] \\ \Sigma_{y,k} - \Sigma_{x,k+1} &\in [-\epsilon, \epsilon] \end{aligned}$$

where  $\mu_{y,k}$  and  $\Sigma_{y,k}$  are the results of the uncertainty propagation of  $x_k$  and  $\epsilon, \varepsilon$  are tolerances.

Numerical values for the additional parameters can be found in appendix A.

### 9.1.1 Integration of Distributions

The last step needed for the stochastic MPFC scheme is a tool allowing us to predict the state over the horizon. In the nominal MPFC scheme, the next state is predicted using a standard order 4 Runge-Kutta method (RK4) given the continuous time dynamics. This method must be extended to handle stochastic dynamics. In this case, the dynamics are described by a function  $\dot{x} = f(x, u)$  that given  $x \sim \mathcal{N}(\mu, \Sigma)$  and  $u$  supplies the mean and covariance of  $\dot{x}$  using the exact moment matching algorithm. The resulting distribution of  $\dot{x}$  can be integrated using an approximate RK4 integrator. The approximate RK4 integrator is given by

$$\begin{aligned} k_1 &\sim f(x, u) \\ k_2 &\sim f\left(x + \frac{h}{2}k_1, u\right) \\ k_3 &\sim f\left(x + \frac{h}{2}k_2, u\right) \\ k_4 &\sim f(x + hk_3, u) \\ x(t+h) &\sim x + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

where  $h$  is the time step. The distributions are summed using affine transformations of Normal random variables (see appendix B). More details about this integrator can be found in [24].

Using the described scheme, the stochastic MPFC can now be tested on simulation and on the experimental setup.

## 9.2 Simulation Results

In order to run the stochastic MPFC on the simulation, Gaussian processes trained using simulation data is needed. The GPs used contain only a linear explicit basis and are the ones presented in fig. 8.2b (section 8.3).

Fig. 9.1 presents the simulation results using the proposed stochastic MPFC scheme. The tracking results over 2 loops are seen in fig. 9.1a. As can be seen, the tracking is very good, with only slight deviations from the reference path. As the controller is stochastic, it associates a distribution to its prediction at any point of time. In other words, the confidence of the controller's prediction is measured by the uncertainty, or the variance, of the distribution. Fig. 9.1b presents the trajectory during one figure-8, accompanied by the controller's uncertainty. The area delimited by the purple and green lines represents the 95% confidence region (pointwise mean plus and minus two standard deviations). As shown in the figure, the confidence region is relatively small, indicating a high level of confidence of the controller.

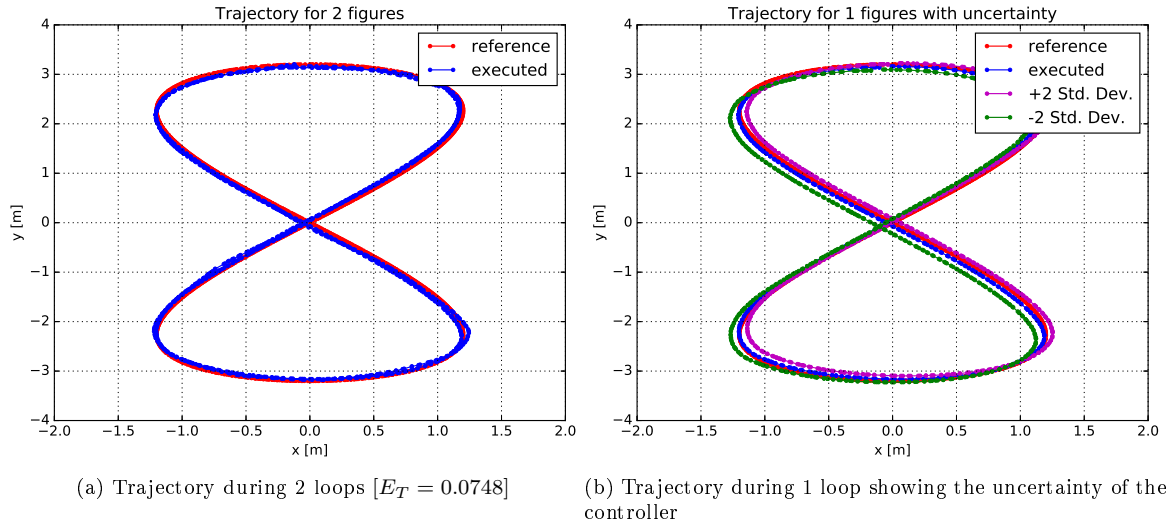


Figure 9.1: Stochastic MPFC (simulation)

Another interesting insight into the stochastic controller can be seen in fig. 9.2. This figure presents the prediction of the optimal state sequence over the horizon at a fixed point of time (a single optimal solution of the OCP). In other words, it shows the Gaussian distribution of each  $q_k^*$  at the 10 steps in the horizon ( $k = 1, \dots, 10$ ). The mean of each distribution is planned according to the optimal solution for the tracking problem. The variance is also propagated, following the stochastic system's dynamics. Naturally, we would expect the variance to grow at each prediction step as the prediction becomes less and less confident. While this occurs in the system, the growth is minimal to the point that it is hard to notice. The controller's prediction is thus highly confident, which is of course a good sign, though expected in the perfect conditions of the simulation.

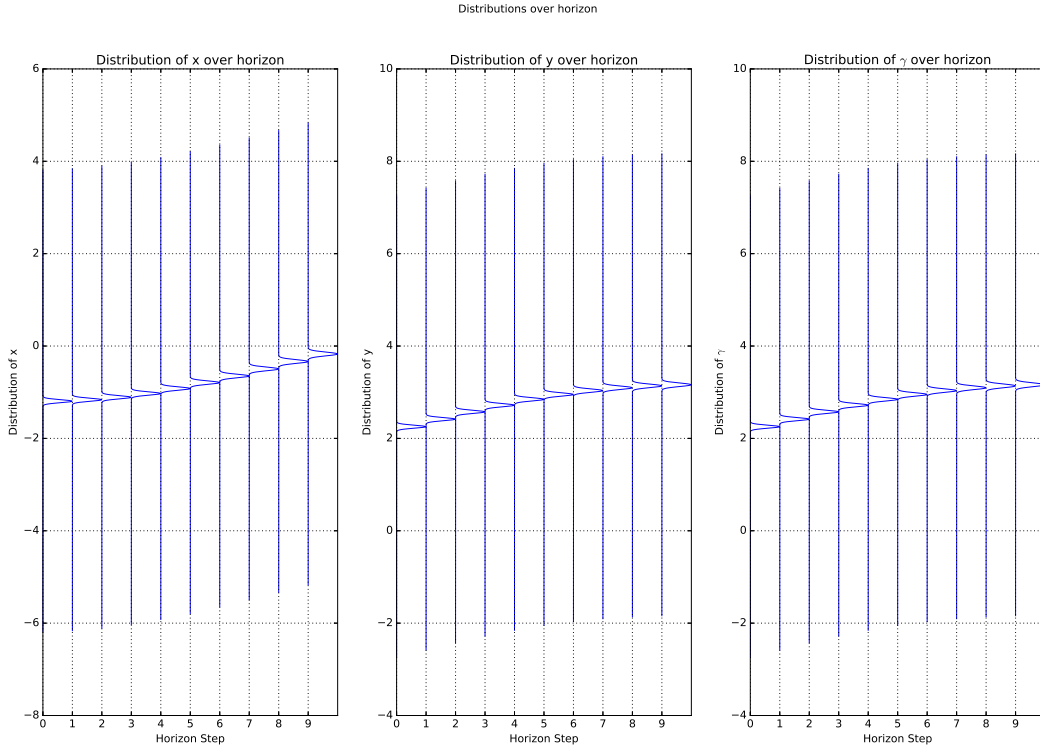
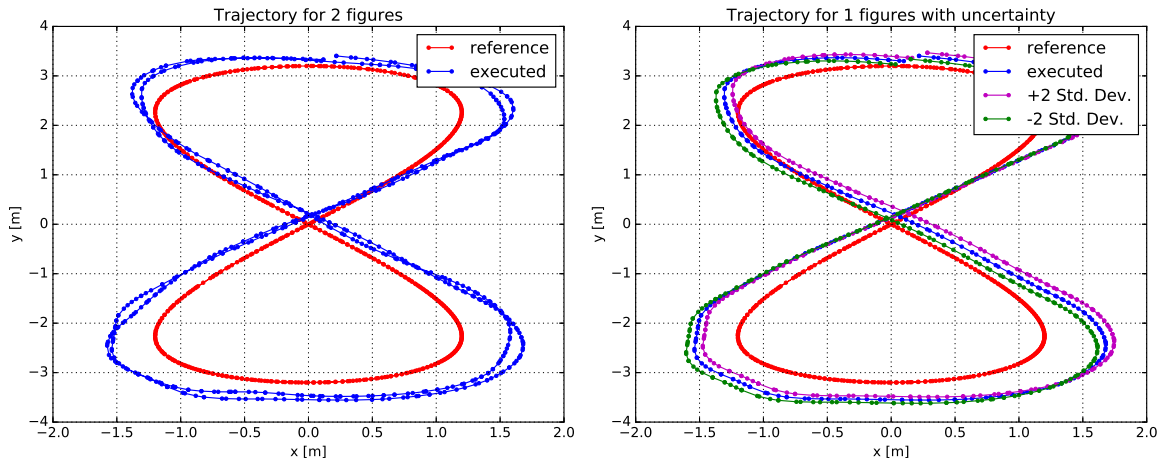


