Università degli Studi di Padova

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di Laurea Magistrale in Ingegneria Meccatronica

Ball and Plate MPC Control of a 6 DOF Stewart Platform

Relatore:
Prof. Roberto Oboe

Laureando:
Federico Dal Cero
1236507

Anno Accademico 2021/2022

Abstract

MPC is a cutting-edge control technique which is nowadays deeply studied, since it permits to overcome several disadvantages related to simpler and more traditional control methods. In industry, Proportional Integral Derivative controllers are still widespread, due to their low complexity and tuning procedure. However, they do not have the capability to cope with coupled multi-input multi-output systems, as well as constrained and non-linear ones. On the other hand, Model Predictive Control is able to deal with these features at the price of computational complexity. The latter is related to its structure: MPC is indeed an optimization-based control technique which require to solve an optimization problem within every timeslice, and this implies high computational burden. Therefore, its use has been so far mainly limited to academic applications or chemical processes, where the timescale is quite large (around the minute). However, recent advances in both embedded hardware and MPC software libraries have made Model Predictive Control a viable candidate also for systems with fast dynamics, e.g. robotics. With regard to this, the objective of this master thesis is to explore the suitability of Model Predictive Control for motion control applied to fast-timescale systems. The hardware under test consists in a Stewart platform, i.e., a robotic system governed by six servo-actuators which can move a flat plate. The implemented controller aims at balancing a ball towards the middle of the plate.

The used MPC library is based on acados, a collection of solvers for fast embedded optimization. Although it is implemented in C, acados is able to interface with higher-level interfaces, such as Matlab, Python and C++, feature which makes acados really flexible. On the other hand, this library provides also efficiency, as its core is written in a low-level language.

The project firstly develops simulations by the use of Matlab, in order to observe the feasibility of MPC in terms of computational time and capability to control the system. In a second time, the hardware is employed to verify the simulation results. This includes the setting of the communication in accordance to the serial protocol developed for the embedded systems platform within the hardware.

Acknowledgements

I would like to express a special thank to my academic supervisor, Roberto Oboe, for his support and guidance throughout the whole project. His suggestions during the meetings we had every month were crucial for every positive outcome of my master thesis.
Furthermore, I am profoundly grateful to my supervisor Caspar Grujithuijsen, who has been a great guide in teaching both technical and organizational aspects and allowing me to grow professionally.
Moreover, a big thank certainly goes to the company I worked for during my master thesis, Sioux Technologies. Their trust in me had allowed me to make a great experience in a different country as the Netherlands is, as well as to experience the dynamic and cutting-edge working environment which the city of Eindhoven is able to offer.
Eventually, I would like to thank all my professors of Mechatronic Engineering at the Department of Management and Engineering in Vicenza: they have been essential in teaching me several academic and human concepts for my academic and work success.

Dedication

This work is dedicated to my family: my mother Alessandra, my father Maurizio and my brother Davide; my grandparents, uncles and aunts and my cousins. They have always been an unconditioned and great support to me in every situation, both academic and personal.
I will always be profoundly grateful to them.

Federico Dal Cero
Vicenza, Italy
April 14, 2022

Contents

List of Figures

List of Tables

Introduction

Nowadays, manufacturing operations are quickly taking the direction of modernization and upgrade in order to meet the demand for improving quality and just-in-time delivery. With regard to these requirements, a strong transition is taking place from manual to automated processes. *Motion control* is crucial for this shift, since it refers to that sub-field of automation whose goal is to move the singular parts of a machine in a controlled way. Its main fields of application are precision engineering, bio and nanotechnology, as well as plants such as production lines, where robotics and automatic machines are involved.
Motion control relates to a set of individual components that together contribute to the building of a controlled movement in machines. A first essential component is the controller, i.e., an electronic-related device that implements the code to control the motors and drives of the machines. According to the complexity of the plant, the number of controllers may vary. Each controller is fed with instructions from the main computer and provides the latter with a feedback response. A second part of motion control is comprised by the motors, which can assume different characteristics and applications. Nevertheless, their main function is to turn the received inputs into motion. One of the most widespread types of motors is the servo, which provides high accuracy to control angular motion. In addition, a linear actuator is often used to convert the rotational motion of a motor into linear motion, which is needed e.g. in production lines. Finally, motor drives act as a converting element between the controller and motors. They receive the electrical signal from the controller and feed the motor with a reinterpreted power signal. Moreover, other components like sensors and cabling are needed.
Motion control may be based on open or closed loop. The former does not implement a feedback, so that the controller is only aware of the reference without knowing the real motion of the final element. On the other hand, closed loop represents a much more widespread control category, since it is able to achieve better accuracy and disturbances rejection due to the feedback provided by a

sensor and/or an estimator.
Control of robotic manipulators is a significant part of motion control, since the greatest part of robotic manipulators are driven by electrical servos, which represent thus the final actuator comprising the motion control part. The following subsection will focus on motion control dealing with robotic applications.

Motion Control in Robotics

Recently, robotics has been becoming a crucial part of composite systems often responsible to carry out complex tasks, such as robotic-assisted surgeries or production lines. The latter example often comprises a pick-and place manipulator responsible for manipulating objects on a line. In this context, motion control of robotic components enables the correct development of these complex tasks, provided that the robot is equipped with the proper end effector and motion law. The definition of the combined actions between the two of them to achieve the correct way of solving the task is rarely trivial. The end effector is responsible for executing the work on the manipulated element, while the joints of the robot are controlled through motion control. The established relationship between these two is often complex and may require several equations and control solutions which are only solvable in an iterative way. Moreover, many robots possess more degrees of freedom than necessary, resulting in kinematically redundant robots. This feature increases both the power and the struggle in controling them. The difficulty of control arises from the infinite number of ways to solve a task, for instance the execution of a point-to-point path. To ensure the correctness of the choice, a possibility is the inclusion of path constraints in it, which nonetheless makes the motion control part much more complex and advanced. Robotic joints have furthermore speed and acceleration limits, while actuators have upper and lower bounds for torque and force. The combination of the latter constraints, related to the power of the single devices, with the path constraints, renders robot motion control quite challenging and gives rise to the need of relying on cutting-edge control techniques.
Because of the previous considerations, a *Stewart platform* has been considered as the hardware under test. This is indeed a robotic parallel manipulator with 6 degrees of freedom, even though the final task is to balance a ball in the middle of the upper plate, which requires only two degrees of freedom (the angles α, β in the $x-$, $y-$ axes). This gives rise to several possibilities of achieving the

two desired angles starting from a previous couple of them. Furthermore, the path to reach the center of the plate starting from a different position is to be constrained: the plate surface has indeed spacial limits and the ball has to always remain attached to it. Therefore, the task of balancing a metallic ball through a Stewart platform presents all the features that make the implementation of motion control in robotics challenging and computationally demanding.
The next section will focus on the control techniques which may be applied in robotics: previous works with respect to the Stewart platform will be summarized and the implemented plant will be presented.

Control Techniques in Robotics: Overview and Proposed Framework

Because of the increasing involvement of robotics within industrial processes, the application of control theory in this field is currently one of the most popular concerns in automation engineering. Therefore, several control techniques have been tested within the robotic field, with both positive and negative outcomes. Stewart platform represents a deeply utilized hardware to test that variety of techniques, since it is a rather simple and cheap device which provides easiness in its study and modelization. Nonetheless, it is capable of introducing the majority of the struggles usually related to control applied to the robotics field. A comparison between different methods of controlling a Stewart platform has been carried out by Kassem A., Haddad H., Albitar C. [1]. Their work models the system through the Euler-Lagrangian equations and takes into consideration 4 control methods: *proportional integral derivative regulator* (PID), *linear quadratic regulator* (LQR), *sliding mode controller* and *fuzzy controller.* The first one has been already mentioned within the abstract, since it is the golden standard in industry: it consists of following a reference by evaluating its error through a proportional, an integral and a derivative part. LQR is on the other hand an optimal control technique which is still deeply employed in industry: it requires a linear model of the system in state-space form and provides an input given two matrices Q, R which express the penalties on the state, input variables through a cost function $J(x(\cdot), u(\cdot))$. If Q is positive semi-definite, R is positive definite and the horizon length of the cost function is infinite, the output error and the optimal input are related through a time-invariant coefficient K: $u(k) = -K(y(k)-r(k))$. The third control technique, i.e., sliding mode control, falls within the category

of variable structure controllers, which are characterized by a suite of feedback control laws and a decision rule [2].In particular, sliding mode control provides the system with a discontinuous control signal: the state-feedback control may thus switch from one continuous structure to another, depending on the current values of the state space variables. Eventually, a fuzzy controller is applied to the platform. As the name suggests, a fuzzy control system is based on *fuzzy logic* and *fuzzy decision making* [3]: the main feature is here the degree of membership of the variables, which can contemporarely belong to different sets. Therefore, fuzzy control may be defined as a rule-based control technique where the transitions between different sets are not as well-established as in traditional rule-based techniques. The results reached by Kassem et alii highlights the validity of sliding mode control, thanks to the fact that the considered model is non-linear and dynamic. Therefore, the main concern is to implement a control technique which is able to correctly approximate the Stewart platform model, given that it is variable with the configuration of the plate. To tackle this issue, Oravec M. and Jadlovska A. proposed *Model Predictive Control* [4]. In their work, MPC faces the space-varying behavior of the system by introducing constraints which allow to linearize the model achieving sufficient consistency with the real model. The control algorithm is verified in Matlab and Simulink with positive results.
Given this overview about the already explored control techniques applied to the Stewart platform, the last mentioned control technique, i.e., MPC, has been chosen as the candidate to be applied to the real hardware provided by *Sioux Technologies.* After a deep study of the considered control technique, several simulation tests have been carried out in Matlab, with the help of the library *acados.* After the verification of its applicability and the implementation of a Kalman filter to estimate the full state vector, a Qt application implementing the controller and the communication has been developed in order to interact with the hardware. Eventually, the experimental tests have been performed. The next chapters will lead the reader throughout the steps of the project. Chapter 1 presents the Stewart platform in details: the equations of the ball-and-plate system and the inverse kinematics are here illustrated. Afterwards, Model Predictive Control is deeply explained through chapter 2: its general formulation and the algorithm are firstly explained, after which the stability and robustness issues are tackled. Moreover, the principal involved solvers are underlined. Chapter 3 introduces two estimators which can work in strict correlation with the MPC: the Kalman filter and the Moving Horizon Estimator. Model Predictive Control

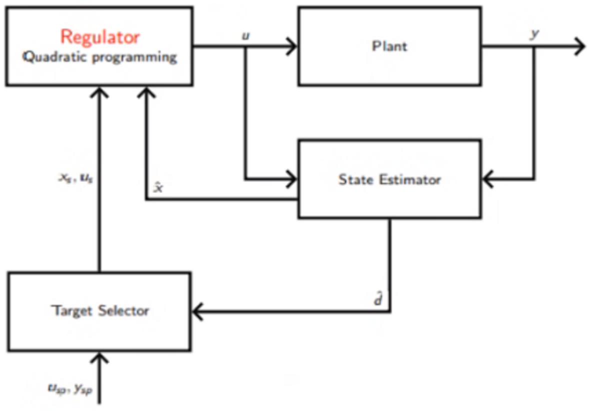


Figure 1: Block scheme structure of MPC and estimator communicating with the real plant

needs indeed to be provided with the full state vector, hence an estimation of certain components might be necessary, as fig. 1 illustrates. Subsequently, chapter 4 provides a description of the whole system: firstly, the hardware components are highlighted, then the serial communication protocol and the interaction with the hardware are explained; lastly, the software libraries and the design of MPC are presented. Chapter 5 and 6 illustrate respectively the simulation and experimental results. The former aims at verifying the feasibility of MPC and the validity of the approximations for the model, as well as the possibility to combine MPC with KF. The latter has the objective of validating the modelization of the hardware and the MPC applied to the real platform. Finally, the whole project is summarized and future works are proposed.

Chapter 1

Stewart Platform

The origin of the Stewart platform dates back to 1954, when the automotive engineer *Eric Gough* firstly designed it for tires testing. It consists of a parallel manipulator for positioning and motion control which is made of two parallel plates and a variable number of legs led by servo-motors. The legs connect the two plates and aim at moving and controling the upper plate through the servos. The base plate represents the static part of this device: it is indeed attached to the floor. The number and the kind of servo motors may differ: the used platform relies on six rotary servo actuators, as illustrated through the model in fig. 1.1. Since the six-legged is its most widespread design, the Stewart platform is usually also referred to as hexapod. The rotary actuators may complicate the inverse kinematics, i.e., the calculation of the servo motors position given the angles of the upper plate. This process will be further explained within section 1.2. Each servo is to move a leg which connects the static base plate and the moving upper plate.

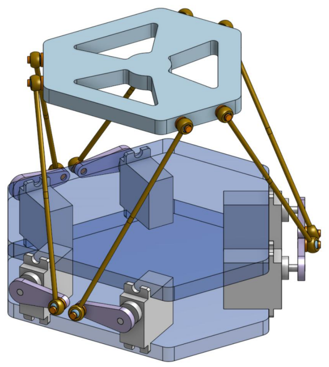


Figure 1.1: Stewart platform with 6 rotary servo-actuators

Stewart platform classification as a parallel manipulator is due to the different number of linkages between the upper and the base plate. A strong advantage regards the cumulative error: serial manipulators accumulate indeed the positioning error of each linkage, drawback which is not present within parallel manipulators. As previously mentioned, the Stewart platform was initially employed in the testing of tires. However, several different uses have arisen with time: nowadays this robotic device is proficiently used for medical and surgical applications, wearable items such as smart watches, training and entertainment simulators, engineering research applications and positioners' algorithm research. In our case, the Stewart platform aims at balancing a metallic ball through a cutting-edge control algorithm, i.e, Model Predictive Control, and thus combines the two last applications of the list. The balancing of the ball is made through the movement of the center point of the top plate in $3D$ space. To this end, every leg changes its configuration and thus its virtual length (which in case of a linear actuator corresponds to its real one). A desirable pose of the upper plate, i.e., the 6 degrees-of-freedom vector values of the center of this plate, is reached through the inverse kinematics, which calculates the position of the servos given the pose of the plate. Subsequently, every servo is controlled to achieve the required position. Therefore, devices placed on the top plate can be moved and controlled in the six degrees of freedom: the three linear movements x, y, z (lateral, longitudinal, vertical) and the three rotations x, y, z (pitch, roll, yaw).
The main issues of every Stewart platform are the accuracy of the center location and the response time against a position change command. The first critical point mainly deals with the correctness of the implemented inverse kinematics, which has to take into account extremely accurate values regarding the involved mechanical parameters, such as the lengths of the legs and the rotations of the servos. If not, errors can cumulate and lead to a real configuration which is different from the desired one. Furthermore, a position change command is to be put into action as fast as possible, so that the configuration and thus the required optimal position has not changed in the meantime. This mainly deals with applications where an object must be controlled on the upper plate: the command concerning the pose of the plate has to rely on a measurement of the object position which should be as updated as possible. These applications are referred to a further issue, i.e., the accuracy of the measured object position. This is entirely related to the measurement process implemented within the platform, which is usually a sensing camera or, as in the case under exam, a touch screen inside the plate.

The next section will focus on the modelization of the system representing the Stewart platform: first of all, the model of the ball and plate is obtained starting from the Euler-Lagrangian equation. Secondly, the inverse kinematics is presented to reach the required position of the servos given the pose of the upper plate.

1.1 Equations of the Ball-and-Plate System

The modelization may be done either by analyzing the forces and torques of the system through the *Newton-Euler formalism*, as in [5] for the $1D$ case and [6] for the $2D$, or through the *Euler-Lagrangian equation*, as in The system expressing a ball balancing table has been found here starting from the *Euler-Lagrangian equation*:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_i} - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_i \tag{1.1}$$

Here, q_i is the ith-direction coordinate, T is the kinetic energy of the system and V the potential one. Q is the external force, which is assumed to be zero. This preliminary system allows indeed to access the angles α, β directly. The chosen coordinates are x_b, y_b, i.e., the cartesian components of a system in the middle of the plate, and α, β, the angles which represent the inclination of the plate (α is around x while β around y). Furthermore, several assumptions are made: first of all, there is no slipping for the ball, which is homogeneous and symmetric. Moreover, the friction is neglected and the plate and the ball are always in contact. The energy equations need to be explicitly written:

$$T_b = \frac{1}{2}m_b\left(\dot{x}_b^2 + \dot{y}_b^2\right) + \frac{1}{2}I_b\left(\omega_x^2 + \omega_y^2\right) = \frac{1}{2}\left(m_b + \frac{I_b}{r_b^2}\right)\left(\dot{x}_b^2 + \dot{y}_b^2\right) \tag{1.2}$$

$$T_p = \frac{1}{2}\left(I_p + I_b\right)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + \frac{1}{2}m_b\left(x_b\dot{\beta} + y_b\dot{\alpha}\right)^2 \tag{1.3}$$

$$V = m_b g h = m_b g(x_b \sin\beta + y_b \sin\alpha) \tag{1.4}$$

Where T_b is the kinetic energy of the ball, T_p is the kinetic energy of the plate, V is the potential energy. Only the ball contributes to that last one. Therefore, the following equations are achieved:

$$\frac{\partial T}{\partial \dot{\alpha}} = (I_p + I_b)\dot{\alpha} + m_b x_b y_b \dot{\beta} + m_b y_b^2 \dot{\alpha} \tag{1.5}$$

$$\frac{\partial T}{\partial \dot{\beta}} = (I_p + I_b)\dot{\beta} + m_b x_b y_b \dot{\alpha} + m_b x_b^2 \dot{\beta} \tag{1.6}$$

$$\frac{\partial T}{\partial \dot{x}_b} = \left(m_b + \frac{I_b}{r_b^2} \right) \dot{x}_b \tag{1.7}$$

$$\frac{\partial T}{\partial \dot{y}_b} = \left(m_b + \frac{I_b}{r_b^2} \right) \dot{y}_b \tag{1.8}$$

Furthermore, chosen $L = V - T$, it results:

$$\frac{\partial L}{\partial \alpha} = m_b g y_b \cos\alpha \tag{1.9}$$

$$\frac{\partial L}{\partial \beta} = m_b g x_b \cos\beta \tag{1.10}$$

$$\frac{\partial L}{\partial x_b} = m_b g sin\beta + m_b x_b \dot{\beta}^2 + m_b y_b \dot{\alpha}\dot{\beta} \tag{1.11}$$

$$\frac{\partial L}{\partial y_b} = m_b g sin\alpha + m_b y_b \dot{\alpha}^2 + m_b x_b \dot{\alpha}\dot{\beta} \tag{1.12}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\alpha}} = (I_p + I_b)\ddot{\alpha} + m_b y_b^2 \ddot{\alpha} + 2 m_b y_b \dot{y}_b \dot{\alpha} + m_b x_b y_b \ddot{\beta} + m_b \dot{x}_b y_b \dot{\beta} + m_b x_b \dot{y}_b \dot{\beta} \tag{1.13}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\beta}} = (I_p + I_b)\ddot{\beta} + m_b x_b^2 \ddot{\beta} + 2 m_b x_b \dot{x}_b \dot{\beta} + m_b x_b y_b \ddot{\alpha} + m_b \dot{x}_b y_b \dot{\alpha} + m_b x_b \dot{y}_b \dot{\alpha} \tag{1.14}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{x}_b} = \left(m_b + \frac{I_b}{r_b^2} \right) \ddot{x}_b \tag{1.15}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{y}_b} = \left(m_b + \frac{I_b}{r_b^2} \right) \ddot{y}_b \tag{1.16}$$

The Euler Lagrange equations in scalar form are thus:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\alpha}} + \frac{\partial L}{\partial \alpha} = (I_p + I_b)\ddot{\alpha} + m_b y_b^2 \ddot{\alpha} + 2 m_b y_b \dot{y}_b \dot{\alpha} + m_b x_b y_b \ddot{\beta} + m_b \dot{x}_b y_b \dot{\beta} + m_b x_b \dot{y}_b \dot{\beta} + m_b g y_b \cos\alpha = 0 \tag{1.17}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\beta}}+\frac{\partial L}{\partial \beta} = (I_p+I_b)\ddot{\beta}+m_b x_b^2\ddot{\beta}+2m_b x_b\dot{x}_b\dot{\beta}+m_b x_b y_b\ddot{\alpha}+m_b\dot{x}_b y_b\dot{\alpha}+m_b x_b\dot{y}_b\dot{\alpha}+m_b g x_b\cos\beta = 0 \tag{1.18}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{x}_b}+\frac{\partial L}{\partial x_b} =\left(m_b+\frac{I_b}{r_b^2}\right)\ddot{x}_b+m_b g sin\beta+m_b x_b\dot{\beta}^2+m_b y_b\dot{\alpha}\dot{\beta}=0 \tag{1.19}$$

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{y}_b}+\frac{\partial L}{\partial y_b} =\left(m_b+\frac{I_b}{r_b^2}\right)\ddot{y}_b+m_b g sin\alpha+m_b y_b\dot{\alpha}^2+m_b x_b\dot{\alpha}\dot{\beta}=0 \tag{1.20}$$

As previously mentioned, the angles α, β, or rather their accelerations, are supposed to be directly accessible, therefore the forces Q_i are set to zero. The non-linear system is reached by imposing the following state and input vectors:

$$x=\left[x_b\ \dot{x}_b\ \alpha\ \dot{\alpha}\ y_b\ \dot{y}_b\ \beta\ \dot{\beta}\right]^T \tag{1.21}$$

$$u=\left[\ddot{\alpha}\ \ddot{\beta}\right]^T \tag{1.22}$$

The components of the vector $\dot{x}=\left[\dot{x}_b\ \ddot{x}_b\ \dot{\alpha}\ \ddot{\alpha}\ \dot{y}_b\ \ddot{y}_b\ \dot{\beta}\ \ddot{\beta}\right]^T$ are expressed by the equations below:

$$\ddot{x}_b=\frac{1}{m_b+I_b/r_b^2}\left(m_b x_b\dot{\beta}^2+m_b y_b\dot{\alpha}\dot{\beta}+m_b g\sin\beta\right) \tag{1.23}$$

$$\ddot{y}_b=\frac{1}{m_b+I_b/r_b^2}\left(m_b y_b\dot{\alpha}^2+m_b x_b\dot{\alpha}\dot{\beta}+m_b g\sin\alpha\right) \tag{1.24}$$

and thus:

$$\dot{x}_1=x_2,\ \dot{x}_3=x_4,\ \dot{x}_4=u_1,\ \dot{x}_5=x_6,\ \dot{x}_7=x_8,\ \dot{x}_8=u_2 \tag{1.25}$$

$$\dot{x}_2=\frac{1}{m_b+I_b/r_b^2}\left(m_b x_1 x_8^2+m_b x_5 x_4 x_8+m_b g\sin x_7\right) \tag{1.26}$$

$$\dot{x}_6=\frac{1}{m_b+I_b/r_b^2}\left(m_b x_5 x_4^2+m_b x_1 x_4 x_8+m_b g\sin x_3\right) \tag{1.27}$$

The system above is quite complicated and can be simplified withouth much loss of generality. Centrifugal forces related to the angles rates are indeed usually much smaller than the gravity component. Hence, the equations expressing $\ddot{x}_b$, $\ddot{y}_b$ are:

$$\ddot{x}_b = \frac{1}{m_b + I_b/r_b^2}\big(m_b g \sin\beta\big) \tag{1.28}$$

$$\ddot{y}_b = \frac{1}{m_b + I_b/r_b^2}\big(m_b g \sin\alpha\big) \tag{1.29}$$

The two equations are now decoupled, which means they can be treated as two separate SISO systems.
In addition, the assumption of small angles, e.g. $|\alpha| \leq 10^o$, $|\beta| \leq 10^o$, leads to a linear system:

$$\ddot{x}_b = \frac{1}{m_b + I_b/r_b^2}\big(m_b g \beta\big) \tag{1.30}$$

$$\ddot{y}_b = \frac{1}{m_b + I_b/r_b^2}\big(m_b g \alpha\big) \tag{1.31}$$

The linearized system in state space form is the following:

$$x = \begin{bmatrix} x_b & \dot{x}_b & y_b & \dot{y}_b \end{bmatrix}^T, u = \begin{bmatrix} \alpha & \beta \end{bmatrix}^T \tag{1.32}$$

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 0 & 0 \\ 0 & \frac{m_b g}{m_b + I_b/r_b^2} \\ 0 & 0 \\ \frac{m_b g}{m_b + I_b/r_b^2} & 0 \end{bmatrix} \cdot u \tag{1.33}$$

The moment of inertia of a solid ball of mass m and radius r is $\frac{2}{5}mr^2$ and thus $m_b + I_b/r_b^2 = \frac{7}{5}m_b$. The linearized 4×4 system is finally:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 0 & 0 \\ 0 & \frac{5}{7}g \\ 0 & 0 \\ \frac{5}{7}g & 0 \end{bmatrix} \cdot u \tag{1.34}$$

1.2 Inverse Kinematics Analysis

Since the inputs of the previously achieved system are the angles of the plate α, β, inverse kinematics analysis is needed to establish the required angles of the six servo-motors [7], [8].
The Stewart Platform makes use of 6 rotary actuators, which rely on servo-motors

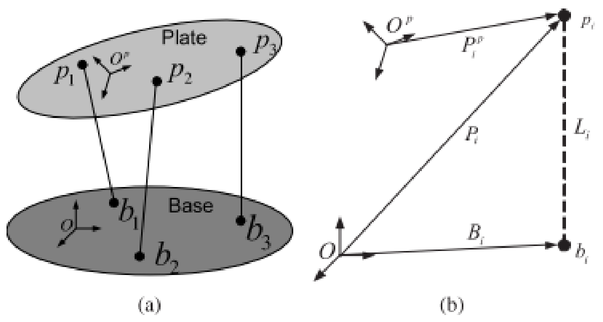


Figure 1.2: Schematic Illustration of a Stewart Platform. The coordinates of the system and the vectors used for the inverse kinematics are illustrated

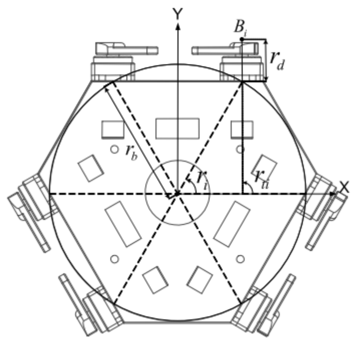


Figure 1.3: Illustration of the angles and the vector B_i

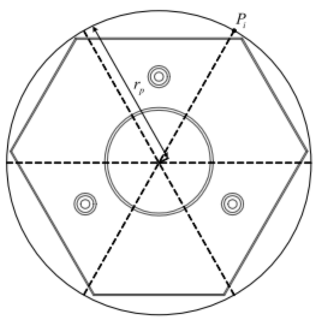


Figure 1.4: Upper plate and illustration of the vector P_i

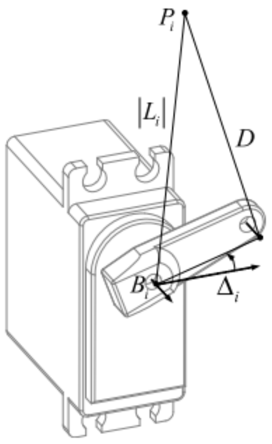


Figure 1.5: Virtual leg L_i: illustration of the variable and constant parameters

and servo-horns. The rotation and translation matrixes R_b, T_b vary in accordance to the angles and position of the plate:

$$R_b = \begin{bmatrix} \cos\beta & \sin\alpha\sin\beta - \cos\alpha & \cos\alpha\sin\beta + \sin\alpha \\ 0 & \cos\alpha & -\sin\alpha \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}, T_b = \begin{bmatrix} x_{oP} \\ y_{oP} \\ z_{oP} \end{bmatrix} \tag{1.35}$$

The position vector of joints attached to the plate is thus $P_i = R_b P_i^p + T_b$, while the length of the virtual legs (real if prismatic joints) is $L_i = P_i - B_i$. Here, P_i^p is the position vector related to the plate reference system, while P_i is referred to the base reference frame. B_i refers to the rotation center of each servo motor. These vectors are illustrated within fig. 1.2. We can express $B_i = [r_b\cos r_i + r_d \sin r_{ti} \;\; r_b \sin r_i + r_d \cos r_{ti} \;\; 0]^T$, $P_i^p = [r_p \cos r_i^p \;\; r_p \sin r_i^p \;\; 0]^T$ where r_b is the radius of the circle which touches every servo-motor, r_i is the angle between the x-axis and the line for the origin and the contact between the base and the center line of the servo motor. Moreover, r_{ti} is the angle of the servo motor, r_d is the distance between the end of the base and the center of the joint attached to the horn, r_p is the distance between the center of the joint and the origin of the plate, r_i^p is the angle of the joints attached to the plate. These quantities are better visualized in fig. 1.3, 1.4.

The length of the equivalent prismatic actuators is therefore achieved. However, the hardware requires the achievement of the rotation angles of the servo motors, which are to be calculated starting from L_i. The position vector of the joints attached to the horns is $M_i = B_i + R_z(r_t)R_y(-\Delta i) \cdot [|R_m| \; 0 \; 0]^T$, where R_m is the length of the horn. The well-known y and z rotation matrixes are:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}, R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.36}$$

Δ_i is the angle of the i-th servo motor, the variable of interest. From the explicit expressions of $R_y(\theta)$, $R_z(\theta)$, M_i can be rewritten:

$$M_i = \begin{bmatrix} |R_m| \cos\Delta_i \cos r_t + x_i \\ |R_m| \cos\Delta_i \sin r_t + y_i \\ |Rm| \sin\Delta_i + z_i \end{bmatrix} \tag{1.37}$$

Moreover, as illustrated in fig. 1.5, the lengths R_m, D, $|L_i|$ follow the relationships below:

$$\begin{aligned} R_m^2 &= (M_i - B_i)^T (M_i - B_i) \\ D^2 &= (P_i - M_i)^T (P_i - M_i) \\ |L_i|^2 &= (P_i - B_i)^T (P_i - B_i) \end{aligned} \tag{1.38}$$

where D is the length of the leg. From 1.37 and 1.38 an implicit expression of Δ_i can be reached:

$$c_i = a_i \sin\Delta_i + b_i \cos\Delta_i \tag{1.39}$$

where:

$$\begin{aligned} a_i &= 2|R_m|(z_i^p - z_i) \\ b_i &= 2|R_m|((x_i^p - x_i)\cos r_{ti} + (y_i^p - y_i)\sin r_{ti}) \\ c_i &= |L_i|^2 - D^2 + R_m^2 \end{aligned} \tag{1.40}$$

Finally:

$$\Delta_i = \sin^{-1}\left(\frac{c_i}{\sqrt{a_i^2 + b_i^2}}\right) - atan2\big(b_i, a_i\big) \tag{1.41}$$

Chapter 2

Model Predictive Control

As already underlined, several robotic systems, among which the Stewart platform, are kinematically redundant and therefore not trivial to control. Constraints are one of the possibilities to guarantee a proper control choice. Furthermore, the achieved model of the system is in a state space form. As a consequence, a control technique able to explicitly deal with constraints and state space systems should be chosen. Finally, optimization-based techniques have shown their ability for automatic control in several circumstances [9]. The main candidate becomes therefore Model Predictive Control, whose capability to control a Stewart platform has been already proven in several projects. MPC is an advanced control technique which is nowadays of great interest. Its main limit is the high computational effort it requires, feature that has limited its deployment in industry and confined its use mainly to academic applications. So far, industrial applications of MPC have been restricted to slow varying processes, such as chemical plants. Latter's time steps are indeed long enough to easily allow the completion of the control algorithm. MPC requires in fact to solve an optimization problem based on the model of the system within every time slice. Moreover, it takes into account the values of the state vector as the initial condition. Therefore, when the full state is not measured, MPC is combined with an estimate procedure, e.g. Kalman filtering or Moving Horizon Estimator. The latter represents the dual of MPC for estimates, as Kalman filter is for linear quadratic regulator. Although MPC has been successfully applied to slowly variable systems, recent improvements in embedded hardware and software libraries make it a valuable alternative for motion control applied to high-dynamics plants, where the time scale is much faster. Indeed, while chemical plants have time scales of minutes, motion applications require time scales of seconds. Therefore, the capability of controlling them by the use of MPC requires high-performance embedded hardwares and fast embedded solvers.

2.1 Model Predictive Control Details

2.1.1 Overview

[10] provides an extensive overview of Model Predictive Control, including its formulation, advantages and disadvantages, design choices, stability and robustness. Model Predictive Control has an optimal control formulation, therefore it handles systems represented through a state space form. A cost function J is optimized which takes into account the state vector and the input of the system within a finite horizon N. Moreover, it relies on a model of the system, which may be either linear or non-linear. The first case leads to linear MPC, while the second gives rise to the more complex non-linear MPC. Regardless the terminology, MPC is always a non-linear control method, since it deals with system constraints. This makes Model Predictive Control a non-exact control technique, meaning that the achieved solutions may be only sub-optimal. Model Predictive Control is also known as a combination of open loop and closed loop control. The first is due to its reliance on a model of the system. This is to be as accurate as possible, in order to not degrade the performance of the control technique. Therefore, MPC internally optimizes an open loop finite horizon control problem. Nevertheless, the cost function takes into account the latest available values of the state vector to compensate for any existing mismatch between the real and the nominal model. There is therefore a state feedback. Indeed, if a feedback did not happen, any difference between the model and the real system would create time-increasing errors between the actual and the predicted state components. On the other hand, the availability of the latest measured or estimated state vector allows to introduce some simplifications for the nominal model. For instance, a non-linear system might be linearized around the working point, so that the computational effort required by MPC decreases.

