UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER THESIS IN AUTOMATION ENGINEERING

DEVELOPMENT OF A REAL-TIME MOTION CUEING ALGORITHM BASED ON NONLINEAR MODEL PREDICTIVE CONTROL TECHNIQUES

SUPERVISOR
PROF. ING. ALESSANDRO BEGHI
UNIVERSITÀ DI PADOVA

CO-SUPERVISORS
ING. MATTIA BRUSCHETTA
DOTT. ING. ENRICO PICOTTI
UNIVERSITÀ DI PADOVA

MASTER CANDIDATE
SILVIA FANO

16 DECEMBER 2019 - ACADEMIC YEAR 2018/ 2019

Abstract

Dynamic driving simulators are nowadays a common tool in the automotive field. The effectiveness of such devices is strictly related to the quality of the driver's motion experience. The limited operational space does not allow a one-to-one reproduction of the vehicle behaviour, hence Motion Cueing Algorithms are necessary to generate a realizable set of control inputs to the platform, transforming the motion of a real vehicle into platform motions. The goal of this work is to develop a Motion Cueing Algorithm based on Model Predictive Control, which generate a trajectory that minimize a cost function, while not exceeding the simulator limits, exploiting the prediction of the future linear accelerations and angular velocities on the driver's head. Demonstrations of how the weighting matrices of the cost function affect the platform behaviour are provided, and considerations about computational time are made. Later, two receding horizon strategies are presented, with the aim to decrease the computational burden. A valuable MPC-based MCA requires a prediction horizon wide enough to anticapiate and prevent limits violation, thus maximizing the platform capabilities. In the last part of this thesis a proof of how the prediction window length affect the algorithm performances is given.

Contents

1 Introduction

The role of driving simulators has become increasingly important in recent years, thanks to the great variety of fields in which they operate [1]. Originally, simulators were mainly employed in the racing area, useful for driver training and vehicle set-up. Nowadays, they represent an essential device for engineering applications, since they facilitate the planning of road infrastructures and vehicles, by the design and the test of the interior of the car and the human-machine interface, allowing to save money and time. They are also involved in many research topics addressed to medical rehabilitation, driving assistance, security control systems, driving in critical conditions, emergency maneuvers and strategies to avoid accident. Moreover, driving simulators are crucial for studies about driver behaviour in rare or dangerous situations, such as the effects of tiredness and drugs, or driver distractions. The aim of a driving simulator is to reproduce the cues that the driver would perceive in a real car, by means of visual and auditory feedback in the case of static simulators, or adding the inertial stimulus in dynamic simulators. Hence, the effectiveness of these devices is measured by the quality of the driver's motion experience. In the dynamic case, since the limited platform operational space does not allow a one-to-one reproduction of the vehicle behaviour, a determinant role is played by the Motion Cueing Algorithms (MCAs). In their classical approach, MCAs make use of a combination of high-pass and low-pass filters [2] [3]: the former extract the fast dynamics and reproduce the related accelerations by means of small linear displacements, while the latter reproduce the low-frequency motions by the use of *tilt coordination*, i.e. the tilt of the platform to exploit the gravity force. This method, however, presents some disadvantages. First of all, the tuning phase is a difficult and non-intuitive process, since the parameters to be tuned do not have physical meaning, and also small changes in the values may translate in unexpected behaviour of the platform. Moreover, the platform constraints cannot be explicitly dealt, so parameters have to be tuned taking into account the worst-case motion, thus not exploiting the available workspace.

To overcome these issues, modern MCAs exploit the Model Predictive Control (MPC) technique [4][5][6], whose key properties are:

- it is model-based, so the controller can deal with the entire dynamics. Moreover, models of the human perception system can be adopted;
- it is a predictive technique, so the controller can anticipate the platform maneuvers, preventing constraints violation;
- the control input is calculated through the minimization of a cost function, and the tunable parameters are clearly associated to physical quantities;
- constraints can be explicitly declared, and this allows to best exploit the platform workspace.

The goal of this thesis is to develop a MPC-based motion cueing algorithm for a dynamic driving simulator consisting of a Hexapod Stewart Platform [7][8]. The kinematic model of the platform is implemented, imposing constraints on legs lengths, velocities and accelerations. Great attention is paid to the tuning phase. Later, considerations about computational time and how constraints treatment affects it are made, discussing two receding horizon strategies to decrease computational burden.
In details, the thesis is organized as follows:

- *Chapter 2* is divided in two parts. The first part introduces driving simulators, with a description of their fidelity-based classification and an overview of the main application fields. History and development of simulators are briefly presented. In the second part the most important classical motion cueing algorithms are described.
- *Chapter 3* illustrates the model predictive control techniques for motion cueing algorithms, highlighting its advantages and the most important differences with classical MCAs. The theory and the main features behind the method, together with its mathematical implementation, are deeply described.
- *Chapter 4* analyzes the kinematic model of the hexapod Stewart platform, with a detailed description of the geometry of the simulator and of the constraints on legs lengths, velocities and accelerations.
- *Chapter 5* concerns the implementation in MATLAB of the MPC process, together with the description of the constraints relaxation and the utilized cost function.
- *Chapter 6* shows the simulation results, investigating how the tunable parameters affects the algorithm performances. Different strategies to decrease the computational time are introduced.
- *Chapter 7* provides a recap of the work, and concluding remarks are given. At last, some considerations about possible future investigations are made.

2 Driving simulator and Motion Cueing Algorithms

The aim of motion simulation is the reproduction, in a virtual environment, of the stimuli and perceptions that a driver would feel in a real vehicle. To best meet the needs of this task, the driver has to receive visual, acoustic, vestibular and haptic information, named *cues*.

Motion simulators are classified according to fidelity [1]:

- Low-level simulator consists of a seat fixed to the ground and a fixed screen, with the view angle as large as possible. The driver receives force feedback from steering wheels and pedals, in addition to audio feedback;
- Mid-level simulator includes a car that can accelerate in one direction (often along y or x-axis or around z-axis) and a screen that can be fixed or can move together with the car. Audio and force feedback is given to the driver;
- High-level simulator is composed by a payload actuated in at least 6 DOF, but redundant DOF can be introduced to allow longer planar excursions. Graphics projected cover a view angle of at least 220°.

Driving simulators are employed in many and different areas of use, shortly described in the following.

Entertainment

This field, which mainly includes video games, employs fix screen simulators with at most 0 or 1 DOF. The work of military and entertainment industries increases the benefit of the low-level simulators, thanks to technical development which leads to higher fidelity systems.

Research

Driving simulators play a fundamental role in research, facilitating the planning of road infrastructures or the design and test of the interior of the car and the human-machine interface. In addition to this, they allow studies about driver behaviour in situations which are dangerous to reproduce, such as the effects of tiredness and drugs or driver distractions.

Training

The use of driving simulators for training sessions results to be advantageous for many reasons. They can be economical in cases when the real vehicle is too expensive, and they allow training in a risk-free situation, in which it is possible to monitor and advise students.

In the following, a general overview of the development of the simulators is presented, and then classical motion cueing algorithms are briefly proposed.

2.1 History of driving simulators

The birth of the motion simulator dates back at the beginning of the 20th century in the flight simulation field, to cope with the large cost of repair due to heavy landings by student.
In 1909 the Antoinette company starts producing complete aircraft by their own design [9], but the operation for the control was not easy and after some crashes the company developed a simulator to teach pilots to generate the correct maneuvers. The first simulator, shown in Fig. 2.1, comprised two halves of a barrel connected by a flexible joint, with one half forming a base and the other a cockpit. The cockpit was moved manually by the instructor, while the student used the control to correct the roll and pitch of the simulator and line up a reference bar with the horizon.

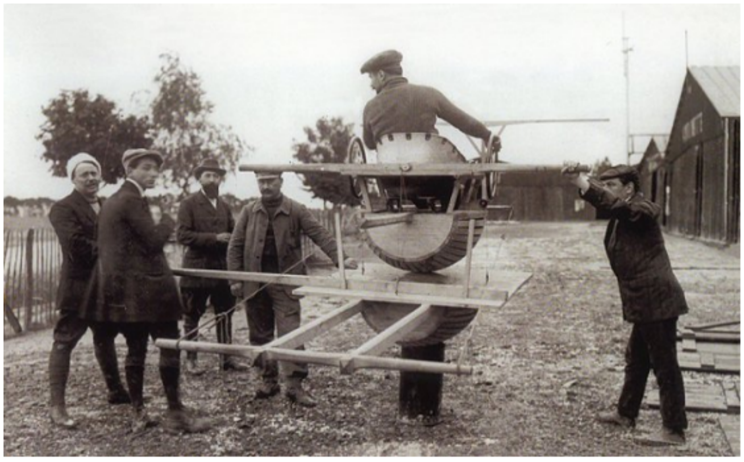

Figure 2.1: First motion simulator created by the Antoinette company

An important turning point in the history of the simulator coincides with the introduction of the Gough-Stewart Platform. In 1962 Stewart, starting from the idea of Gough, designed the parallel 6DOF system, consisting of two platforms and six actuators. The structure, also called *hexapod*, is composed of a fixed base, a moving top platform and moving, independent actuators, attached in pairs to three positions on the base, crossing over to three attachment point on the top disk. The kinematic structure of the hexapod allows movement of the payload following the 6 degrees of freedom, great accuracy in the positioning and satisfactory high acceleration of the platform.