Figure 9.2: Prediction of the state over the horizon (simulation)

### 9.3 Experimental Results

After seeing the good performance on the simulation, we turn to test the stochastic MPFC scheme on the experimental setup. The same scheme is used, apart from replacing the Gaussian processes that model the stochastic system. For the experimental setup, we use GPs trained on data from the experiments, presented in fig. 8.3b (section 8.3).

Fig. 9.3 presents the results on the experimental setup. Similar to the simulation, fig. 9.3a shows the tracking results during 2 loops and fig. 9.3b shows the tracking including the controller's confidence region. As can be seen, even on the experimental setup where the data is noisier and cannot be fitted as well as in the simulation, the confidence region is still quite small. The tracking itself is similar to previous results using other MPFC schemes.



(a) Trajectory during 2 loops [ $E_T = 0.3990$ ]

(b) Trajectory during 1 loop showing the uncertainty of the controller

Figure 9.3: Stochastic MPFC at normal velocity (experimental)

We can again plot the prediction of the state over the horizon, which can be seen in fig. 9.4. The plot shows comparable results to the simulation, with a slow growth of the variance over the horizon.

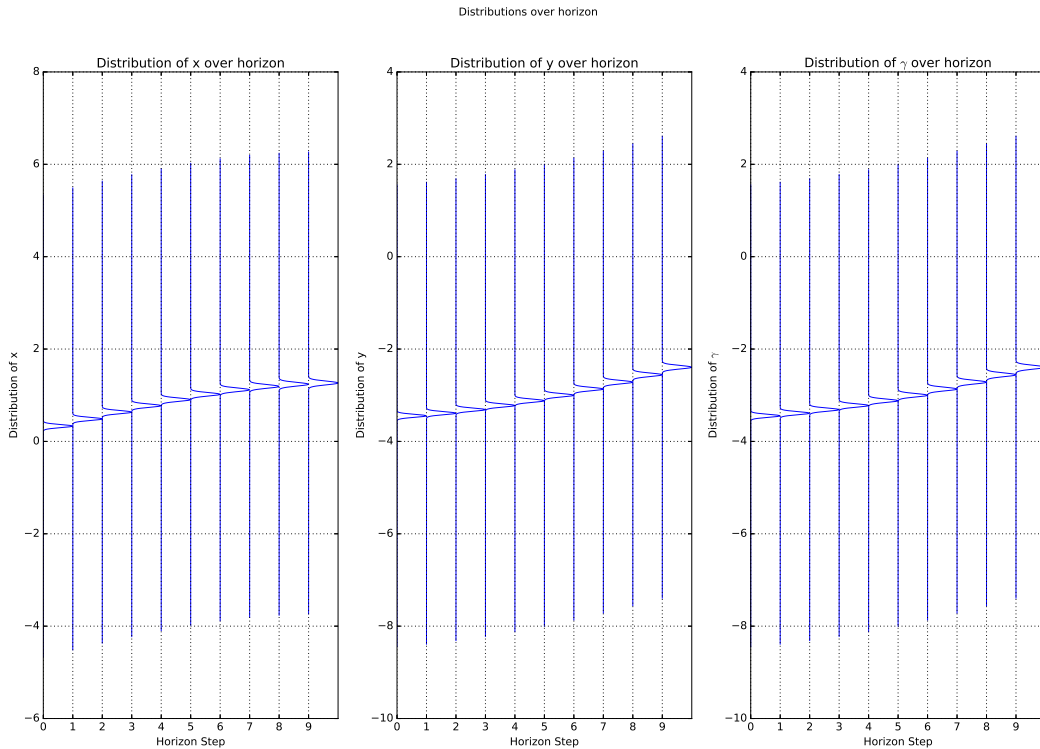


Figure 9.4: Prediction of the state over the horizon (experimental)

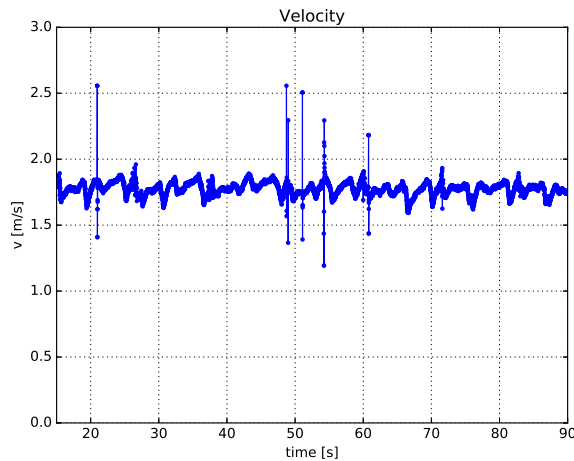


Figure 9.5: Velocity profile for the normal velocity stochastic MPFC experiment

A final test that can be done is to verify the effect of increasing the velocity of the car. In the experiment presented in fig. 9.3 the car's velocity is around  $1.75\text{ m/s}$ , as can be seen in fig. 9.5. In a second test, the car was driven at a velocity of around  $3\text{ m/s}$ . The tracking results are shown in fig. 9.6 as well as the velocity profile. As can be seen, the tracking results are worse at high velocity, mostly due to the limited turning radius of the car. In fact, at high speed the car struggles to turn as sharply as needed which causes the big overshoot of the reference path. Above  $3\text{ m/s}$  the testing room available isn't sufficiently large for the car to complete a figure-8. Remark that even under these conditions, the stochastic MPFC scheme manages to keep stability and feasibility.