2.1.2 Pros and Cons

MPC has the objective to overcome several limitations related to more classical and simpler control methods, such as PID or optimal control. First of all, it can handle both input and state constraints. This is of great importance in several

applications, especially in the field of motion control. For instance, in robotics, the planned path of a robotic arm is always to be confined inside the workspace, and this may be easily handled by setting constraints within the cartesian coordinates. With regard to the system under test, the dimension of the plate provides some constraints for the position of the ball, which must always be attached to the plate. A second advantage of MPC is its capability to cope with coupled MIMO systems, which nonetheless is not the case of the simplified system taken into consideration. Furthermore, this control technique can push the plant to its limits of performance. This is related to its predictive behavior: the cost function J considers the state and input references within a finite time window, starting from the current time instant, therefore a change of one of the reference components is detected N time slices before and the provided inputs consider this in advance. On the contrary, a PID controller relies only on the current error between the references and the actual variables: a future reference is not considered, hence an error needs to emerge before the PID changes the system input. Therefore, PID does not have any predictive behaviour and this makes the reference tracking less effective.
The complexity of MPC gives nevertheless birth to some disadvantages, e.g. its high computational burden. The capability to handle system constraints, together with the optimization problem which is solved during every time-slice, gives rise indeed to a high computational complexity if compared to PID or linear quadratic regulator. Hence, plants involving MPC call for extremely efficient optimization solvers and libraries.
The next subsections will deeply dive into these concepts.

2.1.3 Formulation

Model Predictive Control optimizes the following cost function J:

$$\begin{aligned} J\big(x(\cdot),u(\cdot)\big)=&\sum_{i=1}^{N}\big(x(k+i)-x_{ref}(k+i)\big)^T Q\big(x(k+i)-x_{ref}(k+i)\big)+\\ &\sum_{i=0}^{N-1}\big(u(k+i)-u_{ref}(k+i)\big)^T R\big(u(k+i)-u_{ref}(k+i)\big) \end{aligned} \tag{2.1}$$

where $x(\cdot)$ is the state space vector and $u(\cdot)$ is the input vector of the system. The system is thus to be expressed in a state space form. Even though MPC is able to deal with non-linear systems, an effort towards achieving a linear system in the following form should be carried on:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \tag{2.2}$$

Here, A, B, C are respectively the state, input and output matrices, while x, u, y are the state, input and output vectors.
Moreover, the matrices Q and R handle the different weights regarding the state and input components. As it can be inferred by 2.1, MPC optimizes the trajectory of the state and input components within a finite time horizon $N \cdot T$, where T is the period of the discretized system. Furthermore, this technique requires an accurate model of the system to be able to provide high quality performances. The whole optimization problem solved by MPC can be summarized as follows:

$$\begin{aligned} J\big(x(\cdot), u(\cdot)\big) = &\sum_{i=1}^{N} \big(x(k+i) - x_{ref}(k+i)\big)^T Q \big(x(k+i) - x_{ref}(k+i)\big) + \\ &\sum_{i=0}^{N-1} \big(u(k+i) - u_{ref}(k+i)\big)^T R \big(u(k+i) - u_{ref}(k+i)\big) \end{aligned} \tag{2.3}$$

subject to:

$$\dot{x} = f(x, u), \quad x \in X, \quad u \in U \tag{2.4}$$

The function $\dot{x} = f(x, u)$ represents the dynamics of the system in a state space form. Furthermore, X and U are sets of allowable values for the state and input vectors x, u, taking into account the system constraints.
As visible in 2.1, the cost function considers the state, input references of the next N time steps. Therefore, a change within them is detected N steps in advance and can be better followed. On the contrary, the majority of control methods, e.g. PID and LQR, take into account only the reference of the present time slice. The time window considered by the cost function covers N time steps, however its beginning (and thus its end) changes every time step, making this window sliding: it always starts from the present instant and considers the next N ones, feature also referred to as receiding horizon.
After solving the optimization problem, a sequence of N inputs $u(k)$, $u(k+1)$, $\dots$, $u(k+N-1)$ is achieved. Nonetheless, only the first input $u(k)$ is applied to the plant, since during the next time step another optimization problem will be solved and thus a new sequence will be obtained. This last sequence could rely

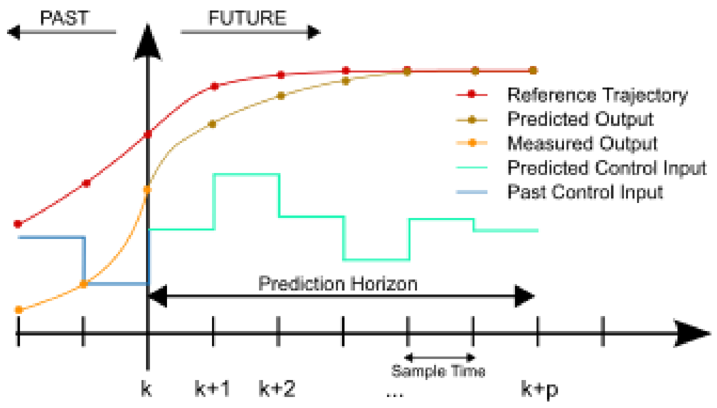


Figure 2.1: Illustration of the sliding horizon concept of MPC

on more updated values for the state vector, which might be essential in order to compensate for a plant-model mismatch. Fig. 2.1 provides an illustration of MPC concept: the time line is discretized into time samples and the prediction horizon involves N of them. A piecewise control input trajectory is performed within this prediction horizon and the first piece of this trajectory is applied. The goal is to have a measured output as closer as possible to the reference trajectory.

2.1.4 Implementation Details

In case of linear MPC with quadratic cost function and symmetric constraints, the optimization problem can be rewritten as a linearly constrained quadratic problem. The original optimization problem is as follows:

$$\begin{aligned}\min_{x_N,u_N} &\sum_{i=1}^{N}\big(x(k+i)-x_{ref}(k+i)\big)^T Q\big(x(k+i)-x_{ref}(k+i)\big)+\\ &\sum_{i=0}^{N-1}\big(u(k+i)-u_{ref}(k+i)\big)^T R\big(u(k+i)-u_{ref}(k+i)\big)\end{aligned} \tag{2.5}$$

subject to:

$$x(k+i+1)=Ax(k+i)+Bu(k+i),\ i=0,1,\dots,N-1 \tag{2.6}$$

$$|u(k+i)|\le u_{max},\ i=0,1,\dots,N-1 \tag{2.7}$$

$$|x(k+i)|\le x_{max},\ i=0,1,\dots,N-1 \tag{2.8}$$

The following matrices of values are defined:

$$x_N = \begin{bmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+N) \end{bmatrix}, \; u_N = \begin{bmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+N-1) \end{bmatrix} \tag{2.9}$$

These matrices arise from the state, input vectors considered within subsequent time samples. The achieved linearly constrained quadratic problem is represented by the following expressions:

$$\min_{\tilde{x}} \frac{1}{2}\tilde{x}^T H \tilde{x} + f^T \tilde{x} \tag{2.10}$$

subject to

$$A_e \tilde{x} = b_e, \; A_i \tilde{x} \leq b_i \tag{2.11}$$

where the following matrix has been defined:

$$\tilde{x} = \begin{bmatrix} x_N \\ u_N \end{bmatrix} \tag{2.12}$$

The matrices and vectors in the linearly constrained quadratic problem are as follows:

$$H = 2 \begin{bmatrix} Q & & & & & \\ & \ddots & & & & \\ & & Q & & & \\ & & & R & & \\ & & & & \ddots & \\ & & & & & R \end{bmatrix} \tag{2.13}$$

$$f = -H \begin{bmatrix} x_{ref}(1) \\ \vdots \\ x_{ref}(N) \\ u_{ref}(0) \\ \vdots \\ u_{ref}(N-1) \end{bmatrix} \tag{2.14}$$

$$A_i = \begin{bmatrix} I_{n_x \times N} & \\ -I_{n_x \times N} & \\ & I_{n_u \times N} \\ & -I_{n_u \times N} \end{bmatrix} \tag{2.15}$$

$$b_i = \begin{bmatrix} x_{max} \\ \vdots \\ x_{max} \\ u_{max} \\ \vdots \\ u_{max} \end{bmatrix} \tag{2.16}$$

$$A_e = \begin{bmatrix} I_{n_x} & & & & -B & & & \\ -A & I_{n_x} & & & & -B & & \\ & \ddots & \ddots & & & & \ddots & \\ & & -A & I_{n_x} & & & & -B \end{bmatrix} \tag{2.17}$$

$$b_e = \begin{bmatrix} Ax(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.18}$$

The problem is convex if the matrices Q, R are positive semi-definite. Furthermore, the matrices A_e, A_i are sparse, which means less computational burden. However, the optimization variables are $N \cdot (n_x + n_u)$: these may be reduced to $N \cdot n_u$, but at the cost of sparsity. The Matlab function to solve this optimization problem is the following one:

```
x_tilde = quadprog(H, f, Ai, bi, Ae, be)
```

2.1.5 MPC Design Choices

The optimization problem related to MPC has been so far represented by a quadratic cost function subject to a plant model and some constraints. Nonetheless, the cost function and the constraints may be expressed in different forms, as well as the prediction model.

Prediction Model

Given that the system representing the plant is to be in a state space form, several types of model are possible. The most straightforward and less critical is the linear state space system, resulting in the linear MPC. On the contrary, a non-linear state space system is related to the non-linear MPC and increases a lot the computational complexity of the optimization problem. Finally, a linear model with uncertainties results in the robust MPC. It must be remembered that MPC is always a non-linear control technique, even tough the model of the plant may be linear. Moreover, a linear system leads to a much more efficiently solvable optimization problem. Therefore, an effort should be done towards reaching a linear model which fits the plant sufficiently well.

Cost Function

So far, the cost function has always been represented through a quadratic form with weight matrices Q and R. This is by far the most common form, however different kinds of cost function J are possible. The general cost function is expressed through the following expression:

$$J(\boldsymbol{x}_N, \boldsymbol{u}_N) = \sum_{i=0}^{N-1} l(x_k, u_k) + F(x_N) \tag{2.19}$$

Even though the most widespread cost function is the quadratic one, a linear form is also possible, i.e., $w_x|x_k - x_{ref,k}| + w_u|u_k - u_{ref,k}|$. In alternative, cost functions in terms of Δx_k, Δu_k are viable candidates. Another tuning parameter of J is the horizon length N. Its increase leads to a more computationally complex problem, since the number of optimization variables $N \cdot (n_x + n_u)$ increases. However, the problem is more likely to be stable. Therefore, N should be chosen big enough with regard to all the dynamics of the system.

Constraints

The most common constraints limit the state and the input vectors x, u to convex sets X, U. Moreover, a combination of these two vectors represents a possibility, e.g. when an output constraint is present: $y = Cx + Du \in Y$. Typically, the constraint is expressed in the following form, depending on whether it is a state, an input or an output constraint:

$$|x_k| \leq x_{k,max}, \ |u_k| \leq u_{k,max}, \ |y_k| \leq y_{k,max} \tag{2.20}$$

Furthermore, constraints on rates of change can occur as well: $|\Delta u_k| \leq u_{k,max}$. The previous constraints are referred to as hard constraints, since they are to be strictly satisfied. However, setting too many hard constraints may make the system infeasible. In order to overcome infeasibility, soft constraints can be introduced: these are related to the cost function J by a parameter ϵ. An example of soft constraints regarding the state vector can be set as follows:

$$|x_k| \leq x_{k,max} + \epsilon_k, \quad J^{sc} = J + \rho_k l_\epsilon(\epsilon_k) \tag{2.21}$$

In this case, constraints may be violated and therefore they are introduced for less important limitations. Generally, hard constraints are used until there is no degree of freedom left. After that, all the other constraints are to be soft ones.

2.1.6 Stability

The classical way to determine whether a closed loop system is stable involves the transfer functions of the controller $C(z)$ and the plant $P(z)$. There are several methods to do that, such as through the Bode or Nyquist diagrams, or simply by analyzing the poles of $C(z)P(z)$. However, the system comprised by the MPC and the plant does not provide an easy way to achieve transfer functions. Therefore, new ways to determine the stability of the plant are needed. The most widespread stability verification is based on two steps: the first proves the recursive feasibility, i.e., determines whether the controller is well-defined for all the time instants k. The second step involves the Lyapunov function to check if the trajectories converge to an equilibrium point.

Recursive Feasibility

Recursive feasibility means that, an optimization problem which is feasible for time k, is also feasible for $k+1$, therefore for all $k+i$ with $i \geq 0$. The feasible region is defined as the one in state space comprised by all states x for which the MPC problem results feasible. The problem requires thus to build a feasible optimal solution for the time $k+1$, given the optimal solution at time k, i.e.,

$x(k) \in X$. The optimal solution at time k is feasible, since it is optimal. The cost function is expressed in the generic form:

$$J = \min_{x_N(k), u_N(k)} \sum_{i=0}^{N-1} l(x(k+i|k), u(k+i|k)) + F(x(k+N|k)) \tag{2.22}$$

To simplify the recursive feasibility verification, a strong assumption is made, i.e., there is no plant-model mismatch. Therefore, the state vector at time $k+1$ predicted at k is equal to the real state vector at time $k+1$: $x(k+1|k) = x(k+1|k+1)$. This last equation is valid also for the next time steps $k+i$, with $i = 1, \dots, N$. In order to achieve recursive feasibility, three conditions are to be verified: firstly, the final state vector predicted at time k, i.e., $x(k+N|k) \in X_N$, has to be part of the feasible region X. This is satisfied if $X_N \subseteq X$. Secondly, the two terminal vectors at time $k+1$, which are $x(k+N+1|k+1)$, $u(k+N|k+1)$, are to be inside the feasible region:

$$x(k+N+1|k+1) \in X_N, \; u(k+N|k+1) \in U \tag{2.23}$$

To do that, a locally stabilizing controller k_N is supposed to be known: $u_K = k_N(x_k)$. This implies that $x_{k+1} = f(x_k, k_N(x_k))$ is locally stable. By choosing $u(k+N|k+1) = k(x(k+N|k))$, $x(k+N+1|k+1) = f(x(k+N|k), k(x(k+N|k)))$, conditions expressed by 2.23 are satisfied.

Lyapunov Stability

The Lyapunov stability principle states that, if $\exists V(x)$ such that for some region X_f around 0 it results $V(x_{next}) < V(x)$, $\forall x \in X_f \setminus 0$, $V(0) = 0$, then all trajectories starting within X_f asymptotically evolve towards 0. With regard to the MPC application, X_f is chosen as the feasible region, while $V(x)$ is the optimal cost value of the MPC optimization problem for a given $x \in X_f$. The goal is thus to prove that $V(x)$ is a Lyapunov function, i.e., $V(x(k+1)) < V(x(k))$, $\forall x(k) \neq 0$. This corresponds to verify the following inequality:

$$J(x(k+1), x_N(k+1), u_N(k+1)) < J(x(k), x_N(k), u_N(k)) \tag{2.24}$$

The following relationship between the two cost expressions can be set:

$$\begin{aligned} J(x(k+1), x_N(k+1), u_N(k+1)) - J(x(k), x_N(k), u_N(k)) = \\ F(f(x(k+N|k), k_N(x(k+N|k))) - F(x(k+N|k)) + \\ +l(x(k+N|k), k_N(x(k+N|k))) - l(x(k), u(k)) \end{aligned} \tag{2.25}$$

This should be < 0. Since $l(x(k), u(k)) \geq 0$, a straightforward condition, even though conservative, to achieve Lyapunov stability is the following:

$$F(x) - F(x, k_N(x)) \geq l(x, k_N(x)), \ \forall x \in X_N \tag{2.26}$$

2.1.7 Robustess

The stability of the whole plant is a requirement even in presence of uncertainties. Concerning this issue, Robust MPC is able to deal with both uncertain models and disturbances. Its aim is to keep recursive feasibility once the size of the model uncertainties and disturbances is approximately known. The uncertainty of the model can be expressed through a time-variant state space system:

$$x_{k+1} = A(k)x_k + B(k)u_k \tag{2.27}$$

There are mainly two ways to express this uncertainty. The first is through a politopic region with apexes $[A_1 \ B_1]$, $\ldots$, $[A_L \ B_L]$, also referred to as linear parameter-varying state space model with politopic uncertainty description. The second is known as linear parameter-varying state space model with norm-bounded uncertainty description and the matrices are expressed as follows:

$$\begin{cases} A(k) = A_0 + B_p \Delta_k C_q \\ B(k) = B_0 + B_p \Delta_k D_{qu} \end{cases} \tag{2.28}$$

On the other hand, the disturbances are typically bounded, $w_k \in W$. The latter set may be described in two ways: either though a politope, $W = Co\{w_1, \ldots, w_n\}$, or with a simple inequality, $W = \{w | \ A_w w \leq 1_v\}$. A trivial condition to achieve well-posedness is $W \subseteq X$, i.e., the bounded set W is to be a subset of the one describing the constraints. This way, the recursive feasibility is ensured, since the set X is part of the feasible region. To guarantee the stability of the plant in presence of uncertainties and disturbances, some modifications are necessary. Firstly, the predictions become uncertain, however the constraints are to be satisfied anyway. This is referred to as worst-case constraint satisfaction over all predictions:

$$x(k+i|K) \in X \ \forall [A(k+j|k) \ B(k+j|k)], \quad j = 0, \dots, i-1, \ i = 0, \dots, N \tag{2.29}$$

The state components $x(k+i|K)$ depend linearly on $[A(k+j) \ B(k+j)]$, $j = 0, \dots, i-1$, while Ω is a polytopic set. As a result, a sufficient condition is to impose constraints on the vertices of the polytopic region:

$$x(k+i|k) \in X, \quad \begin{aligned} &\forall [A(k+j) \ B(k+j)] \in \{[A_1 \ B_1], \dots, [A_L \ B_L]\} \\ &j = 0, \dots, i-1, \ i = 0, \dots, N \end{aligned} \tag{2.30}$$

Moreover, the optimization problem relies on uncertain matrices $A(k)$, $B(k)$, therefore the cost function needs to take into account a worst-case scenario as well:

$$\min_{x_N(k), u_N(k)} \max_{\substack{[A(k+j|k) \ B(k+j|k)] \in \Omega \\ j = 0, \dots, N-1}} \sum_{i=0}^{N-1} l(x(k+i|k), u(k+i|k)) + F(x(k+N|k)) \tag{2.31}$$

Fortunately, once again it is sufficient to take into account only the vertices of the uncertainty polytope.
Additionally, the terminal cost has now to satisfy multiple Lyapunov inequalities. The non-robust stability condition for terminal cost is expressed by 2.26. To reach an explicit inequality in presence of uncertainties, some simplifications are considered. First of all, the stabilizing feedback controller is supposed to be linear, i.e., $u_k = k_N(x_k) = -kx_k$. Secondly, the quadratic cost is expressed through a quadratic form, $J(x,u) = \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T Q_N x_N$. Hence, the robust stability condition for terminal cost becomes:

$$x^T Q_N x - x^T (A - BK)^T Q_N (A - BK) x \geq x^T Q x + x^T K^T R K x \quad \forall x \in X_N, \forall [A \ B] \in \Omega \tag{2.32}$$

or, equivalently:

$$Q_N - (A - BK)^T Q_N (A - BK) \succeq Q + K^T R K \quad \forall [A \ B] \in \Omega \tag{2.33}$$

The inequality is satisfied if and only if:

$$Q_N - (A_i - B_i K)^T Q_N (A_i - B_i K) \succeq Q + K^T R K \\ i = 1, \dots, L \tag{2.34}$$

Finally, the terminal constraint has to be a robust invariant set. Recursive feasibility is indeed guaranteed by three conditions, as presented within the previous subsection. The first two remain unchanged, i.e., $X_N \subseteq X$, $k_N(x) \in U$. However, the third condition $f(x, k_N(x)) \in X_N$, is to be modified to take model uncertainty into account and to result in a robust positive invariance condition. If a linear stabilizing controller is considered, i.e. $k_N(x) = -kx$, the closed loop system results as follows:

$$x_{k+1} = \Phi(k)x_k,\ \Phi(k) \in \Omega^* = Co\{\Phi_1,\ \dots,\ \Phi_L\},\ \Phi_i = A_i - B_i k \tag{2.35}$$

The robust positive invariance is satisfied if, for $x \in S$, it results $\Phi x \in S$, $\forall \Phi \in \Omega^*$. Once again, the inclusion is to be satisfied only for the apexes, i.e., $\forall \Phi_i$, $i = 1, \dots, L$.

2.1.8 MPC and LQR

This subsection has the objective to compare two optimization-based control techniques, Linear Quadratic Regulator and Model Predictive Control. The goal is to highlight the flexibility of the second, which allows the user to tackle a variety of critical characteristics. Linear Quadratic Regulator, i.e., the most popular control method based on the optimization of a cost function, has the following formulation:

$$J\big(x(\cdot), u(\cdot)\big) = \sum_{i=1}^{\infty} \big(x(k+i) - x_{ref}(k+i)\big)^T Q \big(x(k+i) - x_{ref}(k+i)\big) + \\ \sum_{i=1}^{\infty} \big(u(k+i) - u_{ref}(k+i)\big)^T R \big(u(k+i) - u_{ref}(k+i)\big) \tag{2.36}$$

subject to:

$$\dot{x}(k) = Ax(k) + Bu(k) \tag{2.37}$$

If existing, the optimal solution can be expressed by the form $u(k) = Kx(K)$, and therefore LQR provides an explicit and linear solution. Furthermore, it does not require any computational effort, since the values of the gain matrix K can be performed off-line by solving the Riccati equation. These are two advantages if compared with MPC, which does not provide any explicit solution and possesses high computational complexity. These cons of MPC are due to its complexity and its power to cope with constraints, as well as non-linear systems. Moreover, the predictive behaviour of MPC illustrated within subsection 2.1.2 is not a feature possessed by LQR, since every input is obtained based on the present state estimate and reference.

Hence, even though both are based on the minimization of a cost function and on a state-space system, MPC and LQR are really different in applications and requirements. The first is more complex and advanced, it requires high-performance solvers and may be difficult to apply to systems with really small discretization time steps. On the other hand, Linear Quadratic Regulator is more straightforward since it does not require powerful processors. However, the limitations of LQR, e.g. the inability to deal with constraints and the lack of predictive behaviour, are overcome by MPC, which thus results in a more complex and performing control technique.

2.2 Dynamic Optimization Methods

As stated above, MPC is an optimization-based control technique. Before further diving into the relation between the MPC problem and dynamic programming, it is proper to illustrate some concepts used in the optimization theory. Secondly, the available methods will be presented.

2.2.1 Optimization Basics

An optimization problem can be expressed through the following general form:

$$\begin{aligned} min_x f(x) \\ s.t.\ h(x) = 0 \\ g(x) \leq 0 \end{aligned} \tag{2.38}$$

Here, f is the cost function with $x \in \mathbb{R}^n$ vector of optimization variables; h deals with the equality constraints, while g handles the inequality constraints. x^* is the solution of the problem, with the optimal function value $f^* = f(x^*)$. Two important concepts regarding the cost function are the gradient vector and the

Hessian matrix. They are specified through the following equations:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}, \ \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.39}$$

After choosing a $n-$dimensional point $x_0 \in \mathbb{R}^n$, the gradient vector points in the direction of the steepest ascent of $f(x_0)$, while the Hessian describes the local curvature of $f(x_0)$ through its eigenvectors $v_1, \ldots, v_n$ and eigenvalues $\lambda_1, \ldots, \lambda_n$. With regard to an unconstrained problem, a $n-$dimension point x^* is optimal if $\nabla f(x^*) = 0$. This optimal point is a minimum if $\nabla^2 f(x^*) \prec 0$, a maximum if $\nabla^2 f(x^*) \succ 0$, a saddle point otherwise.

On the other hand, if the problem is constrained, two additional sets of parameters are needed: the Lagrange multipliers of the inequality constraints λ_i and of the equality constraints μ_i. Afterwards, the Lagrangian can be defined as follows:

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{l} \lambda_i g_i(x) + \sum_{i=1}^{m} \mu_i h_i(x) \tag{2.40}$$

The constrained optimum of equation 2.38 can be found solving the problem below:

$$\begin{cases} \max_{\lambda,\mu} \min_x L(x, \lambda, \mu) \\ s.t. \ \lambda \geq 0 \end{cases} \tag{2.41}$$

Therefore, a min-max problem is to solved: the minimization is over the variable x, while the maximization over λ, μ. The problem can be split into two parts: a first order optimality condition in x and a maximization problem over λ, μ. The first consists of finding the optimal solution x^* of the following equation:

$$\nabla f(x^*) + \sum_{i=1}^{l} \lambda_i \nabla g_i(x^*) + \sum_{i=1}^{m} \mu_i \nabla h_i(x^*) = 0 \tag{2.42}$$

The two additional terms act as forces against the gradient of the function: their aim is to keep the optimum either on or on the right side of respectively the equality and inequality constraints.

If $l(\lambda, \mu)$ is defined as $l(\lambda, \mu) = \inf_x L(x, \lambda, \mu)$, the following maximization problem may be solved:

$$\begin{cases} \max_{\lambda,\mu} l(\lambda,\mu) \\ \text{s.t. } \lambda \geq 0 \end{cases} \tag{2.43}$$

Hence, x is pointed out as the primal variable, while λ, μ are the dual variables. Lastly, the Karush-Kuhn-Tucker conditions specify the ones necessary to achieve constrained optimality:

$$\begin{cases} \nabla f(x^*) + \sum_{i=1}^{l} \lambda_i \nabla g_i(x^*) + \sum_{i=1}^{m} \mu_i \nabla h_i(x^*) = 0 \\ h_i(x_i^*) = 0 \\ g_i(x_i^*) \leq 0 \\ \lambda_i^* \geq 0 \\ \lambda_i^* g(x_i^*) = 0 \end{cases} \tag{2.44}$$

2.2.2 Convex Optimization

A constrained optimal problem can be either convex or non-convex. Nevertheless, the first is much more widespread and straightforward to be solved. First of all, instead of providing local solutions, it guarantees a global optimum. Moreover, several efficient solvers are available, according to the specific convex optimal problem. An optimization problem in the form of 2.38 is convex if and only if, for any two feasible points x, y, the following apply:

$$\begin{aligned} \lambda x + (1-\lambda)y \quad \text{is feasible } \forall \lambda \in [0,\ 1] \\ f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)g(x),\ \forall \lambda \in [0,\ 1] \end{aligned} \tag{2.45}$$

These are satisfied if and only if the following conditions are true:

- the cost function is convex
- the equality constraints are either linear or absent
- the inequality constraints define a convex region

The main kinds of convex optimization problems are, from the more computationally efficient to the more general: linear programming, quadratic programming, second order cone programming, semi-definite programming.

Linear Programming

A linear programming problem (LP) is described as follows:

$$\begin{aligned} &\min_x f^T x \\ \text{s.t. } &A_e x = b_e \\ &A_i x \leq b_i \end{aligned} \tag{2.46}$$

In this case, the optimal solution is at a corner of the region of inequality constraints. Linear programming permits furthermore to eliminate the equality constraints by reparametrizing the optimization vector $x = Cy + d$, where $d = A_e^{-1} b_e$, $A_e C = 0$ with C full-column rank matrix. The optimization problem thus becomes:

$$\begin{aligned} &\min_y f^T C y \\ \text{s.t. } &A_i C y \leq b_i - A_i d \end{aligned} \tag{2.47}$$

Quadratic Programming

A more general class of constrained optimal problems is referred to as quadratic programming (QP), described in the following way:

$$\begin{aligned} &\min_x x^T H x + 2 f^T x \\ \text{s.t. } &A_i x \leq b_i \end{aligned} \tag{2.48}$$

To be convex, the problem is required to count on a semi-positive definite matrix $H \succeq 0$. Quadratic programming is widespread in several engineering domains. It is worth noting that linear programming is a special case of quadratic programming with $H = 0$.

Second Order Cone Programming

The general form for second order cone programming (SOCP) is expressed below:

$$\begin{aligned} &\min_x f^T x \\ \text{s.t. } &A_i x \leq b_i \\ ||M_i x + n_i|| \leq\ &c_i^T x + d_i,\ i = 1, \ldots, N \end{aligned} \tag{2.49}$$

The symbol $||\cdot||$ stands for the norm of a vector, $||u|| = \sqrt{u^T u}$. The last inequality represents the second order cone constraint. This form of optimal problem is always convex and it is employed within engineering applications which involve sum of squares or robust programming. In particular, quadratic programming (and indeed linear programming) is a special case of second order cone programming.

Semi-Definite Programming

The general form of SDP is:

$$\begin{aligned} &\min_x f^T x \\ &\text{s.t.} F(x) \succeq 0 \end{aligned} \tag{2.50}$$

where $F(x) = F_0 + \sum_{i=1}^n x_i F_i$, F_i symmetric $\forall i = 1, \dots, n$. This kind of optimal problem is widely used in systems and control theory, since it is able to reformulate problems involving eigenvalues as an inequality constraint. In particular, second-order cone programming is a special case of semi-definite programming. Therefore, the four types of optimization problems may be seen through a set representation, from the least general (linear quadratic programming) to the broadest (semi-definite programming). Figure 2.2 gives a visual representation of the four types of optimal programming. The four kinds refer to the class of

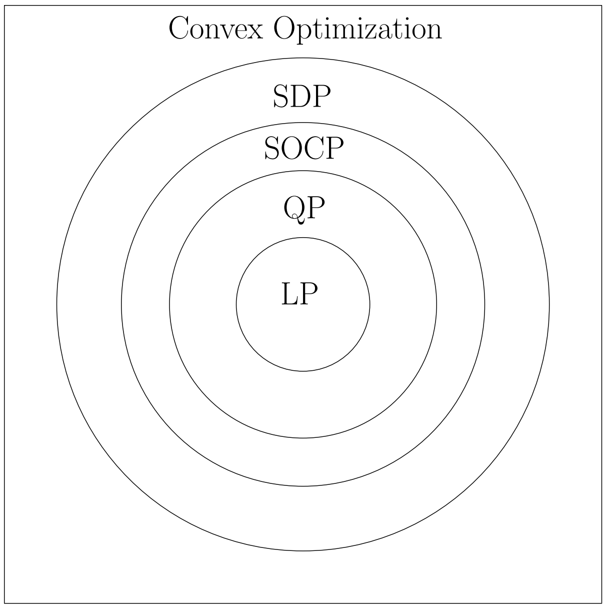


Figure 2.2: Set representation of the main kinds of optimal problems

convex optimization, which nonetheless are not entirely described by these. However, a convex optimization problem not representable through one of these four forms is really difficult to be solved, since it involves a high degree of complexity. On the contrary, a problem enclosed in one of the previous classes can be easily solved, as many toolboxes have been developed that provide high efficiency and low computational time.

2.3 Dynamic Programming and MPC

As already stated above, the MPC algorithm provides that an optimal input sequence $u_N(k)$ is calculated within every time step k. This sequence is found by means of dynamic programming. Depending on the specific structure of the optimal problem describing the MPC algorithm, there may be several formulations. For instance, in case of least squares cost function and linear state space system, the optimization problem can be represented through a quadratic programming formulation. The typical MPC problem is indeed stated as follows:

$$\begin{aligned} &\min_{x_N,u_N} \sum_{k=1}^{N}(x_k^T Q x_k) + \sum_{k=0}^{N-1}(u_k^T R u_k) \\ \text{s.t. } & A_u u_k \leq 1_v, \qquad\qquad k = 0,\dots,N-1, \\ & A_x x_k \leq 1_v, \qquad\qquad k = 1,\dots,N, \\ & x_{k+1} = Ax_k + Bu_k, \; k = 0,\dots,N-1 \end{aligned} \tag{2.51}$$

The optimization vector is thus $x_{opt} = \begin{bmatrix} u_0 & \dots & u_{N-1} & x_1 & \dots & x_N \end{bmatrix}$, while the cost function is expressed below:

$$\begin{aligned} J(x_{opt}) &= x_{opt}^T H x_{opt} + 2f^T x_{opt} \\ H &= \begin{bmatrix} R & & & & & \\ & \ddots & & & & \\ & & R & & & \\ & & & Q & & \\ & & & & \ddots & \\ & & & & & Q \end{bmatrix}, \; f = 0 \end{aligned} \tag{2.52}$$

The condition for convexity is $H \succeq 0$, thus $Q \succeq 0$, $R \succeq 0$. The equality and inequality constraints rely on the following matrices and vectors:

$$A_e = \begin{bmatrix} -B & & & I & & & \\ & -B & & -A & I & & \\ & & \ddots & & \ddots & \ddots & \\ & & -B & & & -A & I \end{bmatrix} \quad b_e = \begin{bmatrix} Ax_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$
$$A_i = \begin{bmatrix} A_u & & & & & \\ & \ddots & & & & \\ & & A_u & & & \\ & & & A_x & & \\ & & & & \ddots & \\ & & & & & A_x \end{bmatrix} \quad b_i = \begin{bmatrix} b_u \\ \vdots \\ b_u \\ b_x \\ \vdots \\ b_x \end{bmatrix} \tag{2.53}$$

Alternatively, in case of linear model, constraints and objective function, MPC may be represented by linear programming:

$$\begin{aligned} \min_{x_N, u_N} & \sum_{k=1}^{N} w_x^T |x_k - r_k| + \sum_{k=0}^{N-1} w_u^T |u_k| \\ \text{s.t. } & |u_k| \le u_{max}, \qquad k = 0, \dots, N-1, \\ & |x_k| \le x_{max}, \qquad k = 1, \dots, N, \\ & x_{k+1} = Ax_k + Bu_k, \; k = 0, \dots, N-1 \end{aligned} \tag{2.54}$$

This option provides a slightly faster but more chattering solution. The latter indeed jumps around, therefore is less smooth.
Finally, in presence of a non-linear model, the optimization problem falls into the field of non-linear programming. This means much more computational complexity, hence a linear model should be chosen which approximates the real plant in the most accurate way.