The use of hydraulic actuation, typical of the first simulators, has been replaced by the development of electric technology, which led to an improvement in the driving simulation experience. In the following some of the most high-level fidelity simulators are introduced:

Volkswagen: In the early 70s, Volkswagen built the first driving simulator, composed by a car moving by a roll, pitch, and yaw mechanism, i.e. a 3DOF simulator. The driver received visual inputs from a flat screen situated in front of the seat.

IFAS: In 1984 IFAS produced a 1DOF simulator, moving on a hydraulically driven y-sled. The visuals were projected in a box in front of the driver.

VTI: In 1984 the Swedish Road and Traffic Research Institute VTI created a 4DOF simulator, a half car mounted on a motion platform that can accelerate in roll, pitch and yaw, on a y-sled. The simulator was equipped with a fixed screen. In 2004 VTI increased the size of the simulator, also adding a vibration table, to reproduce high frequent road rumbles.

Daimler-Benz: In 1985 Daimler-Benz introduced the first simulator that can be driven in 6DOF. It was an hydraulic hexapod, with a car situated inside a dome. Moreover, six CRT projectors display a 180° field of view. In 1993 the simulator improved due to an extension of the motion system in lateral direction.

JARI, **Nissan**: JARI and Nissan, respectively in 1996 and 1999, built a 6DOF hydraulic simulator. The hexapod, equipped with three additional cylinders, attaches the payload at the height of the driver's head, since the idea is that it is easier to tilt the payload around the driver's vestibular system.

NADS-1: In 2002 the North American Driving Simulator is the most advanced driving simulator. It is a 9DOF driving simulator, consisting of an hexapod mounted on a xy-table. On the top of the hexapod there is a yaw turntable, which provides yaw-accelerations and supported a dome with a full-size car inside.

UoLDS: Since 2006, the simulator of University of Leeds Driving Simulator is one of the most advanced driving simulators in research environment. In particular it is employed in intelligent speed adaptation and effects of automated systems on safety.

2.2 Classical Motion Cueing Algorithms

Although in the last decades driving simulators have reached a considerable level of realism, thanks to advanced graphic systems, good real-time vehicle models and high-fidelity steering torque motors, it is not totally clear how to best produce motion cues from the real vehicle motion. One of the main issues that arises from the use of simulators is the limited workspace of the motion platform, considerably smaller than the space used by a real car to perform the maneuver. So, transformations are needed to compute a realizable set of platform motions from the motion of the real vehicle [3]. These transformations are referred to as *Motion Cueing Algorithms*, the most important of which are presented in the following.

Classical washout algorithm

Classical washout algorithm, whose scheme is presented in Fig. 2.2, is the first motion cueing algorithm presented in the literature. It uses high-pass filters to remove the low-frequency motion components, allowing the reproduction of the higher frequency ones, and guaranteeing that the platform does not exceed its limits. Most classical algorithms make use of tilt coordination, i.e. use roll and pitch rotations, at a rate below the perception threshold, to reproduce, respectively, sustained lateral and longitudinal accelerations. The low-pass filter used for tilt coordination is 2nd-order and the cut-off frequency is chosen such that all the translational motion is reproduced, through platform translation or by platform tilt. The filter cut-off frequencies, damping factors, and gains are tuned through a trial-and-error process, considering the worst-case vehicle acceleration. For a non-experienced user, it is not intuitive to decide which parameters to vary and in which direction, in order to achieve the desired platform response.

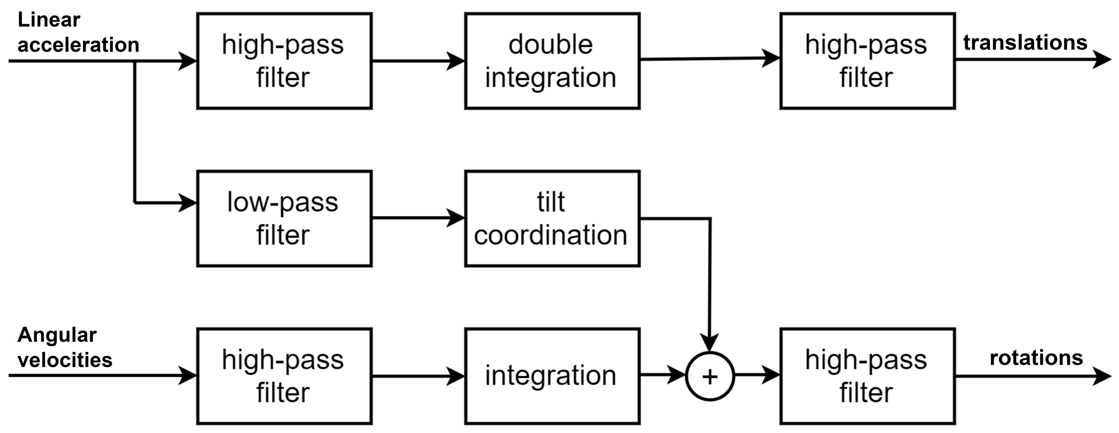

Figure 2.2: Classical motion cueing scheme

Adaptive washout algorithm

This algorithm, which employs filters with adaptive gains, aims to minimize a cost function that penalizes motion error, motion magnitude and the variation of the adaptive parameters from their initial values. The resulting motion is a trade-off between a faithful reproduction of a real car performance and the limitation of the platform movement. The pitch/longitudinal and roll/lateral axis pairs are treated together and the cost function to be minimized is the following:

$$J_y = \frac{1}{2}[W_1(a - a_s)^2 + W_2(\dot{\theta} - \dot{\theta}_s)^2 + W_3\dot{y}_s^2 + W_4 y_s^2 + W_5\dot{\theta}_s^2 + W_6\theta_s^2 + \sum_{i=1}^{2} W_{i+6}\Delta P_i^2]$$

where W_i are the weights used to tune the filters, and P_i are the adaptive filters gains. Once again, the tuning process is based on the worst-case acceleration. If on the one hand the difficulties of how to choose the parameters remain, on the other it is more intuitive to tune the cost weights.

Optimal control-based algorithm

The algorithms use the approach typical of the tracking problem represented in Fig. 2.3: the acceleration perceived by the driver in the platform should track the acceleration perceived in a real car, hence the aim is to minimize the error *e*(s), i.e. the difference between the two, while respecting the simulator workspace. For this purpose it is necessary to understand the relationship between the body motion and the motion perceived by the brain, and so the vestibular model is introduced. The washout filter is generated by a Linear Quadratic Regulator design process and results to be the one that minimizes the cost function

$$J = E\{e^T Qe + x_d^T R_d x_d + u_s^T RU_s\}$$

where *e* is the perception error, x_d a vector comprising the states of the platform, u_s the input for the platform motion, and Q, R_d and R are cost weight matrices.
The tuning process is easier and intuitive for the non-expert: filter parameters are completely removed and the tuning is done by just regulating the cost function weights.

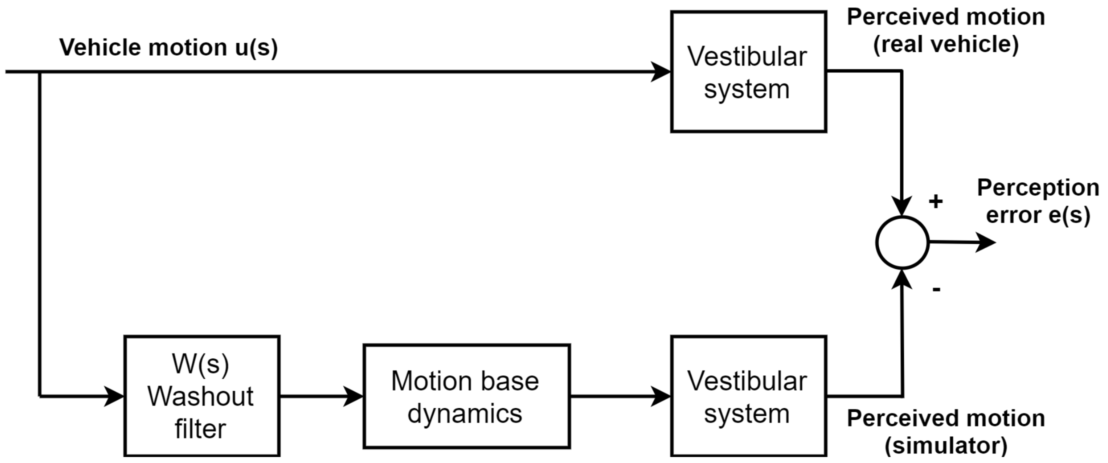

Figure 2.3: Motion tracking problem

The three algorithms described present some weaknesses, among which:

- The tuning process, except for optimal control-based algorithms, is complex and non-intuitive, due to the fact that the filter parameters do not have physical meaning, and also small value variations may have unexpected results on driver's perception;
- Platform limits are not taken into account explicitly so it is necessary to tune the algorithm for the worst-case motion;
- It is not possible to use any information on driver's behavior in the future.

An algorithm that tries to compensate for these shortcomings will be presented in the next chapter.