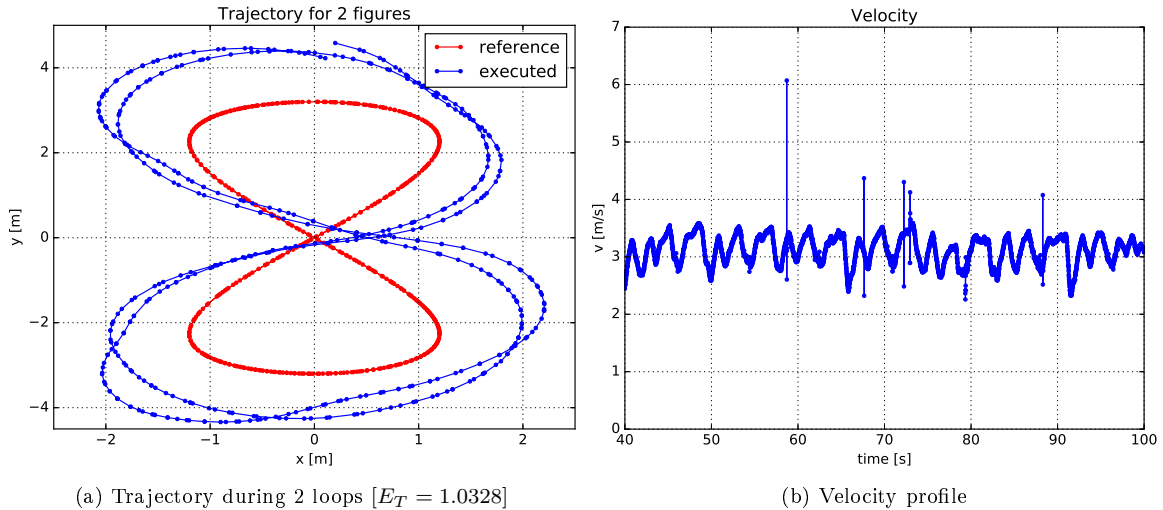


Figure 9.6: Stochastic MPFC at high speed (experimental)

## 9.4 Summary

This section presented a stochastic MPFC scheme using Gaussian processes. The nominal MPFC scheme presented in previous sections was adapted to the use of a stochastic model for the car's dynamics. In stochastic MPC, as the dynamic model is stochastic, the MPFC controller predicts the state over the horizon as a sequence of distributions, defined by their means and variances. This introduces a confidence measurement into the prediction, such that the controller is more robust to disturbances, noise and uncertainties

The stochastic MPFC scheme was presented in details. The stochastic representation of the state was introduced and the changes with respect to the nominal MPFC scheme were explained. The system was modeled using Gaussian processes obtained by fitting measurements from either the simulation or the experiments. A method to integrate over the stochastic dynamical system using RK4 was also presented.

The stochastic MPFC scheme was tested first on simulation and then on the experimental setup. The simulation results showed good tracking of the reference path. The confidence level of the of the controller while tracking was shown to be relatively low. In addition, the evolution of the state's prediction over the horizon was shown for a fixed point in time. The optimal solution's variance was shown to slowly grow, which indicates a slow decrease in the controller's confidence over the horizon.

Similar results were presented for the experimental setup. The tracking performance was shown to be good and comparable to results achieved using other MPFC schemes. The confidence level of the controller was shown to be relatively high and the uncertainty over the horizon to be slowly increasing. Considering the uncertainties in the system in the experimental setup, this indicates the good adaptation of the MPFC scheme to uncertainties and disturbances.



## 10 Uncertainty Propagation for Gaussian Processes

The last part of this thesis will concentrate on the propagation of uncertainty for Gaussian processes. Given a Gaussian process, we want to find the distribution of the output given a random variable input. Both random variables are defined by their mean  $\mu$  and variance  $\Sigma$ , thus the problem becomes predicting the output's  $(\mu_y, \Sigma_y)$  given the input's  $(\mu_X, \Sigma_X)$ .

Given the input distribution  $p(X_q)$ , the output distribution is given by

$$p(y_q|X, y) = \int_{\mathcal{X}} p(y_q|X, y, X_q) p(X_q) dX_q \quad (10.1)$$

Two methods to compute this integral will be presented in this section.

### 10.1 Testing System

In order to test the uncertainty propagation, we use a simple system modeled by the function  $f(x)$ , named thereafter the “true function”. The function  $f$  is sampled at  $N$  random points which are used to train a Gaussian process. The uncertainty propagation algorithm is tested by propagating over 10 steps, starting from the initial distribution  $x_0$ .

Fig. 10.1 shows the true function and the fitting of a Gaussian process. The numerical values for  $f$ ,  $N$  and  $x_0$  can be found in appendix A.

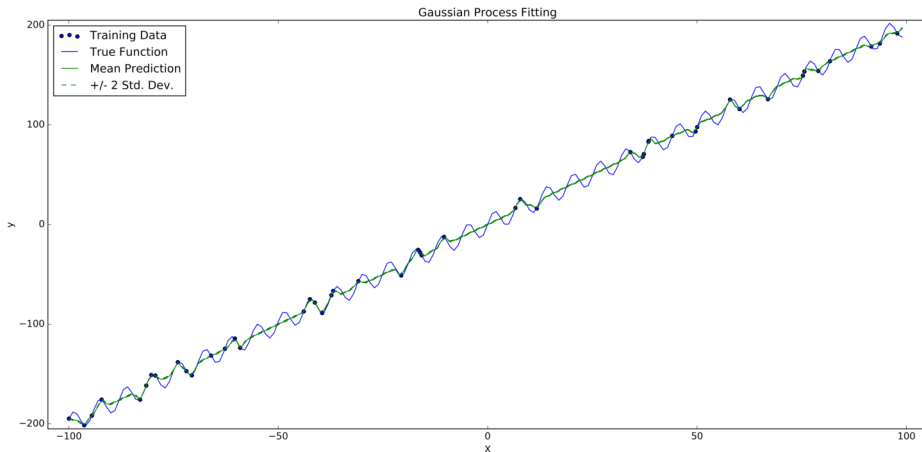


Figure 10.1: Gaussian process fitting to the true function using squared exponential kernel and linear explicit basis

### 10.2 Exact Moment Matching

Given a Gaussian input distribution  $p(X_q) \sim \mathcal{N}(\mu, \Sigma)$ , the integral in equ. 10.1 can be computed analytically for certain types of basis and kernels, including the squared exponential and polynomial kernels and the linear basis. Thus, the first and second moments of the output distribution can be found exactly, hence the name “exact moment matching”.

[25] (section 3.4) derives the moment matching expressions for Gaussian processes containing only Gaussian kernels. [24] extends these results to Gaussian processes containing an additional linear explicit basis. Using these references, the exact moment matching algorithm can be implemented using a symbolic framework.

Fig. 10.2 shows the uncertainty propagation using exact moment matching. As can be seen, the initial distribution is very narrow and becomes more spread out over the iterations. This is natural, as at each propagation the variance of the output is bigger than that of the input.