2.3.1 Optimization Algorithms

To solve one of the optimization constrained problems previously presented, two main kinds of algorithms are employed: active set methods and interior methods.

Active Set Methods

Active set methods rely on the following steps:

- choose an initial feasible point x_0
- make an initial guess about the active constraints employed for the optimal solution
- convert these into equality constraints, ignore the other constraints
- solve the resulting equality constrained quadratic programming
- use the result as a search direction and proceed until a new constraint is met
- add the latter to the active set of constraints
- repeat from the second step until convergence

Two relevant advantages of active set methods regard hot start, which means that previous information, e.g. the active set of constraints, is reused in the next iterations, and the fact it provides feasible intermediate results.

Interior Point Methods

The steps regarding interior point methods are specified below:

- choose an initial point $(x,\, \lambda_i,\, \mu_i)$
- linearize the Kasush-Kuhn-Tucker conditions around the current point
- solve the linearized KKT system to reach a search direction
- calculate the step length such that $\lambda_i \geq 0$, $g_i(x) \leq 0$
- repeat from the second step until convergence

The main advantage of interior point methods is that they can exploit sparsity.

2.4 Sequential Quadratic Programming

2.4.1 Overview

To solve any non-linear programming problem in the form of 2.38, a possibility is to make use of Sequential quadratic programming. SQP refers to an iterative method for constrained non-linear optimization. It aims to find the optimal triple (x^*, λ^*, μ^*), according to what illustrated within subsection 2.2.1. In order to be suitable for SQP, a mathematical problem is to possess an objective function and contraints which are twice continuously differentiable. The procedure consists in solving a sequence of optimization subproblems that optimize a quadratic model of the objective subject to a linearization of the constraints. An embedded SQP algorithm should provide the following features:

- numerical integration of the continuous-time dynamics
- generation of first and second-order sensitivities of the objective subject, constraints
- an approximation of the Hessian matrix
- a QP solver
- strategies for warm-starting the algorithm for the next problem

Chapter 4 will focus on acados, the software library used to implement the model predictive control. The SQP algorithm implemented in acados has the following structure [11]:

$$\begin{aligned} w(i+1) &= w(i) + \Delta w_{QP}, \quad i = 0, 1, \dots \\ \pi(i+1) &= \pi_{QP}, \quad i = 0, 1, \dots \\ \mu(i+1) &= \mu_{QP}, \quad i = 0, 1, \dots \end{aligned} \tag{2.55}$$

Here, $w(i) = \begin{bmatrix} x_0(i)^T & \dots & u_0(i)^T & \dots & x_N(i)^T \end{bmatrix}^T$ is the primal iterate at SQP iteration i, $\pi(i)$ and $\mu(i)$ are the dual iterates. Δw_{QP} is found by solving the following quadratic programming problem:

$$\begin{aligned}
\min_{\substack{\Delta x_0, \ \dots, \ \Delta x_N \\ \Delta u_0, \ \dots, \ \Delta u_N \\ s_0, \ \dots, \ s_N}} \ & \sum_{k=0}^{N-1} \begin{bmatrix} \Delta x_k & \Delta u_k \end{bmatrix}^T \begin{bmatrix} Q_k & S_k \\ S_k & R_k \end{bmatrix} \begin{bmatrix} \Delta x_k & \Delta u_k \end{bmatrix} + \begin{bmatrix} q_k & r_k \end{bmatrix}^T \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} \\
& + \Delta x_N^T Q_N \Delta x_N + q_N^T \Delta x_N \\
& + \sum_{k=0}^{N} s_k^T P_k s_k + p_k^T s_k
\end{aligned} \tag{2.56}$$

subject to:

$$\begin{aligned}
\Delta x(k+1) = A_k \Delta x(k) + B_k \Delta u(k) + \Phi(k)^x - x(k+1), &\quad k = 0, \dots, N-1 \\
\Delta x_0 = \bar{x}_0 - x_0, & \\
-g(k) \geq G(k)^x \Delta x(k) + G(k)^u \Delta u(k) + s(k), &\quad k = 0, \dots, N-1 \\
-g(N) \geq G(N)^x \Delta x(N) + s(N), &\quad k = 0, \dots, N-1 \\
0 \leq s(k), &\quad k = 0, \dots, N-1
\end{aligned} \tag{2.57}$$

In this notation, $x : \mathbb{R} \to \mathbb{R}^{n_x}$ indicates the differential states, $u : \mathbb{R} \to \mathbb{R}^{n_u}$ the control inputs, while $s : \mathbb{R} \to R^{n_s}$ denotes the slack variables. The latter are introduced to reach a formulation with soft constraints.

2.4.2 The Online Dilemma

The computational time for an SQP iteration varies in a small range and, given an initial value $x(t_0)$, several SQP iterations are required to achieve a sufficiently exact solution. Supposing that n iterations are needed, each of ones taking a time ϵ, the optimal feedback control $u_0^*(x(t_k))$ is available only at time $t_k + n\epsilon$, i.e., with the delay $n\epsilon$. Nevertheless, by the time $t_k + n\epsilon$, the system state has already moved to a different system state $x(t_k + n\epsilon) \neq x(t_k)$. Hence, $u_0^*(x(t_k))$ is not the exact feedback anymore, being the exact value equal to $u_0^*(x(t_k + n\epsilon))$. Best case scenario, the system state has not changed much meanwhile, so that $u_0^*(x(t_k)) \approx u_0^*(x(t_k + n\epsilon))$. If not, a possibility is to predict the most probable system state $x(t_k+n\epsilon)$. However, this approach increases the reliance on the open-

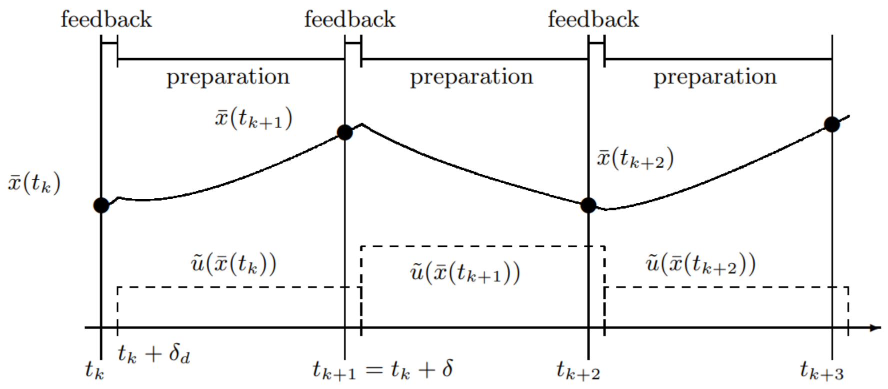


Figure 2.3: Division of the computational time made by the real time iteration scheme: a preparation and a feedback phase are present

loop part of MPC, since the future state is a prediction based on the nominal model of the system. Moreover, the feedback regarding a disturbance comes with a delay δ_d of one full sampling time: $\delta_d = \delta = n\epsilon$. In addition to this, another problem related to the execution of several SQP iteration is the respect of the time limits. A specific answer to these problems is presented within the following subsection.

2.4.3 Real Time Iteration

As already mentioned, embedded applications require to make use of real-time control settings and thus to deal with solving NLP in sequence and under rigid time conditions. Because of the continuously changing environment, an approximate solution is much more useful than a high-accuracy solution obtained after the deadline. In this regard, one online method of great interest is the real-time iteration scheme, [13], [14]. Its peculiarity consists in solving an inequality-constrained QP in each iteration. Hence, within each real-time iteration, only one full iteration of an SQP-type scheme is performed. Moreover, its feasibility for controlling a ball-and-plate system has been proven in [15].
It is possible to divide the computation time of each cycle into a short feedback phase and a usually much longer preparation phase. The first is only used to evaluate the accuracy of the approximation $\tilde{u}_0(x(t_k))$, while the second is to prepare the next feedback $\tilde{u}_0(x(t_)k+1)$ without the knowledge of $x(t_{k+1})$. This aims at achieving delays δ_d which are much smaller than the ones presented within the previous subsection, i.e., $n\epsilon$. A critical choice regards the best approximation $\tilde{u}_0(t_k)$. According to the NLP notation, the problem to be solved is the following:

$$P(x(t_k)) : \qquad \min_w a(w) \quad \text{subject to} \quad b_{x(t_k)}(w) = 0, \quad c(w) \geq 0 \qquad (2.58)$$

The aim is to achieve the solution $w^*(x(t_k))$ of each problem $P(x(t_k))$ as fast as possible. At the basis of the real-time iteration scheme there are two ideas, which make the algorithm even quicker:

- Most of the operations needed for the first iteration can be done before the initial value $x(t_k)$ is known. Hence, the delay time can be further reduced if the majority of the computations are performed before t_k. At time t_k the feedback response $\tilde{u}_0(x(t_k))$ is computed, so that the feedback delay δ_d turns to be even smaller than a single SQP iteration, $\delta_d < \epsilon$.
- An approximate solution of the optimal control problem has been considered so far. Therefore, instead of iterating the SQP until convergence, a single iteration per sampling time would considerably decrease the preparation time. This would also permit shorter sampling intervals with the duration of one SQP iteration, $\delta = \epsilon$. A strong advantage is the much smaller difference between subsequent states $x(t_k)$ and $x(t_{k+1})$ due to the shorter sampling time.

Therefore, the real-time iteration scheme allows to reach feedback delays δ_d that are much smaller than a sampling time, as well as sampling times δ equal to a single SQP iteration ϵ. All these achievements are of great interest in the field of embedded optimization.

Chapter 3

State Estimators

As discussed within chapter 2, MPC needs the full state vector components at every time instant k, so as to provide an optimal sequence of inputs u_N of which the first one is applied to the plant. Nonetheless, the state is rarely fully measured. With regard to the system under test, i.e., the Stewart platform, the only measurements come from the touch screen of the plate, which indeed provide the user with the position of the ball. The remaining vector components, such as the velocity of the ball, as well as the angles of the plate, are not available. Moreover, in order to achieve an offset-free behaviour, modeled disturbances are to be introduced, estimated and afterwards compensated, as tackled in [16], [17]. Therefore, an estimator is needed to be combined with MPC. Several kinds of estimators are available in literature, however two candidates have been chosen to be better deepened: the *Kalman filter* and the *moving horizon estimator*. These are indeed both optimal estimators, i.e., they rely on a cost function as MPC and LQR do. The former estimator is the dual of LQR: starting from three matrices Q, R, P_0, it minimizes a cost function J in order to provide a state estimate in a noisy environment. The latter, on the other hand, is strictly correlated to MPC: it provides an optimal sequence of state vectors within a finite horizon N, hence the concept of receiding horizon is present. In addition to a state estimate, both are able to estimate modeled disturbances. The following subsections further deep into the concepts related to these two estimation techniques.

3.1 Kalman Filter

3.1.1 Overview

The ability of Kalman filter to provide MPC with a trustworthy estimate of the full state vector has already been proven in [18]. Moreover, its capability of estimating modeled disturbances in order to achieve an offset-free behaviour has been faced in [19]. Kalman filtering is an algorithm that provides the estimates of some unknown variables given the measurements observed over time. This kind of estimators possesses a relatively straightforward form and requires low computational burden. Its main advantage is the capability of dealing with a noisy environment, where both the state and the output components are affected by an aleatoric disturbance. The following equations express a state space system where the state and the output are corrupted by two random disturbances w_c, v.

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu + w(t) \\ y(t) = Cx(t) + v(t) \end{cases} \tag{3.1}$$

The signals $w(t)$, $v(t)$ are to be white noise processes, i.e., zero-mean, stationary and indipendent. The challenge of KF is to build a variable gain estimator that provides an optimal estimate of $x(k)$. Figure 3.1 illustrates the plant: KF receives the discrete signals $u(k)$, $y(k)$ and gives back an estimate of the full state $x_{est}(k)$ and if necessary of modeled disturbances $d_{est}(k)$. The algorithm of Kalman fil-

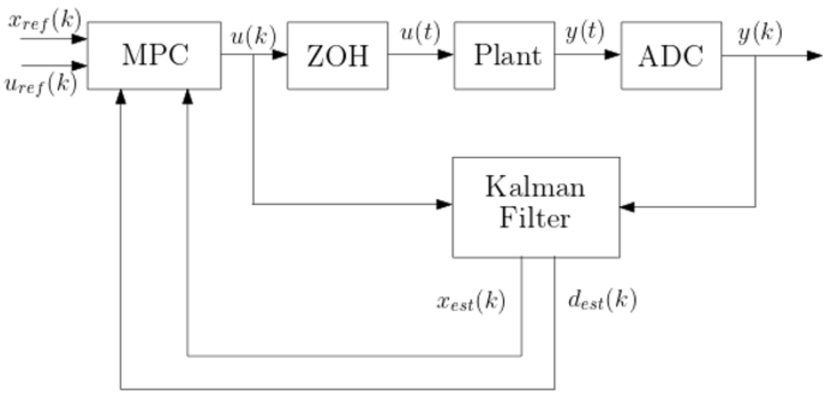


Figure 3.1: Block scheme of the plant comprising Kalman filter and MPC

tering takes origin from the concept of least squares estimation, illustrated in the next subsection.

3.1.2 Least Squares Estimation

Least squares estimation aims at achieving an optimal estimate of unknown components, starting from p corrupted measurements. $y(k) \in \mathbb{R}^p$ is the measurements vector, related to the not-fully known vector $x(k) \in \mathbb{R}^n$ through the following relationship:

$$y_m(k) = Hx(k) + v(k) \tag{3.2}$$

where $v(k)$ is the disturbance to be minimized in order to achieve an optimal estimate $x_{est}(k)$. The cost function is thus $J = \frac{1}{2}v^T v$ and results minimized if its partial derivative in x is zero:

$$\begin{aligned} J = \frac{1}{2}v^T v = \frac{1}{2}(y - Hx)^T(y - Hx) \\ \frac{\partial J}{\partial x} = (y - Hx)^T(-H) = 0 \end{aligned} \tag{3.3}$$

which implies $H^T y = H^T Hx$ and thus:

$$x_{est} = (H^T H)^{-1} H^T y \tag{3.4}$$

From 3.4, the square matrix $H^T H \in \mathbb{R}^{n \times n}$ needs to be invertible. Therefore, the number of measurements p is to be greater than the dimension of the vector x, $p \geq n$. The error between the estimate x_{est} and the real quantity x is:

$$x_{est} - x = (H^T H)^{-1} H^T (Hx + v) - x = (H^T H)^{-1} H^T v \tag{3.5}$$

Hence, if v is a zero-mean quantity, the error is zero-mean as well. Therefore, x_{est} is an unbiased estimate, since a zero-mean disturbance leads to a zero-mean error within the estimation. The error covariance matrix is $P = E[(x_{est} - x)(x_{est} - x)^T]$, while the disturbance covariance matrix is $R = E[vv^T] = \sigma^2 I$. σ is the variance of every random variable $v(k)$, $k = 0, \ldots, t$, since these variables are supposed to be indipendent and identically distributed. Therefore, the following expression for P is reached:

$$\begin{aligned} P &= E[(H^T H)^{-1} H^T vv^T H (H^T H)^{-1}] \\ &= (H^T H)^{-1} H^T E[vv^T] H (H^T H)^{-1} = \sigma^2 (H^T H)^{-1} \end{aligned} \tag{3.6}$$

An alternative is to use a weighted least squares estimation, in the sense that not every measurement contributes with the same weight. Assuming W as the weight matrix, the cost function becomes $J = \frac{1}{2}v^T W v$. The estimate and the covariance of the error are expressed below:

$$x_{est} = (H^T W H)^{-1} H^T W y \tag{3.7}$$

$$P = (H^T W H)^{-1} H^T W E[vv^T] W H (H^T W H)^{-1} \tag{3.8}$$

If $E[vv^T] = R$, an approach is to choose $W = R^{-1}$, so that:

$$x_{est} = (H^T R^{-1} H)^{-1} H^T R^{-1} y \tag{3.9}$$

$$P = (H^T R^{-1} H)^{-1} \tag{3.10}$$

In order to achieve a more useful algorithm, a recursive formulation should be used. This means that an estimate reached from p measurements needs to be changed after another measure is available, but without repeating the whole process from the very beginning. For instance, a vector of p measurements $y_0 = H_0 x + v_0$ is the starting point, to which a new measurement $y_1 = H_1 x + v_1$ is added. The weights are based on the inverses of the covariance matrices, $W_0 = R_0^{-1}$, $W_n = R_n^{-1}$. By rearranging eq. 3.9, the following equation is obtained:

$$\begin{bmatrix} H_0 \\ H_n \end{bmatrix}^T \begin{bmatrix} R_0^{-1} & 0 \\ 0 & R_n^{-1} \end{bmatrix} \begin{bmatrix} H_0 \\ H_n \end{bmatrix} x_{est} = \begin{bmatrix} H_0 \\ H_n \end{bmatrix}^T \begin{bmatrix} R_0^{-1} & 0 \\ 0 & R_n^{-1} \end{bmatrix} \begin{bmatrix} y_0 \\ y_n \end{bmatrix} \tag{3.11}$$

The plan is to provide a formulation of the updated estimate x_{est} which relies on the old one x_0 and a variation dependent only on the new measurement: $x_{est} = x_{est,0} + \delta_x$. Since x_0 is the solution at the previous step, i.e., $(H_0^T R_0^{-1} H_0) x_{est,0} = H_0^T R_0^{-1} y_0$, from eq. 3.11 it results:

$$\begin{aligned} \delta_x &= (H_0^T R_0^{-1} H_0 + H_n^T R_n^{-1} H_n)^{-1} H_n^T R_n^{-1} (y_n - H_n x_{est,0}) = \\ &= (P_0^{-1} + H_n^T R_n^{-1} H_n)^{-1} H_n^T R_n^{-1} (y_n - H_n x_{est,0}) \end{aligned} \tag{3.12}$$

The covariance of the new estimate is:

$$E[(x_{est} - x)(x_{est} - x)^T] = P_n = (P_0^{-1} + H_n^T R_n^{-1} H_n)^{-1} \tag{3.13}$$

Therefore, the whole procedure could be summarized as follows:

- Obtain P_0, $x_{est,0}$ from the first p measurements
- Collect another measure y_n
- Calculate $P_n = (P_0^{-1} + H_n^T R_n^{-1} H_n)^{-1}$, $x_{est} = x_{est,0} + P_n H_n^T R_n^{-1}(y_n - H_n x_{est,0})$

Therefore, if P_0 has high values, which means that $x_{est,0}$ is quite inaccurate, $P_n \approx (H_n^T R_n^{-1} H_n)^{-1}$, $x_{est} \approx (H_n^T R_n^{-1} H_n)^{-1} H_n^T R_n^{-1} y_n$. On the contrary, if $P_0 \approx 0$, i.e., $x_{est,0}$ is really accurate, $P_n \approx P_0$, $x_{est} \approx x_{est,0}$.

3.1.3 Formulation

Kalman filter takes the form of a current estimator:

$$x_{est}(k|k) = x_{est}(k|k-1) + L(k)[y(k) - Hx_{est}(k|k-1)] \tag{3.14}$$

Hence, the system expressed by 3.1 is to be discretized:

$$\begin{cases} x(k+1) = \Phi x(k) + \Gamma u(k) + \Gamma_1 w(k) \\ y(k) = Hx(k) + v(k) \end{cases} \tag{3.15}$$

where $w(k)$, $v(k)$ are the discretized noises. They must be indipendent, zero-mean, stationary random processes. KF aims at estimating the state vector $x(k)$ minimizing the quadratic sum of the innovation $e(k) = y - Hx_{est}$. Therefore, it is a least square estimation technique and thus takes origin from the algorithm presented in subsection 3.1.2. Regarding the symbols, the updated innovation covariance matrix is $P_n = P(k)$ while the open loop one is $M(k)$, the measurement covariance matrix is $R_n = R_v$ and the output matrix $H_n = H$. Hence, the estimator gain and the innovation covariance matrix are expressed as follows:

$$L(k) = P(k)H^T R_v^{-1} \tag{3.16}$$

$$P(k) = (M(k)^{-1} + H^T R_v^{-1} H)^{-1} \tag{3.17}$$

The open loop and closed loop estimates of the state are respectively:

$$x_{est}(k|k-1) = \Phi x(k-1|k-1) + \Gamma u(k-1) \tag{3.18}$$

$$x_{est}(k|k) = x_{est}(k|k-1) + L(k)(y(k) - Hx_{est}(k|k-1)) \tag{3.19}$$

The open loop innovation covariance matrix is:

$$\begin{aligned} M(k) = E[(x(k) - x_{est}(k|k-1))(x(k) - x_{est}(k|k-1))^T] \\ = E[\Phi(x(k-1) - x_{est}(k-1|k-1))(x(k-1) - x_{est}(k-1|k-1))^T\Phi^T + \\ +\Gamma_1 w(k-1)w(k-1)^T\Gamma_1^T] \\ = \Phi P(k-1)\Phi^T + \Gamma_1 R_w \Gamma_1^T \end{aligned} \tag{3.20}$$

Hence, the knowledge of $P(k-1)$ is needed firstly to achieve $M(k)$ and then $P(k)$.
In summary, the Kalman filter algorithm comprises two steps: the prediction, or a-priori estimate, and the update, or a-posteriori estimate. The former is performed before the measurement $y(k)$ is available, while the latter takes place afterwards. The starting point is the initial guess of the state $x_{est,}(0)$, its covariance matrix $P(0)$ and the covariance matrices R_w, R_v, of the i.i.d. random processes $w(k)$, $v(k)$.

A-priori estimate

- Perform the open loop estimation $x_{est}(k|k-1) = \Phi x_{est}(k-1|k-1) + \Gamma u(k)$
- Calculate the open loop innovation covariance matrix $M(k) = \Phi P(k-1)\Phi^T + \Gamma_1 R_w \Gamma_1^T$

A-posteriori estimate

- Calculate the closed loop innovation matrix $P(k) = M(k) - M(k)H^T(HM(k)H^T + R_v)^{-1}HM(k)$
- Achieve the KF gain $L(k) = P(k)H^T R_v^{-1}$
- Perform the closed loop estimation $x_{est}(k|k) = x_{est}(k|k-1) + L(k)(y - Hx_{est}(k|k-1))$

3.1.4 Tuning

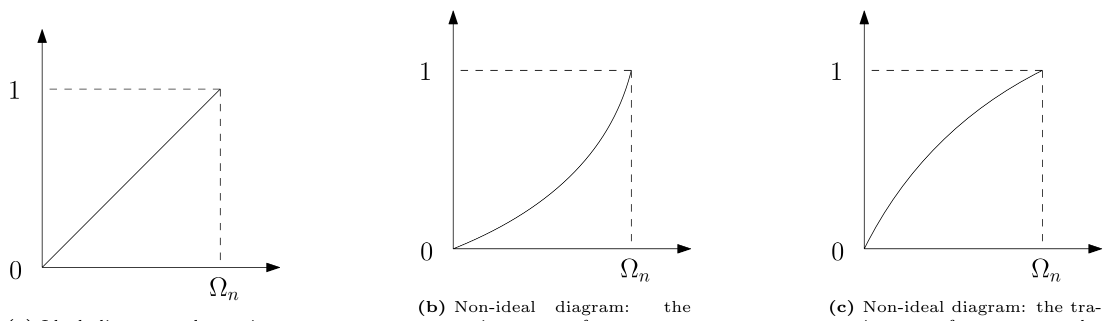
The main drawback concerning KF is the difficulty of tuning. From the previous subsection, the quantities $x_{est}(0)$, $P(0)$, R_w and R_v are to be known as a starting point. Furthermore, from the continuous system 3.1, a discretized one is achieved expressed by 3.15. However, the disturbances are continuous-time signals, not piecewise-constant such as the input $u(t)$, therefore achieving two discrete quantities $w(k)$, $v(k)$, as well as the matrix Γ_1, is not trivial. $v(k)$ represents any output disturbance, hence it can be imputed to e.g. the quantization of the output. On the contrary, $w(k)$ is the model disturbance, which might be related to random disturbances or uncertainty of the parameters. To achieve a more straightforward way of tuning the covariance matrices, the lowest number of tuning parameters should be involved. The simplest tuning method of KF arises from the following hypothesis: differences between the nominal and the real model, thus imputable to $w(t)$, can be represented with an equivalent input disturbance $w_c(t)$. However, as already highlighted, $w_c(t)$ is not piecewise-constant like $u(t)$ is. Hence, achieving a discretized matrix which explains a relationship between the discretized signals $x(k)$ and $w_c(k)$ is not elementary. Starting from the continuous system, the discretization happens as follows:

$$\dot{x}(t) = Ax(t) + B(u(t) + w_c(t)) \tag{3.21}$$

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) + \int_0^T e^{A\sigma} B w_c(\sigma) d\sigma \\ &= \Phi x(k) + \Gamma u(k) + W_d(k) \end{aligned} \tag{3.22}$$

The expression of the matrix $M(k+1) = \Phi P(k) \Phi^T + \Gamma_1 R_{w_d} \Gamma_1^T$ illustrates how the quantity of interest is $Q_w = \Gamma_1 R_{w_d} \Gamma_1^T$, not directly R_{w_d}. Therefore, instead of trying to reach a relation between $w_d(k)$ and $x(k)$, the representation of the discretized input disturbances makes use of $W_d(K)$ and its covariance matrix $Q_w = E[W_d(k) W_d(K)^T]$:

$$\begin{aligned} Q_w &= E\left[\int_0^T \int_0^T e^{A\sigma} B w_c(\sigma) w_c^T(\eta) B^T e^{A^T\eta} d\sigma d\eta\right] \\ &= \int_0^T \int_0^T e^{A\sigma} B E\big[w_c(\sigma) w_c^T(\eta)\big] B^T e^{A^T\eta} d\sigma d\eta \\ &= \sigma_{w_c}^2 \int_0^T \int_0^T e^{A\sigma} B \delta(\eta - \sigma) B^T e^{A^T\eta} d\sigma d\eta \\ &= \sigma_{w_c}^2 \int_0^T e^{A\sigma} B B^T e^{A^T\sigma} d\sigma \end{aligned} \tag{3.23}$$



(a) Ideal diagram: the trajectory refers to a cumulative periodogram of a white noise process

(b) Non-ideal diagram: the trajectory refers to a cumulative periodogram with dominance of high-frequency components

(c) Non-ideal diagram: the trajectory refers to a cumulative periodogram with dominance of low-frequency components

Figure 3.2: Different possible trajectories for the cumulative periodogram of the innovation $e(k)$

$E[w_c(\sigma)w_c^T(\delta)] = \sigma_{w_c}^2\delta(\eta - \sigma)$ is due to the fact that w_c is supposed to be a white noise, thus its covariance matrix is a diagonal matrix with the variance σ_{w_c} as diagonal value. Another important remark regards the integral $\int_0^T e^{A\sigma}BB^Te^{A^T\sigma}d\sigma$, which happens to be the controllability Gramian Q_w.

On the other hand, the measurement noise is usually well-featured as it can be ascribed to well-known causes, such as the quantization of the output. Considering a SISO system, the tuning parameters are $\sigma_{w_c}^2$ for Q_w, σ_v^2 for R_V, and only their ratio is relevant. A starting point can be to set $\sigma_v^2 = q^2/12$ and modify σ_{w_c} according to the observations. The procedure of tuning these parameters is referred to as whiteness test. The innovation $e(k)$ is indeed to be a white noise, therefore the Kalman filter is considered well-tuned when the resulting innovation behaves like a white noise. To verify that, the cumulative periodogram is analyzed, cumsum$|\text{fft}(e(k))|$. Figure 5.23 illustrates the three possible kinds of cumulative periodograms achieved. 3.2a refers to the ideal one, while 3.2b, 3.2c represent undesired trajectories. The first has an excess of high-frequency components: this means that $\sigma_{w_c}^2$ must be lowered. On the other hand, an excess of low-frequency components, like in 3.2c, points out that $\sigma_{w_c}^2$ must be highered.

3.2 Moving Horizon Estimator

Another possibility for estimating the full state vector and possible modeled disturbances or unknown parameters is the use of moving horizon estimator, which is an optimal state estimator, similarly to the Kalman filter. However, it is able to deal with state and input constraints, and therefore it is considered the dual of model predictive control. Hence, MHE solves a dynamic optimization problem within every time slice which can easily cope with state constraints. The non-linear discrete time system is described as follows:

$$\begin{aligned} x(k+1) &= f(x(k), u(k), p(k)) \\ y(k) &= h(x(k)) \end{aligned} \tag{3.24}$$

where $x(k) \in \mathbb{R}^{n_x}$, $u(k) \in \mathbb{R}^{n_u}$, $p(k) \in \mathbb{R}^{n_p}$ $y(k) \in \mathbb{R}^{n_y}$ are respectively the state, input, uncertainty and output vectors. In case $p(k) = 0$, the considered system is equal to the nominal one. The feasibility of MHE in embedded applications has been proven in [20], where a real time algorithm has been proposed. Furthermore, its combination with Model Predictive Control has been tackled within [21].

Two different formulations of MHE in combination with MPC are possible, as explained in [22]. The first supposes that the uncertainty structure of the model is unknown. Therefore, state and output disturbances are to be introduced to compensate for any plant-model mismatch. This approach is similar to the one used with the Kalman filter. The second formulation requires instead the knowledge of the uncertainty structure, i.e., which parameters inside the system are uncertain. In this latter case, the MHE provides the estimation of both the plant state and the uncertainty parameters. The following subsections present the two possible approaches.

3.2.1 MHE with State and Output Disturbances

In the first scenario, which expects that the uncertainty structure is unknown, the parameter $p(k)$ is set to zero and the offset-free behaviour is achieved taking into account state and output disturbances. The MHE problem is therefore expressed as follows:

$$\begin{aligned} min \sum_{i=0}^{N} (\eta_{k-N+i}^T W_y \eta_{k-N+i} + w_{k-N+i}^T W_w w_{k-N+i}) + \\ (\hat{x}_{k-N} - \bar{x}_{k-N})^T W_x (\hat{x}_{k-N} - \bar{x}_{k-N}) \end{aligned} \tag{3.25}$$

subject to:

$$\begin{aligned} \hat{x}_{k-N+i+1} &= f(\hat{x}_{k-N+i}, u_{k-N+i}, 0) + w_{k-N+i} \\ \hat{y}_{k-N+i} &= h(\hat{x}_{k-N+i}) \\ \eta_{k-N+i} &= y_{k-N+i} - \hat{y}_{k-N+i} \\ \hat{x}_{k-N+i} \in X,\ \eta_{k-N+i} &\in \Omega_\eta,\ w_{k-N+i} \in \Omega_w, \\ j &= 0, \ldots, N \end{aligned} \tag{3.26}$$

W_y, W_w, W_x represent the weight matrices, that are to be symmetric positive definite, while N is the estimation horizon length. The estimated state and output are $\hat{x}_k$ and $\hat{y}_k$, while w_k and η_k are the state and output disturbances. $\bar{x}_{k-N}$ is the most likely prior value of x_{k-N}, i.e., the estimated state evaluated within the previous timeslice.

3.2.2 MHE with State and Parameter Estimation

In case the uncertainty parameter structure is known, MHE is able to simultaneously estimate both the state and the uncertain parameters. This means that the model is modified online. The formulation of MHE is described through the following expressions:

$$\begin{aligned} min \sum_{i=0}^{N} (\eta_{k-N+i}^T W_y \eta_{k-N+i}) + \hat{p}_K^T W_p \hat{p}_K + \\ (\hat{x}_{k-N} - \bar{x}_{k-N})^T W_x (\hat{x}_{k-N} - \bar{x}_{k-N}) \end{aligned} \tag{3.27}$$

subject to:

$$\begin{aligned} \hat{x}_{k-N+i+1} &= f(\hat{x}_{k-N+i}, u_{k-N+i}, \hat{p}_k) \\ \hat{y}_{k-N+i} &= h(\hat{x}_{k-N+i}) \\ \eta_{k-N+i} &= y_{k-N+i} - \hat{y}_{k-N+i} \\ \hat{x}_{k-N+i} \in X,\ \eta_{k-N+i} &\in \Omega_\eta,\ \hat{p}_k \in \Omega_p \\ j &= 0,\ \ldots,\ N \end{aligned} \tag{3.28}$$

$\hat{p}_k$ is here the estimated uncertainty parameter. Moreover, similarly to the previous formulation, η_{k-N+j} is the output disturbance. However, it is worth noting that the state disturbance is not present. The model of the system $f(\hat{x}_k, \hat{u}_k, \hat{p}_k)$ is updated online since the parameter $\hat{p}_k$ is estimated by the MHE. The solution of the optimization problem leads thus to the optimal $\hat{p}_k$ that minimizes the difference between the estimated output $\hat{y}$ and the measured one y over the entire horizon N.