3
MPC based Motion Cueing Algorithms

In this chapter the *Model Predictive Control* technique will be illustrated. MPC occupies an important role in the field of control engineering, both in the industrial and academic research sphere. The approach differs from the other classical motion cueing algorithms mainly in four aspects:

1. It is **model-based**, and this provides important information to the algorithm since the controller can deal directly with the entire dynamics. An accurate model can reach consistent prediction of the future.
2. It is a **predictive technique**, hence the algorithm optimizes the motion of the platform over a prediction horizon. Thanks to the knowledge of future reference trajectories, the controller can anticipate the platform maneuvers, for an optimal simulation experience.
3. It is an **optimal control strategy**, since the minimization of a cost function allows to calculate the optimal control input. The cost function can be formulated to define the best behaviour of the system, and can depend on inputs, states, and outputs.
4. It can **handle constraints**, soft or hard, which are explicitly defined. This aspect is of fundamental importance to deal with the strict limits required by the use of simulators, and allows to best exploit platform workspace.

The model predictive control computes the input sequence $\mathbf{u}$ to optimize the future behaviour of the output $\mathbf{y}$. The optimization is performed within a limited time window giving plant information at the start of the time window.
In the following sections the method will be described in detail, and in particular the model, the cost function and the constraints will be analyzed. The last two sections are reserved to mathematical formulation and optimization.

3.1 Advantages of MPC

The significant development in control engineering technology has a strong impact on all areas of the control discipline, such as theory, controllers, actuators, sensors, industrial processes, computer methods and so on. Much of these challenges involved the industrial field, which benefited from the Richalet's model predictive control technique from the late 1960s [10].
MPC is one of the more popular advanced techniques for industrial process applications and, in the last years, the development of robust and efficient implementations, and performing software tools made the design of MPC algorithms easier.
The main advantages that distinguish MPC from other methods can be summarized as follows:

- it can be used in different levels of the process control structure;
- the design formulation is easy and intuitive. Changing model or specifications does not require complete redesign;
- the parameter tuning process is simple for non-expert users, thanks to its intuitiveness;
- it can deal with soft and hard constraints, which can be explicitly defined;
- the optimization process can be performed on-line;
- it can handle Multi-Input Multi-Output system;
- non-linearities in both system dynamics and constraints can be introduced;
- since the algorithm considers the platform motion over a future time window, the effect of future disturbances can be anticipated and removed. Moreover, the knowledge of future reference trajectories can be used to best exploit platform workspace.

Model Predictive Control is introduced in the following by consulting Fabio Maran's Ph.D. thesis [11], however, for a more detailed illustration, Wang [10] is recommended.

3.2 Theory of MPC

Model Predictive Control technique is based on iterative, finite-horizon optimization of the future control trajectory of the plant, subject to operational constraints. To introduce the theory behind MPC, a discrete time domain problem is considered. Assume that at time $k \in \mathbb{Z}$ a reference trajectory $\mathbf{r}(t|k)$, $t \geq k$, and a current measurement of the output $\mathbf{y}(k)$ are available; the current input is not yet computed. Suppose to have a complete model of the process to be controlled and that the state of the system (or an estimate of it) is available. Hence, considering a time window of length N_p (N_p is called *prediction horizon length*), it is possible to compute the future output sequence

$$\mathbf{y}(k+i|k), \qquad i = 1, ..., N_p,$$

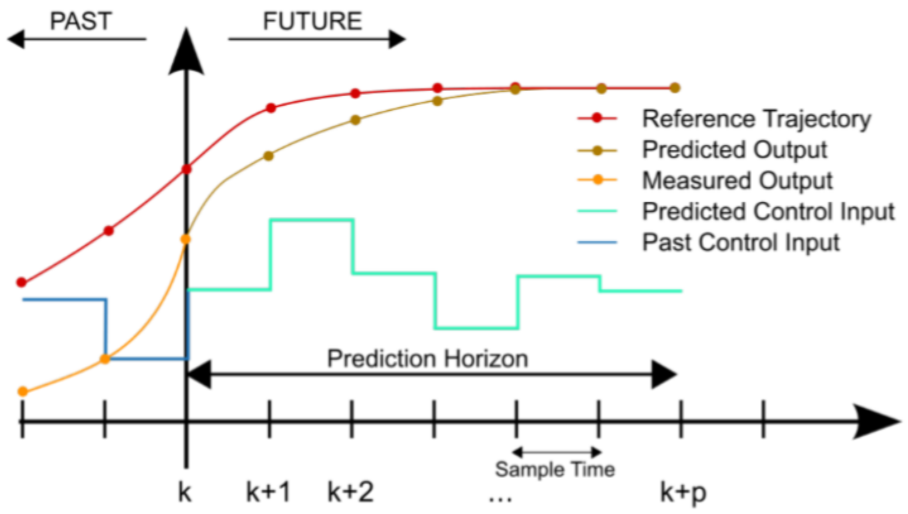

Figure 3.1: Model Predictive Control scheme

resulting from the input

$$\mathbf{u}(k+i|k), \qquad i = 1, ..., N_p - 1.$$

Fig. 3.1 shows a graphical representation of this principle. The main idea is to calculate the input sequence $\hat{\mathbf{u}}(k+i|k)$ that minimizes a *cost function* $\mathbf{J}$, typically related to the tracking error, i.e. the difference between the desired and the actual outputs

$$\boldsymbol{\epsilon}(k+i|k) = \mathbf{r}(k+i|k) - \mathbf{y}(k+i|k) \qquad i = 1, ..., N_p$$

while respecting a set of constraints that can be imposed on inputs, states, outputs or other variables related to the problem. Considering the optimal control sequence just computed, only its first element is applied, thus

$$\mathbf{u}(k) = \hat{\mathbf{u}}(k|k)$$

is used to produce the new output $\mathbf{y}(k+1)$. Iteratively, at time $k+1$ a new optimal input sequence is computed, and once again only its first element is considered. This technique is referred to as *receding horizon*: at each time step, the prediction horizon is shifted in order to deal with N_p instants in the future, starting from the current time. In this way the MPC algorithm can use the information provided by the new measurements to correct errors due to disturbances, errors in the model or system variations. This gives the algorithms a certain degree of robustness. Note that in general the first element of the new optimal control sequence $\hat{\mathbf{u}}(k+1|k+1)$ is different from the second element of the previous control sequence $\hat{\mathbf{u}}(k+1|k)$, since the output $\mathbf{y}(k)$ depends on past input $\mathbf{u}(k-1)$, $\mathbf{u}(k-2)$,..., and not by actual input $\mathbf{u}(k)$. Thus, at each time step the algorithm takes advantage of the information on new computation and this improves the MPC performances.

3.3 Process model

The process model is extremely important in model predictive control, since the effectiveness of the control is strictly related to the accuracy of system representation, even if the receding horizon technique can compensate some inaccuracies of the model. In recent years, state-space models have been preferred among various model structures, such as FIR models, step-response models, impulse-response models and transfer function models. This because, thanks to the simplicity of the design framework, they make immediately accessible state variables and, if not directly available, are particularly well suited to design state estimator. Moreover, they are the best way to represent MIMO systems. In this thesis, state-space models are used for the implementation of a Motion Cueing algorithm.

Consider a linear, discrete state-space model of the form

$$\begin{aligned} \mathbf{x}_m(t+1) &= A_m\mathbf{x}_m(t) + B_m\mathbf{u}(t) \\ \mathbf{y}(t) &= C_m\mathbf{x}_m(t) \end{aligned} \tag{3.1}$$

where $\mathbf{u}$ is the input variable, $\mathbf{x}_m$ is the state variable and $\mathbf{y}$ is the output. Following the formulation proposed by Wang, it is convenient to approximate the input derivative in order to include $\Delta\mathbf{u}(t)$ in the cost function.
Hence the control variable is

$$\Delta\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}(t-1). \tag{3.2}$$

Considering then the difference of the state variable

$$\Delta\mathbf{x}_m(t) = \mathbf{x}_m(t) - \mathbf{x}_m(t-1), \tag{3.3}$$

the state equation can be expressed in the new form

$$\Delta\mathbf{x}_m(t+1) = A_m\Delta\mathbf{x}_m(t) + B_m\Delta\mathbf{u}(t). \tag{3.4}$$

The output difference results to be

$$\mathbf{y}(t+1) - \mathbf{y}(t) = C_mA_m\Delta\mathbf{x}_m(t) + C_mB_m\Delta\mathbf{u}(t) \tag{3.5}$$

and a new state variable vector is introduced to enclose $\Delta\mathbf{x}_m(t)$ and $\mathbf{y}(t)$, obtaining

$$\mathbf{x}(t) = \begin{bmatrix} \Delta\mathbf{x}_m(t)^T & \mathbf{y}(t)^T \end{bmatrix}^T \tag{3.6}$$

The previous equation leads to the following expression of the state-space model:

$$\begin{aligned}\mathbf{x}(t+1) &= \left[\begin{array}{c|c} A_m & \mathbf{0} \\ \hline C_m A_m & \mathbf{I} \end{array}\right] \mathbf{x}(t) + \left[\begin{array}{c} B_m \\ \hline C_m B_m \end{array}\right] \Delta\mathbf{u}(t) \\ \mathbf{y}(t) &= \left[\begin{array}{c|c} \mathbf{0} & \mathbf{I} \end{array}\right] \mathbf{x}(t)\end{aligned} \tag{3.7}$$