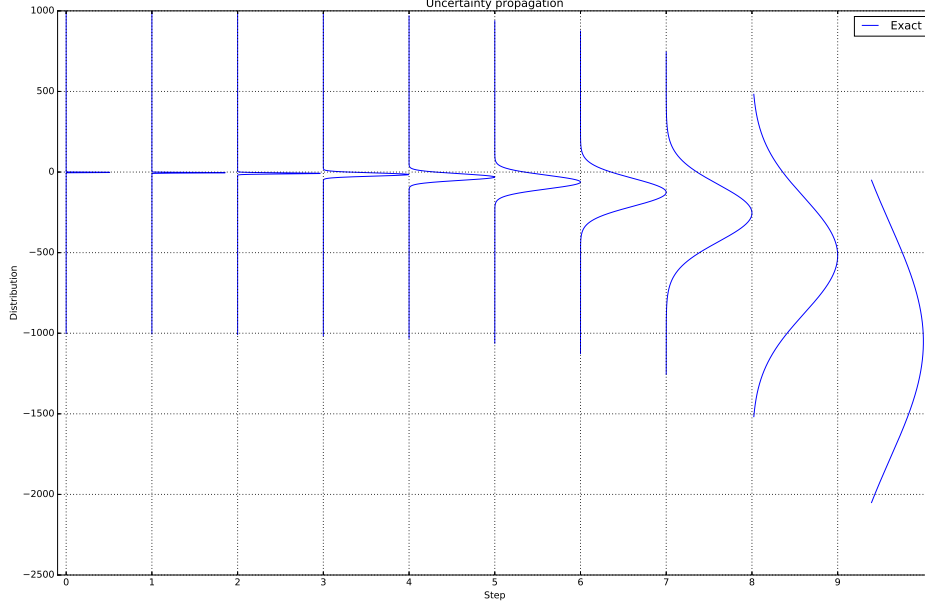


Figure 10.2: Uncertainty propagation using Exact Moment Matching

### 10.3 Approximate Moment Matching

Exact moment matching is impossible for other types of kernels, as the integral in equ. 10.1 cannot be computed analytically. For such cases, one could use the approximate moment matching method, where the covariance function is approximated using a Taylor series around the mean  $u$ , such that

$$C(x, x_i) \approx C(u, x_i) + (x - u)^T C'(u, x_i) + \frac{1}{2} (x - u)^T C''(u, x_i) (x - u) \quad (10.2)$$

**Proposition 4** (Approximate Moment Matching). *Consider a Gaussian process with the kernel function  $C(x_1, x_2)$  and a linear explicit basis. Given the Gaussian uncertain input  $x \sim \mathcal{N}(\mu, \Sigma)$ , the output is a Gaussian distribution  $y \sim \mathcal{N}(m, v)$  where  $m$  and  $v$  are given by*

$$m(u, \Sigma) = m_{kernel} + m_{basis} = \sum_{i=1}^N \alpha_i l_i + u^T \beta$$

$$v(u, \Sigma) = v_{kernel}(u, \Sigma) + v_{basis}(u, \Sigma) - v_{mixed}(u, \Sigma)$$

with

$$v_{kernel}(u, \Sigma) = \sigma_n + C(u, u) + \frac{1}{2} tr(C''(u, u)\Sigma) - \sum_{i,j=1}^N S_{ij} tr(C'(u, x_i)C'(u, x_j)^T \Sigma)$$

$$- \sum_{i,j=1}^N (S_{ij} + \alpha_i \alpha_j) C(u, x_i) C(u, x_j)$$

$$- \frac{1}{2} \sum_{i,j=1}^N (S_{ij} + \alpha_i \alpha_j) [C(u, x_i) tr(C''(u, x_j)\Sigma) + C(u, x_j) tr(C''(u, x_i)\Sigma)]$$

$$v_{basis}(u, \Sigma) = tr(M^{-1}\Sigma) + u^T M^{-1}u + \beta^T \Sigma \beta$$

$$v_{mixed}(u, \Sigma) = 2 \sum_{i=1}^N E \left( u l_i + tr(C'(u, x_i)\Sigma) \right) - 2 m_{kernel} m_{basis}$$

$$l_i = C(u, x_i) + \frac{1}{2} tr(C''(u, x_i)\Sigma)$$

*Proof.* See Appendix C. ■

Fig. 10.3 presents the uncertainty propagation using approximate moment matching for the same system used previously, with the exact moment matching results for comparison. As can be seen, the approximate results are close to the exact propagation, though not exactly the same.

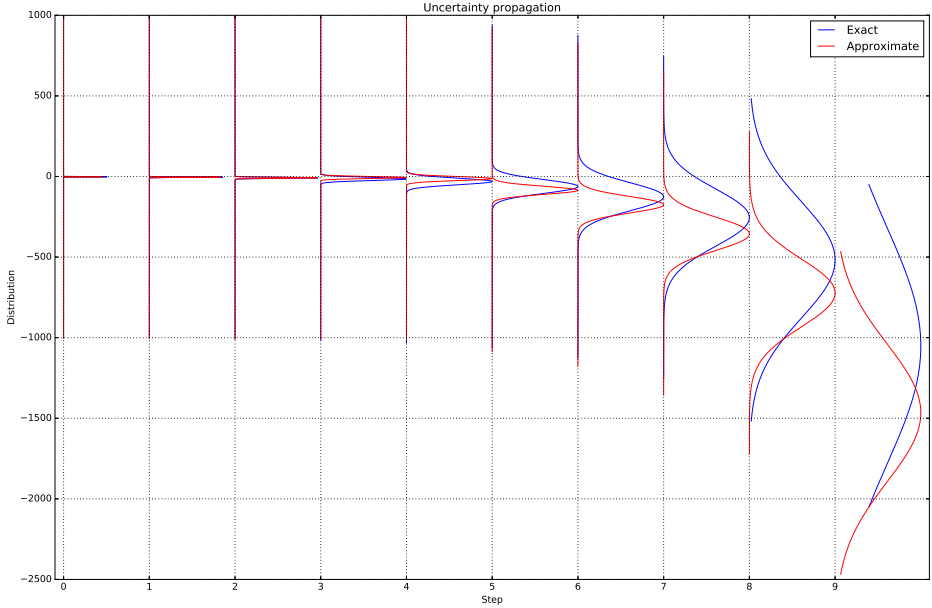


Figure 10.3: Uncertainty propagation using Approximate and Exact Moment Matching



## 11 Conclusions and Future Work

Path-following is a very common task in many applications. Many solutions were proposed over the years to handle path-following, each with its own advantages and disadvantages. Inspired by Airborne Wind Energy systems, where a kite must track a path in the sky in order to generate optimal power, this thesis explores the use of Model Predictive Path-Following (MPFC) schemes to track a reference path. The platform chosen to test the MPFC schemes is a micro-scale RC car.

This work presents a complete implementation for a path following system. A low level tracker is combined with a high level MPFC planner in order to follow a predefined reference path. The optimization problem is presented, as well as the real and virtual systems used in the MPFC scheme. Different Model Predictive Control schemes are presented and tested on both simulation and an experimental setup.

A nominal MPFC scheme is presented and tested in experiments. Due to challenges arising from the real world application, regularization terms must be added. The experimental results are presented along with two imperfections in the system - a variation of the velocity and a delay in the low level tracker's performance. The influence of the controller's parameters is investigated. It is shown that the tracking of the reference path can be vastly improved by reducing the penalty of the slack variables, equivalent to reducing the regularization used by the controller. However, due to numerical limitations and real world imperfections, completely removing the regularization causes the controller to jump between local solutions and results in an unsmooth trajectory. The penalty on the tracking error is also tuned but is shown to have a minor effect on the tracking performance.