Chapter 4

System Description

The considered motion system is a Stewart platform, i.e., a parallel manipulator with six servo actuators attached to two flat plates. While the first plate represents the base of the platform, the other is moved by the actuators according to the inputs provided externally. The six degrees of freedom of the platform are the rotations and the traslations within the three axes x, y, z. Furthermore, a ball is placed on the top of the upper plate. The whole plant, comprised by the platform, the MPC solver and the serial communication between them, aims at balancing the ball in the middle of the plate. The communication happens with the use of an USB cable that connects the laptop to the hardware. The Stewart platform provides the latest measures of the position of the ball on the upper plate, while MPC feeds the plant with the inclination of the plate according to the received outputs. Since Model Predictive Control requires the knowledge of the full state vector, a Kalman filter is implemented to achieve it starting from the ball position in the x, y-axes. The whole plant comprises therefore an implementation of the MPC algorithm by the use of the acados library and the creation of a multi-threading Qt program where the control technique is combined with the communication process and a thread to periodically print useful results, as the state vector components and the inputs. Moreover, a small graphical user interface is created to start and stop the program. The software library acados is a collection of solvers able to provide both high performance and embedded deployability and this justifies its choice. Regarding the language, Qt creator relies on C++. A deeper description of the several components will take place in the following sections.

GPIOs groups	8
Operating Voltage	$1.8v - 3.3V$
Type of GPIO	Integrated 4−wire touchscreen controller
ADC	6−input 12−bit
Buffer depth	128
System Level	$25kV$ air-gap ESD protection
Device Level	$4kV$ HBM ESD protection

Table 4.1: Features of the STMPE610 GPIO

4.1 Components of the Stewart Platform

The Stewart platform employed for this project comprises different components, described in the following part of the section.
The upper plate of the platform under test is able to detect the position of an object through a touch screen. The sensing ability of the touch panel varies in accordance with the weight of the object, which in this case is a metallic ball. Therefore, the position of a heavier ball is more easily detected. The touch screen is controlled by an Adafruit STMPE610, i.e., an advanced touch screen controller with 6-bit port expander. It is a general purpose input-output port expander able to interface a main digital ASIC via a two-line bidirectional bus. A 4-wire touchscreen controller is built into the STMPE610, which is enhanced with a movement tracking algorithm to avoid excessive data. Furthermore, it possesses a 128×32 bit buffer and a programmable active window feature. Being a general purpose input-output, its function is to allow the microcontroller to interact with another peripheric, in this case the touch screen panel. Table 4.1 summarizes the characteristics of the implemented GPIO.
The six utilized servo actuators are Savox SC-0254MG, i.e., 3-pole DC motors which provide an accurate and fast response with fine resolution. Table 4.2 describes in further details these servo actuators.
Moreover, the hardware relies on an embedded systems platform development board and a polar PCB (printed circuit board). The resulting system makes use of a serial protocol, i.e., specifics rules to set up a connection between the platform and an external device. This protocol is described in the following subsection.

4.2 Serial Protocol

A universal asynchronous receiver-transmitter (UART) is a computer hardware device for asynchronous serial communication. The data format of this com-

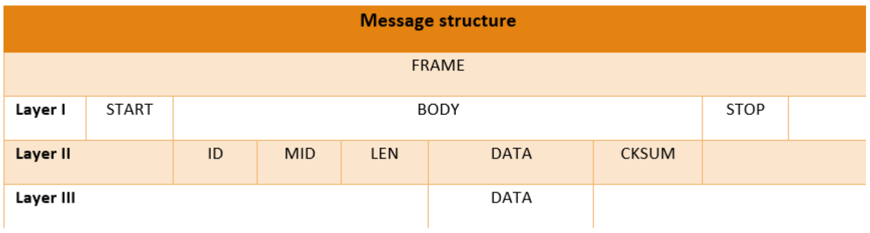
Servo Technology	Digital Low Voltage
Servo Case	Composite
Input Voltage	$4.8V - 6.0V$
Servo Type	Standard
Motor Type	Brushed DC Motor
Gear Material	Metal Gears
Ball Beared	Yes
Servo Power-Torque	$5 - 10Kg/cm$
Servo Speed	$0.10 - 0.15s/60^o$
Length	$40.7mm$
Width	$20.0mm$
Height	$39.4mm$
Weight	$49g$
Speed	$0.14s/60^o$
Torque	$7.2Kg/cm$
Connector Type	JR
Spline Size	25T Spline

Table 4.2: Technical Specifications of the Savox SC-0254MG servo motors

Baudrate	115200
Parity	Odd
Data Bits	8
Stop Bits	1
Hardware Flow Control	None

Table 4.3: UART settings

munication is configurable. The settings used in the considered application are summarized in table 4.3 Moreover, the message structure must be set. The implemented packet is described in 4.1. The start byte value is $0x02$, while the stop and the esc values are respectively $0x03$ and $0x10$. The latter is used to point out that the following byte is not to be interpreted as a special one, i.e., it is not a start, stop or an esc byte even though the value may indicate one of these. Before a message is accepted, its $8-$bit XOR-checksum is to be checked


Message structure

FRAME

Layer I	START	BODY					STOP	
Layer II		ID	MID	LEN	DATA	CKSUM		
Layer III					DATA			

Figure 4.1: Serial message structure

Field	Bytes
Frame	inf
Start	1
Stop	1
ID	1
MID	1
Length	1
Data	127
Checksum	1

Table 4.4: Size of message components

ID	Type	Content	Min	Max	Comment
1	Position	Uint32_t X_t	0	4000	Ball position
		Uint32_t Y	0	4000	
2	Get set point	Uint32_t X	800	3400	Target ball position with boundaries
		Uint32_t Y	600	3500	
3	Orientation	Int32_t tX	-	-	Set platform orientation
		Int32_t tY	-	-	
		Int32_t tZ	-	-	
		Int32_t rX	-	-	
		Int32_t rY	-	-	
		Int32_t rZ	-	-	
4	Set set point	Uint32_t X	800	3400	Target ball position with boundaries
		Uint32_t Y	600	3500	
5	Mode	Uint8 tM	0	4	Mode select/update

Table 4.5: Content of the message according to the ID field

against the content of the message, START and STOP bits excluded. Hence, if the checksum is correct the message can be processed further. Moreover, each message has a specific ID which determines its function, as welll as an MID which is only used in Multi-Point systems. The size of the different components of a message is explicited within table 4.4. The content of the message depends on the ID field, as showed in tab. 4.5. Moreover, several operating modes are planned:

- Balancer: the ball is balanced om the set point in the middle of the platform through the PID regulator previously implemented.
- Entertain: the ball moves along a predetermined path on the platform thanks to the PID regulator previously implemented.
- Follow: The user can decide a path using a Nunchunk controller or an external program.

- Stewart: An external interface can be used to manually set the orientation of the platform.
- Extern: An external interface can be used to manually set a new set point where the platform will balance the ball.

A message regarding the orientation of the plate, i.e., the one with ID 3, works only if the platform is in mode Stewart. In that mode indeed a program can set the orientation of the platform once the position of the ball is known. This is also the goal of this project: to write a code that, given the position of the ball, calculates the orientation of the plate with the use of the Model Predictive Control.

4.3 Interaction with the Hardware

The hardware is commanded through a serial communication that involves a laptop (and in particular a Qt program) and a cable to interact with the platform. The latter connects the embedded systems platform of the hardware to the laptop via usb communication.

4.3.1 Utilized Laptop

The used notebook is an HP Pavilion 15-ec1020nl. It provides an AMD Ryzen 7 4800H processor, a 8 gb ram, a solid state drive of 512 gb. Moreover, the graphics card is a NVIDIA geForce GTX 1650 4 gb.

4.3.2 Qt Program State Flow

To interact with the hardware Qt creator is used. This is a software that integrates C++, Javascript and QML with the possibility of creating graphical user interface applications. A small window is indeed created with three buttons: *start*, *stop*, *end program*. The user can command the flow of the program through these buttons. The *start* button starts the control of the platform, the *stop* one ends it, while the button *end program* closes the program.
The code implements a multi-threading program, where one thread is dedicated to the MPC controller, one to the serial communication, one to the log, i.e., the printing of the variables values, while the last one is the main thread. Qt creator provides signals and slots, which establish a clean and efficient way to

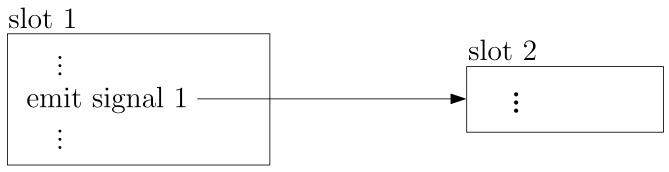
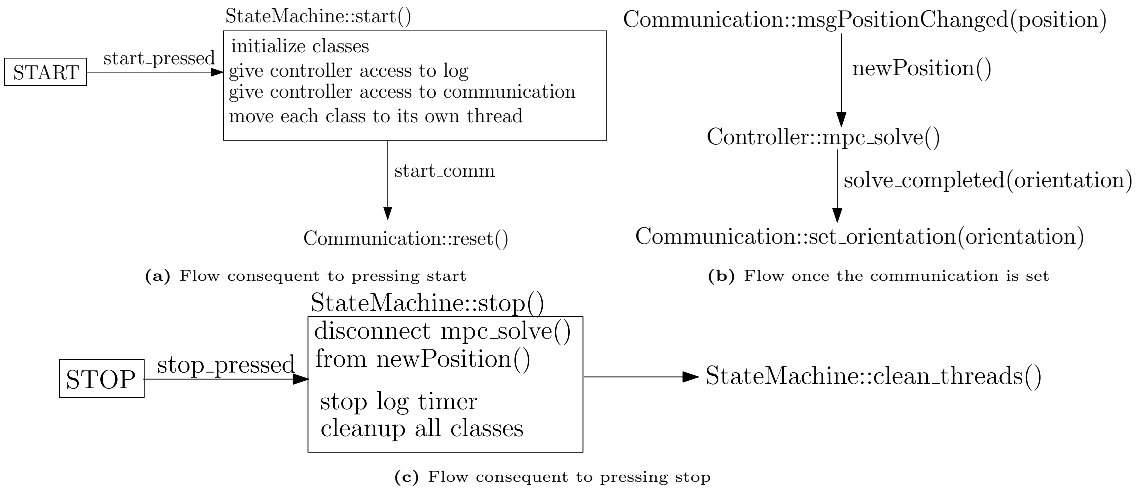


Figure 4.2: Signals and slots graphical representation: the slot 1 emits a signal 1 which has been previously connected to a slot 2. Therefore, the latter starts as soon as the signal 1 is emitted

move between threads. Signals and slots are connected: when either a slot or a function emit a signal, the slot connected to this signal starts to execute from the proper thread. Therefore, signals and slots define the flow of the program. Figure 4.2 gives a graphical representation of the signals-and-slots concept.
The flow of the program is not trivial and is to be illustrated graphically. The class *state machine*, which is driven by the three buttons previously presented, executes four slots: start(), set_timer(), stop() and clean_threads(). The first is connected to the signal emitted by starting the *start* button and does a series of operations which are useful to initialize every other class, i.e., the *controller* class, the *log* class and the *serial communication* class. Furthermore, it gives the controller access to both the log and the serial communication and moves every class to its own thread. Once the slot comes to its end, the signal start_comm() is emitted, which is connected to the *serial communication* class slot reset(). This resets the communication by doing proper operations and eventually emits the signal reset_done(), that is connected to the *state machine* class slot set_timer(). This connects the log to a timer, so that the variables values are printed on a regularly basis. After the reset of the serial communication, the hardware regularly sends a signal specifying the position of the ball (every $10ms$). This interval may be adjusted by modifying the firmware of the platform. When the signal is recognized to be in position, it is decoded and the *serial communication* class slot msgPositionChanged(position) is executed, which emits the signal newPosition(). The latter is connected to the *controller* class slot mpc_solve(). Therefore, every $10ms$ a new position is sent and the mpc controller calculates the optimal input. Since the controller has the angular acceleration as input of the model, while the platform needs to be fed with an orientation, the angles are calculated from the just calculated angular accelerations and the previous knowledge of the angular



StateMachine::start()
START
start_pressed
initialize classes
give controller access to log
give controller access to communication
move each class to its own thread
start_comm
Communication::reset()
(a) Flow consequent to pressing start
Communication::msgPositionChanged(position)
newPosition()
Controller::mpc_solve()
solve_completed(orientation)
Communication::set_orientation(orientation)
(b) Flow once the communication is set
StateMachine::stop()
STOP
stop_pressed
disconnect mpc_solve() from newPosition()
stop log timer
cleanup all classes
StateMachine::clean_threads()
(c) Flow consequent to pressing stop

Figure 4.3: Flow of Qt creator program. Once the start button is pressed, the classes controller, communication and log are initialized. Also, the controller is given access to log, communication and the three classes are moved to their own threads. After the communication is set, a new position received by the platform starts the mpc solving which eventually sends the orientation achieved. Once the stop button is pressed, the connection between the mpc solver and the received position is ended, the log timer is stopped and the classes are cleanup. Finally, the threads are quit.

velocities, amplitudes:

$$\begin{aligned}
\dot{\alpha}(k) &= \ddot{\alpha}(k) \cdot T_s + \dot{\alpha}(k-1), \\
\dot{\beta}(k) &= \ddot{\beta}(k) \cdot T_s + \dot{\beta}(k-1) \\
\alpha(k) &= (\dot{\alpha}(k) + \dot{\alpha}(k-1)) \cdot T_s/2 + \alpha(k-1), \\
\beta(k) &= (\dot{\beta}(k) + \dot{\beta}(k-1)) \cdot T_s/2 + \dot{\beta}(k-1)
\end{aligned} \tag{4.1}$$

Once the angles are reached, the signal solve_completed(a, b) is emitted: this is connected to the *serial communication* class slot send_orientation(a, b) which sends the angles to the platform. Afterwards, once a new position of the ball is received, the procedure is repeated again. This loop can be stopped by pressing the *stop* button, which calls the *state machine* slot stop(). This disconnects the slot mpc_solve() from the signal newPosition() and does the cleanup. Finally, it calls the *state machine* function clean_threads(), which quits every thread.

4.4 acados

4.4.1 Overview

To implement the MPC functions, as well as the simulation solver, a software library called *acados* is used. This is a collection of solvers for fast embedded optimization, intended for fast embedded applications. The main advantage of this library is its combination of both flexibility and efficiency. The former arises from its ability to interface with high-level languages, such as MATLAB and Python. This makes also acados user-friendly. On the other hand, the core of acados is written in C, which provides efficiency. The prerequisite for implementing MPC with real-time application is indeed the use of efficient optimal control methods written in low-level languages, e.g. C. Therefore, acados represents a strong candidate to implement an embedded optimal control method, like MPC is, for a motion control system, as the Stewart platform. acados does not rely on automatic code generation. This choice is due to the lack of flexibility that often rises subsequently to an automatic code generation process: the generated code works only for the fixed dimensions of the problem, so that any change requires the user to regenerate and recompile the whole solver. Moreover, an automatically generated code is sometimes hard to be read by the user, thus the debugging becomes a demanding procedure. Instead of an automatic code generation, with regard to the linear algebra operations, acados relies on the high-performance linear algebra package BLASFEO. On the other hand, the used library defining the quadratic programming problem is HPIPM. It defines three QP types (dense QP, OCP QP and tree-structured OCP QP), in addition to a broad set of routines to create, manage and solve the QPs. HPIPM uses the BLASFEO library to implement the QP solvers. Furthermore, the code is organized in a modular fashion, with formal interfaces between the different algorithmic components. This aims at reaching a straightforward way to interchange solvers, routines and libraries needed for the embedded control algorithm. Finally, the *CasADi* modeling language is used. This is based on expression graphs, which often leads to shorter instruction sequences and smaller and faster code. This is also crucial for embedded applications.

4.4.2 BLASFEO

Embedded optimization libraries always rely on linear algebra operations: their implementation may be either through a small set of linear algebra routines or a call to a specialized linear algebra library. In this sense, BLASFEO is a linear algebra package which achieves good performances with regard to small matrices, the ones typically encountered in embedded optimization [23]. Moreover, a packed matrix format called panel major is used: this optimizes the cache usage so that high performance is achieved for matrices of sizes uo to hundreds. In conclusion, BLASFEO defines a complete linear algebra framework and enables a considerable speedup in the matrix computations, except for really small sizes matrices.

4.4.3 HPIPM

As previously mentioned, HPIPM is a quadratic programming library that develops three QP types, i.e., dense QP, OCP QP, tree-structured OCP QP [24].It also defines several routines to deal with the QPs and to convert between the different QP types. For instance, condensing routines convert an OCP QP into a dense QP, while expansion routines transform a dense QP solution ino an OCP QP solution. The reliance on HPIPM leads therefore to a wide set of QP types and routines, which render acados pretty flexible and suitable for different optimization applications.

4.4.4 acados Core Library

In order to be both user-friendly at a high level and efficient at a low level, acados has a core library written in C with Python and MATLAB interfaces. A modular fashion exists between the embedded optimization algorithms, which means there are clear interfaces between these algorithms. Each of these algorithmic components are designed as separate modules, some of which may be used by themselves while others need to be combined together. Table 4.6 illustrates an overview of all modules present in acados and their algorithmic variants. All modules look identical in their signature, so that an extension of acados through the insertion of another module is straightforward. Moreover, some modules may comprise other modules, e.g. an integrator is needed by an SQP solver. In this case, the comprised module , i.e., the integrator, takes the name of submodule. An example of the relationship between an NLP solver and its submodules is showed in figure 4.4. In summary, the core library of acados contains mostly a collection of modules, each with variants of solvers and different data types. However, directly using the core library can be error-prone and cumbersome, reason why high-level interfaces are present as well.

4.4.5 C Interface

The low-level constructs of the acados core are encapsulated by the acados C interface. One of these regards the choice of the solvers. All the functions within

Module	Variants
OCP QP	HPIPM qpDUNES HPMPC OOQP OSQP
Dense QP	HPIPM qpOASES OOQP
Condensing	Full condensing (HPIPM) Partial condensing (HPIPM)
Simulation	ERK IRK GNSF-IRK lifted IRK
OCP NLP	Gauss-Newton SQP Gauss-Newton SCQP Exact-Hessian SQP RTI
Regulatization	Projection Mirroring Convexification
Nonlinear function	CasADi generated functions C-code functions

Table 4.6: Software modules in acados

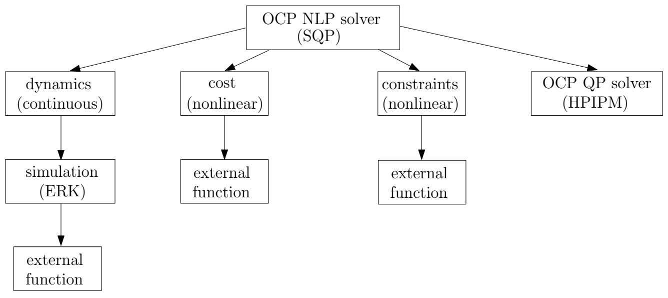


Figure 4.4: Example of the relation between modules and submodules in acados

the core library are indeed specific to one variant of a module. The goal is thus to facilitate the switch between solvers, and this is reached through the definition of a plan. The latter is a struct that incorporates several fields representing a particular combination of solvers.
In addition, a structure for passing the chosen options is to be implemented. Since dictionaries are not available in C, a specific struct is used where functions are represented via strings. The strings encapsulate both the modules of the options and the names of the options.
Another aspect concerns the memory management. Since allocating memory manually might be cumbersome, some routines that manage the process are available. Finally, the C interface provides the user with some helper routines, called setters and getters. Their function is to access the low-level structs of the acados core.

4.4.6 High-Level Interfaces

acados offers interfaces to three popular languages for scientific computing: C++, MATLAB and Python. This aims at offering the possibility to use the acados library even to non-expert users, that may face difficulties in writing C code manually. These high-level interfaces use CasADi, a C++ code, as a modeling language. The first benefit of using CasADi is the modeling and code generation of non-linear functions and derivatives. Furthermore, the solution provided by acados can be compared with solutions from different optimization libraries, given that these are interfaced with CasADi too. The workflow provides that firstly a description of the OCP to be solved is implemented by the high-level interfaces. Secondly, a self-contained C project can be generated that contains all the necessary function evaluations and the OCP and NLP solvers. In addition, a MATLAB S-function and a build system are generated which can be utilized in order to compile the code. Therefore, a high-performance and self-contained solver is reached from a description in a high-level interface. This can be finally deployed with reference to embedded applications.

4.5 Design of the MPC

As already illustrated in section 2.1, and in particular within subsections 2.1.3 and 2.1.5, several design choices are necessary to obtain a proper controller. In particular, crucial decisions concern the modelization of the plant to control, the

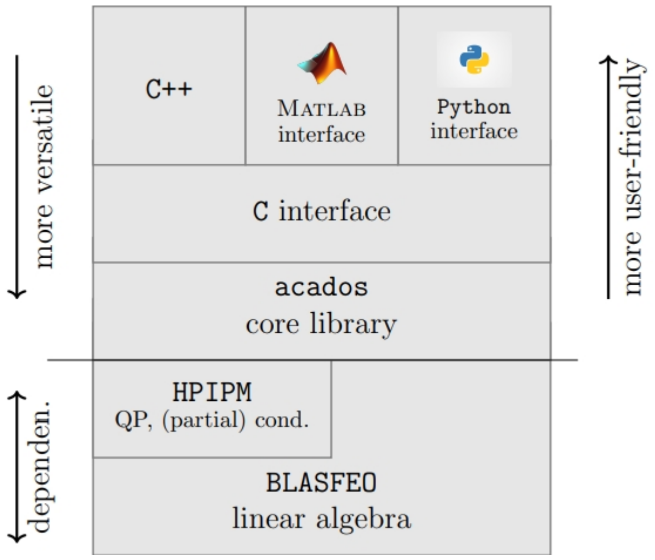


Figure 4.5: acados structure: its dependencies, the core C library and its high-level interfaces

form of the cost function and the form of the constraints. Other possible decision options, such as the tuning of the cost function and constraints and the features of the solver, will be tackled directly within the chapters regarding the simulation and experimental results.

4.5.1 Prediction Model

The employed model of the system, referred to as prediction model in subsection 2.1.5, is the linearized one presented in 4.5.1. However, the 4×4 system expressed through equation 1.34 does not allow to set a proper set of constraints, since these may concern only the state and the input vectors components. The considered system permits indeed to set constraints on the two angles of the upper plate, while its angular velocities and accelerations are free. However, the servos are subject to velocity and acceleration restrictions, according to the data sheet, and therefore the upper plate has limitations regarding velocity and acceleration as well. In order to set angular velocities and accelerations constraints, the system is augmented to a 8×8 linear one:

$$
\begin{aligned}
x &= \begin{bmatrix} x_b & \dot{x}_b & \alpha & \dot{\alpha} & y_b & \dot{y}_b & \beta & \dot{\beta} \end{bmatrix}^T, \\
u &= \begin{bmatrix} \ddot{\alpha} & \ddot{\beta} \end{bmatrix}^T, \\
\dot{x} &= \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{5}{7}g & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & \frac{5}{7}g & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \cdot x + \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1
\end{bmatrix} \cdot u
\end{aligned} \tag{4.2}
$$

The modified system is now able to deal with constraints regarding the angular velocities and accelerations of the plate, that translate also in velocity and acceleration constraints of the servo-motors.

4.5.2 Cost Function

A quadratic cost function is employed according to the following expression:

$$
J(x(\dot{}), u(\dot{})) = \sum_{i=0}^{N} (x(k+i)^T Q x(k+i)) + \sum_{i=0}^{N-1} (u(k+i)^T R u(k+i)) \tag{4.3}
$$

The weighting matrices Q, R are to be positive semi-definite. A quadratic cost function permits to solve the problem through the use of quadratic programming and consequently to employ sequential quadratic programming, possibly with the real time iteration scheme. The latter allows to solve the MPC with higher speed and is crucial to control the experimental system in a proper way, as will be clear from the experimental results.

4.5.3 Constraints

As explained in subsection 4.5.1, the model is augmented to include also the angular velocity and acceleration within respectively the state and the input vector. The main reason is the willing to set constraints also for these quantities. Therefore, the constraints regard the following physical variables:

- The position of the ball within both the $x-$ and $y-$ axes, i.e., x_b and y_b
- The velocity of the ball within both the $x-$ and $y-$ axes, i.e., $\dot{x}_b$ and $\dot{y}_b$
- The angular rotation of the plate within both the $x-$ and $y-$ axes, i.e., α and β
- The angular velocity of the plate within both the $x-$ and $y-$ axes, i.e., $\dot{\alpha}$ and $\dot{\beta}$
- The angular acceleration of the plate within both the $x-$ and $y-$ axes, i.e., $\ddot{\alpha}$ and $\ddot{\beta}$

For all these variables, the chosen form of the constraints is the classical one, which is a bound between two limits:

$$a_{min} \leq a \leq a_{max} \tag{4.4}$$

where a is the variable trajectory and a_{min} and a_{max} are respectively the lower and upper limits.

Chapter 5

Simulation Results

The simulation attempts are carried out in Matlab with the use of the acados library. The latter includes an ocp model, which represents the MPC, and a simulated model. The first calculates the optimal inputs starting from the knowledge of the actual state, which is provided either by the simulated system or by a combination of it and the Kalman filter. Indeed, at first the simulation trials pretend to have full access to the state vector. A comparison between different choices for the ocp model is made: within section 5.1 the MPC relies on the real model, i.e., the same as the simulation system. This first sections aims at verifying the applicability of Model Predictive Control with the required time and variables constraints. On the contrary, section 5.2 will introduce a mismatch between the two models: in particular, several approximations will be made for the ocp model, such as the neglect of some force components as well as a linearization of the system. Subsequently, section 5.3 will analyze the best choice for the angles constraints and section 5.4 the best one for the control sample rate. Moreover, an angle calibration error will be taken into account within section 5.5, which brings to steady state error in position. To compensate for this error, two methods are proposed: section 5.6 includes the integral of the position inside the state vector, while section 5.7 implements a Kalman filter with an augmented state space vector. This filter is useful in case of unmeasured state components, which is the case of the experimental setup. It is indeed able to estimate the full state vector in presence of a noisy environment. Its behaviour will be shown within the section.

5.1 Behaviour of the Nominal System

The first MPC simulation relies on the same simulation and ocp models, which are expressed by equations 1.25, 1.26, 1.27. The initial state is fixed at $x_0 = \begin{bmatrix} x_b \ \dot{x}_b \ \alpha \ \dot{\alpha} \ y_b \ \dot{y}_b \ \beta \ \dot{\beta} \end{bmatrix}^T = \begin{bmatrix} 0.03 \ 0 \ 0 \ 0 \ 0.05 \ 0.1 \ 0 \ 0 \end{bmatrix}^T$. The weight matrices of this first simulation, as well as the ones used in the subsequent sections, are:

$$Q = \begin{bmatrix} 10^5 &&&&&&& \\ & 10^2 &&&&&& \\ && 10^{-2} &&&&& \\ &&& 10^{-3} &&&& \\ &&&& 10^5 &&& \\ &&&&& 10^2 && \\ &&&&&& 10^{-2} & \\ &&&&&&& 10^{-3} \end{bmatrix}, \qquad R = \begin{bmatrix} 10^{-5} & \\ & 10^{-5} \end{bmatrix} \tag{5.1}$$

On the other hand, the following constraints are set:

$$|x| \leq \bar{x}, \ |u| \leq \bar{u}$$
$$\bar{x} = \begin{bmatrix} 0.2 \\ 2 \\ 0.4 \\ 20 \\ 0.2 \\ 2 \\ 0.4 \\ 20 \end{bmatrix}, \ \bar{u} = \begin{bmatrix} 200 \\ 200 \end{bmatrix} \tag{5.2}$$

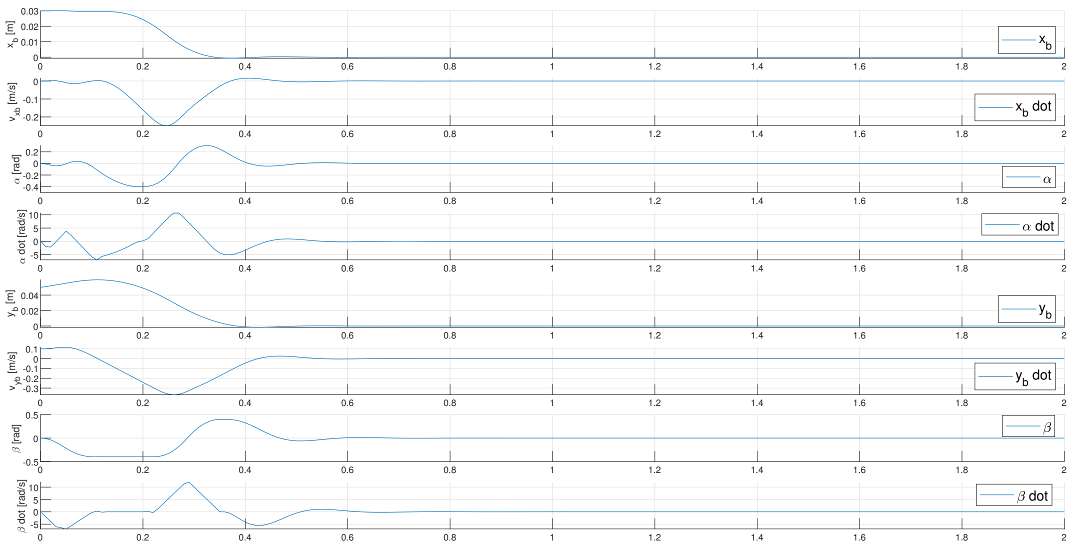


Figure 5.1: Trajectory of the states

Figures 5.1, 5.2 and 5.3 illustrate respectively the state and input trajectories and the required computational time. The final state error is:

$$e_x = \begin{bmatrix} 0.0011 \\ -0.0018 \\ 0.0180 \\ -0.2202 \\ 0.0025 \\ -0.0063 \\ -0.0383 \\ 0.5161 \end{bmatrix} \cdot 10^{-3}$$

This error is due to the sub-optimal solution found by the MPC, due to constraints. The computational time is $T_{COMP} = 1.0781ms$, split into $t = 0.725944ms$ for the linearization of the system, $t = 0.15991ms$ for the QP solution and $t = 0.191982$ for remaining actions. These values are quite high, especially the time required for the linearization. A better solution would demand a smaller computation time and in particular a quicker linearization process. In the following section an effort is thus carried out in that direction.

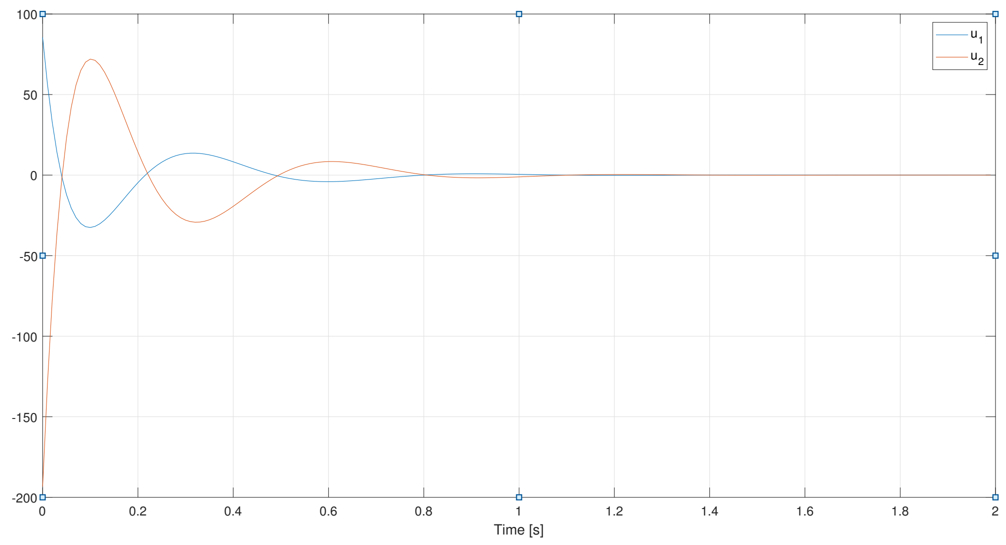


Figure 5.2: Trajectory of the input

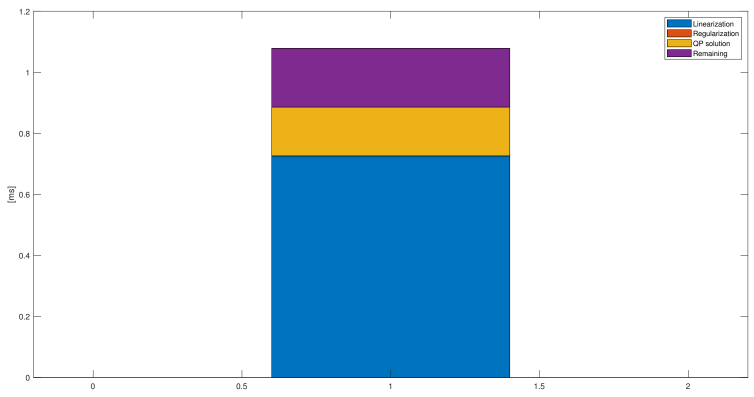


Figure 5.3: Computation time

5.2 Consistency between the simplified and the real systems

In this section an examination will be carried out of the errors and the imperfections which the simplification of the model leads to, as well as the improvements in the computational time. In order to do that, the ocp model is based on a simplified system, while the sim model relies on the nominal system. The final state error, together with the state and input trajectories and the computation time, are analyzed. The initial state is always fixed at $x_0 = \big[0.03\ 0\ 0\ 0\ 0.05\ 0.1\ 0\ 0\big]^T$.