3.4 Cost function

The optimal control input sequence is obtained by minimizing a cost function, which, basing on the specific features of the system, is designed to follow the reference signals, while respecting the plant limits expressed by soft or hard constraints. To reduce complexity in resolution, the cost function $\mathbf{J}$ has a quadratic form, and it can consider, for examples, the error between the predicted trajectory and the future reference in the prediction horizon of length N_p, the future inputs and input difference in the control horizon N_c:

$$\begin{aligned} J(t) = &\sum_{j=1}^{N_p} \delta(j)||\mathbf{y}(t+j|t) - \mathbf{r}(t+j)||^2 + \sum_{j=0}^{N_c-1} \lambda(j)||\mathbf{u}(t+j)||^2 + \\ &+ \sum_{j=0}^{N_c-1} \gamma(j)||\Delta\mathbf{u}(t+j)||^2 \end{aligned} \tag{3.8}$$

$\mathbf{J}(t)$ has to be minimized with respect to $\mathbf{u}(t)$ and $\Delta\mathbf{u}(t)$. Since system inputs comprise longitudinal accelerations, that are high-frequency and discontinuous signals, sometimes it may be useful to introduce in the cost function the weighting term $\gamma(j)$, to act on their (approximate) derivative, in order to achieve a certain degree of regularity and to avoid excessive stress of the actuators and possible unfeasibilities in the optimization problem. It is important that the cost function is expressed in a quadratic form to reduce the computational cost, however, the choice of which terms have to be included is not unique and depends on the problem needs.

In equation (3.8), the control horizon length N_c is introduced. It corresponds to the length of the input sequence that is calculated by the algorithm at each step, so $N_c \leq N_p$. If $N_c < N_p$, the last $N_p - N_c$ elements of the control sequence are considered constant and equal to the last one of the control horizon, hence

$$\mathbf{u}(t+N_c-1) = \mathbf{u}(t+N_c) = \mathbf{u}(t+N_c+1) = ... = \mathbf{u}(t+N_p) \tag{3.9}$$

or, in terms of $\Delta\mathbf{u}$

$$\Delta\mathbf{u}(t+N_c) = \Delta\mathbf{u}(t+N_c+1) = ... = \Delta\mathbf{u}(t+N_p) = 0 \tag{3.10}$$

This approach is useful in cases in which fast computation is needed, since in this way the size of the optimal sequence to be computed is smaller, and the problem resolution easier. During the tuning phase, the parameters N_c and N_p have to be tuned accurately to obtain satisfactory performance.

3.5 Constraints

In many real situations, the presence of constraints on signals is unavoidable, due to consumer specifications, safety restriction or physical limitation of the simulator workspace. Controller parameters have to be tuned to keep the platform within its bounds, but at the same time the control system should drive the process towards the constraints as close as possible, since closer to limits in general means better performances. One of the most important characteristics of MPC is that it is a constrained optimal control procedure, since it takes constraints into account in its formulation. Therefore, the optimal unconstrained solution is modified in such a way that constraints are not violated. This necessarily implies an increase of complexity in finding the solution, since the introduction of constraints makes the problem impossible to be solved in an analytical way, and heuristic solvers have to be used. An important aspect has to be considered: the introduction of constraints leads to possible infeasibility of the problem, i.e., given a certain set of constraints, there might be particular situations in which it is impossible to find a solution. The solver should be able to handle this critical case, for example by relaxing some of the constraints in the so called *soft constraint*, in opposition to *hard* ones which cannot be violated. In this context, constraints are translated in expressions concerning bounds on input, state or output

$$\mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max} \tag{3.11}$$

$$\mathbf{x}_{min} \leq \mathbf{x}(t) \leq \mathbf{x}_{max} \tag{3.12}$$

$$\mathbf{y}_{min} \leq \mathbf{y}(t) \leq \mathbf{y}_{max} \tag{3.13}$$

Note that constraints can be also defined over others system variables. Considering the process model (3.7), constraints on inputs variations are set

$$\Delta\mathbf{u}_{min} \leq \Delta\mathbf{u}(t) \leq \Delta\mathbf{u}_{max} \tag{3.14}$$

The minimization of the cost function taking constraints into account ensures the best possible control input within limits, while better exploiting the operational space.

3.6 Mathematical formulation

The aim is to manipulate the MPC problem in order to obtain a QP one, where only $\Delta\mathbf{u}$ has to be minimized. It is important to notice that the cost function (3.8) is quadratic, and constraints

(3.11), (3.12) and (3.13) are linear.

A Quadratic Programming (QP) problem is an optimization problem in which:

1. the function to be minimized has a quadratic form;
2. the constraints are linear.

Dealing with QP problem has several advantages: it has been studied in detailed in literature, it is not too complex to solve and different optimizer are provided.

Consider model (3.7) and define matrices A, B and C as follows:

$$A = \left[\begin{array}{c|c} A_m & \mathbf{0} \\ \hline C_m A_m & \mathbf{I} \end{array}\right] \qquad B = \left[\begin{array}{c} B_m \\ \hline C_m B_m \end{array}\right] \qquad C = \left[\begin{array}{c|c} \mathbf{0} & \mathbf{I} \end{array}\right] \tag{3.15}$$

then the state evolution is

$$\begin{aligned} \mathbf{x}(t+1|t) &= A\mathbf{x}(t) + B\Delta\mathbf{u}(t) \\ \mathbf{x}(t+2|t) &= A\mathbf{x}(t+1|t) + B\Delta\mathbf{u}(t+1) = A^2\mathbf{x}(t) + AB\Delta\mathbf{u}(t) + B\Delta\mathbf{u}(t+1) \\ &\vdots \\ \mathbf{x}(t+N_p|t) &= A^{N_p}\mathbf{x}(t) + A^{N_p-1}B\Delta\mathbf{u}(t) + A^{N_p-2}B\Delta\mathbf{u}(t+1) + ... + \\ &+ A^{N_p-N_c}B\Delta\mathbf{u}(t+N_c-1) \end{aligned} \tag{3.16}$$

and considering $N_c < N_p$ the corresponding outputs are given by

$$\begin{aligned} \mathbf{y}(t+1|t) &= CA\mathbf{x}(t) + CB\Delta\mathbf{u}(t) \\ \mathbf{y}(t+2|t) &= CA\mathbf{x}(t+1|t) + CB\Delta\mathbf{u}(t+1) = \\ &= CA^2\mathbf{x}(t) + CAB\Delta\mathbf{u}(t) + CB\Delta\mathbf{u}(t+1) \\ &\vdots \\ \mathbf{y}(t+N_p|t) &= CA^{N_p}\mathbf{x}(t) + CA^{N_p-1}B\Delta\mathbf{u}(t) + CA^{N_p-2}B\Delta\mathbf{u}(t+1) + ... + \\ &+ CA^{N_p-N_c}B\Delta\mathbf{u}(t+N_c-1). \end{aligned} \tag{3.17}$$

As can be seen, the output variables are expressed as a function of the current state $\mathbf{x}(t)$ and the input sequence $\Delta\mathbf{u}(t+i)$, with $i = 0, 1, ..., N_c - 1$. It is possible to introduce the vectors

$$\mathbf{Y} = \text{vec}\{\mathbf{y}(t+i|t)\}_{i=1,...,N_p} = \begin{bmatrix} \mathbf{y}(t+1|t) \\ \mathbf{y}(t+2|t) \\ \vdots \\ \mathbf{y}(t+N_p|t) \end{bmatrix} \in \mathbb{R}^{(N_p \cdot n_{out}) \times 1} \tag{3.18}$$

$$\Delta \mathbf{U} = \text{vec}\{\Delta \mathbf{u}(t+i|t)\}_{i=1,\dots,N_c-1} = \begin{bmatrix} \Delta \mathbf{u}(t) \\ \Delta \mathbf{u}(t+1) \\ \vdots \\ \Delta \mathbf{u}(t+N_c-1) \end{bmatrix} \in \mathbb{R}^{(N_c \cdot n_{in})\times 1} \tag{3.19}$$

where n_{out} and n_{in} specify the size of $\mathbf{y}$ and $\Delta\mathbf{u}$ respectively, and represent the input-output prediction evolution using the single equation

$$\mathbf{Y} = F\mathbf{x}(t) + \Phi \Delta \mathbf{U} \tag{3.20}$$

where matrices F and Φ are defined as follows

$$F = \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{N_p} \end{bmatrix}, \quad \Phi = \begin{bmatrix} CB & 0 & 0 & \dots & 0 \\ CAB & CB & 0 & \dots & 0 \\ CA^2B & CAB & CB & \dots & 0 \\ \vdots & & & & \vdots \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & \dots & CA^{N_p-N_c}B \end{bmatrix} \tag{3.21}$$