To improve the system's model used by the MPFC controller, an adaptive scheme is proposed. The MPFC controller is coupled with an adaptation script which provides it with updated estimations of the system's parameters. The adaptation mechanism uses measurements to update the parameters' estimates according to an update rule. Thus plant-model mismatch is reduced and the performance is improved.

The first adaptation algorithm proposed in this work is the Recursive Least Squares (RLS), based on the Robbins-Monro algorithm. Several versions of RLS are derived for two different representations of the state. A sufficient condition for convergence is derived for the second representation, as well as a filter RLS extension. The RLS algorithm is then adapted to the car system and is tested on both simulation and the experimental system. It is shown how by re-parameterizing the system and using an adaptation algorithm, the parameters learned can correspond to any type of input. In this case, the input is chosen to represent normalized steering commands, such that the low level tracker can be discarded.

The adaptive MPFC scheme is tested using two learning rates for two different models of the system. The first model states that the angular velocity is a multiple of the steering input. Testing on simulation, this results in a convergence of the parameters to their true values. The second model adds a degree of freedom to the model in the form of a bias term. It is shown that the RLS algorithm is not well suited for this case, as it tends to associate the effect of one parameter to another. While the prediction error is reduced, a plant-model mismatch is formed, which harms the tracking performance.

These results are confirmed on the experimental setup, where the parameters' estimations converge to a neighborhood of a fixed value. The phenomenon of attributing an effect to the wrong parameter appears again. For the experimental setup however, the car's turning rate is very low. A bigger reference path is needed in order to reduce the turning rate required. Even so, under the limitations of the hardware, the problem may not be feasible at all times. The low level tracker is thus used again.

The second adaptation algorithm presented in this thesis is based on Gaussian processes. Gaussian processes are introduced and a method to model the car's dynamics using a special explicit basis is presented. Different types of Gaussian processes fitting are compared. It is shown that a linear basis GP is most suited for the car's system. An adaptive MPFC scheme is then proposed using recursive GP-based adaptation algorithm. Its performance is evaluated and the convergence of the estimated parameters is shown.

Using Gaussian processes, the system dynamics can be modeled as a stochastic system. By replacing the deterministic model used in the nominal MPFC, a stochastic MPFC scheme is created. In such scheme, the state is modeled as a Gaussian distribution and the prediction is done in a stochastic manner. The stochastic nature makes the MPFC controller more robust in the face of uncertainties

and disturbances. The stochastic MPFC scheme is presented, including a method allowing approximate integration over a stochastic dynamical model. The tracking performance of the stochastic controller are presented, showing tracking comparable to other MPFC schemes. The confidence level of the controller while tracking is investigated, as well as the evolution of the planned states over the horizon. In both cases, low uncertainty is shown as well as an extremely slow increase in the uncertainty over the horizon.

The last section presents the derivation and implementation of two uncertainty propagation methods for Gaussian processes. For certain kernel functions, such as the squared exponential, an analytic computation of the propagation can be done, resulting in Exact Moment Matching. For other kernel functions, an analytic solution isn't available. Using Taylor series of the kernel, the Approximate Moment Matching method is derived.

During the work on this thesis, certain difficulties were encountered. Some challenges have forced us to change our approach and to come up with new solutions. Others, however, deserve their own work and could be the center of a following project. With limited time, choices had to be made to determine which challenges could be solved and which must be overlooked for the time being.

The main issue encountered in all of the MPFC schemes proposed in this thesis is the need of regularization. An underlying issue causes the solver to jump significantly between local solutions which causes rapid variations of the input and results in an unsmooth trajectory. This issue was overcome by penalizing the difference between the new solution and the previous one. However, this penalty is somewhat counter-intuitive as it encourages the MPC controller to less modify its solution and less adapt to the situation. In addition, the theory upon which the controller is based doesn't include this regularization and thus the theoretical support is weakened. In fact, it was seen that reducing the regularization highly improves the tracking, but was not possible due to the jumping issue.

Another direction for future work could be replacing the low level tracker. It seems that the low level tracker, added as a necessity, may be a weak point of the controller. While the planner can function perfectly, its plan is not executed directly, but is tracked by the low level controller. Thus, any issue in the low level tracking may cause a deviation from the optimal plan and reduce the tracking performance. For this thesis, we were satisfied with a decent performance of the low level tracker. However, finding a method to directly apply the MPFC plan could benefit the path-following. This might also allow to overcome the issue presented in section 7.4.

Finally, we believe that this research could highly benefit from an improvement of the experimental setup. As one could remark while reading this thesis, the experimental results don't show a perfect tracking. From simulation results, we believe that the MPC schemes proposed are not to blame. While lower performance is expected from any imperfect real world system, we believe that the reduced performance can be attributed to experimental conditions that can be improved. First, the delay in the system should be reduced as much as possible. In a real time system such as a car, any delay in executing the commands can harm the path-following. Second, the car's turning rate given the current hardware is very limited. This limits the possible trajectories it can track and can often lead to infeasible optimization problem. The car used also exhibits a relatively high level of slipping, under-steer and over-steer, which must be either eliminated or incorporated into the dynamic model as they can affect the tracking results.

To conclude, as seen in previous studies and in this thesis, model predictive control is very well suited to path-following problems. A basic MPFC scheme can be easily implemented and can provide good path-following. The results can be then improve using a variety of MPC schemes, whose derivation and performance were presented. An adaptive MPC can improve the tracking by better estimating the parameters using either RLS or Gaussian processes. A stochastic MPC can include a stochastic representation of the system and thus better handle noisy and uncertain systems. Even though some difficulties were presented, we believe that future work could tackle these issues and show an even higher performance for various path-following applications.



## A Numerical Values of Parameters and Results

This appendix includes numerical values for various parameters and results used throughout this thesis.

### Low Level Tracking

$$K_p = -450, T_i = 0, T_d = 0, bias = 485, u_{max} = 385 \quad (\text{A.1})$$

### Nominal Model Predictive Path-Following Control

*MPFC parameters*

$$Q = \begin{pmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{pmatrix}, \quad Q_f = \begin{pmatrix} 500 & 0 & 0 \\ 0 & 500 & 0 \\ 0 & 0 & 150 \end{pmatrix}, \quad Q_E = 1 \quad (\text{A.2})$$

$$R_u = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.01 \end{pmatrix}, \quad R = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix}, \quad N = 10, \quad \delta = 0.1 \text{ sec} \quad (\text{A.3})$$

$$\mathcal{Q} = \{(x, y, \gamma) | x \in [-20; 20], y \in [-20; 20], \gamma \in [-\infty; \infty]\} \quad (\text{A.4})$$

$$\mathcal{U} = \{(u_\gamma, u_\tau) | u_\gamma \in [-20; 20], u_\tau \in [0; 3]\} \quad (\text{A.5})$$

*Reference path*

$$a = 0.2, \quad h = 0 \quad (\text{A.6})$$

### Gaussian Processes Parameters

Signal variance:  $\sigma_f^2 = 0.1$

Length-scale:  $l = 1.0$

### Stochastic Model Predictive Path-Following Control

$$\mathcal{S} = \{\Sigma | \Sigma \in [10^{-27}; 10^{32}]\}, \quad \varepsilon = \epsilon = 10^{-9} \quad (\text{A.7})$$

### Uncertainty Propagation

True function:  $f(x) = 2x + 10\sin x$

Initial distribution:  $x_0 \sim \mathcal{N}(-1, 0.1)$

Number of sample points:  $N = 50$

### Basis Coefficients for Gaussian Processes

Coefficients for Gaussian processes fitting using differential wheels basis.