5.2.1 Real Sim vs Simplified OCP

Within this subsection the simulation is built on the following equations:

$$\dot{x}_1 = x_2,\ \dot{x}_3 = x_4,\ \dot{x}_4 = u_1,\ \dot{x}_5 = x_6,\ \dot{x}_7 = x_8,\ \dot{x}_8 = u_2$$

$$\dot{x}_2 = \frac{5}{7}\big(x_1 x_4^2 + x_5 x_4 x_8 + g \sin x_7\big)$$

$$\dot{x}_6 = \frac{5}{7}\big(x_5 x_8^2 + x_1 x_4 x_8 + g \sin x_3\big)$$

On the other hand, the OCP follows the equations below:

$$\dot{x}_1 = x_2,\ \dot{x}_3 = x_4,\ \dot{x}_4 = u_1,\ \dot{x}_5 = x_6,\ \dot{x}_7 = x_8,\ \dot{x}_8 = u_2$$

$$\dot{x}_2 = \frac{5}{7} g \sin x_7$$

$$\dot{x}_6 = \frac{5}{7} g \sin x_3$$

The state error is represented by the following vector:

$$e_x = \begin{bmatrix} 0.0014 \\ -0.0048 \\ 0.0207 \\ -0.3075 \\ 0.0032 \\ -0.0070 \\ -0.0515 \\ 0.6666 \end{bmatrix} \cdot 10^{-3}$$

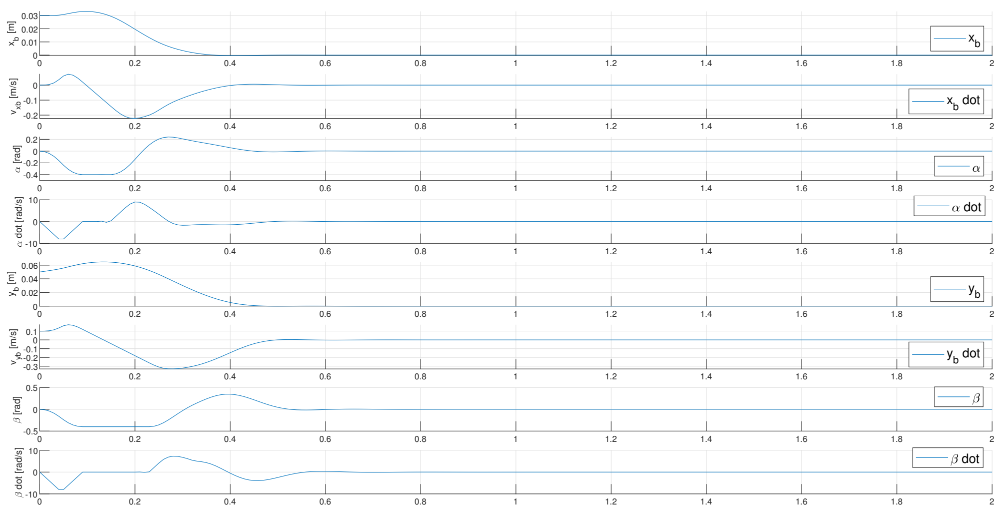


Figure 5.4: Trajectory of the states with the simplification of the ocp model equations

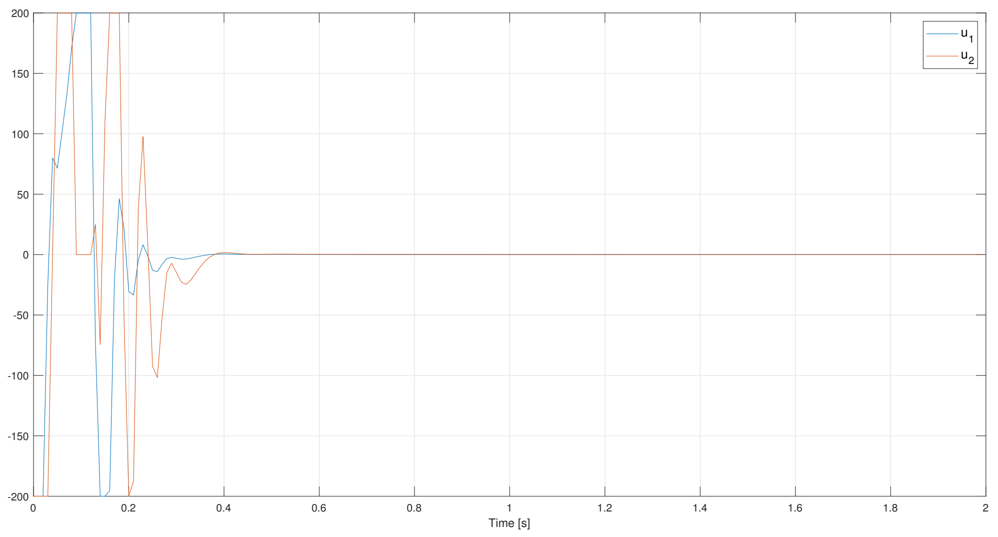


Figure 5.5: Trajectory of the input with the simplification of the ocp model equations

Fig. 5.4, 5.5, 5.6 illustrate the trajectories of the state, input and the computational time. In particular, the required computation time is $T_{COMP} = 0.936604ms$: $t = 0.617481ms$ for the linearization, $t = 0.145668ms$ for the QP solution and $t = 0.173243$ for the remaining actions. The decrease of the time required by the linearization process is visible, even though still quite big. On the other hand, the final error state shows a deterioration regarding some components, e.g. $x_2 = \dot{x}_b$, being nonetheless still acceptable.

5.2.2 Real Sim vs Discretized OCP

An even bigger simplification of the ocp model is now provided, i.e., the linearization and the discretization of it. Therefore, the new ocp model is described by

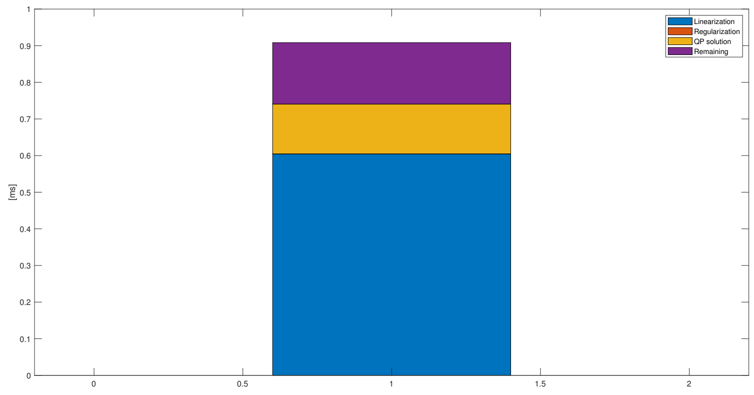


Figure 5.6: Computation time with the simplification of the ocp model equations

the equations below:

$$\dot{x}_1 = x_2,\ \dot{x}_3 = x_4,\ \dot{x}_4 = u_1,\ \dot{x}_5 = x_6,\ \dot{x}_7 = x_8,\ \dot{x}_8 = u_2 \tag{5.3}$$

$$\dot{x}_2 = \frac{5}{7} g x_7 \tag{5.4}$$

$$\dot{x}_6 = \frac{5}{7} g x_3 \tag{5.5}$$

The sim model is always represented through what is supposed to be the real system, i.e. equations 1.25, 1.26, 1.27. The final state error becomes:

$$e_x = \begin{bmatrix} 0.0014 \\ -0.0047 \\ 0.0208 \\ -0.3062 \\ 0.0032 \\ -0.0058 \\ -0.0522 \\ 0.6501 \end{bmatrix} \cdot 10^{-3}$$

This is very similar to the previous one, showing some small improvements or deteriorations according to the different state variables. Fig. 5.7, 5.8, 5.9 illustrate the trajectories of the variables and the computation time. The latter in particular shows a significant improvement, becoming indeed $T_{COMP} = 0.426027ms$: the QP solution requires $t = 0.144074ms$, the remaining $t = 0.161077$ and the discretization $t = 0.120778$. However, the latter is referred to the discretization

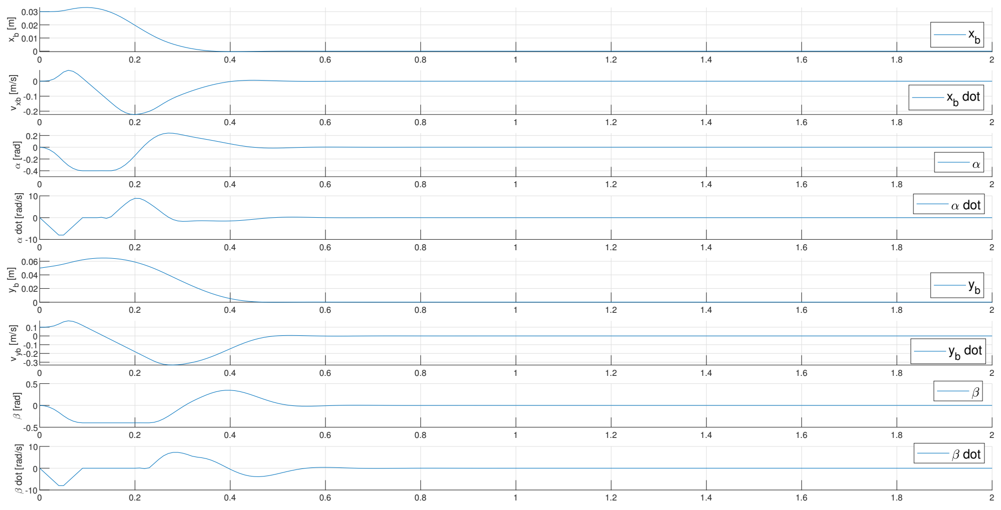


Figure 5.7: Trajectory of the states with the discretization of the ocp model equations

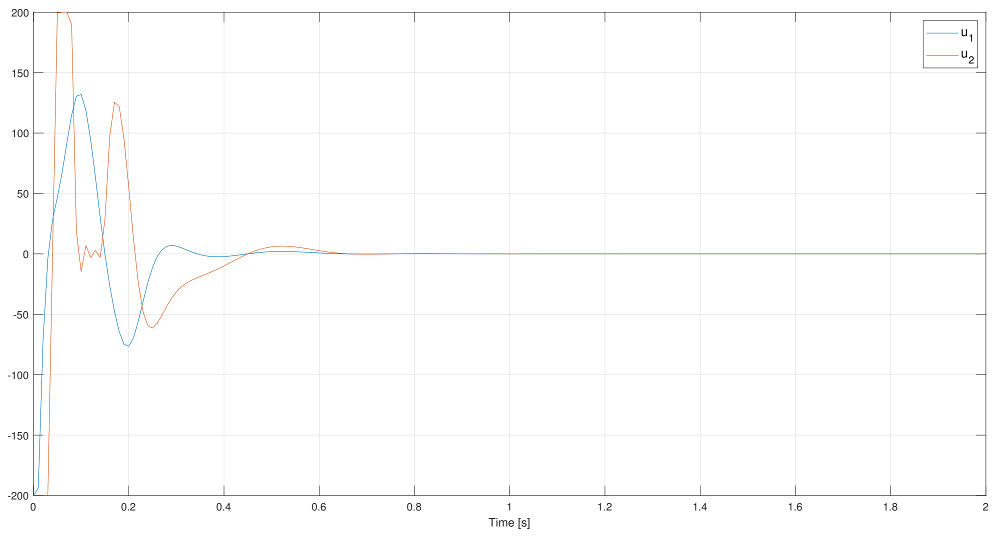


Figure 5.8: Trajectory of the input with the discretization of the ocp model equations

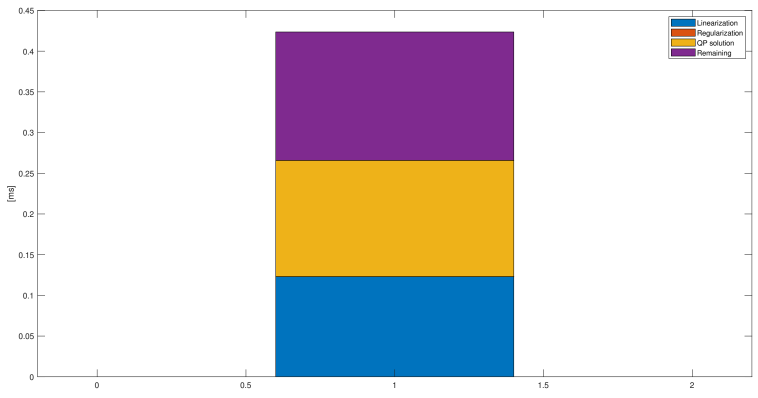


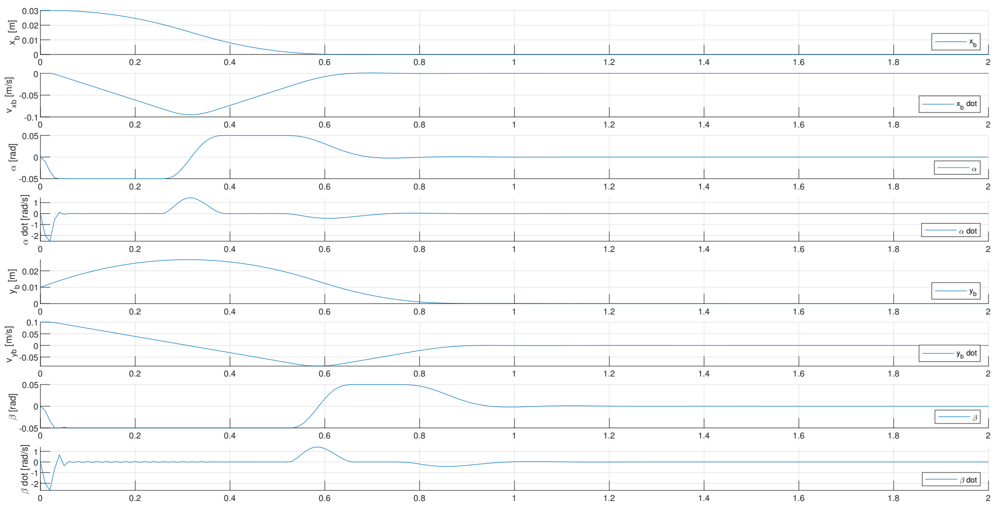
Figure 5.9: Computation time with the discretization of the ocp model equations

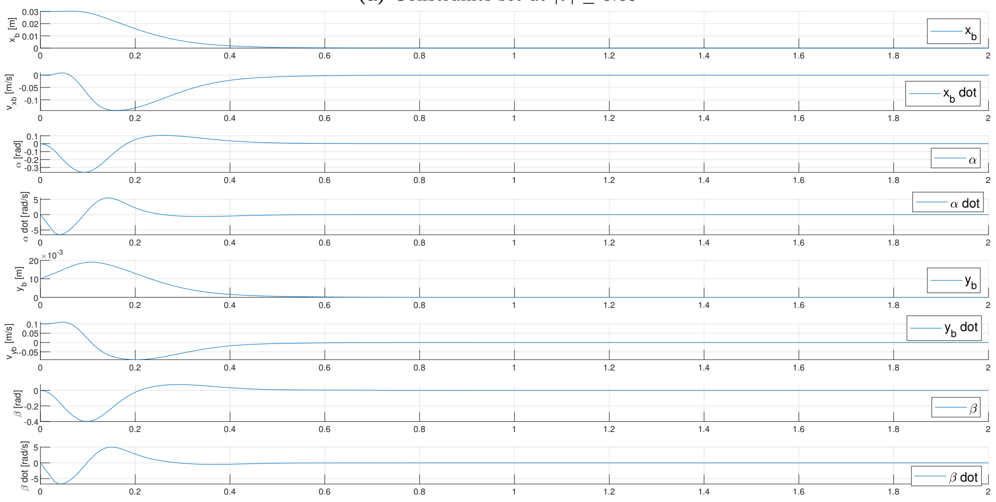
of the simulation model rather than the ocp model, which is already a discrete system. Therefore, this time would not be present with an experimental system replacing the simulated one. The concern about the high computation burden required by the linearization of the ocp model seems therefore to be solved.

5.3 Angles Constraints

As it can be seen from fig. 5.7, β rages between $-0.4rad$ and $0.3rad$. These values are a bit too far from the origin, which means that the approximation $\sin\theta \approx \theta$ is not so precise. In particular, $\sin(0.4rad) = 0.389$ and $sin(0.3rad) = 0.295$. Constraints are therefore to be established for the angles amplitudes. In particular, the stricter the constraints are, the smaller the final state error is. However, a smaller interval $[\theta_{min}, \ \ \theta_{max}]$ requires more computational time and the convergence results to be slower. Anyway, constraints are to be set, since a different initial state could require even bigger angles and therefore unacceptable inaccuracies. The trade-off between computational effort and accuracy of the approximation is to be analyzed and properly set. In order to do this, the initial state is always set to $x_0 = \begin{bmatrix}0.03 \ 0 \ 0 \ 0 \ 0.01 \ 0.5 \ 0 \ 0\end{bmatrix}^T$ and different behaviours are compared. Initially, the acceptable interval is $[-\pi \ \ \pi]$, which is equivalent to not having any constraint. Secondly, a pretty strict interval is set, i.e., $[-0.05 \ \ 0.05]$. This would almost perfectly satisfy the approximation $\sin\theta \approx \theta$, giving rise however to other issues. These are compared to $[-0.4 \ \ 0.4]$ and $[-0.5 \ \ 0.5]$. The first choice, i.e., setting no constraints, results in an unstable system, as expected.
Figures 5.10 illustrate the different state trajectories according to the chosen constraints. The final state errors are respectively:

$$e_{0.05} = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0001 \\ -0.0592 \\ -0.0045 \\ 0.0500 \\ -0.0002 \end{bmatrix}, \quad e_{0.4} = \begin{bmatrix} -0.0009 \\ -0.0013 \\ -0.0196 \\ 0.2419 \\ -0.0030 \\ 0.1540 \\ 0.2506 \\ 0.5993 \end{bmatrix} \cdot 10^{-5}, \quad e_{0.5} = \begin{bmatrix} -0.0001 \\ -0.0016 \\ -0.0046 \\ 0.0178 \\ -0.0023 \\ 0.0873 \\ -0.0954 \\ -0.6547 \end{bmatrix} \cdot 10^{-4}$$


(a) Constraints set at $|\theta| \leq 0.05$


(b) Constraints set at $|\theta| \leq 0.4$

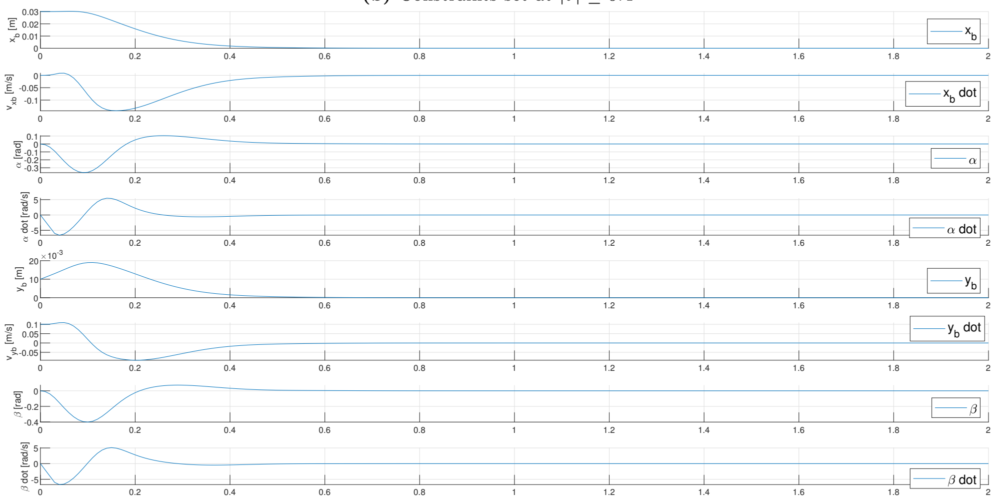
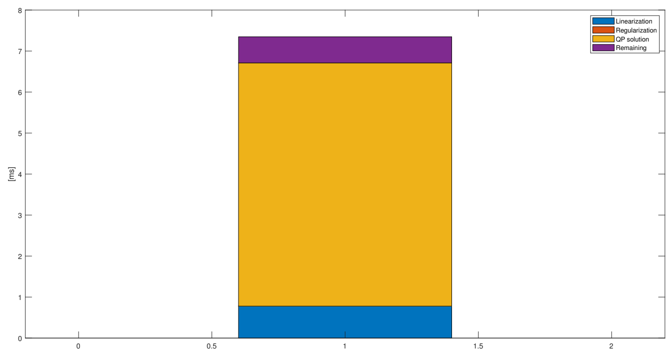
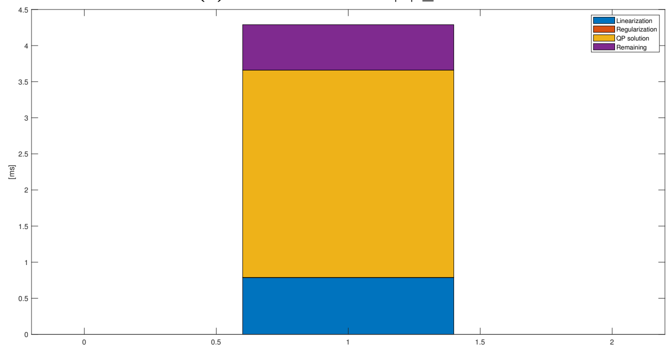

(c) Constraints set at $|\theta| \leq 0.5$

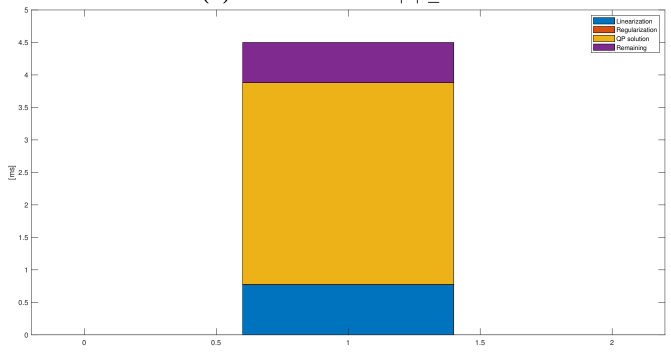
Figure 5.10: State trajectory according to the different constraints



(a) Constraints set at $|\theta| \leq 0.05$



(b) Constraints set at $|\theta| \leq 0.4$



(c) Constraints set at $|\theta| \leq 0.5$

Figure 5.11: Computation time according to the different constraints

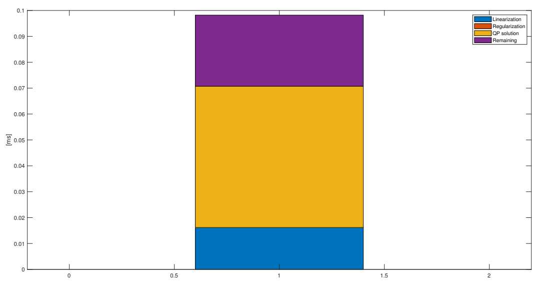
The higher error $e_{0.05}$ is due to the more time required to achieve the setpoint with stricter constraints. In addition, the computation times are $T_{0.05} = 7.34723ms$, $T_{0.4} = 4.28977ms$, $T_{0.5} = 4.50007ms$. As one would expect, the computation time decreases from $|\theta| \leq 0.05$ to $|\theta| \leq 0.4$. The fact $T_{0.5} \geq T_{0.4}$ is related to the random component included in the computation time: the latter depends indeed from e.g. initial guesses and their suitability to the optimal trajectory, so that it may slightly vary from one execution to the other. It may be assumed thus $T_{0.05} \approx T_{0.04}$. From these considerations, the constraints $\alpha_{min} = \beta_{min} = -0.4$, $\alpha_{max} = \beta_{max} = 0.4$ are chosen for the next simulations.

5.4 Control Sample Rate

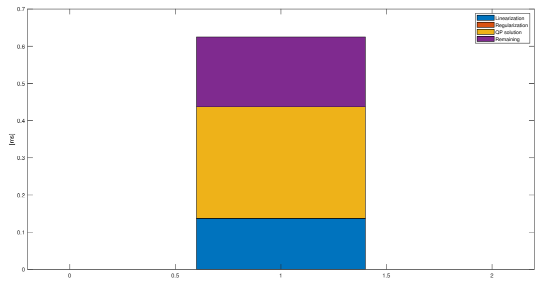
The aim is now to compare different control sampling rates and to select the most proper one. The degree of freedom is thus the number of control steps N, which determines also the time slice $T_s = T_{OCP}/N$. The minimum value is $N = 5$, due to the QP solver. Different values which have been tested are $N = 10$, $N = 20$, $N = 50$, $N = 100$, $N = 500$, $N = 1000$. A higher N does not allow the software to properly solve the problem. With the increase of the number of steps, the trajectories become smoother at the expense of the computational time. The different state trajectories are highlighted through figures 5.14, 5.15, while images 5.12 illustrate the different computation times. The initial state is always fixed at $\begin{bmatrix}0.03, & 0, & 0, & 0, & 0.05, & 0.1, & 0, & 0\end{bmatrix}^T$. As illustrated in fig. 5.12, the computational time deeply increases with the number of time steps. In particular, the QP solution is the most affected part. Figure 5.13 illustrates the esponential trend of the required computational time. Therefore, a too high number of N is not acceptable. Figures 5.14, 5.15 show the different trajectory of the state vector. Increasing N leads to a behavior which is closer to the reality. However, since the computation time increases as well, the choice is directed towards a number of steps that permits to have an acceptable reality approximation without requiring too much computational burden. In order to do that, also the final error is evaluated. The errors related to $N = 50$, $N = 100$ are similar, despite a double computation time for $N = 100$. Therefore, the best choice seems to be $N = 50$.

5.4.1 Finer Grid for the Simulation Model

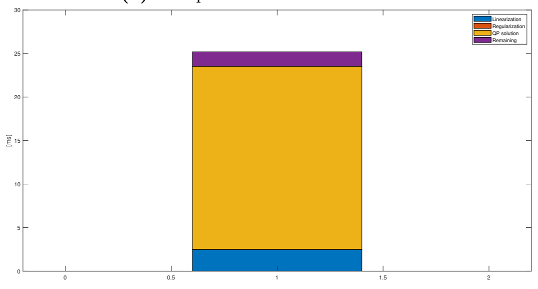
Another possibility provided by the software consists in using a finer grid for the sim model. This means that the simulation divides the single time step of



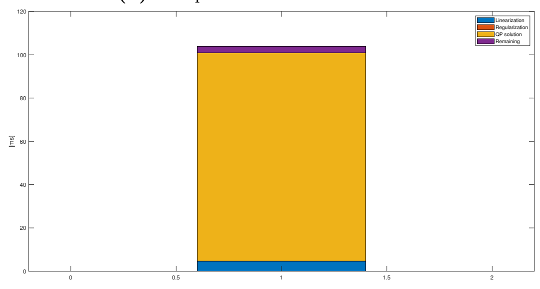
(a) Computation time with $N = 5$



(b) Computation time with $N = 50$



(c) Computation time with $N = 500$



(d) Computation time with $N = 1000$

Figure 5.12: Computation time according to different control steps

the optimal control problem in sub-slices in order to better approximate the real plant, which operates in the continuous time. However, if the number of timesteps N is properly chosen, the finer grid does not provide different behaviors. The final error changes indeed only if $N = 5$ or $N = 10$, switching for instance between $N' = 1$ and $N' = 10$ for the finer grid. This means that, by choosing a proper number of time steps, the software is able to simulate properly the whole system without the need of a finer grid for the simulation model. Therefore, $N = 50$, $N' = 1$ can be used.

5.5 Angle Calibration Error

MPC is a control technique where open-loop and closed-loop coexist: the cost function is indeed based on an open-loop system, however the state feedback is assumed as an initial condition in every time step. There should be therefore a certain grade of rejection regarding plant-model mismatch or unmeasured disturbances. This can be observed by assuming the angles amplitudes are wrong: in simulation, this can be easily implemented by writing $\sin(\theta + e)$ instead of $\sin\theta$ within the sim model, without modifying the ocp model. Fig. 5.16 show the behaviour of the system in presence of an error $e = 0.2rad$. Thanks to the state feedback and its weight, the system is capable of detecting the presence of a disturbance (in this case an angle calibration error) and tries to stabilize the sys-

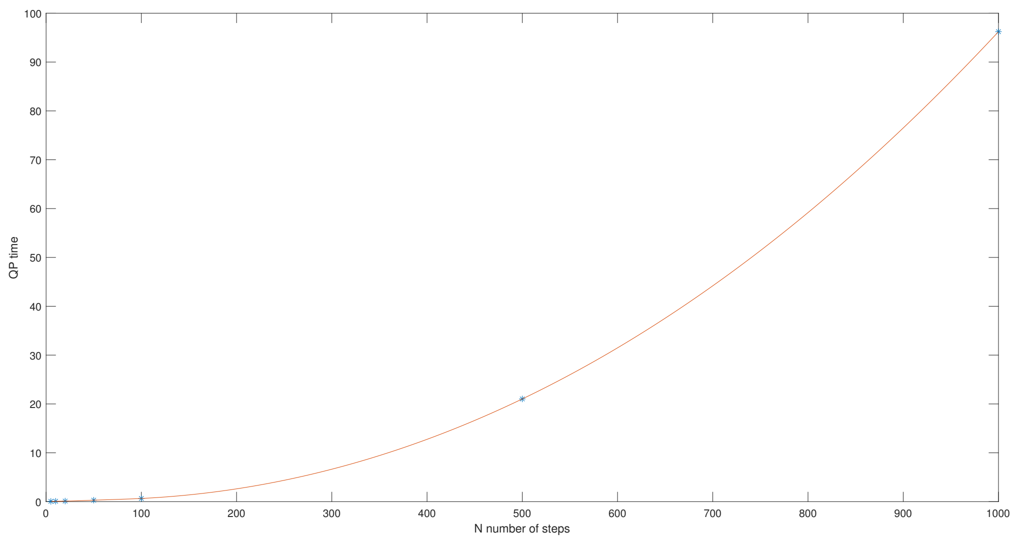
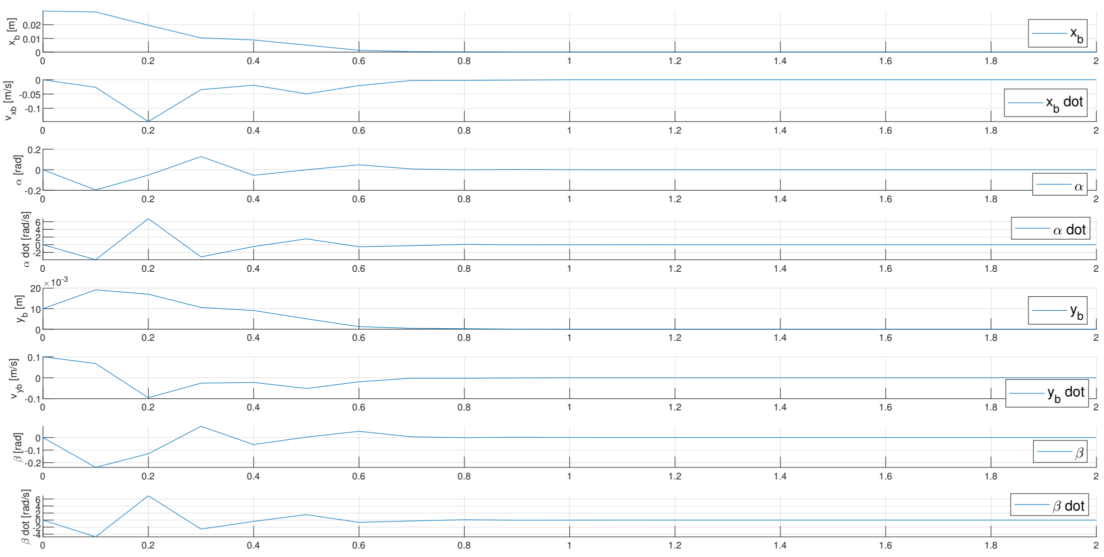


Figure 5.13: Computation time trend

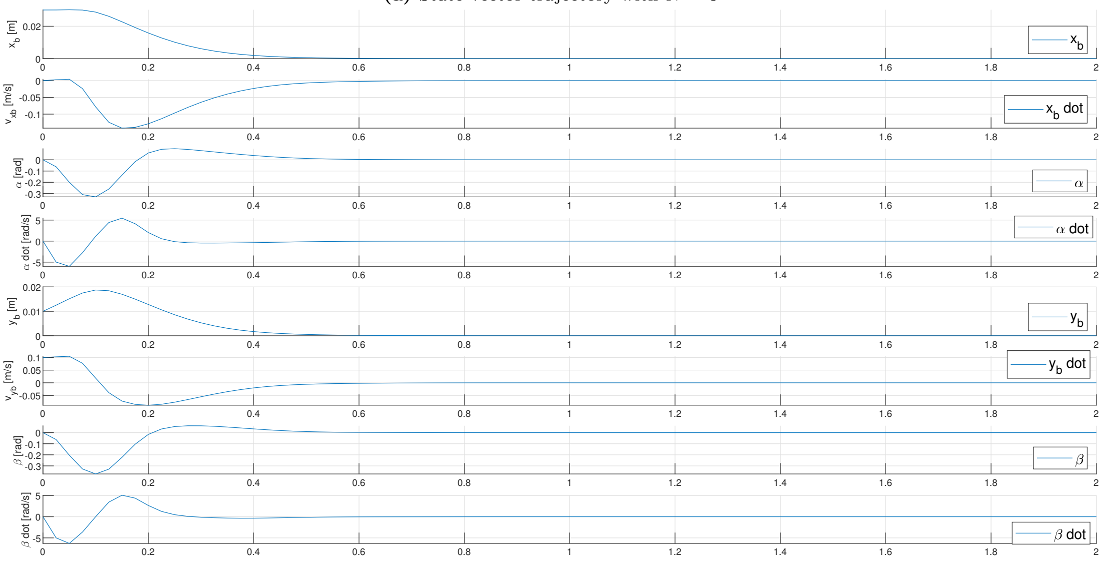
tem. In particular, at steady state the two estimated angles become $\alpha = -0.2rad$, $\beta = -0.2rad$ in order to stop the ball velocity. With these values the plate is indeed horizontal. Nevetheless, there is a steady state offset, since the ball is now still but not at its setpoint, i.e., $x_{b,f} \neq 0$, $y_{b,f} \neq 0$. The amplitude of this steady state error depends on the cost function J, the constraints and the equations of the system. This is exactly what one would expect, as an MPC without any extension is similar to a PD controller regarding the rejection of disturbances. Therefore, a method to reject modeled disturbances is to be included. This will be illustrated in the following sections through two different approaches: firstly, the integral component of the position is included in the state vector. This way, the steady state error moves from the position of the ball to its integral, leading to zero steady state error in position. This is the same principle of passing from a PD regulator to a PID. Secondly, a Kalman filter is included to estimate the modeled disturbances.