Introducing a vectorization of the reference signal $\mathbf{r}(t+i), i = 1, \dots, N_p$, the cost function can be rewritten as

$$J(\Delta\mathbf{U}) = (\mathbf{R}_s - \mathbf{Y})^T Q (\mathbf{R}_s - \mathbf{Y}) + \mathbf{U}^T S \mathbf{U} + \Delta\mathbf{U}^T R \Delta\mathbf{U} \tag{3.22}$$

where Q, S, R are block diagonal weight matrices.
Let express the input sequence $\mathbf{u}(t+i), i = 0, \dots, N_c - 1$ as a function of $\Delta\mathbf{U}$ in order to obtain a cost function depending only on $\Delta\mathbf{U}$

$$\underbrace{\begin{bmatrix} \mathbf{u}(t) \\ \mathbf{u}(t+1) \\ \vdots \\ \mathbf{u}(t+N_c-1) \end{bmatrix}}_{\mathbf{U}} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{I} & \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{0} \\ \vdots & & & & \vdots \\ \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \end{bmatrix}}_{T} \underbrace{\begin{bmatrix} \Delta\mathbf{u}(t) \\ \Delta\mathbf{u}(t+1) \\ \vdots \\ \Delta\mathbf{u}(t+N_c-1) \end{bmatrix}}_{\Delta\mathbf{U}} + \underbrace{\begin{bmatrix} \mathbf{u}(t-1) \\ \mathbf{u}(t-1) \\ \vdots \\ \mathbf{u}(t-1) \end{bmatrix}}_{\bar{\mathbf{U}}_i} \tag{3.23}$$

or in vectorized form

$$\mathbf{U} = T\Delta\mathbf{U} + \bar{\mathbf{U}}_i \tag{3.24}$$

The substitution of (3.24) and (3.20) in (3.22) leads to

$$J(\Delta\mathbf{U}) = (\mathbf{R}_s - F\mathbf{x}(t) - \Phi\Delta\mathbf{U})^T Q(\mathbf{R}_s - F\mathbf{x}(t) - \Phi\Delta\mathbf{U}) + \tag{3.25}$$
$$+ (T\Delta\mathbf{U} + \bar{\mathbf{U}}_i)^T S(T\Delta\mathbf{U} + \bar{\mathbf{U}}_i) + \Delta\mathbf{U}^T R\Delta\mathbf{U} \tag{3.26}$$

which, discarding the constant terms not depending on $\Delta\mathbf{U}$ and thus not affecting the results of the minimization, and after some manipulations, becomes

$$J(\Delta\mathbf{U}) = \Delta\mathbf{U}^T(\Phi^T Q\Phi + R + T^T ST)\Delta\mathbf{U} + 2\Delta\mathbf{U}^T(\Phi^T Q(F\mathbf{x}(t) - \mathbf{R}_s) + T^T S\bar{\mathbf{U}}_i). \tag{3.27}$$

Defining

$$H = 2(\Phi^T Q\Phi + R + T^T ST) \tag{3.28}$$
$$f = 2(\Phi^T Q(F\mathbf{x}(t) - \mathbf{R}_s) + T^T S\bar{\mathbf{U}}_i) \tag{3.29}$$

(3.27) can be rewritten as follows

$$J(\Delta\mathbf{U}) = \frac{1}{2}\Delta\mathbf{U}^T H\Delta\mathbf{U} + \Delta\mathbf{U}^T f \tag{3.30}$$

which is the expression of the cost function for a QP problem, where $\Delta\mathbf{U}$ is the variable to be minimized.

Problem without constraints

If no constraints are taken into consideration, the optimization problem can be solved by imposing the derivative of the cost function with respect to $\Delta\mathbf{U}$ equal to 0. Assuming that the inverse of H exists, it results

$$\frac{\partial J}{\partial \Delta\mathbf{U}} = 0 \quad \Rightarrow \quad \Delta\mathbf{U}_{\text{opt}} = H^{-1} f \tag{3.31}$$

Problem with constraints on inputs and outputs

If constraints are taken into account, the QP problem comprises also the inequality

$$A\Delta\mathbf{U} \leq b \tag{3.32}$$

with A and b containing the information on constraints expressed as a function of $\Delta\mathbf{U}$.
It is necessary to reformulate constraints, referring them to inputs variation. Limits on $\Delta\mathbf{u}(t)$

are easier to set, leading to

$$\underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & -\mathbf{I} & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & -\mathbf{I} \end{bmatrix}}_{M_1} \underbrace{\begin{bmatrix} \Delta\mathbf{u}(t) \\ \Delta\mathbf{u}(t+1) \\ \vdots \\ \Delta\mathbf{u}(t+N_c) \end{bmatrix}}_{\Delta\mathbf{U}} \leq \underbrace{\begin{bmatrix} \Delta\mathbf{u}_{max} \\ \Delta\mathbf{u}_{max} \\ \vdots \\ \Delta\mathbf{u}_{max} \\ -\Delta\mathbf{u}_{min} \\ -\Delta\mathbf{u}_{min} \\ \vdots \\ -\Delta\mathbf{u}_{min} \end{bmatrix}}_{n_1} \tag{3.33}$$

Equation (3.24) allows translating constraints on $\mathbf{u}(t)$ to $\Delta\mathbf{u}(t)$, resulting in

$$\underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{I} & \mathbf{I} & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ -\mathbf{I} & -\mathbf{I} & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ -\mathbf{I} & -\mathbf{I} & \dots & -\mathbf{I} \end{bmatrix}}_{M_2} \underbrace{\begin{bmatrix} \Delta\mathbf{u}(t) \\ \Delta\mathbf{u}(t+1) \\ \vdots \\ \Delta\mathbf{u}(t+N_c) \end{bmatrix}}_{\Delta\mathbf{U}} \leq \underbrace{\begin{bmatrix} \mathbf{u}_{max} - \mathbf{u}(t-1) \\ \mathbf{u}_{max} - \mathbf{u}(t-1) \\ \vdots \\ \mathbf{u}_{max} - \mathbf{u}(t-1) \\ -\mathbf{u}_{min} + \mathbf{u}(t-1) \\ -\mathbf{u}_{min} + \mathbf{u}(t-1) \\ -\mathbf{u}_{min} + \mathbf{u}(t-1) \\ \vdots \\ -\mathbf{u}_{min} + \mathbf{u}(t-1) \end{bmatrix}}_{n_2} \tag{3.34}$$

Observe that n_2 is time-variant, since it depends on $\mathbf{u}(t-1)$ with t the current time. Hence it has to be updated at each time step and cannot be defined offline.
The same considerations hold for constraints on $\mathbf{y}(t)$, which are translated into constraints on $\Delta\mathbf{u}(t)$, by using equation (3.20). The following formulation is obtained

$$\underbrace{\begin{bmatrix} \Phi \\ -\Phi \end{bmatrix}}_{M_3} \Delta\mathbf{U} \leq \underbrace{\begin{bmatrix} \mathbf{Y}_{max} - F\mathbf{x}(t) \\ -\mathbf{Y}_{min} + F\mathbf{x}(t) \end{bmatrix}}_{n_3} \tag{3.35}$$

where $\mathbf{Y}_{max}$ and $\mathbf{Y}_{min}$ are, respectively, the vectorizations of the upper and lower bounds of $\mathbf{y}_{max}$ and $\mathbf{y}_{min}$.
Finally, by combining inequalities (3.33), (3.34) and (3.35) constraints can be expressed in the

classic form for a QP problem

$$A\Delta\mathbf{U} \leq \mathbf{b} \qquad \text{with} \qquad A = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} \tag{3.36}$$

The procedure described above allows to convert an MPC problem with quadratic cost function and linear constraints into a QP problem, where the control sequence $\Delta\mathbf{u}(t+i), i = 0, ...,$ $N_c - 1$ has to be calculated by minimizing the cost function $\mathbf{J}$.

3.7 Optimization

To solve QP problem, the use of appropriate solvers is necessary, since the presence of constraints makes impossible to find an analytical solution. Generic QP solvers are inadequate for the strict real-time requirements, since fast dynamics needs small computation times. Therefore, particular algorithms that exploit the structure of QPs arising in MPC are needed, and can be of two different types: offline and online solvers.

The major aspect of offline optimizers is the offline precomputation of a solution for all the possible instances arising in the problem. Constraints are used to divide the state space in polyhedral critical regions, in each of which the optimal control law is an affine function of the state. The online procedure consists of evaluating the region associated with the current state and its corresponding affine function. The main drawback of this strategy is that it can only deal with low-dimensional problems, since the number of regions increases with the number of constraints and the offline computation burden has exponential growth, making the evaluation slower. Moreover, the online tuning, of crucial importance for the presence of the human driver in the loop, becomes very difficult due to the offline computation.

For these reasons, in this applications the use of online QP solvers has been preferred. Online quadratic optimization comprise two main categories of algorithms:

1. Interior Point (IP): the idea is to replace the inequality constraint with a log barrier function in the objective. Exploiting the convexity of the cost function, these methods are appropriate for dealing with linear and QP problems. However, they lack effective hot-start strategies, i.e. the capability of obtaining a smart starting point for the current problem using the results derived at the previous step, hence reducing the computational effort.

2. Active Set (AS): its idea is similar to the one used in the offline approach, but AS aim at calculating the current critical region without a pre-computation phase. When an iterate violates some inequality constraints, these are treated as equality constraints (active constraints) until the algorithm moves off of the inequality. At this point those constraints are ignored. These methods provide a hot start strategy assuming that the active set in one QP does not change much from the previous one during the MPC process. Moving from the old problem to the new one along directions derived from the parameters, exploiting the convexity of the parameter space and considering only a subset of constraints in order

to reduce problem complexity, intermediate QPs are solved, until the global solution is found. In case of boundary crossing of the current set, the new active set is calculated online.

To deal with the MPC problem, an Active Set method has been preferred.