$$\begin{aligned} \text{Simulation : } \beta^T &= \begin{pmatrix} 3.3e-9 & 1.499 & 3.8e-9 & 1.7e-7 \\ -2.6e-8 & 9.5e-9 & 1.499 & 1.0e-8 \\ 7.6e-7 & 1.1e-6 & 7.4e-8 & 9.599 \end{pmatrix} \\ &\Rightarrow v_x = v_y = 1.499, k_0 \approx 0, k_1 = 9.599 \end{aligned}$$

$$\begin{aligned} \text{Experiment : } \beta^T &= \begin{pmatrix} 5.9e-3 & 1.635 & -1.6e-2 & 5.2e-2 \\ 1.0e-2 & -1.3e-2 & 1.649 & -1.2e-2 \\ -0.15 & 7.1e-2 & 0.1 & 1.885 \end{pmatrix} \\ &\Rightarrow v_x = 1.635, v_y = 1.649, k_0 = -0.15, k_1 = 1.885 \end{aligned}$$

## B Affine transformations of Normal random variables

In this appendix, we present formulas for affine transformations of Normal random variables used throughout this thesis. See [26] for more details.

Be  $X$  and  $Y$  independent random variables,  $a$  and  $b$  real numbers.

### Scalar Addition and Multiplication

$$\begin{cases} E[aX + b] &= aE[X] + b \\ Var[aX + b] &= a^2Var[X] \end{cases} \quad (\text{B.1})$$

Thus, for  $X \sim \mathcal{N}(\mu, \sigma^2)$

$$aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2) \quad (\text{B.2})$$

### Addition of Independent Random Variables

$$\begin{cases} E[X + Y] &= E[X] + E[Y] \\ Var[X + Y] &= Var[X] + Var[Y] \end{cases} \quad (\text{B.3})$$

Thus, for  $X \sim \mathcal{N}(\mu, \sigma^2)$  and  $Y \sim \mathcal{N}(\eta, \nu^2)$

$$X + Y \sim \mathcal{N}(\mu + \eta, \sigma^2 + \nu^2) \quad (\text{B.4})$$



## C Approximate Moment Matching - Derivation

This appendix details the derivation of the approximate moment matching method for uncertainty propagation. The method used is similar to the one presented in [25] (sections 3.2-3.3) for Gaussian processes without an explicit basis. Certain steps missing in their derivation will also be presented.

Given a Gaussian input distribution  $X_p \sim \mathcal{N}(u, \Sigma)$ , we would like to find the distribution of the output  $y_p \sim \mathcal{N}(m, v)$ . From [24], the posterior distribution is defined by

$$\mu(u, \Sigma) = K_* \alpha + \phi(X_p)^T \beta \quad (\text{C.1})$$

$$\begin{aligned} \sigma^2(u, \Sigma) = & \sigma_n + C(X_p, X_p) - K_* S K_*^T + \phi(X_p)^T (M^{-1} + \beta \beta^T) \phi(X_p) \\ & - K_* E \phi(X_p) - \phi(X_p)^T E^T K_*^T - \mu(u, \Sigma)^2 \end{aligned} \quad (\text{C.2})$$

where

$$\begin{aligned} K_* &= C(X_p, X) \\ \alpha &= K^{-1}(\bar{y} + \phi^T \beta) \\ \beta &= M^{-1} [B^{-1} b + \phi K^{-1} \bar{y}] \\ S &= K^{-1} (K - \phi^T M^{-1} \phi) K^{-1} - \alpha \alpha^T \\ E &= K^{-1} \phi^T M^{-1} - \alpha \beta^T \end{aligned}$$

and  $C(x_1, x_2)$  is the kernel function,  $\phi$  is the explicit basis,  $b$  and  $B$  are the prior mean and variance,  $K_{ij} = C(x_i, x_j) + \delta_{ij} \sigma_n$  and  $\sigma_n$  is the noise variance. Notice that in the case of  $\phi = 0$ , we get the same expressions as in [25].

In order to find the output's distribution the integral  $p(y_q | X, y) = \int_{\mathcal{X}} p(y_q | X, y, X_q) p(X_q) dX_q$  must be computed. In particular, for the first and second moments separately, we get (see [24] for more details)

$$\begin{aligned} m(u, \Sigma) &= E[y_p | X, y] = \int_{\mathcal{X}} \mu(u, \Sigma) p(X_q) dX_q \\ v(u, \Sigma) &= E[y_p | X, y] - E[y_p | X, y]^2 = \int_{\mathcal{X}} \sigma^2(u, \Sigma) p(X_q) dX_q - m(u, \Sigma)^2 \end{aligned}$$

where  $E[\cdot]$  is the expected value.

$\mu$  and  $\sigma^2$  can then be replaced by equ. C.1 and C.2.

### Mean of propagated distribution

$$\begin{aligned} m(u, \Sigma) &= \int_{\mathcal{X}} (K_* \alpha + \phi(X_p)^T \beta) p(X_q) dX_q \\ &= \sum_i \left( \int_{\mathcal{X}} K_{*i} \alpha_i p(X_q) dX_q \right) + \int_{\mathcal{X}} \phi(X_p)^T \beta p(X_q) dX_q \end{aligned}$$

The first part, involving the kernel, can be written as

$$m_{kernel} = \sum_i \left( \int_{\mathcal{X}} K_{*i} \alpha_i p(X_q) dX_q \right) = \sum_{i=1}^N \alpha_i E_x [C(x, x_i)] \quad (\text{C.3})$$

We thus need to compute the expected value of  $C(x, x_i)$ . For cases in which the integral cannot be computed analytically,  $C(x, x_i)$  can be approximated by its Taylor series around the mean  $u$  of  $x$ ,

$$C(x, x_i) \approx C(u, x_i) + (x - u)^T C'(u, x_i) + \frac{1}{2} (x - u)^T C''(u, x_i) (x - u) \quad (\text{C.4})$$

Note that  $C^{(p)}(u, x_i) \forall p$  in this expression are constants independent of  $x$ . Using this approximation, the expected value can be computed for any kernel function, as follows

$$\begin{aligned}
E_x [C(x, x_i)] &= E_x [C(u, x_i)] + E_x [(x - u)^T C'(u, x_i)] + E_x \left[ \frac{1}{2} (x - u)^T C''(u, x_i) (x - u) \right] \\
&= C(u, x_i) + E_x [(x - u)^T] C'(u, x_i) + \frac{1}{2} E_x [(x - u)^T C''(u, x_i) (x - u)]
\end{aligned}$$

where

$$E_x [(x - u)^T] = E_x [x] - u = u - u = 0 \quad (\text{C.5})$$

$$\begin{aligned}
E_x [(x - u)^T C''(u, x_i) (x - u)] &= E_x [(x - u)^T] C''(u, x_i) E_x [(x - u)] + \text{tr} (C''(u, x_i) \text{Cov}[x - u]) \\
&= 0 + \text{tr} (C''(u, x_i) \Sigma)
\end{aligned}$$

where we used the properties of the expected value and  $E_x [x^T A x] = E_x [x]^T A E_x [x] + \text{tr}(A \text{Cov}[x])$ . Finally

$$E_x [C(x, x_i)] = C(u, x_i) + \frac{1}{2} \text{tr} (C''(u, x_i) \Sigma) = l_i \quad (\text{C.6})$$

and thus

$$m_{kernel} = \sum_i \left( \int_{\mathcal{X}} K_{*i} \alpha_i p(X_q) dX_q \right) = \sum_{i=1}^N \alpha_i \left[ C(u, x_i) + \frac{1}{2} \text{tr} (C''(u, x_i) \Sigma) \right] = \sum_{i=1}^N \alpha_i l_i \quad (\text{C.7})$$