5.6 Inclusion of the Integral Component

In order to achieve zero steady state offset, the integral of the ball position is initially included in the state vector. Therefore, the state space system becomes



(a) State vector trajectory with $N = 5$

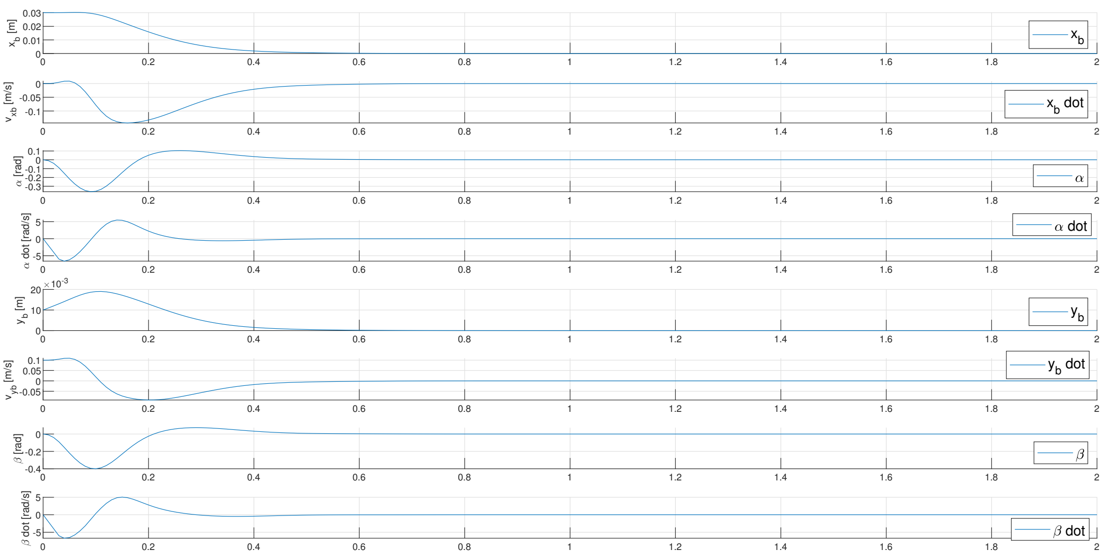


(b) State vector trajectory with $N = 20$

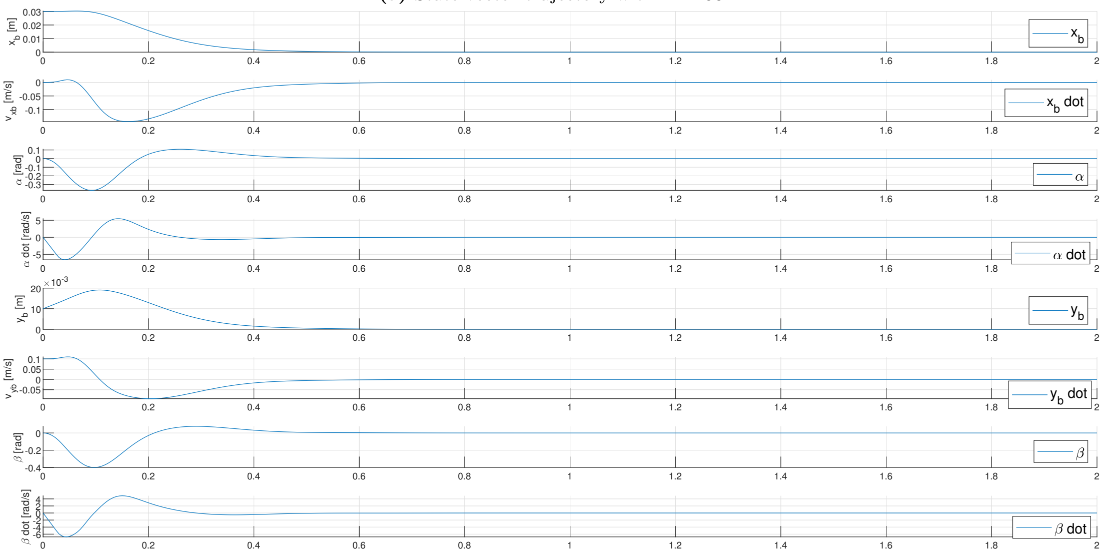
Figure 5.14: State vector trajectory according to different control steps ($N = 5$, $N = 20$)

as follows:

$$x = \begin{bmatrix} \int x_b dt \\ x_b \\ \dot{x}_b \\ \int y_b dt \\ y_b \\ \dot{y}_b \end{bmatrix}, \; u = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{5.6}$$

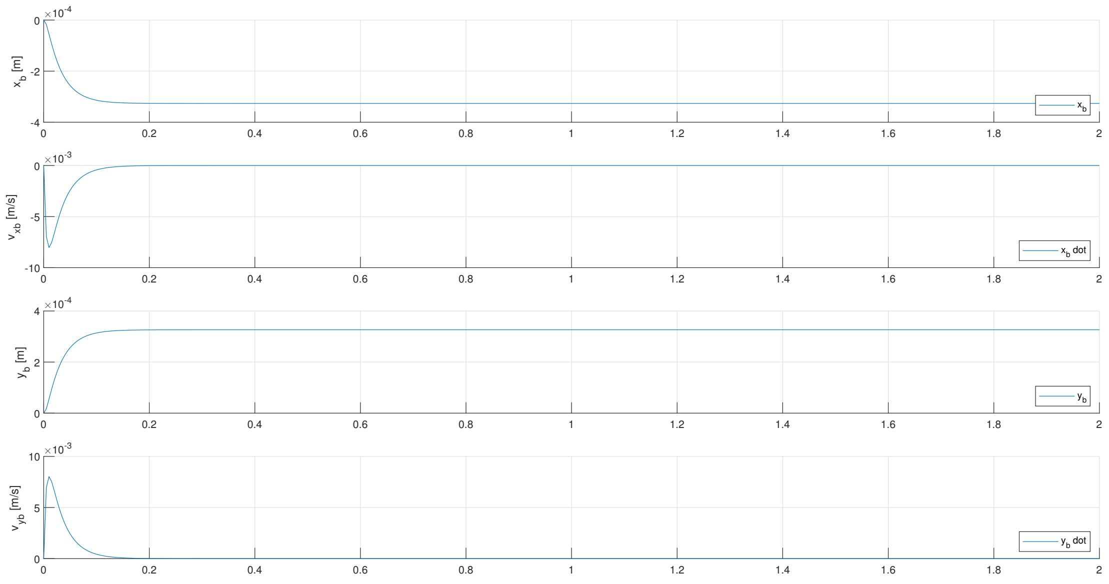


(a) State vector trajectory with $N = 50$

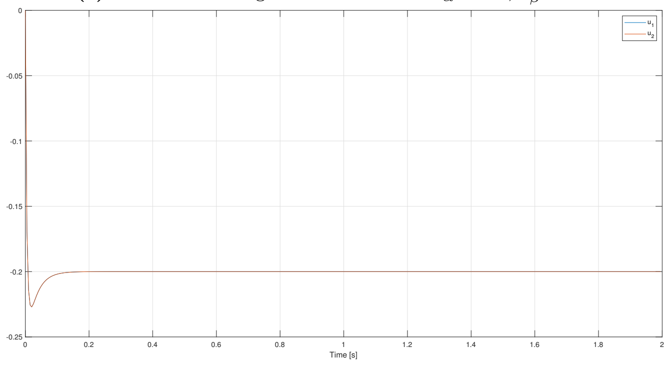


(b) State vector trajectory with $N = 100$

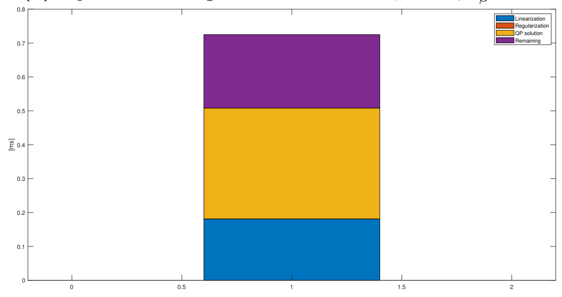
Figure 5.15: State vector trajectory according to different control steps ($N = 50$, $N = 100$)



(a) State with an angle calibration error $e_\alpha = 0.2$, $e_\beta = 0.2$



(b) Input with an angle calibration error $e_\alpha = 0.2$, $e_\beta = 0.2$



(c) Computation time

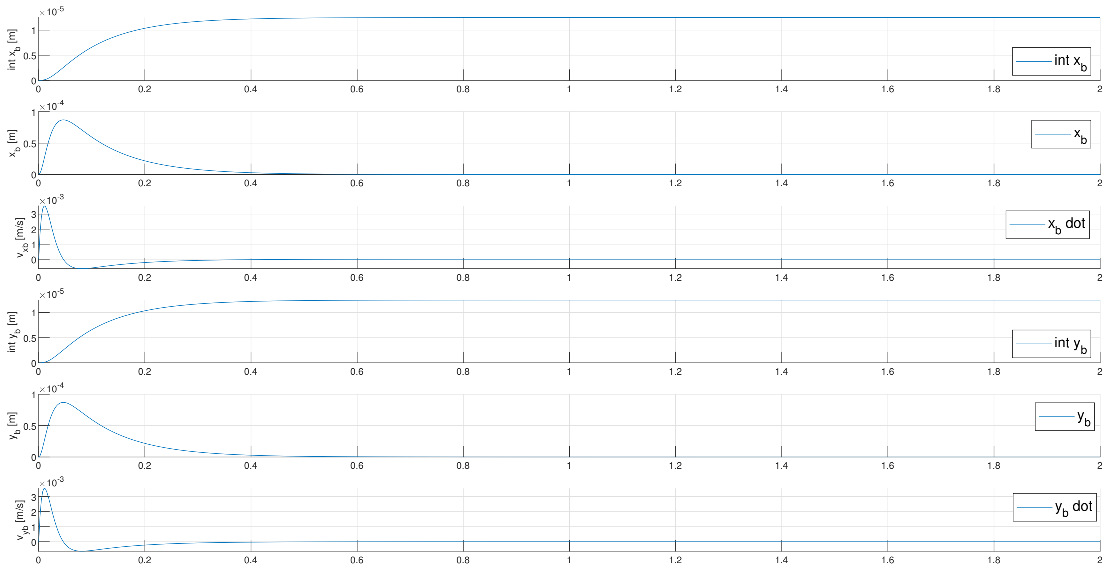
Figure 5.16: System behaviour without any compensation of the angle calibration error

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{5}{7}g & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{5}{7}g \end{bmatrix} \cdot u \tag{5.7}$$

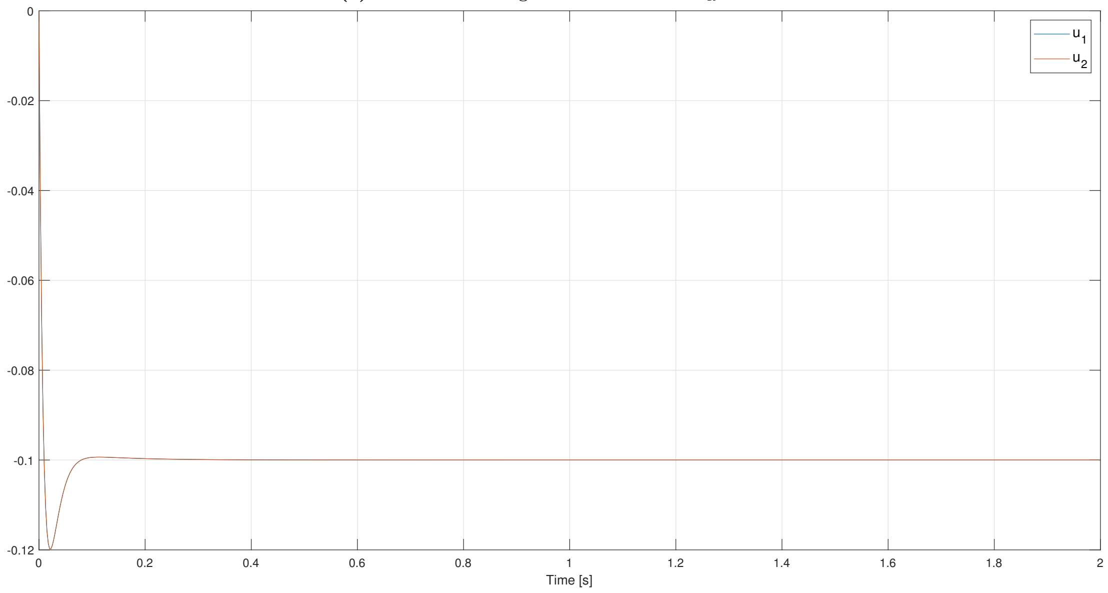
That way, the integral is weighted as well. The crucial point is that now the input relies not only on the current state, but also on the past history of the system. Fig. 5.17 illustrate the result: the position offset is now zero. Supposing to access the angles accelerations, the weighting matrices are now $W_x = diag(10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3}, 10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3})$, $W_u = diag(10^{-5}, 10^{-5})$, while the setpoint is $x_{ref} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$. On the other hand, if the inputs are the two angles, the last values of the cost functions disappear.
So far the integral weight has been used only to carry the ball to the origin starting from an angle calibration error. If this error is combined with a position setpoint different from the origin, the setpoint of the integral component is to be changed as well. The position and its integral must indeed have consistent references, as they are not indipendent. In particular, $x_1^* = \int x_2^* dt$. Therefore, if the position reference is a non-zero constant number, the integral reference will be a ramp with slope $\Delta = x_2^*$, i.e., $x_1^* = x_2^* \cdot t$. To show the behaviour of the system, the setpoint position is changed to $x_b^* = 0.05$, $y_b^* = 0.03$, thus $x_1^* = 0.05 \cdot t$, $x_6^* = 0.03 \cdot t$. Fig. 5.18 illustrates the trajectory of the state components with weights $Q = diag(10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3}, 10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3})$, $R = diag(10^{-5}, 10^{-5})$. The integral trends reach respectively $0.05 \cdot 2 = 0.1$, $0.03 \cdot 2 = 0.06$ as final values.

5.6.1 Choice of the Proper Weight

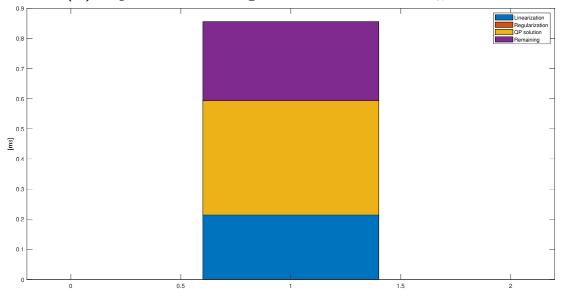
As already mentioned within the previous section, fig. 5.17 presents the weights $Q = diag(10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3}, 10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3})$, $R = diag(10^{-5}, 10^{-5})$. These however give rise to excessive overshoots and undershoots within the position, which means that the position and the velocity weights should be increased. Therefore, an adjustment of the weights is necessary. A possibility is to use the values $Q = diag(10^4, 10^3, 10^0, 10^{-2}, 10^{-3}, 10^4, 10^3, 10^0, 10^{-2}, 10^{-3})$, $R = diag(10^{-5}, 10^{-5})$, which allow to achieve a better behaviour. However, the behaviour provided by the latter matrices is excessively overdumped. This is



(a) State with an angle calibration error $e_\alpha = 0.2$



(b) Input with an angle calibration error $e_\alpha = 0.2$



(c) Computation time

Figure 5.17: System behaviour with the inclusion of the integral $\int x_b$, $\int y_b$ in presence of angle calibration error

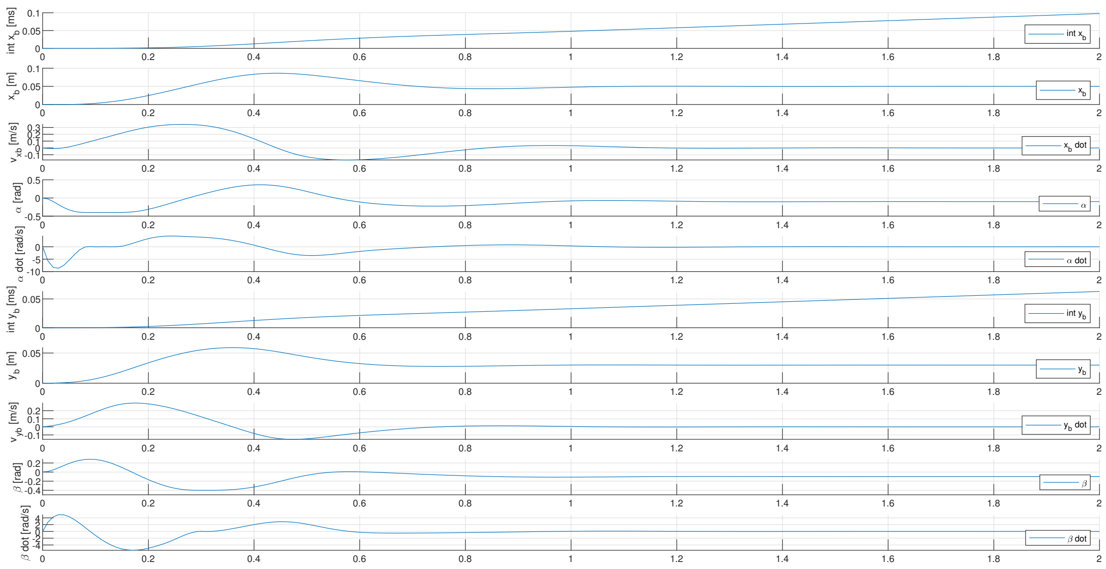


Figure 5.18: state vector trajectory with weights $Q = diag(10^4, 10^2, 10^{-1}, 10^{-2}, 10^{-3}),\ R = diag(10^{-5}, 10^{-5})$

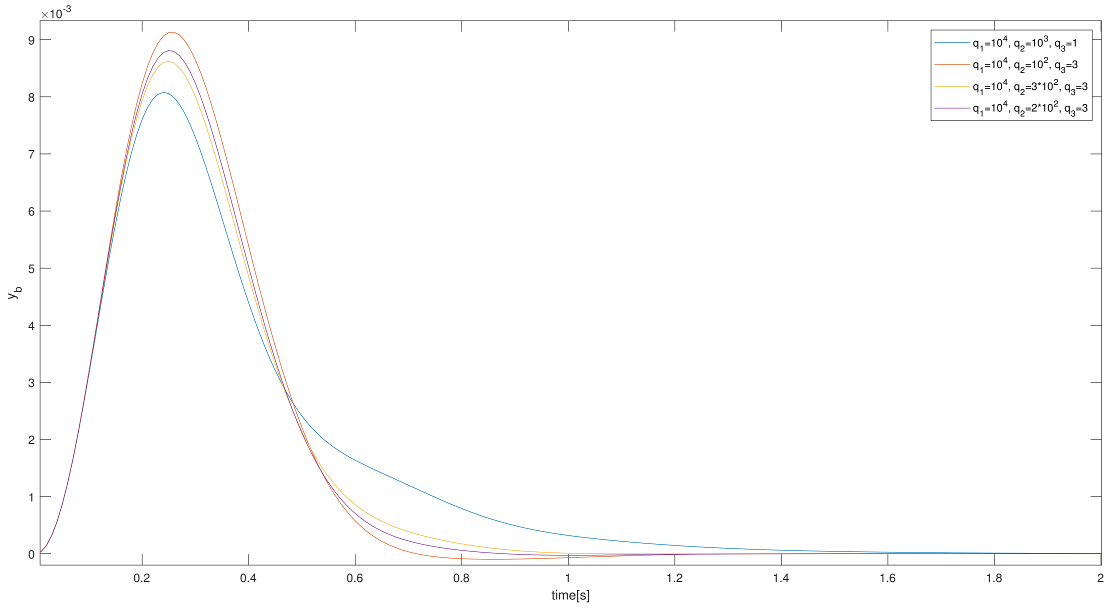


Figure 5.19: Position of the ball y_b according to the weights matrices. The initial and the final positions are the origin. The angle calibration error initially displaces the ball from the equilibrium point

shown in fig. 5.19: the ball weights $10^4, 10^3, 10^0$ allow to have a smaller movement of the ball, however the recovery time is higher. On the other hand, the weights $10^4, 10^2, 3$ and $10^4, 2 \cdot 10^2, 3$ lead to a slightly underdamped behaviour. The perfect choice thus seems to be $10^4, 3 \cdot 10^2, 3$, with a critically damped trajectory.

5.7 Inclusion of the Kalman Filter

As already presented in chapters 2 and 3, model predictive control needs the knowledge of the full state vector. So far, the assumption of directly being able to access the state vector from the simulation model has been made. However, to reach a more realistic simulation, the only accessible state components are now the ball positions in the $x-$, $y-$ axes. The remaining state components are estimated through a Kalman filter, whose structure has been presented within section **??**. In addition to the inaccessible state components, this filter is implemented to estimate the modeled disturbances, i.e., the angles calibration errors: the state vector is thus augmented to include the disturbances as well. The Kalman filter relies on the following discrete model of the system:

$$\begin{gathered}\xi = \begin{bmatrix} x & d_\alpha & d_\beta \end{bmatrix}^T = \begin{bmatrix} x_b & \dot{x}_b & \alpha & \dot{\alpha} & y_b & \dot{y}_b & \beta & \dot{\beta} & d_\alpha & d_\beta \end{bmatrix}^T u = \begin{bmatrix} \ddot{\alpha} & \ddot{\beta} \end{bmatrix}^T \\ \xi(k+1) = \begin{bmatrix} A_d & A_d(:;3,7) \\ 0_{n_x \times n_x} & I \end{bmatrix} \xi(k) + \begin{bmatrix} B_d \\ 0_{n_p \times n_u} \end{bmatrix} u(k), \quad y(k) = \begin{bmatrix} x_b & y_b \end{bmatrix}^T \end{gathered} \tag{5.8}$$

The notation $A_d(:;3,7)$ means that the third, seventh columns of the matrix A_d are selected. Kalman filter aims thus at estimating both the inaccessible state components and the disturbances, which are then fed to the ocp model.
Figures 5.20, 5.21, 5.22 illustrate the state and input trajectories, as well as the required computational time. These are achieved by setting the following matrices:

$$\begin{aligned} W_x &= \text{diag}(10^5,\ 10^4,\ 10^{-2},\ 10^{-3},\ 10^5,\ 10^4,\ 10^{-2},\ 10^{-3}), \quad W_u = \text{diag}(10^{-4},\ 10^{-4}), \\ P(0) &= \text{diag}(10^{-10},\ 10^{-5},\ 10^{-10},\ 10^{-10},\ \ 10^{-10},\ 10^{-5},\ 10^{-10},\ 10^{-10},\ \ 10^{-10},\ 10^{-10}), \\ Q &= \text{diag}(10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ \ 10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ \ 10^{-7},\ 10^{-7}), \\ R &= \text{diag}(10^{-7},\ 10^{-7}) \end{aligned} \tag{5.9}$$

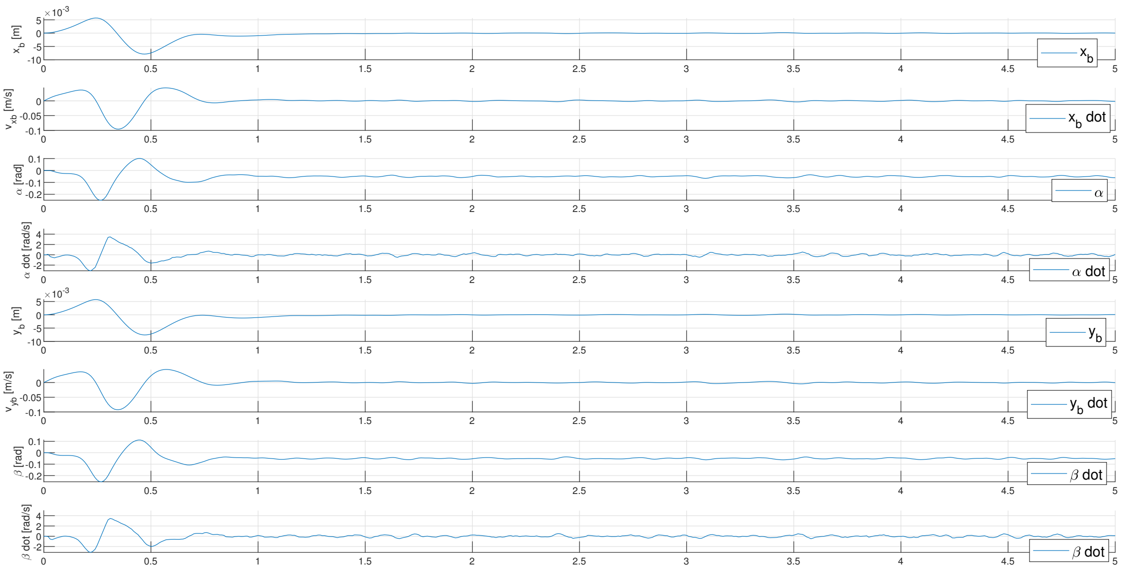


Figure 5.20: State trajectory with the use of Kalman filter to estimate the full state vector and the modeled disturbances

The bigger weights for the velocities within $P(0)$ are due to the initial uncertainty of them, which are generically set to zero. On the other hand, the modeled disturbances require a faster estimation with respect to the other augmented state components, hence the disturbances weights inside Q are bigger than the other components. The next subsection will focuses on a proper tuning of the Kalman filter matrices Q and R with the cumulative periodogram method, which has been explained within subsection 3.1.4.

5.7.1 Tuning of the Kalman filter

As presented in subsection 3.1.4, the tuning parameters of the Kalman filter are the two covariance matrices Q, R (or better, their ratio). The correctness of the tuning may be analyzed through the cumulative periodogram of the innovation $e(t) = y(t) - \hat{y}(t)$. Therefore, the trajectory of cumsum|fft($e(k)$)| is to be a straight line between the frequencies 0 and the Nyquist frequency Ω_N. The main limit of tuning the Kalman filter through simulation regards the introduction of disturbances. These are indeed modeled directly by the user, while an experimental setup would introduce these disturbances by itself without the need to model them. Therefore, a more useful tuning procedure will be carried out with the use of the platform.

The simulation provides an open loop test where the simulation model returns the measure of the ball position and the other state variables are estimated. A

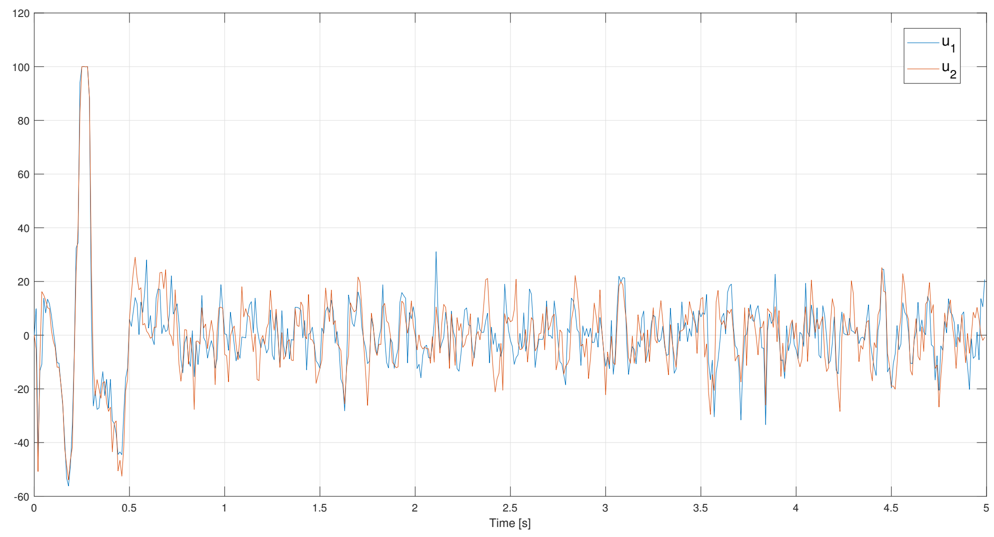


Figure 5.21: Input trajectory with the use of Kalman filter to estimate the full state vector and the modeled disturbances

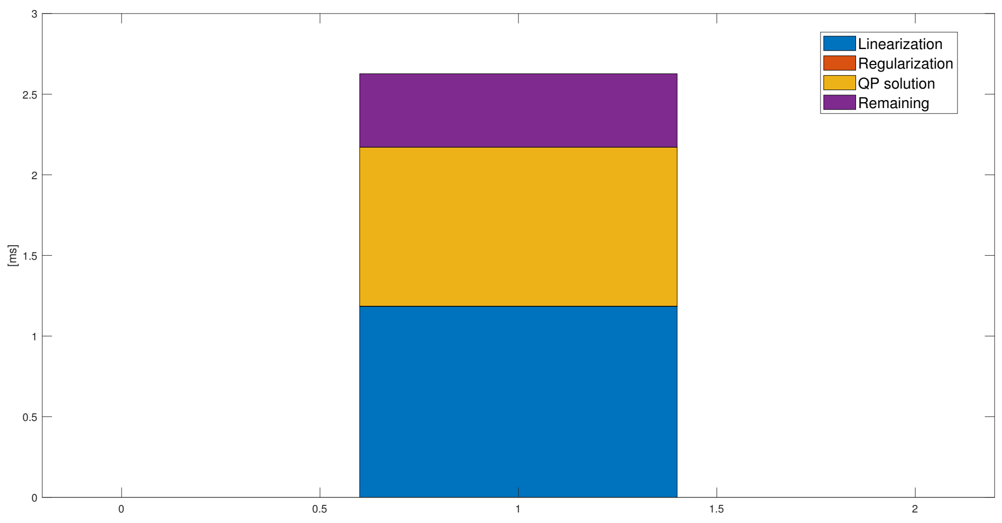
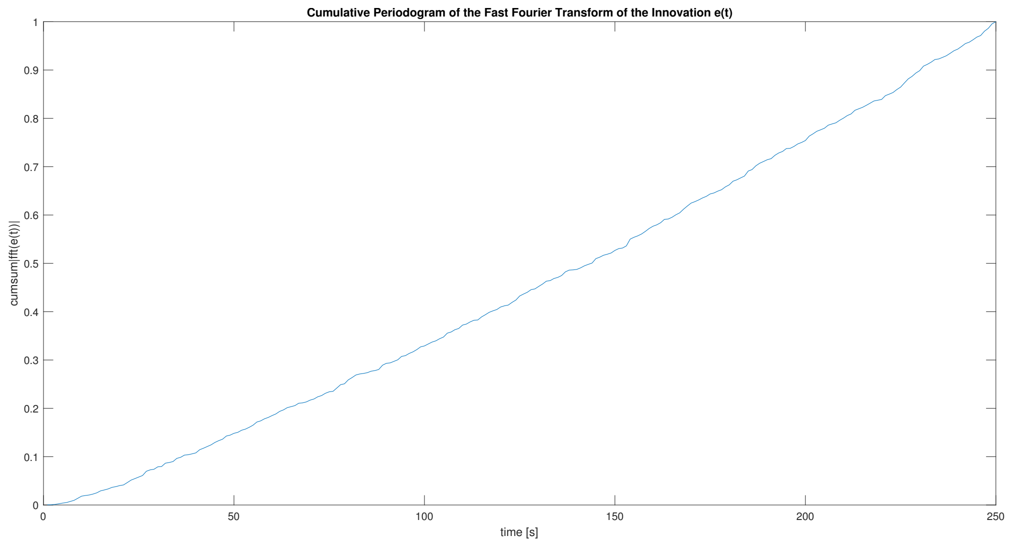


Figure 5.22: Computational time with the use of Kalman filter to estimate the full state vector and the modeled disturbances

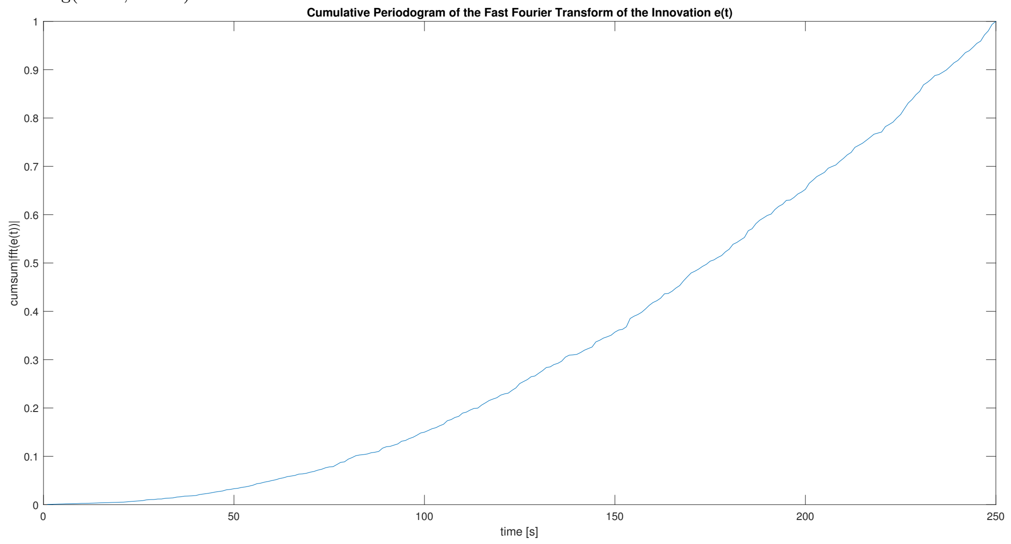
Gaussian output disturbance $N(0, 10^{-7})$ is considered. As it can be seen from figures 5.23a, 5.23b, the augmented state covariance matrix Q is to have smaller weights than the output matrix R, especially for the components of the vector x, due to the lack of state disturbances. The implemented Kalman filter is also able to correctly estimate the modeled disturbances, as shown in fig. 5.24 for the $x-$ axis disturbance.