The main principle of AS, is that, at time t of the MPC, there exists an optimal solution $\Delta\mathbf{U}^*$ which minimizes (3.30), associated to specific values of vector f in (3.29) and constrained vector b in (3.36), both depending on the state $\mathbf{x}(t)$. Consider that starting from an optimum point guarantees the feasibility at the start of the iterations. Subsequently, at the next time step, the parameters $f(\mathbf{x}(t+1))$ and $b(\mathbf{x}(t+1))$ are calculated and the new solution $\Delta\mathbf{U}^{*,new}$ is computed. At this point, in order to reduce the problem complexity, only the set of constraints satisfied by equalities are taken into account, and the corresponding QP problem is solved. The feasibility and the optimality of this intermediate solution has to be verified; if one of this two conditions doesn't hold, a different set of constraints is considered and the process iterated until an optimal solution is found.

The chosen AS algorithm has been proposed by Ferreau and exploit the structure of a QP problem derived from MPC. The idea is that of a smart recalculation way that allows to move on a feasible line from $\mathbf{x}(t)$ to $\mathbf{x}(t+1)$, evaluating the progress through the new optimum when updating vectors f and b, while ensuring the feasibility of the intermediate problems: all the steps from the old optimal solution to the new one are performed along homotopic variations of the parameters. The new optimum is obtained when the maximum step length of homotopy that can be calculated is reached. This interpretation is thought for MPC process, since it can be assumed that the system behaviour does not change much during two consecutive time steps.
The proposed method has a freeware C++ implementation by the OPTEC group at University of Leuven, qpOASES, a ready-to-use package with real-time capabilities that offers some useful solutions for matching different real-time requirements. Major analysis and details can be found in Ferreau [12].

4
Hexapod Stewart platform

In the following section, the plant model will be described, starting from the model in [7]. The simulator employed in this thesis is a Stewart Platform and the relationship between input, state, and output governing the kinematic model will be described, with particular attention to the limitations of the operational space.

4.1 Geometry and workspace

The hexapod is composed by a base $\mathcal{B}$ and a moving top platform $\mathcal{P}$, connected by six prismatic actuators, or legs. As can be seen in Fig.(4.1), to describe the dynamics of the hexapod, it is important to introduce three Cartesian reference frames:

- the Platform frame (P) is attached to the center of the hexapod top disk $\mathcal{O}_P$ and moves together with the platform. The X axis is pointing forward, the Y axis to the right, and the Z axis downward;
- The World frame (W) is an inertial frame, with the origin $\mathcal{O}_W$ coinciding with $\mathcal{O}_P$ when the hexapod is in neutral position. The axes convention is the same used for frame P;
- the Head frame (H) is located at the center of driver's head $\mathcal{O}_H$ and moves together with the driver. The X axis is pointing forward, the Y axis to the left, and the Z axis upward. In this thesis it is assumed that position and orientation of frame H with respect to frame P do not change during the motion. More precisely, the position of driver's head expressed in Platform frame ${}^P\mathbf{r}_H$, and the rotation matrix from Platform to Head frame ${}^HR_P$ are set equal to:

$$
{}^P\mathbf{r}_H = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \qquad\qquad {}^HR_P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\pi) & -sin(\pi) \\ 0 & sin(\pi) & cos(\pi) \end{bmatrix}
$$

It implies that ${}^{P}R_H = {}^{H}R_P^T$.

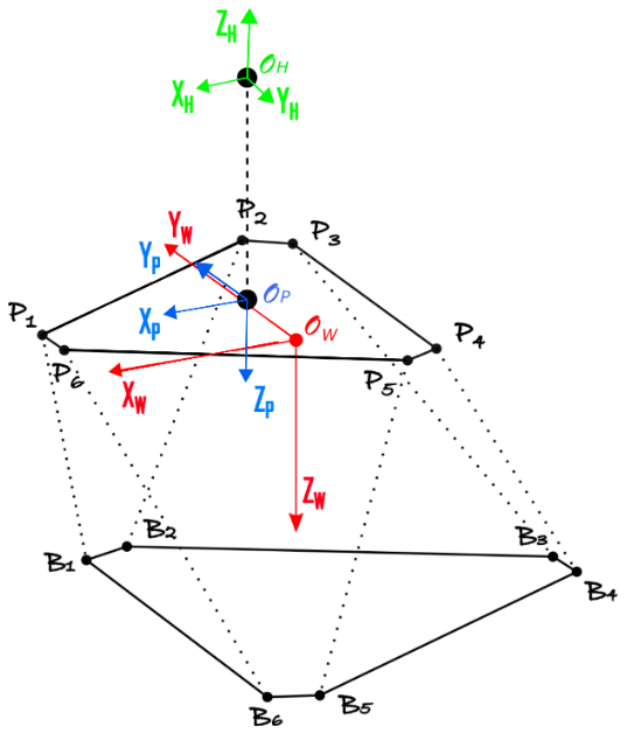

Figure 4.1: Hexapod

Tab.(4.1) reports the coordinates of the legs attachment points, that define the hexapod geometry. The coordinates of the attachment points on the base are expressed in World frame, whereas the ones on the top disk are expressed in Platform frame.
The workspace of the hexapod platform is defined by constraints on legs lengths, velocities and accelerations, whose limits, together with legs lengths in neutral position, are reported in Tab.(4.2).

4.2 Kinematic model

The relationship between input, state, and output that rules the simulator motion is here described.
The state $\mathbf{x}_{hex}$ consists of the pose of the hexapod platform $\mathbf{p} \in \mathbb{R}^{N_p}$, $N_p = 6$, and its first derivatives $\mathbf{v} \in \mathbb{R}^{N_p}$:

$$\mathbf{x}_{hex} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_{lin}^{W} \\ \mathbf{p}_{eul} \\ \mathbf{v}_{lin}^{W} \\ \mathbf{v}_{eul} \end{bmatrix} \tag{4.1}$$

where $\mathbf{p}_{lin}^{W} \in \mathbb{R}^3$ contains the coordinates of the linear position expressed in the World frame, and $\mathbf{p}_{eul} \in \mathbb{R}^3$ includes roll, pitch, yaw Euler angles, defined as Tait-Bryan angles, with the

Corners	x [m]	y [m]	z [m]
B_1^W	0.57	0.99	1.30
B_2^W	0.36	1.06	1.30
B_3^W	-1.34	0.11	1.30
B_4^W	-1.34	-0.11	1.30
B_5^W	0.36	-1.06	1.30
B_6^W	0.57	-0.99	1.30
P_1^P	0.84	0.11	0
P_2^P	-0.54	0.91	0
P_3^P	-0.74	0.79	0
P_4^P	-0.74	-0.79	0
P_5^P	-0.54	-0.91	0
P_6^P	0.84	-0.11	0

Table 4.1: Coordinates of legs attachment points

	min	max	neutral
p_l [m]	1.284	1.934	1.593
v_l [m/s]	-0.465	0.465	0
a_l $[m/s^2]$	-8.6	8.6	0

Table 4.2: Limits on leg length p_l, velocity v_l and acceleration a_l

sequence Z, Y', X" and the convention of intrinsic rotations. $\mathbf{v}_{lin}^{W} = \dot{\mathbf{p}}_{lin}^{W}$ represents the linear velocity in World frame, and $\mathbf{v}_{eul} = \dot{\mathbf{p}}_{eul}$ is the first derivative of Euler angles.

The hexapod input includes the second derivatives of the pose of the top platform:

$$\mathbf{u}_{hex} = \begin{bmatrix} \mathbf{a}_{lin}^{W} \\ \mathbf{a}_{eul} \end{bmatrix} \tag{4.2}$$

where $\mathbf{a}_{lin}^{W} = \dot{\mathbf{v}}_{lin}^{W}$ and $\mathbf{a}_{eul} = \dot{\mathbf{v}}_{eul}$.
By the combination of Eq. (4.1) and Eq. (4.2), the hexapod kinematic model can be expressed as:

$$\dot{\mathbf{x}}_{hex} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{x}_{hex} + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{u}_{hex} \tag{4.3}$$

where $\mathbf{0}, \mathbf{I} \in \mathbb{R}^{N_p \times N_p}$ are null and identity matrices, respectively.