The explicit basis term can be computed for a linear explicit basis  $\phi(x) = \begin{pmatrix} 1 \\ x \end{pmatrix}$ ,

$$m_{basis} = \int_{\mathcal{X}} \phi(X_p)^T \beta p(X_q) dX_q = u^T \beta \quad (\text{C.8})$$

### Variance of propagated distribution

$$\begin{aligned}
v(u, \Sigma) &= \int_{\mathcal{X}} (\sigma_n + C(X_p, X_p) - K_* S K_*^T) p(X_q) dX_q \\
&\quad + \int_{\mathcal{X}} (\phi(X_p)^T (M^{-1} + \beta \beta^T) \phi(X_p)) p(X_q) dX_q \\
&\quad - \int_{\mathcal{X}} (K_* E \phi(X_p) - \phi(X_p)^T E^T K_*^T) p(X_q) dX_q \\
&\quad - m(u, \Sigma)^2
\end{aligned}$$

The first term contains only the kernel, the second only the basis and the third is a mixed term.  $m(u, \Sigma)^2$  can also be decomposed into such three term, such that

$$m(u, \Sigma)^2 = m_{kernel}^2 + m_{basis}^2 - 2 m_{kernel} m_{basis} \quad (\text{C.9})$$

By combining both equations, we get

$$v(u, \Sigma) = v_{kernel}(u, \Sigma) + v_{basis}(u, \Sigma) - v_{mixed}(u, \Sigma)$$

For the kernel term, the integral can be transformed into an expected value

$$\begin{aligned}
\int_{\mathcal{X}} (\sigma_n + C(X_p, X_p) - K_* S K_*^T) p(X_q) dX_q &= E_x [\sigma_n + C(X_p, X_p) - K_* S K_*^T] \\
&= \sigma_n + E_x [C(x, x)] - \sum_{i,j=1}^N S_{ij} E_x [C(x, x_i) C(x, x_j)]
\end{aligned}$$

The expected values are computed similarly to the development of  $E_x [C(x, x_i)]$ , as follows,

$$\begin{aligned}
E_x [C(x, x)] &= E_x [C(u, u)] + E_x [(x - u)^T C'(u, u)] + E_x \left[ \frac{1}{2} (x - u)^T C''(u, u) (x - u) \right] \\
&= C(u, u) + \frac{1}{2} \text{tr} (C''(u, u) \Sigma)
\end{aligned}$$

$$\begin{aligned}
E_x [C(x, x_i)C(x, x_j)] &= E_x [C(x, x_i)C(x, x_j)] + E_x [(x - u)^T (C(u, x_i)C'(u, x_j) + C'(u, x_i)C(u, x_j))] \\
&\quad + E_x \left[ \frac{1}{2}(x - u)^T (C(u, x_i)C''(u, x_j) + C''(u, x_i)C(u, x_j)) (x - u) \right] \\
&\quad + E_x [(x - u)^T C'(u, x_i)(x - u)^T C'(u, x_j)]
\end{aligned}$$

where terms of order 3 or higher were discarded (equivalent to discarding terms of higher order than  $\Sigma$ ). Thus

$$\begin{aligned}
E_x [C(x, x_i)C(x, x_j)] &= C(u, x_i)C(u, x_j) + \frac{1}{2}C(u, x_i)\text{tr}(C''(u, x_j)\Sigma) + \frac{1}{2}C(u, x_j)\text{tr}(C''(u, x_i)\Sigma) \\
&\quad + \text{tr}(C'(u, x_i)C'(u, x_j)^T\Sigma)
\end{aligned}$$

By introducing these terms into the expression of  $v_{kernel}$  and subtracting  $m_{kernel}^2$ , we get

$$\begin{aligned}
v_{kernel}(u, \Sigma) &= \sigma_n + C(u, u) + \frac{1}{2}\text{tr}(C''(u, u)\Sigma) - \sum_{i,j=1}^N S_{ij}\text{tr}(C'(u, x_i)C'(u, x_j)^T\Sigma) \\
&\quad - \sum_{i,j=1}^N (S_{ij} + \alpha_i\alpha_j) C(u, x_i)C(u, x_j) \\
&\quad - \frac{1}{2}\sum_{i,j=1}^N (S_{ij} + \alpha_i\alpha_j) [C(u, x_i)\text{tr}(C''(u, x_j)\Sigma) + C(u, x_j)\text{tr}(C''(u, x_i)\Sigma)]
\end{aligned}$$

A similar method is used for the explicit basis term, such that

$$\begin{aligned}
v_{basis}(u, \Sigma) &= \int_{\mathcal{X}} (\phi(X_p)^T (M^{-1} + \beta\beta^T)\phi(X_p)) p(X_q) dX_q - m_{basis}^2 \\
&= \text{tr}(M^{-1}\Sigma) + u^T M^{-1}u + \beta^T \Sigma \beta
\end{aligned}$$

Finally, for the mixed term

$$v_{kernel}(u, \Sigma) = 2 \int_{\mathcal{X}} K_* E \phi(X_p) p(X_q) dX_q - 2 m_{kernel} m_{basis} \quad (\text{C.10})$$

For a linear explicit basis  $\phi(x) = \begin{pmatrix} 1 \\ x \end{pmatrix}$  and noting  $E = (E_1 \ E_2)$

$$\int_{\mathcal{X}} K_* E \phi(X_p) p(X_q) dX_q = \sum_{i=1}^N (E_{1i} E_x [C(x, x_i)] + E_{2i} E_x [C(x, x_i) x]) \quad (\text{C.11})$$

The first expected value was already computed and noted as  $l_i$ . The second term can be computed in the same manner, by using the Taylor approximation of  $x C(x, x_i)$ . Following this method, we obtain

$$E_x [C(x, x_i) x] = u \left( C(u, x_i) + \frac{1}{2}\text{tr}(C''(u, x_i)\Sigma) \right) + \text{tr}(C'(u, x_i)\Sigma) = u l_i + \text{tr}(C'(u, x_i)\Sigma) \quad (\text{C.12})$$

Thus, using matrix form, the mixed term can be written as

$$v_{mixed}(u, \Sigma) = 2 \sum_{i=1}^N (E_1 \ E_2) \begin{pmatrix} l_i \\ u l_i + \text{tr}(C'(u, x_i)\Sigma) \end{pmatrix} - 2 m_{kernel} m_{basis} \quad (\text{C.13})$$

## D Implementation Guide

This appendix details the important scripts needed to run the project. The implementation is programmed in Python and uses ROS to communicate between scripts. A terminal running *roscore* must be running. All the scripts are part of the *plane\_control* package. The computer must be connected to the OptiTrack camera tracking system using an EtherCat cable and to the RC controller using the PCTx cable.

### Scripts for setting the environment:

`forward_pos_to_drone.py`: Reads the positioning measurements from the tracking system and publishes them into topic */CameraStream*.

`NonUnifEstimator_car.py`: Estimates the position, velocity,  $\gamma$  as well as the derivatives of  $\gamma$  and the Euler angles.

`EulerAnglesEstimator.py`: Estimates the Euler angles from the position and orientation.

`rosptx.py`: Opens communication to the RC controller through the PCTx cable.