5.8 Simulation Results: Summary

This section aims at summarizing the simulation results. First of all, an approximation of the model can be made in order to decrease the computational time and therefore to be applied to an embedded application. Albeit the real model provides smaller steady state errors, the computational time is indeed too big to an embedded application. Secondly, the proper constraints and control sample rate are chosen. The first is due to the approximation of the ocp model, which considers $sin(\theta) \approx \theta$. For this approximation to be valid, the angle θ is to be small. The control sample rate is on the other hand crucial to achieve a good approximation of the continuous model, without increasing the computational time too much. Indeed, increasing the sample rate leads to an exponential growth of the computation burden, while a too small value of it is not able to properly discretize the model. Moreover, by introducing angles calibration errors a steady state error is seen to appear in position. A rejection of it might be reached through the integral inclusion of the ball position within the state vector. An additional issue concerns the availability of the state components: the experimental setup only measures the position of the ball, while the MPC requires to have access to the full state vector, including the velocity of the ball and the angular components. The proposed solution is a Kalman filter, able to estimate both the unmeasured state components and the modeled disturbances. The next chapter will validate the latter system, i.e., the coexistence of the Kalman filter and Model Predictive Control, with the use of the real hardware.



(a) Cumulative periodogram obtained by setting $Q = \mathrm{diag}(10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-10},\ 10^{-7},\ 10^{-7})$, $R = \mathrm{diag}(10^{-7},\ 10^{-7})$



(b) Cumulative periodogram obtained by setting $Q = \mathrm{diag}(10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7},\ 10^{-7})$, $R = \mathrm{diag}(10^{-7},\ 10^{-7})$

Figure 5.23: Different cumulative periodograms according to different tunings for Q, R

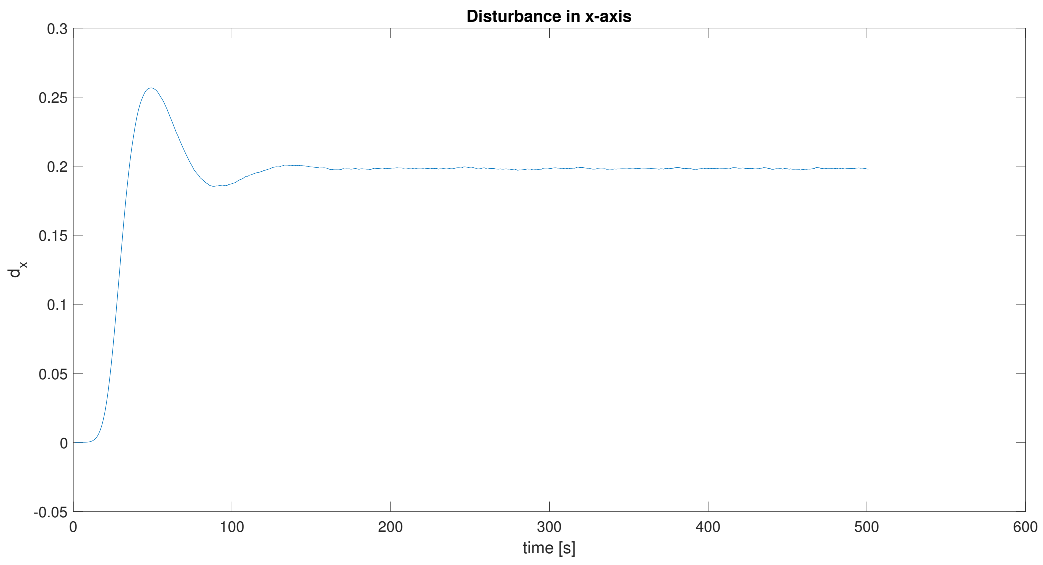


Figure 5.24: Estimated disturbance in $x-$ axis by KF

Chapter 6

Experimental Results

As explained in subsection 4.3.2, the laptop interfaces with the hardware through a Qt program. Since the MPC provides the optimal angular accelerations while the hardware requires the angles directly, equations 4.1 are implemented within the code.
At first, the chosen parameters take into account the simulation results, in particular the ones presented in section 5.7. However, the experimental results will be seen to differ from the simulation ones. Therefore, some adjustments will be deployed in order to enhance the behaviour. The chapter is organized as follows: the first section will illustrate the trajectories with the same parameters as the optimal ones reached in simulation. The next ones will try to enhance the trends: first, a change for the angular weights of the cost function is implemented; afterwards, a compensation of the modeled disturbances is taken into consideration to decrease the steady state error. Finally, a blending between the angular weights is implemented to trade off between speed of convergence and steady state oscillation. Some conclusions will be presented at the end of the chapter.

6.1 First Attempt: Same Parameters as in Simulation

As one might expect, the first attempt is carried out tuning the MPC controller according to the best behaviour achieved in simulation. The only difference regards the angle constraints, which are set at $\pm 0.14 rad$ to not make the plant unstable. On the other hand, the Kalman filter is now working in a real environ-

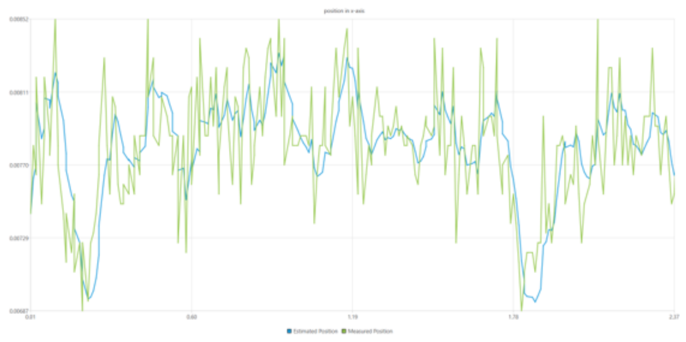


Figure 6.1: Trajectory of measured (green) and estimated (blue) position of the ball in the $x-$ axis

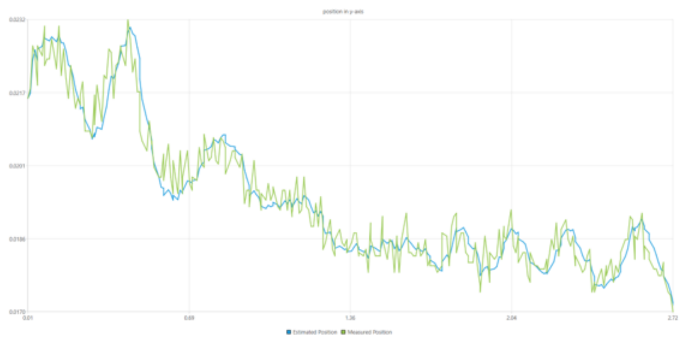


Figure 6.2: Trajectory of measured (green) and estimated (blue) position of the ball in the $y-$ axis

ment, with real disturbances inherent of the touch screen panel, cables and the mismatch between the nominal and the real model. Therefore, its tuning happens as explained in subsection 3.1.4, i.e., through the controllability Gramian. Furthermore, as the angles are calculated from the angular accelerations provided by the MPC, the KF model differs from the one within simulation, in the sense that it estimates both the position and velocity of the ball from the calculated angles. The controllability Gramian is as follows:

$$W_{0\to T} = \int_0^T e^{A\sigma}BB^Te^{A^T\sigma}d\sigma = \begin{bmatrix} \frac{\alpha^2}{3}T^3 & \frac{\alpha^2}{2}T^2 & 0 & 0 \\ \frac{\alpha^2}{2}T^2\alpha^2T & 0 & 0 & \\ 0 & 0 & \frac{\alpha^2}{3}T^3 & \frac{\alpha^2}{2}T^2 \\ 0 & 0 & \frac{\alpha^2}{2}T^2 & \alpha^2T \end{bmatrix} \tag{6.1}$$

The tuning of the Kalman filter is done in open loop by keeping $Q = W_{0\to T}$ and by changing R. The most proper output covariance matrix is $R = 10^{-10}I$, as the cumulative periodograms of the innovation in images 6.1, 6.2 illustrate.

The weights matrices of the MPC cost function are
$Q = diag(10^4,\ 10^3,\ 10^{-2},\ 10^{-3},\ 10^4,\ 10^3,\ 10^{-2},\ 10^{-3})$, $R = diag(10^{-5},\ 10^{-5})$. Although these matrices lead to the best behaviour in simulation, the experimen-

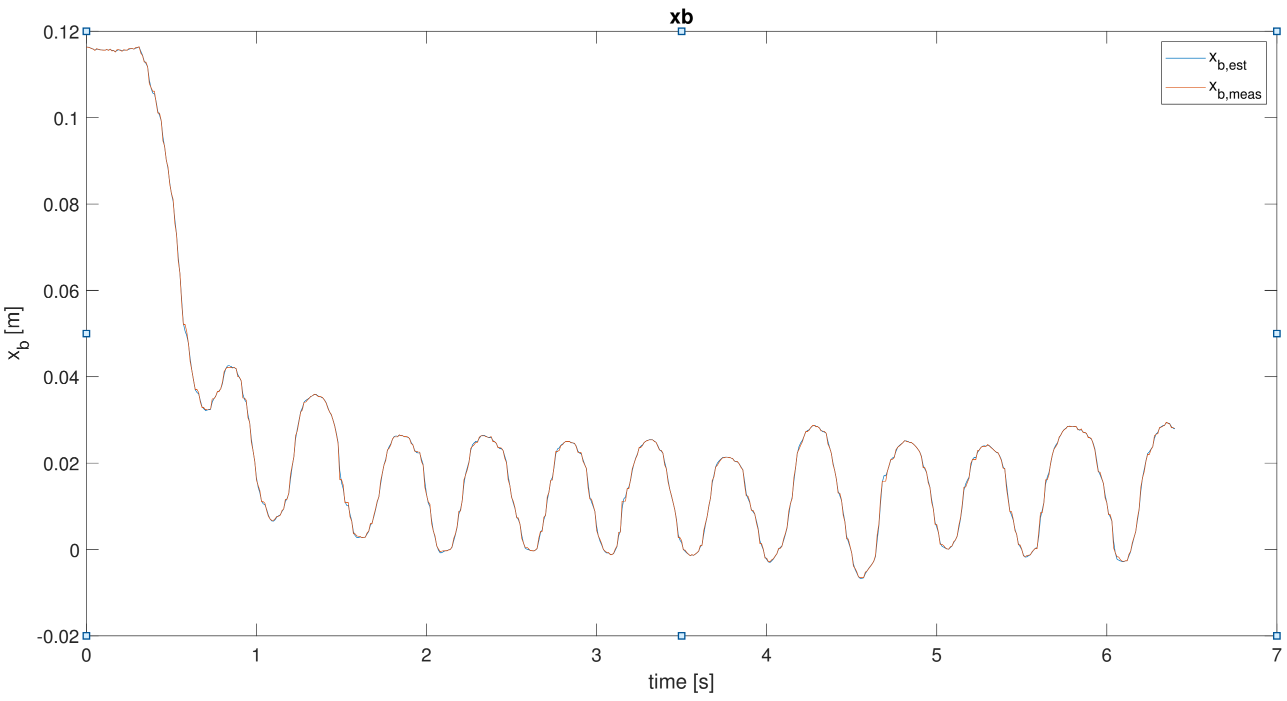


Figure 6.3: Trajectory of the position of the ball in the $x-$ axis with the weights matrices from simulation

tal results are inacceptable, as shown in figures 6.3, 6.4, 6.5, 6.6 Indeed, the steady state trajectory of the ball is oscillating and not even around the origin. The communication delay is the main reason of the oscillating behaviour, since the inputs are related to previous state values. On the other hand, the offset is due to the lack of measurement of the angular calibration errors e_α, e_β. The next sections will try to correct these undesired features.

6.2 Increase of the Angles Weights

The first effort is towards the elimination of the steady state oscillations. Given that the delay cannot be decreased unless a change of the set comunication is provided, the dynamics of the system can be lowered through modifying the weighting matrices, and in particular the weights concerning the angles α, β. Therefore, they are set at $q_\alpha = q_\beta = 5 \cdot 10^4$, which is a value bigger than the velocity weights and smaller than the position ones. The strategy is to have smaller inclinations of the plate so that the ball moves with lower acceleration.

As shown through figures 6.7, 6.8, 6.9, 6.10, the oscillations at steady state are disappeared. The angles indeed do not oscillate between the upper and the lower bound anymore. Nevertheless, the settling time increases, because of the

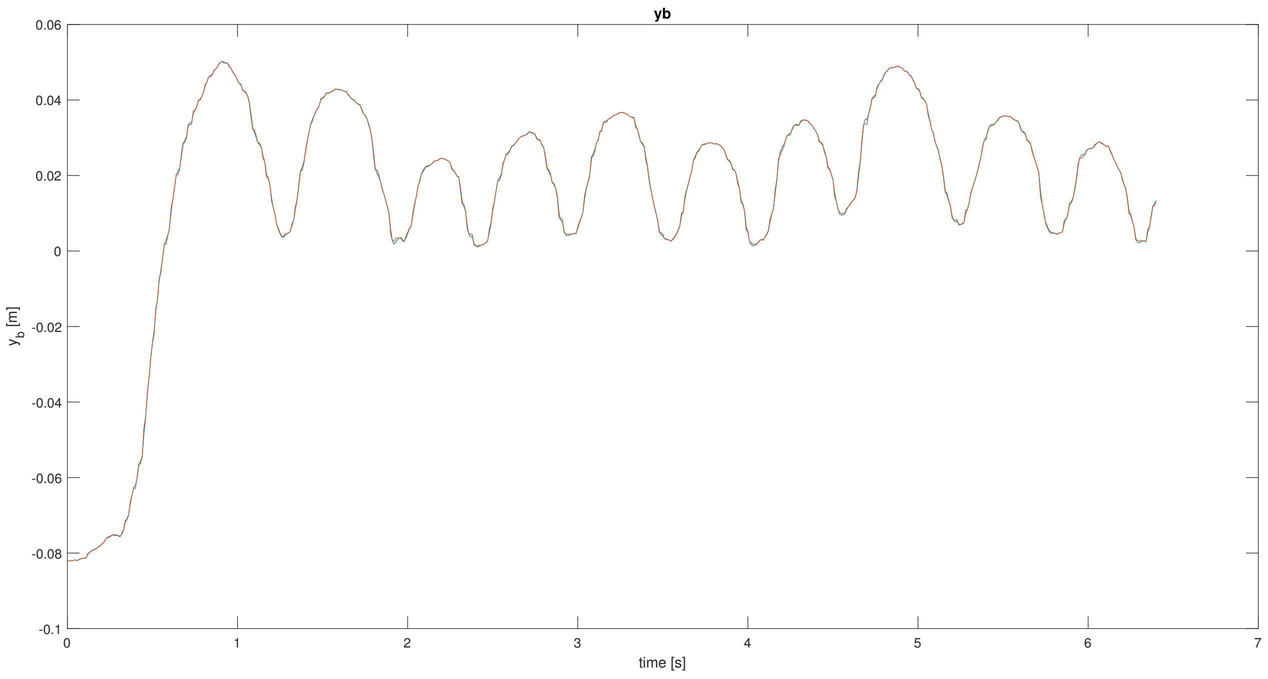


Figure 6.4: Trajectory of the position of the ball in the $y-$ axis with the weights matrices from simulation

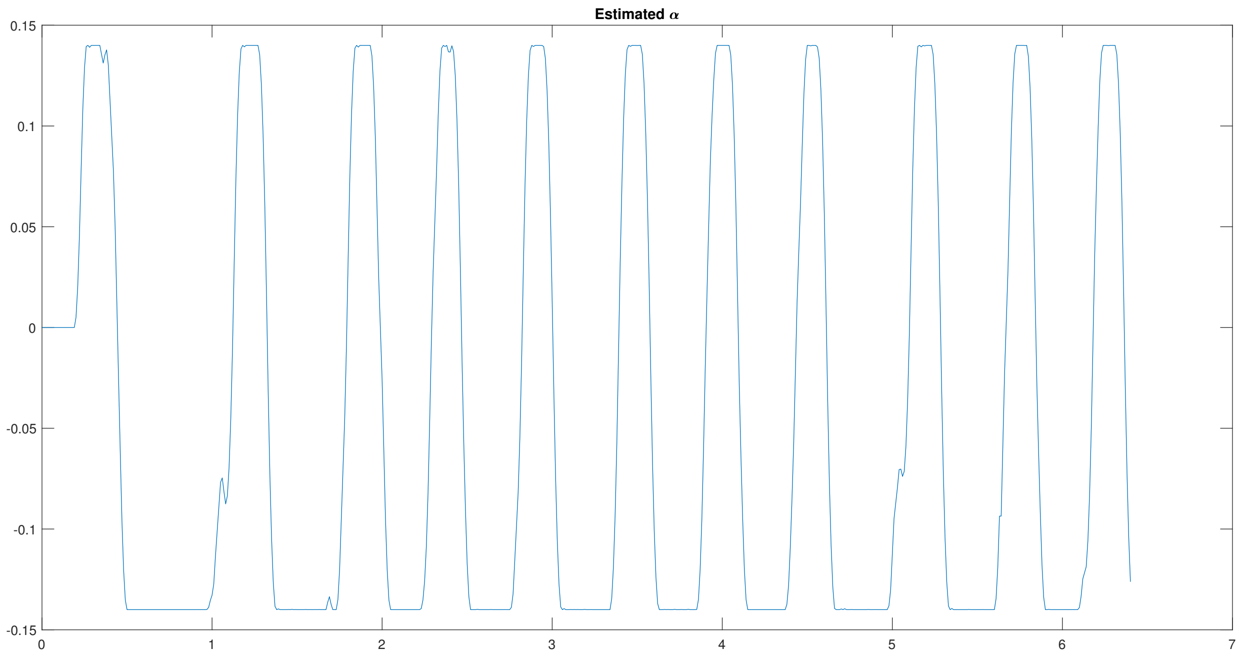


Figure 6.5: Trajectory of the angle of the plate in the $x-$ axis with the weights matrices from simulation

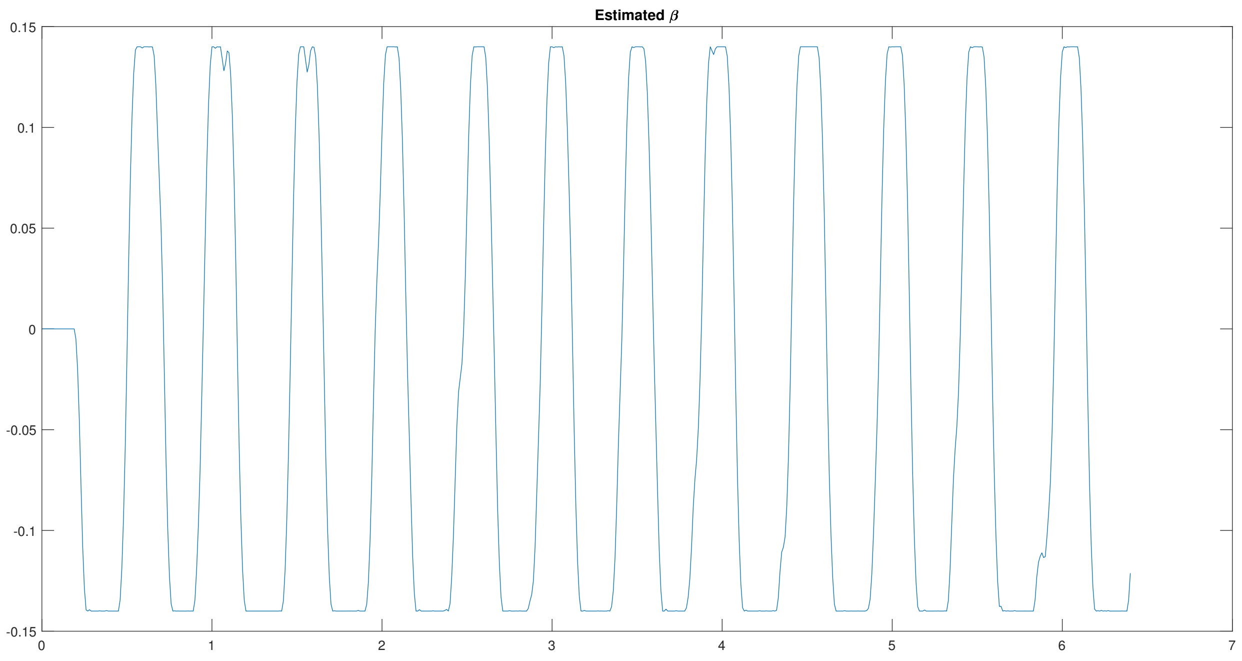


Figure 6.6: Trajectory of the angle of the plate in the $y-$ axis with the weights matrices from simulation

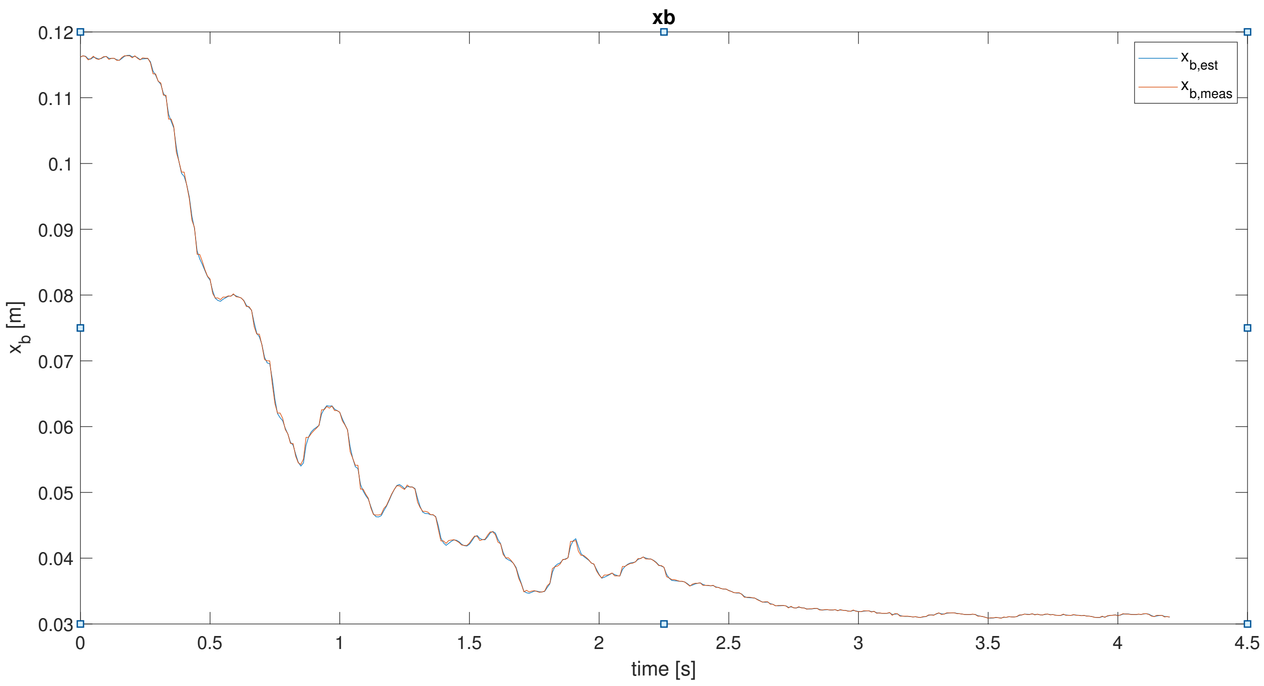


Figure 6.7: Trajectory of the position of the ball in the $x-$ axis with the angular weights increased to $q_\alpha = q_\beta = 5 \cdot 10^4$

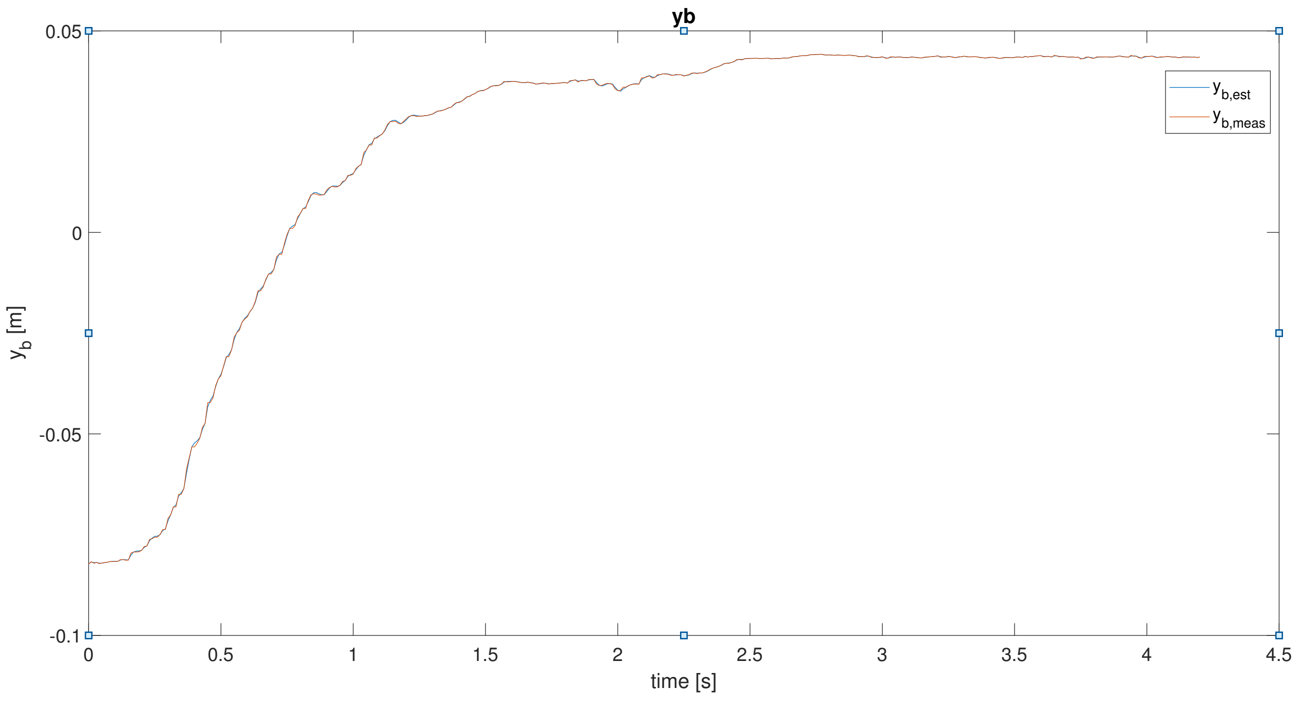


Figure 6.8: Trajectory of the position of the ball in the $y-$ axis with the angular weights increased to $q_\alpha = q_\beta = 5 \cdot 10^4$

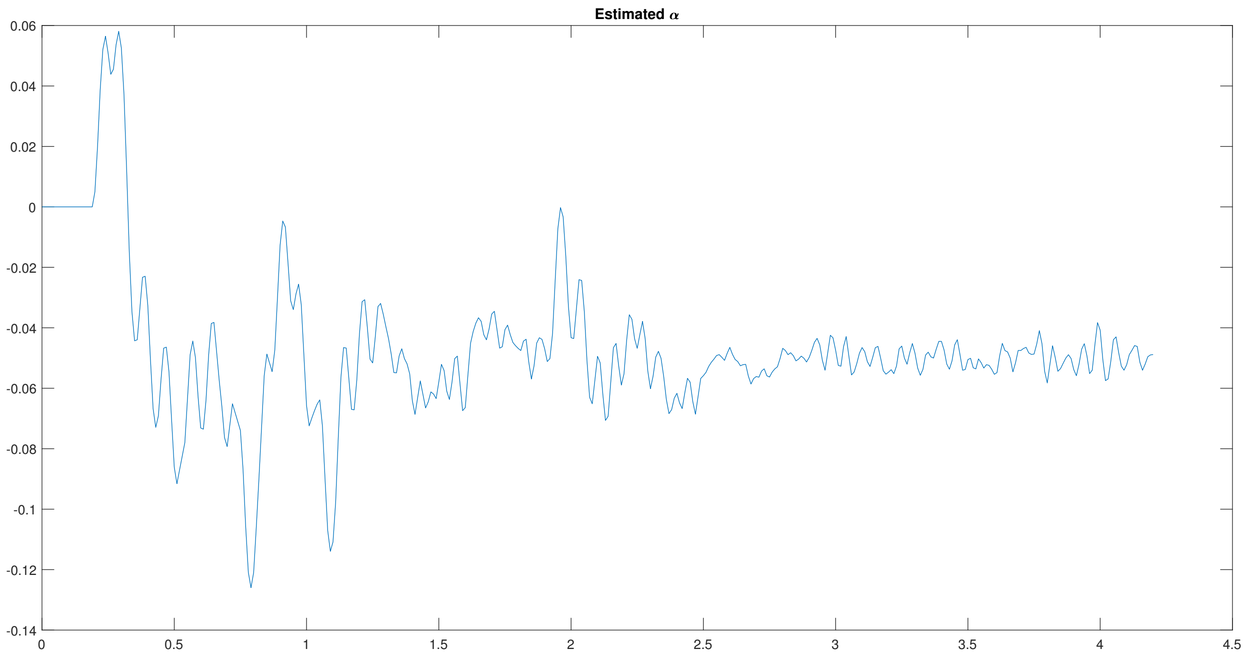


Figure 6.9: Trajectory of the angle of the plate in the $x-$ axis with the angular weights increased to $q_\alpha = q_\beta = 5 \cdot 10^4$

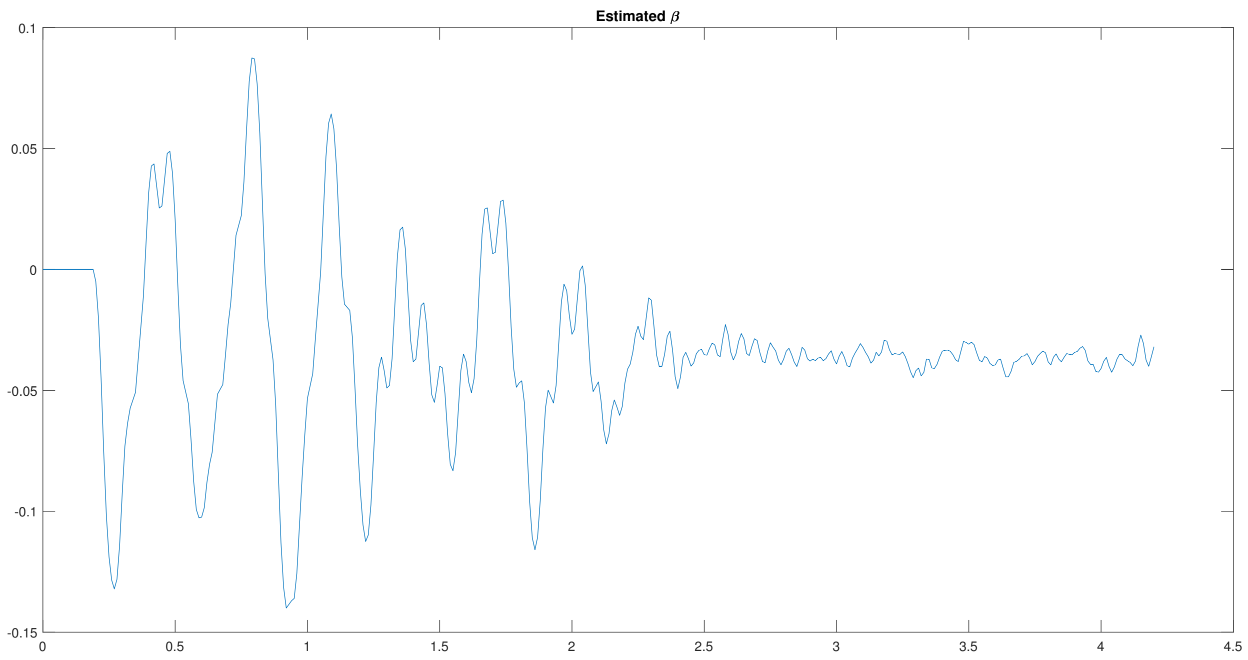


Figure 6.10: Trajectory of the angle of the plate in the $y-$ axis with the angular weights increased to $q_\alpha = q_\beta = 5 \cdot 10^4$

decrease in the plate inclinations. From these considerations, a mediation between the absence of oscillations and a proper settling time is to be found. This will be tackled within subsection

6.3 Compensation of the Modeled Disturbances

The second issue regards the steady state offset, which can be better analyzed withouth steady state oscillations, therefore by setting $q_\alpha = q_\beta = 5 \cdot 10^4$. The offset is due to the lack of the estimation of the disturbances: hence, an estimation procedure is supposed to enhance the performance. The procedure is divided in two steps:

- The constant part of this disturbance is estimated through a calibration procedure, since it corresponds to the opposite of the related steady state angle. Therefore, an average of the steady state angles is done and the opposite is set as the estimated constant disturbance $\bar{d}$.
- The variable part of the disturbance is estimated through the Kalman filter which relies on an augmented state vector, as explained within section 5.7.