Using the components of $\mathbf{p}_{eul}$, and indicating with α, β, and θ the roll, pitch and yaw angle, respectively, the rotation matrix from Platform to World frame can be computed:

$${}^{W}R_P = {}^{P}R_W^T = \begin{bmatrix} c_\beta\, c_\theta & c_\beta\, s_\theta & -s_\beta \\ s_\alpha\, s_\beta\, c_\theta - c_\alpha\, s_\theta & c_\alpha\, c_\theta + s_\alpha\, s_\beta\, s_\theta & c_\beta\, s_\alpha \\ s_\alpha\, s_\theta + c_\alpha\, s_\beta\, c_\theta & c_\alpha\, s_\beta\, s_\theta - s_\alpha\, c_\theta & c_\alpha\, c_\beta \end{bmatrix} \tag{4.4}$$

For easiness of notation, the time dependence of the angles is omitted.
The first and second derivatives, which will be useful later, are now calculated, obtaining:

$${}^{W}\dot{R}_P = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{4.5}$$

where

$$\begin{aligned}
a_{11} &= -s_\beta\, c_\theta\, \dot\beta - c_\beta\, s_\theta\, \dot\theta \\
a_{12} &= c_\beta\, c_\theta\, \dot\theta - s_\beta\, s_\theta\, \dot\beta \\
a_{13} &= -c_\beta\, \dot\beta \\
a_{21} &= s_\alpha\, s_\theta\, \dot\alpha - c_\alpha\, c_\theta\, \dot\theta + c_\alpha\, s_\beta\, c_\theta\, \dot\alpha + c_\beta\, s_\alpha\, c_\theta\, \dot\beta - s_\alpha\, s_\beta\, s_\theta\, \dot\theta \\
a_{22} &= c_\alpha\, s_\beta\, s_\theta\, \dot\alpha - c_\alpha\, s_\theta\, \dot\theta - s_\alpha\, c_\theta\, \dot\alpha + c_\beta\, s_\alpha\, s_\theta\, \dot\beta + s_\alpha\, s_\beta\, c_\theta\, \dot\theta \\
a_{23} &= c_\alpha\, c_\beta\, \dot\alpha - s_\alpha\, s_\beta\, \dot\beta \\
a_{31} &= c_\alpha\, s_\theta\, \dot\alpha + s_\alpha\, c_\theta\, \dot\theta + c_\alpha\, c_\beta\, c_\theta\, \dot\beta - s_\alpha\, s_\beta\, c_\theta\, \dot\alpha - c_\alpha\, s_\beta\, s_\theta\, \dot\theta \\
a_{32} &= s_\alpha\, s_\theta\, \dot\theta - c_\alpha\, c_\theta\, \dot\alpha + c_\alpha\, c_\beta\, s_\theta\, \dot\beta + c_\alpha\, s_\beta\, c_\theta\, \dot\theta - s_\alpha\, s_\beta\, s_\theta\, \dot\alpha \\
a_{33} &= -c_\beta\, s_\alpha\, \dot\alpha - c_\alpha\, s_\beta\, \dot\beta
\end{aligned}$$

and

$$ {}^{W}\ddot{R}_P = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \tag{4.6}$$

where

$$\begin{aligned}
b_{11} &= 2\, s_\beta\, s_\theta\, \dot\theta\, \dot\beta - c_\beta\, c_\theta\, \dot\theta^2 - s_\beta\, c_\theta\, \ddot\beta - c_\beta\, s_\theta\, \ddot\theta - c_\beta\, c_\theta\, \dot\beta^2 \\
b_{12} &= c_\beta\, c_\theta\, \ddot\theta - c_\beta\, s_\theta\, \dot\theta^2 - c_\beta\, s_\theta\, \dot\beta^2 - s_\beta\, s_\theta\, \ddot\beta - 2\, s_\beta\, c_\theta\, \dot\theta\, \dot\beta \\
b_{13} &= s_\beta\, \dot\beta^2 - c_\beta\, \ddot\beta \\
b_{21} &= c_\alpha\, s_\theta\, \dot\alpha^2 + c_\alpha\, s_\theta\, \dot\theta^2 - c_\alpha\, c_\theta\, \ddot\theta + s_\alpha\, s_\theta\, \ddot\alpha - s_\alpha\, s_\beta\, s_\theta\, \ddot\theta - s_\alpha\, s_\beta\, c_\theta\, \dot\alpha^2 - s_\alpha\, s_\beta\, c_\theta\, \dot\beta^2 + c_\alpha\, s_\beta\, c_\theta\, \ddot\alpha + \\
&\quad + c_\beta\, s_\alpha\, c_\theta\, \ddot\beta - s_\alpha\, s_\beta\, c_\theta\, \dot\theta^2 + 2\, s_\alpha\, c_\theta\, \dot\theta\, \dot\alpha - 2\, c_\alpha\, s_\beta\, s_\theta\, \dot\theta\, \dot\alpha - 2\, c_\beta\, s_\alpha\, s_\theta\, \dot\theta\, \dot\beta + 2\, c_\alpha\, c_\beta\, c_\theta\, \dot\beta\, \dot\alpha \\
b_{22} &= c_\alpha\, s_\beta\, s_\theta\, \ddot\alpha - c_\alpha\, c_\theta\, \dot\theta^2 - s_\alpha\, c_\theta\, \ddot\alpha - c_\alpha\, s_\theta\, \ddot\theta - s_\alpha\, s_\beta\, s_\theta\, \dot\alpha^2 - s_\alpha\, s_\beta\, s_\theta\, \dot\beta^2 - c_\alpha\, c_\theta\, \dot\alpha^2 + c_\beta\, s_\alpha\, s_\theta\, \ddot\beta + \\
&\quad - s_\alpha\, s_\beta\, s_\theta\, \dot\theta^2 + s_\alpha\, s_\beta\, c_\theta\, \ddot\theta + 2\, s_\alpha\, s_\theta\, \dot\theta\, \dot\alpha + 2\, c_\alpha\, c_\beta\, s_\theta\, \dot\beta\, \dot\alpha + 2\, c_\alpha\, s_\beta\, c_\theta\, \dot\theta\, \dot\alpha + 2\, c_\beta\, s_\alpha\, c_\theta\, \dot\theta\, \dot\beta \\
b_{23} &= c_\alpha\, c_\beta\, \ddot\alpha - c_\beta\, s_\alpha\, \dot\beta^2 - c_\beta\, s_\alpha\, \dot\alpha^2 - s_\alpha\, s_\beta\, \ddot\beta - 2\, c_\alpha\, s_\beta\, \dot\beta\, \dot\alpha \\
b_{31} &= c_\alpha\, s_\theta\, \ddot\alpha - s_\alpha\, s_\theta\, \dot\alpha^2 - s_\alpha\, s_\theta\, \dot\theta^2 + s_\alpha\, c_\theta\, \ddot\theta - c_\alpha\, s_\beta\, c_\theta\, \dot\alpha^2 - c_\alpha\, s_\beta\, c_\theta\, \dot\beta^2 + c_\alpha\, c_\beta\, c_\theta\, \ddot\beta - c_\alpha\, s_\beta\, c_\theta\, \dot\theta^2 + \\
&\quad + 2\, c_\alpha\, c_\theta\, \dot\theta\, \dot\alpha - s_\alpha\, s_\beta\, c_\theta\, \ddot\alpha - c_\alpha\, s_\beta\, s_\theta\, \ddot\theta + 2\, s_\alpha\, s_\beta\, s_\theta\, \dot\theta\, \dot\alpha - 2\, c_\beta\, s_\alpha\, c_\theta\, \dot\beta\, \dot\alpha - 2\, c_\alpha\, c_\beta\, s_\theta\, \dot\theta\, \dot\beta \\
b_{32} &= s_\alpha\, c_\theta\, \dot\alpha^2 - c_\alpha\, c_\theta\, \ddot\alpha + s_\alpha\, c_\theta\, \dot\theta^2 + s_\alpha\, s_\theta\, \ddot\theta - c_\alpha\, s_\beta\, s_\theta\, \dot\alpha^2 - c_\alpha\, s_\beta\, s_\theta\, \dot\beta^2 + c_\alpha\, c_\beta\, s_\theta\, \ddot\beta - c_\alpha\, s_\beta\, s_\theta\, \dot\theta^2 + \\
&\quad + c_\alpha\, s_\beta\, c_\theta\, \ddot\theta + 2\, c_\alpha\, s_\theta\, \dot\theta\, \dot\alpha - s_\alpha\, s_\beta\, s_\theta\, \ddot\alpha - 2\, s_\alpha\, s_\beta\, c_\theta\, \dot\theta\, \dot\alpha + 2\, c_\alpha\, c_\beta\, c_\theta\, \dot\theta\, \dot\beta - 2\, c_\beta\, s_\alpha\, s_\theta\, \dot\beta\, \dot\alpha \\
b_{33} &= 2\, s_\alpha\, s_\beta\, \dot\beta\, \dot\alpha - c_\alpha\, c_\beta\, \dot\beta^2 - c_\beta\, s_\alpha\, \ddot\alpha - c_\alpha\, s_\beta\, \ddot\beta - c_\alpha\, c_\beta\, \dot\alpha^2
\end{aligned}$$

The hexapod output includes the signals that the driving simulator aims to reproduce: the specific force $\mathbf{f}$, which will be referred to as *linear acceleration*, and the angular velocity $\boldsymbol{\omega}$, both at the center of the driver's head and expressed in the Head frame.