`MPCPlotter.py`: Online plotting while the controller is running. Plots the trajectory as well as many other plots depending on the enabled flags.

### Scripts for running the controllers

`RunMPC_car.py`: Runs the MPFC controller. The used scheme can be chosen by importing from different files. Flags for `isSim`, `isAdaptive` and `isStochastic`.

`PIDSteering.py`: Runs the low level tracker and send steering commands to the RC controller. Flags for `isSim`, `usePID` (if disabled, the steering commands sent are calculated directly by the MPC controller based on  $u_\gamma^*$ ).

`simulation_ContinuousTime.py`: Runs the simulation to simulate the car.

### Additional scripts for MPC control

`adaptiveParameters.py`: Runs the online adaptation algorithm using recursive least squared (memoryless, representation II). Flag for `isSim`.

`adaptiveParameters_oldRep.py`: Runs the online adaptation algorithm using recursive least squared (memoryless, representation I). Flag for `isSim`.

`GP_adaptiveParameters`: Runs the online adaptation algorithm using Gaussian processes fitting (Differential wheels basis, no kernel). Flag for `isSim`.

`PID.py`: Defines a class for PID controllers.

`MPCNom_car.py`: Defines the nominal MPFC scheme. Can be imported by `RunMPC_car.py`.

`MPCAdaptive_gammaku.py`: Defines the adaptive MPFC scheme for the model  $\dot{\gamma} = k \cdot u_\gamma$ . Can be imported by `RunMPC_car.py`.

`MPCAdaptive_gammaARX.py`: Defines the adaptive MPFC scheme for the model  $\dot{\gamma} = k_0 + k_1 \cdot u_\gamma$ . Can be imported by `RunMPC_car.py`.

`MPCStochastic.py`: Defines the stochastic MPFC scheme. Can be imported by `RunMPC_car.py`. Location of GP models to import are defined.

### Gaussian Processes fitting

`GP_Params_BagView.py`: Fits Gaussian processes to stores data using various methods.

`FitGP.py`: Functions to fit Gaussian processes.

### **Bag file Analysis**

MPCBagView.py: Analyzes bag files from running the MPC and plots all the needed plots (depending on enabled flags).

Other bags analysis files are available.

### **Miscellaneous**

RK4.py: Implements the RK4 integrator for both normal and uncertain dynamics.

FiniteDifferencesFunctions.py: Functions to calculate Finite Differences.

ReferencePath.py: Plots the reference path and  $\dot{\gamma}$ .

basicController.py: Basic controller sending commands to the RC.



## References

- [1] Uwe Ahrens, Moritz Diehl, and Roland Schmehl. *Airborne wind energy*. Springer Science & Business Media, 2013.
- [2] Miles L Loyd. Crosswind kite power (for large-scale wind power production). *Journal of energy*, 4(3):106–111, 1980.
- [3] Boris Houska and Moritz Diehl. Optimal control for power generating kites. In *Control Conference (ECC), 2007 European*, pages 3560–3567. IEEE, 2007.
- [4] Boris Houska and Moritz Diehl. Robustness and stability optimization of power generating kite systems in a periodic pumping mode. In *Control Applications (CCA), 2010 IEEE International Conference on*, pages 2172–2177. IEEE, 2010.
- [5] Michael Erhard, Greg Horn, and Moritz Diehl. A quaternion-based model for optimal control of an airborne wind energy system. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 97(1):7–24, 2017.
- [6] Timm Faulwasser and Rolf Findeisen. A model predictive control approach to trajectory tracking problems via time-varying level sets of lyapunov functions. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 3381–3386. IEEE, 2011.
- [7] Sanket Diwale, Andrea Alessandretti, Ioannis Lymperopoulos, and Colin N Jones. A nonlinear adaptive controller for airborne wind energy systems. In *American Control Conference (ACC), 2016*, pages 4101–4106. IEEE, 2016.
- [8] Timm Faulwasser and Rolf Findeisen. Nonlinear model predictive control for constrained output path following. *IEEE Transactions on Automatic Control*, 61(4):1026–1039, 2016.
- [9] Sanket Sanjay Diwale, Timm Faulwasser, and Colin Jones. Model Predictive Path-Following Control for Airborne Wind Energy Systems. In *20th IFAC World Congress*, number EPFL-CONF-223679, 2017.
- [10] Inc. DBA OptiTrack NaturalPoint. Optitrack. <http://optitrack.com/>, 2017. URL <http://optitrack.com/>.
- [11] LLC Horizon Hobby. Dx6i 6-channel full range w/o servos md2. <http://www.spektrumrc.com/products/default.aspx?prodid=spm6600>. URL <http://www.spektrumrc.com/products/default.aspx?prodid=spm6600>.
- [12] Endurance R/C LLC. Pctx. <http://www.endurance-rc.com/pctx.php>, 2016. URL <http://www.endurance-rc.com/pctx.php>.
- [13] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots (Intelligent Robotics and Autonomous Agents series)*, chapter 5.6, pages 227–238. The MIT Press, 2004. ISBN 0-262-19502-X. URL <https://www.amazon.com/Introduction-Autonomous-Mobile-Intelligent-Robotics/dp/026219502X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=026219502X>.
- [14] M K Bowen and Ronald Smith. Derivative formulae and errors for non-uniformly spaced points. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 461, pages 1975–1997. The Royal Society, 2005.
- [15] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes: the art of scientific computing*, chapter 17.1 Runge-Kutta Method. Cambridge University Press, New York, 2007. ISBN 978-0-521-88068-8. URL <https://www.amazon.com/NUMERICAL-RECIPES-WILLIAM-H-PRESS/dp/B0073KC050?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B0073KC050>.
- [16] Georgios B Giannakis. Signal Processing Tools for Big Network Data Analytics. Lecture, February 2017.

- [17] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.
- [18] Ioan Doré Landau, Rogelio Lozano, Mohammed M’Saad, and Alireza Karimi. *Adaptive control: algorithms, analysis and applications*. Springer Science & Business Media, 2011.
- [19] Bing Zhu and Xiaohua Xia. Adaptive Model Predictive Control for Unconstrained Discrete-Time Linear Systems With Parametric Uncertainties. *IEEE Transactions on Automatic Control*, 61(10):3171–3176, 2016.
- [20] C E Rasmussen and C K I Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 026218253X. URL [http://www.ebook.de/de/product/5192092/car1\\_edward\\_rasmussen\\_gaussian\\_processes\\_for\\_machine\\_learning.html](http://www.ebook.de/de/product/5192092/car1_edward_rasmussen_gaussian_processes_for_machine_learning.html).
- [21] Carl Edward Rasmussen. Gaussian processes for machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.
- [22] Milan Korda, Ravi Gondhalekar, Frauke Oldewurtel, and Colin N Jones. Stochastic mpc framework for controlling the average constraint violation. *IEEE Transactions on Automatic Control*, 59(7):1706–1721, 2014.
- [23] Paul J Goulart, Eric C Kerrigan, and Jan M Maciejowski. Optimization over state feedback policies for robust control with constraints. *Automatica*, 42(4):523–533, 2006.
- [24] Sanket Sanjay Diwale. *Gaussian Processes for System Identification, Control and Optimisation*. 2015.
- [25] Agathe Girard. *Approximate methods for propagation of uncertainty with gaussian process models*. phdthesis, University of Glasgow, October 2004.
- [26] Thomas B Schön and Fredrik Lindsten. Manipulating the multivariate gaussian density. *Division of Automatic Control, Linköping University, Sweden, Tech. Rep*, 2011.