Fig. 6.11, 6.12 show the improvements. Although the steady state error is not zero, the values are much smaller than the previous ones. The non-zero values

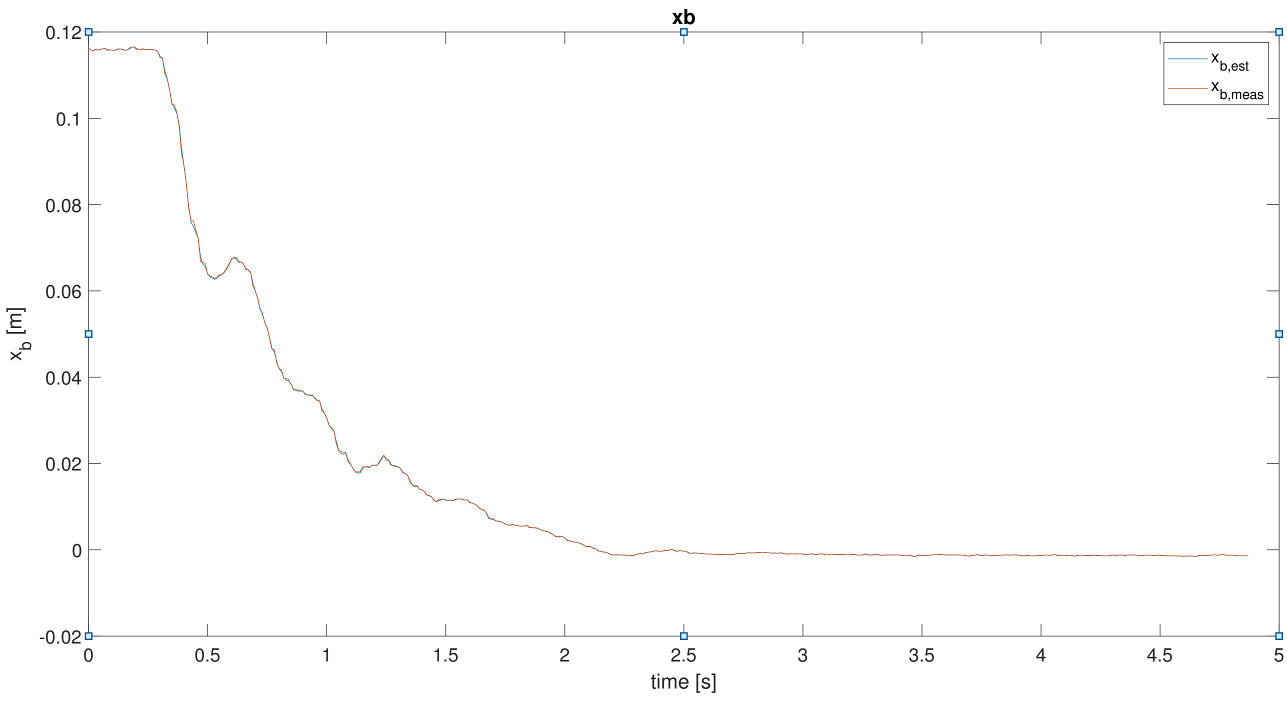


Figure 6.11: Trajectory of the position of the ball in the $x-$ axis with the angular weights increased to $q_\alpha = q_\beta = 5 \cdot 10^4$ with compensation of modeled angular disturbances

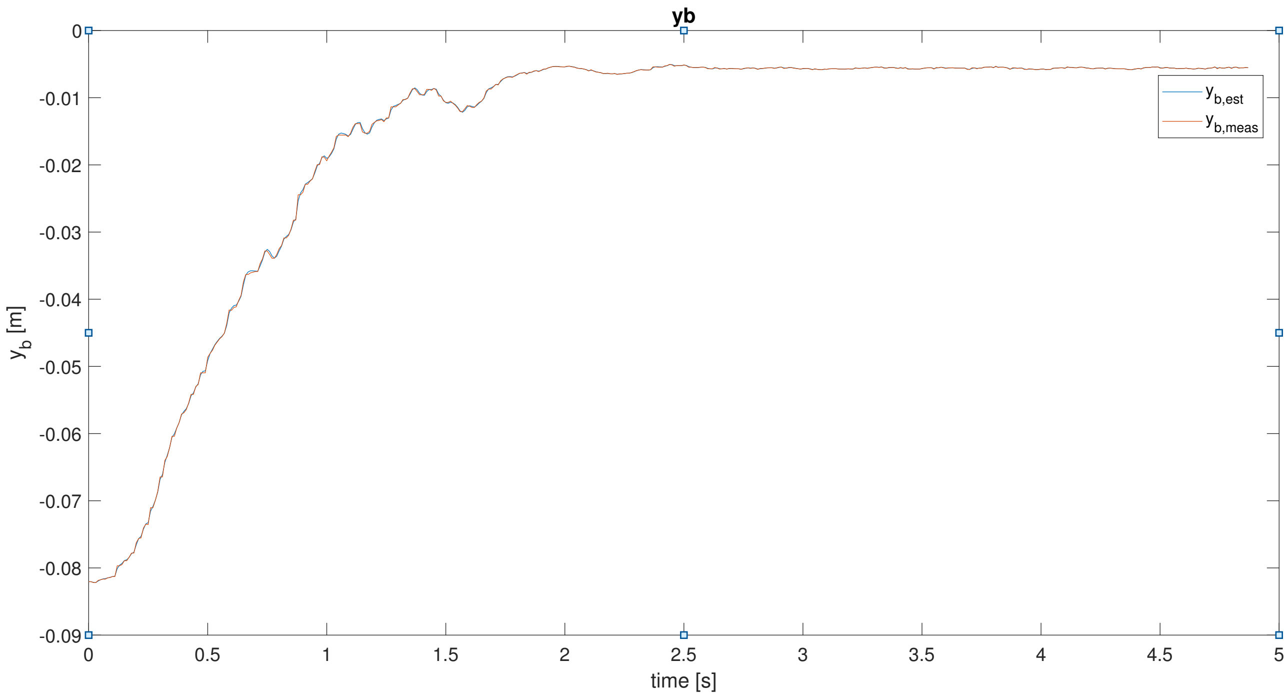


Figure 6.12: Trajectory of the position of the ball in the $x-$ axis with the angular weights increased to $q_\alpha = q_\beta = 5 \cdot 10^4$ with compensation of modeled angular disturbances

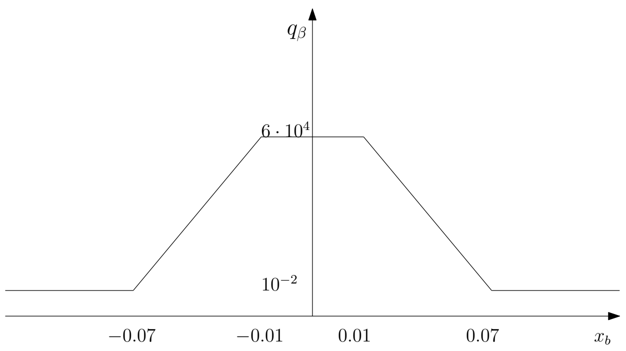


Figure 6.13: Trajectory of the angular weight q_β in relation to the value of the ball position in the $x-$ axis x_b

can be due to several factors, such as the mismatch between the nominal model and the real hardware, due e.g. to the unconsidered friction of the plate.

6.4 Blending between the two Angular Weights

Throughout this chapter a solution to the steady state oscillation and how to decrease the steady state offset have been presented. However, a trade off between the velocity of convergence and the steady state oscillation is to be found, since by just increasing the angular weights the settling time deeply increases as well. The proposed solution takes into account a weighting matrix Q variable with the position of the ball with the following characteristics:

- With $|x_b| \geq 0.07m$, $q_\beta = 10^{-2}$. Similarly, with $|y_b| \geq 0.07m$, $q_\alpha = 10^{-2}$.
- With $|x_b| < 0.01m$, $q_\beta = 5 \cdot 10^4$. Similarly, with $|y_b| < 0.01m$, $q_\alpha = 5 \cdot 10^4$.
- Between $|x_b| = 0.01m$ and $|x_b| = 0.07m$, a blending process is employed: the weight q_β follows a linear trend with angular coefficient $m = \frac{q_{max}-q_{min}}{x_{b,min}-x_{b,max}}$, where $q_{max} = 5 \cdot 10^4$, $q_{min} = 10^{-2}$, $x_{b,max} = 0.07m$, $x_{b,min} = 0.01m$. The offset of the interpolating straight line is equal to $q_{max} - m \cdot x_{b,min}$. The blending procedure is exactly the same for y_b, q_α.

The relationship between the position of the ball and the weight of the angle is visualized within fig. 6.13. Figures 6.14, 6.15, 6.16, 6.17 illustrate the final achieved behaviour: the convergence of the ball towards the origin is much faster than setting high angles weights from the beginning, however the oscillations are not present since high angular weights are considered when the ball is close to the middle of the plate. Hence, the blending procedure is able to eliminate both the disadvantages related to low and high angle weights, due to the fact

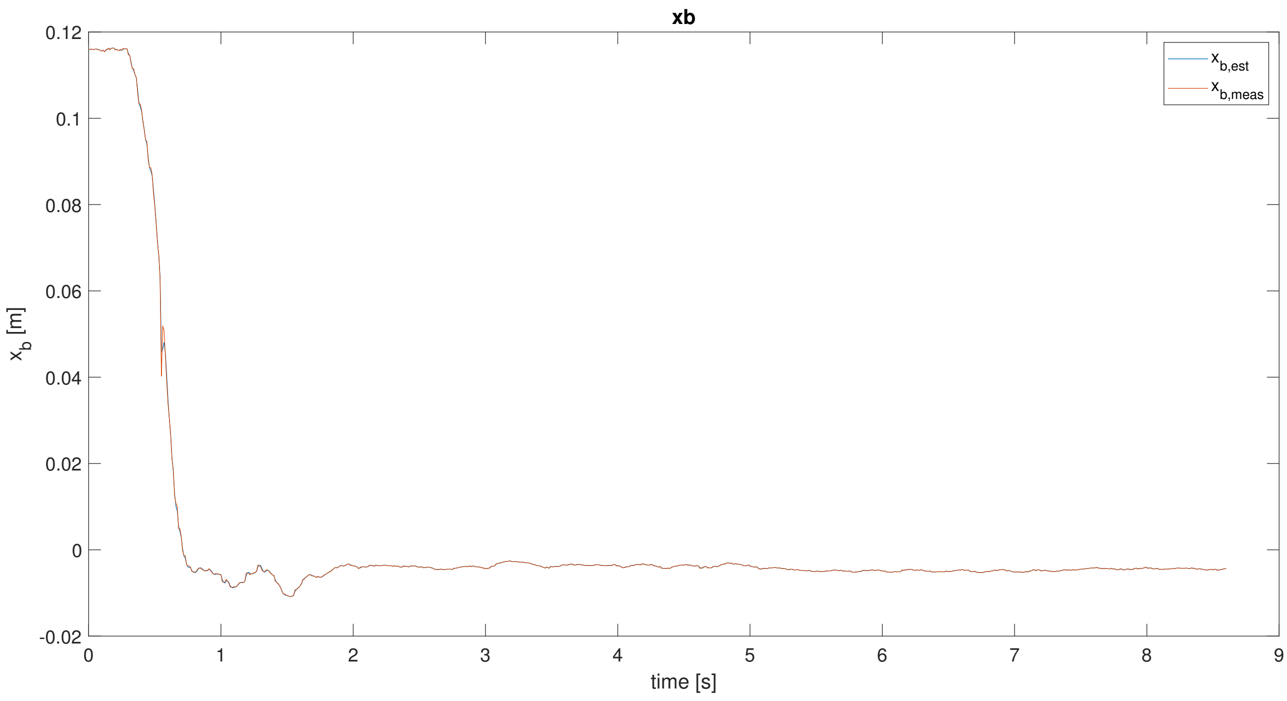


Figure 6.14: Trajectory of the position of the ball in the $x-$ axis with the blending procedure and the disturbance compensation

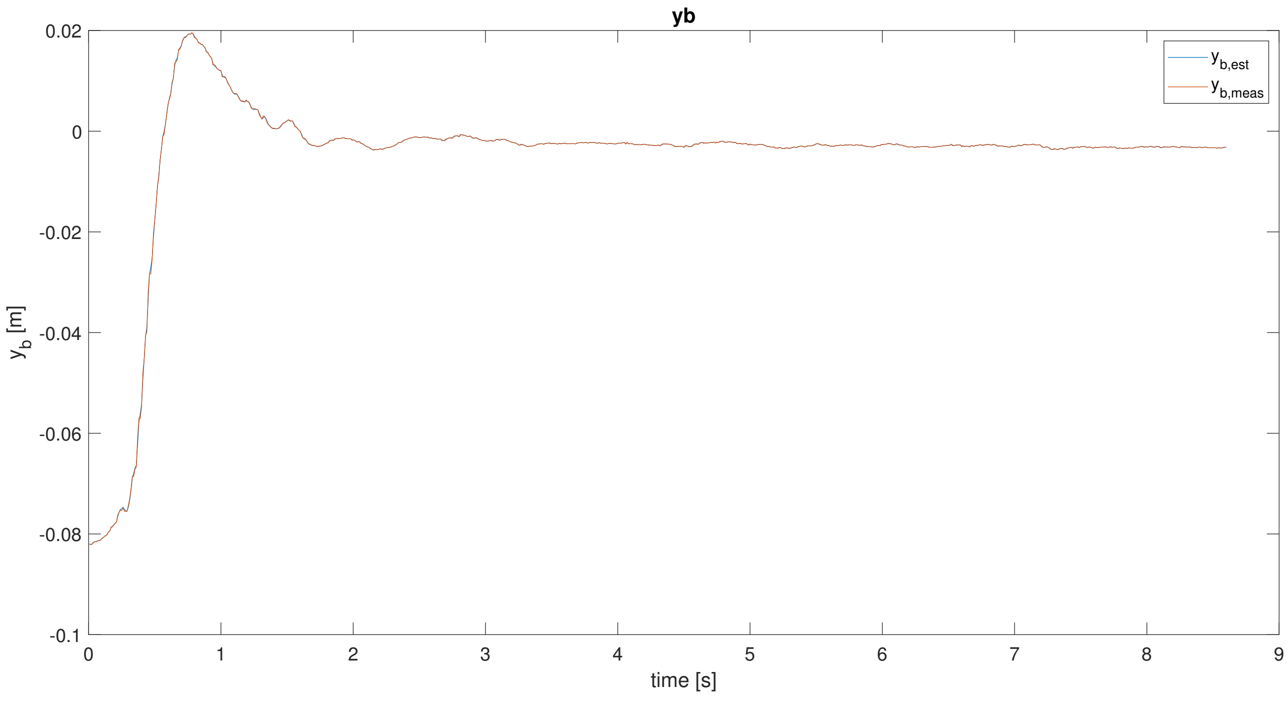


Figure 6.15: Trajectory of the position of the ball in the $y-$ axis with the blending procedure and the disturbance compensation

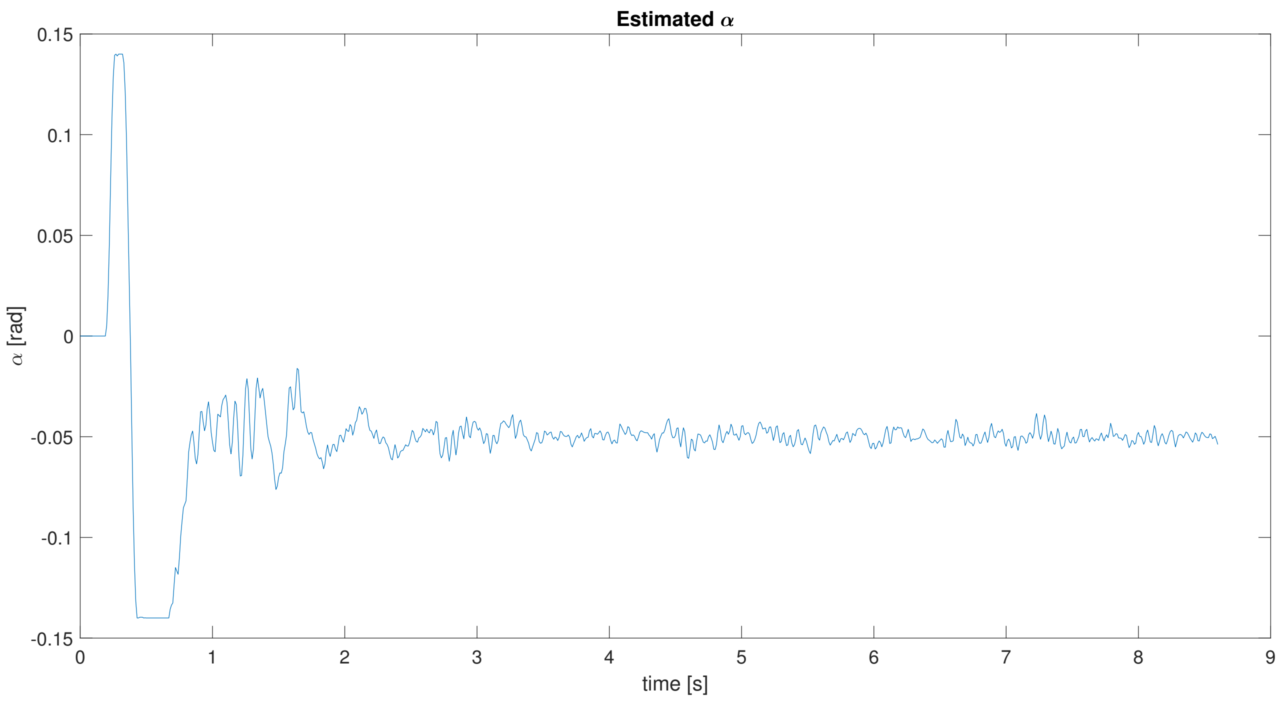


Figure 6.16: Trajectory of the angle of the plate in the $x-$ axis with the blending procedure and the disturbance compensation

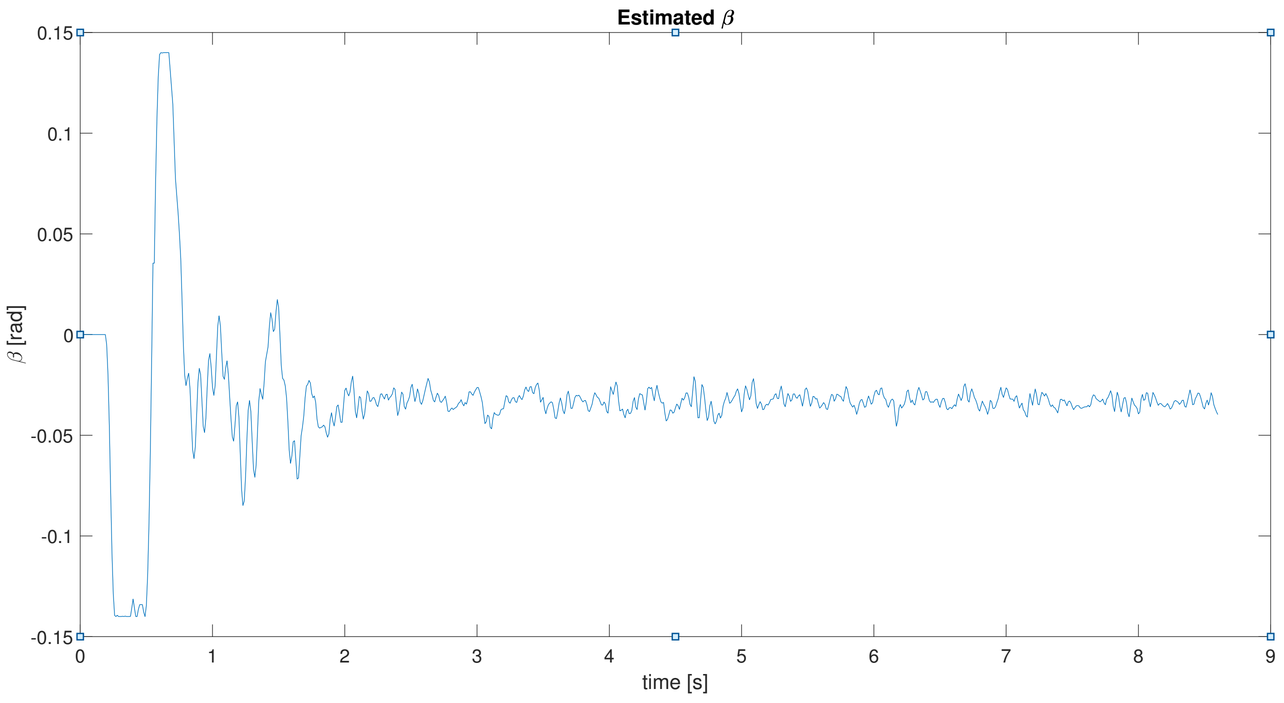


Figure 6.17: Trajectory of the angle of the plate in the $y-$ axis with the blending procedure and the disturbance compensation

that they appear in different areas of the plate: the convergence problem must be tackled with the ball far from the center, while the steady state oscillations appear when the ball is really close to it. A crucial point is here the reliance on the real-time iteration technique explained in subsection 2.4.3. Indeed, the online update of the Q matrix increases the computational burden and thus the required computational time. In order to achieve a fast enough update which does not make the controller infeasible, the real time iteration scheme to find the optimal input provided by the MPC is implemented.

6.5 Experimental Results: Summary

Throughout this chapter, different experimental results with the use of the real hardware have been faced. The introduction of the hardware in the loop leads to some non idealities which were unconsidered before, such as the communication delay. This is the main reason of the inadequacy of the simulation parameters for the cost function J and the constraints. Firstly, the angles constraints have been decrease in order to reach a stable behaviour of the plant. Subsequently, the problem of steady state oscillation in position has been tackled: its elimination is related to increased angular weights q_α, q_β for the cost function J, which nonetheless worsen the convergence behaviour. The solution corresponds to a position-varying weight matrix Q, where the weights q_α, q_β are set according to the ball position. When the ball is far from the middle of the plate the main concern is to bring it towards the center in a fast way, the weights are thus set at a small value. On the contrary, when the ball is close to the center, the elimination of oscillations is achieved through big values for q_α, q_β. With intermediate values for the position of the ball, q_α and q_β follow a linear interpolation between the upper and the lower values. The whole procedure is able to provide a proper trajectory, where the ball converges fast to the center of the plate without any steady state oscillation. A last problem is about the steady state error, due to the lack of compensation for the disturbances. This is faced through an estimate of the angular disturbances, which can be split into a constant part, easily estimated from the steady state angular values, and a variable one, which can be estimated through the Kalman filter.
The final controller is thus able to balance the ball in the middle of the plate with high dynamics and without any steady state oscillation. Moreover, the steady state offset is deeply decreased through the estimation and subsequent

compensation of the angular disturbances.

Conclusion and Future Works

The objective of this last chapter is to summarize the previous considerations about the implemented plant and the simulation and experimental results, as well as to focus on possible future works to enhance the control of the implemented hardware, i.e., the Stewart platform.

Model Predictive Control Applied to the Stewart Platform

The project aims first of all at exploring new control techniques applied within a field of increasing interest, i.e., manifacturing process. The automation of the latter sector is the current direction, since it allows to achieve higher efficiency and therefore save a huge amount of money. In this sense, robotic deviced are one of the major applications of interest since they allow to automate industrial processes. However, apart from the increasing automation, robots also provide some struggles: one of the largest concerns their frequent redundancy, which means that they have more degrees of freedom than the required ones. This is related to the possibility of executing a predefined path in different ways, which nonetheless are not always possible. The considered hardware, which is a Stewart platform, possesses indeed a higher number of DOFs with respect to the ones necessary to move the end-effector, the upper plate. The latter indeed only the rotation around two axes, x and y, to balance the ball, while the six servo-motors provide six degrees of freedom. Therefore, a grade of redundancy is present within the considered hardware. Control techniques which are able to deal with this are necessary. In this sense, Model Predictive Control is able to deal with constraints, which is a method to properly deal with the impossibility of executing certain paths, because of e.g. a fixed workspace, obstacles or delimitations. The following sections thus will face the results of the application of MPC to the Stewart platform, touching the simulation results, the need of an estimator and the final experimental results.

Simulation Results

After the modelization of the platform, a first attempt to apply Model Predictive Control to control the plant has been carried out in simulation. The goal is to verify the applicability of MPC and the consistency between the achieved models. The positive results allow to carry out further investigations, which nonetheless require a method to estimate the missing state components which are not provided directly by the plant. Indeed, Model Predictive Control requires to have access to the full state vector, while the application under consideration measures only the position of the ball. The main need is thus to estimate its velocity. The considered estimation technique is a Kalman filter, since it is able to deal with a noisy environment, which is the case of a real hardware. Moreover, the considered Kalman filter has also the possibility to estimate modeled disturbances, which in this case are within the considered angles. This is related to imperfections of the considered plate angles, arising from errors of the servo motors positions as well as light inclinations of the basement of the platform. Therefore, Kalman filter is a strong candidate to work in strict collaboration with MPC to estimate and subsequently control the plant. The simulation results have been once again positive, therefore the insertion of the real hardware in the loop has been carried out.

Experimental Results

The initial experimental attempt has not been so encouraging, since the use of the same weights as in simulation have given rise to steady state oscillations in position. This is due to several non-idealities which arise with the simulation, like the communication delay. Therefore, the dynamics of the plant has been decreased by considering higher penalties for the angles of the plate, so that the ball moves more slowly. By doing this, the steady state oscillations have disappeared, at the cost of a much slower convergence of the ball. The objective has become thus to find a proper tuning of the weights matrices to tackle both convergence speed and steady state oscillation. The proposed solution consists in angular weights q_α, q_β which vary with the position coordinates of the ball x_b, y_b. In particular, when the ball is far from the middle of the plate, the weights $q_{\alpha,\beta}$

are set at low values, hence the angles α, β are not highly penalized and the ball can quickly move towards the center. This aims at having a better convergence behavior. On the contrary, $q_{\alpha,\beta}$ are highly increased so that α, β are more strictly penalized. The objective of this change is to slow down the dynamics of the whole plant so that the oscillations stop. Furthermore, a blending procedure has been planned to interpolate the two values for $q_{\alpha,\beta}$, in order to not have sudden changes which might possibly degrade the performance. The results have been encouraging, since both the issues have been eliminated or at least their effect deeply reduced. This is because these issues happen in different areas, therefore control tunings varying with the position represent a valuable solution.
Another challenge regards the compensation of disturbances, whose lack leads to a steady state offset in position. In order to take them into account, two steps have been carried out. Firstly, the constant part of the angular disturbances is estimated looking at the steady state angles α and β: the angles indeed balance for that constant part and thus their values are the opposite of them. Secondly, the Kalman filter considers an augmented state vector that includes the disturbances. Therefore, the varying parts of the angular disturbances are estimated by the KF and subsequently included in the MPC, together with the constant part, to compensate for them. The highlighted procedure is able to decrease the steady state offset from around $0.03m$, as shown in fig. 6.7, to less than $0.01m$, as appears in fig. 6.11.

Future Works

To conclude, some considerations about possible future steps which might enhance the performance of the plant are presented. A first unsolved issue concerns the communication delay, which has not been measured or considered within the model of the plant. A solution to deeply decrease the delay is to directly implement the MPC within the microcontroller. Alternatively, the delay might be measured and easily included in the discrete linear system.
Another possibility is to substitute the Kalman filter with another kind of estimator: a suitable one is Moving Horizon Estimator (MHE), which has been described within section 3.1. MHE is considered the dual of MPC, since it shares the moving horizon concept. Moreover, its coexistence with MPC has been already tackled in many works.
In order to tackle the problems of steady state oscillation and velocity of con-

vergence, a blending procedure has been implemented. However, the linear interpolation could be replaced by a smoother one, such as an spline. This might probably enhance the performance of the experimental results.
Finally, the capability of MPC to deal with non-linear state space systems has already been discussed in chapter 2. Since the Stewart platform has been found to be easily represented through non linear equations, a non-linear MPC may be explored to compare its performance with respect to the linear one.

References

[1] Kassem A., Haddad H., Albitar C., *Comparison Between Different Methods of Control of Ball and Plate System with 6DOF Stewart Platform*, 2015.

[2] Edwards C., Spurgeon S., *Sliding Mode Control: Theory and Applications*, 1998.

[3] Moraga C., *Introduction to Fuzzy Logic*, 2005.

[4] Oravec M., Jadlovska A., *Model Predictive Control of a Ball and Plate Laboratory Model*

[5] Kumar J., *Design and Control of Ball on Plate System*, 2016.

[6] Leonov G. A., Zegzhda S. A., Zuev S. M., Ershov B. A., Kazunin D. V., Kostygova D. M., Kuznetsov N. V., Tovstik P. E., Tovstik T. P., Yushkov M. P., *Dynamics and Control of the Stewart Platform*, 2014.

[7] Bang H., Lee Y. S., *Implementation of a Ball and Plate Control System using Sliding Mode Control*, 2015.

[8] Harib K., Srinivasan K., *Kinematic and Dynamic Analysis of Stewart Platform-Based Machine Tool Structures*, 2003.

[9] Ferreau H. J., Almér S., Verschueren R., Diehl M., Frick D., Domahidi A., Jerez J. L., Stathopoulos G., Jones C., *Embedded Optimization Methods for Industrial Automatic Control*, 2017.

[10] Manozca O. M. A., De Moor B., *Introduction to Model Predictive Control (MPC)*, 2017.

[11] Verschueren R., Frison G., Kouzoupis D., van Duijkeren N., Zanelli A., Novoselnik B., Frey J., Albin T., Quirynen R., Diehl M., *acados – a Modular Open-Source Framework for Fast Embedded Optimal Control*, 2019.

[12] Diehl M., Bock H. G., Diedam H., Wieber P. B., *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control*, 2009.

[13] Diehl M., Bock H. G., *A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control*, 2005.

[14] Adhau S., Patil S., Ingole D., Sonawane D., *Implementation and Analysis of Nonlinear Model Predictive Controller on Embedded Systems for Real-Time Applications*, 2019.

[15] Zarzycki K., Lawrynczuk M., *Fast Real-Time Model Predictive Control for a Ball-on-Plate Process*, 2021.

[16] Maeder U., Borrelli F., Morari M., *Linear Offset-Free Model Predictive Control*, 2009.

[17] Pannocchia G., Gabiccini M., Artoni A., *Offset-free MPC Explained: Novelties, Subtleties and Applications*, 2015.

[18] Zenere A., Zorzi M., *Model Predictive Control meets Robust Kalman Filtering*, 2017.

[19] Ohhira T., Shimada A., *Movement Control Based on Model Predictive Control with Disturbance Suppression using Kalman Filter including Disturbance Estimation*, 2018.

[20] Kuehl P., Diehl M., Kraus T., Schloeder J. P., Bock H. G., *A real-time algorithm for moving horizon state and parameter estimation*, 2010.

[21] Copp D. A., Hespanha J. P, *Addressing Adaptation and Learning in the Context of MPC with MHE.*

[22] Huang R., Biegler L. T., Patwardhan S. C., *Fast Offset-Free Nonlinear Model Predictive Control Based on Moving Horizon Estimation*, 2010.

[23] Frison G., Kouzoupis D., Zanelli A., Diehl M., *BLASFEO: Basic Linear Algebra Subroutines for Embedded Optimization*, 2017.

[24] Frison G., Diehl M., HPIPM: a High-Performance Quadratic Programming Framework for Model Predictive Control, 2010.