The linear acceleration is a function of $\mathbf{x}_{hex}$ and $\mathbf{u}_{hex}$ and can be expressed as:

$$\mathbf{f}^H(\mathbf{x}_{hex}, \mathbf{u}_{hex}) = {}^H R_W(\mathbf{x}_{hex})\mathbf{g}^W + {}^H\ddot{\mathbf{r}}_H(\mathbf{x}_{hex}, \mathbf{u}_{hex}) \tag{4.7}$$

where $\mathbf{g}^W$ is the gravitational acceleration expressed in the World frame, and ${}^H R_W$ is the rotational matrix from World to Head frame, derived by:

$${}^H R_W = {}^H R_P \cdot {}^P R_W.$$

The integration of the gravity effect inside the model allows to automatically achieve the tilt coordination correction, giving an important contribution to the tracking of the simulation signals.
To compute ${}^H\ddot{\mathbf{r}}_H$, which is the linear acceleration of the Head frame with respect to the World frame, expressed in Head frame, the second derivative of the position of the center of driver head expressed in the World frame ${}^W\mathbf{r}_H$ is calculated, and then it is rotated by the matrix ${}^H R_W$:

$$\begin{aligned}
{}^W\mathbf{r}_H &= {}^W\mathbf{r}_P + {}^W R_P {}^P\mathbf{r}_H \\
{}^W\dot{\mathbf{r}}_H &= {}^W\dot{\mathbf{r}}_P + {}^W \dot{R}_P {}^P\mathbf{r}_H + {}^W R_P {}^P\dot{\mathbf{r}}_H \\
{}^W\ddot{\mathbf{r}}_H &= {}^W\ddot{\mathbf{r}}_P + {}^W \ddot{R}_P {}^P\mathbf{r}_H + 2{}^W \dot{R}_P {}^P\dot{\mathbf{r}}_H + {}^W R_P {}^P\ddot{\mathbf{r}}_H
\end{aligned}$$

Since it is assumed that position and orientation of the driver's head with respect to the Platform frame are constant during the motion, and noticing that ${}^W\ddot{\mathbf{r}}_P = \mathbf{a}_{lin}$, it results:

$${}^W\ddot{\mathbf{r}}_H = \mathbf{a}_{lin} + {}^W \ddot{R}_P {}^P\mathbf{r}_H \tag{4.8}$$

$${}^H\ddot{\mathbf{r}}_H = {}^H R_W {}^W\ddot{\mathbf{r}}_H \tag{4.9}$$

The angular velocity of the Head frame with respect to the World frame, expressed in Head frame, can be obtained by the components of the skew-symmetric matrix resulting from the following:

$$[\boldsymbol{\omega}^H(\mathbf{x}_{hex})]_\times = {}^H R_W(\mathbf{x}_{hex}) {}^W \dot{R}_H(\mathbf{x}_{hex}) \tag{4.10}$$

$$= \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{4.11}$$

where ${}^W \dot{R}_H$ is given by

$${}^W \dot{R}_H = {}^W \dot{R}_P {}^P R_H$$

To conclude, the output can be written as function of $\mathbf{x}_{hex}$ and $\mathbf{u}_{hex}$:

$$\mathbf{y}(\mathbf{x}_{hex}, \mathbf{u}_{hex}) = \begin{bmatrix} \mathbf{f}^H(\mathbf{x}_{hex}, \mathbf{u}_{hex}) \\ \boldsymbol{\omega}^H(\mathbf{x}_{hex}) \end{bmatrix}. \tag{4.12}$$

Kinematic constraints on legs lengths, velocities and accelerations define the limits of the hexapod workspace. The length ℓ of the *i*-th leg is a function of state and can be written as:

$$\begin{aligned} \ell_i(\mathbf{x}_{hex}) &= \|\mathcal{P}_i^W(\mathbf{x}_{hex} - \mathcal{B}_i^W)\| & (4.13) \\ &= \|{}^W R_P(\mathbf{x}_{hex})\mathcal{P}_i^P + {}^W\mathbf{r}_P(\mathbf{x}_{hex}) - \mathcal{B}_i^W\| & (4.14) \end{aligned}$$

where ${}^W\mathbf{r}_P$ is the position of the origin of the Platform frame expressed in World frame, i.e. $\mathbf{p}_{lin}^W$.
Leg velocities $\dot{\boldsymbol{\ell}}$ and accelerations $\ddot{\boldsymbol{\ell}}$ are the first and second derivatives, respectively, of Eq. (4.14). Referring to the argument of the previous norm with $h\ell_i$

$$h\ell_i = {}^W R_P(\mathbf{x}_{hex})\mathcal{P}_i^P + {}^W\mathbf{r}_P(\mathbf{x}_{hex}) - \mathcal{B}_i^W \tag{4.15}$$

the following equality holds:

$$\ell_i(\mathbf{x}_{hex}) = [(h\ell_i)^T(h\ell_i)]^{\frac{1}{2}} \tag{4.16}$$

obtaining

$$\begin{aligned} \dot{\ell}_i(\mathbf{x}_{hex}) &= \frac{1}{2}[(h\ell_i)^T(h\ell_i)]^{-\frac{1}{2}}[(h\dot{\ell}_i)^T(h\ell_i) + (h\ell_i)^T(h\dot{\ell}_i)] \\ &= \frac{(h\ell_i)^T(h\dot{\ell}_i)}{\ell_i} \end{aligned} \tag{4.17}$$

$$\ddot{\ell}_i(\mathbf{x}_{hex}, \mathbf{u}_{hex}) = \frac{[(h\ell_i)^T(h\ddot{\ell}_i) + (h\dot{\ell}_i)^T(h\dot{\ell}_i)]\ell_i - \dot{\ell}_i(h\ell_i)^T(h\dot{\ell}_i)}{\ell_i^2}. \tag{4.18}$$

By deriving the inverse kinematics $\mathbf{k}$:

$$\begin{bmatrix} \boldsymbol{\ell}(\mathbf{x}_{hex}) \\ \dot{\boldsymbol{\ell}}(\mathbf{x}_{hex}) \\ \ddot{\boldsymbol{\ell}}(\mathbf{x}_{hex}, \mathbf{u}_{hex}) \end{bmatrix} = \mathbf{k}(\mathbf{x}_{hex}, \mathbf{u}_{hex}), \tag{4.19}$$

leg lower and upper bounds ($\underline{\boldsymbol{\ell}}, \overline{\boldsymbol{\ell}}$) can be expressed as follows:

$$\begin{bmatrix} \underline{\boldsymbol{\ell}} \\ \underline{\dot{\boldsymbol{\ell}}} \\ \underline{\ddot{\boldsymbol{\ell}}} \end{bmatrix} \leq \mathbf{k}(\mathbf{x}_{hex}, \mathbf{u}_{hex}) \leq \begin{bmatrix} \overline{\boldsymbol{\ell}} \\ \overline{\dot{\boldsymbol{\ell}}} \\ \overline{\ddot{\boldsymbol{\ell}}} \end{bmatrix}. \tag{4.20}$$

5 Implementation using MATMPC

The Model Predictive Control process, previously described in chapter 3, has been implemented in MATLAB, using MATMPC. This tool, developed by Yutao, aims at providing an easy-to-use nonlinear MPC implementation. Its most important characteristic is that it does not require the user to understand how to make or compile libraries, since all the algorithmic routiness are written in MATLAB and can be compiled into MEX functions. For a more detailed descriptions of MATMPC, visit [13]. In the following sections, the choices adopted in the MPC process implementation, concerning the constraints relaxation, the cost function creation, and the tuning phase will be presented.

5.1 Constraints

As mentioned before, one of the main advantages of MPC is the possibility of explicitly declare the constraints to which the platform is forced. Two types of constraints can be considered: *hard*, which set conditions on variables which under no circumstances can be violated, or *soft*, to which some variable values are associated, that are penalized in the objective function if the conditions are not satisfied.

In this thesis, it has been chosen to use soft constraints in order to examine the system behaviour in conditions where the platform gets close to the limits or violates them, overcoming the fact that, by adopting hard constraints, in the same situations the problem becomes infeasible. Consider, for example, the hard constraints related to legs lengths

$$\underline{\boldsymbol{\ell}} \leq \boldsymbol{\ell}(\mathbf{x}_{hex}) \leq \overline{\boldsymbol{\ell}}.$$

The implementation as soft constraints adding a slack variables $\zeta_l > 0$ results in

$$\underline{\boldsymbol{\ell}} - \boldsymbol{\zeta}_l \leq \boldsymbol{\ell}(\mathbf{x}_{hex}) \leq \overline{\boldsymbol{\ell}} + \boldsymbol{\zeta}_l$$

or, allowing ζ_l to assume any positive or negative value, in the adopted form

$$\underline{\boldsymbol{\ell}} \leq \boldsymbol{\ell}(\mathbf{x}_{hex}) + \boldsymbol{\zeta}_l \leq \overline{\boldsymbol{\ell}}.$$

Introducing slack variables ζ_l, ζ_v, ζ_a, associated to legs lengths, velocities and accelerations respectively, the considered constraints are the following:

$$\underline{\boldsymbol{\ell}} \leq \quad \boldsymbol{\ell}(\mathbf{x}_{hex}) + \boldsymbol{\zeta}_l \quad \leq \overline{\boldsymbol{\ell}} \tag{5.1}$$

$$\underline{\dot{\boldsymbol{\ell}}} \leq \quad \dot{\boldsymbol{\ell}}(\mathbf{x}_{hex}) + \boldsymbol{\zeta}_v \quad \leq \overline{\dot{\boldsymbol{\ell}}} \tag{5.2}$$

$$\underline{\ddot{\boldsymbol{\ell}}} \leq \ddot{\boldsymbol{\ell}}(\mathbf{x}_{hex}, \mathbf{u}_{hex}) + \boldsymbol{\zeta}_a \leq \overline{\ddot{\boldsymbol{\ell}}} \tag{5.3}$$

which have to be added to the cost function using the quadratic formulation.

5.2 Cost function

Motion Cueing algorithms based on Model Predictive Control techniques aim at compute the optimal control input for the system by minimizing a cost function.
In this specific case, the cost function to be minimized has been chosen as:

$$J = \frac{1}{2}\left[\sum_{k=1}^{N_p} ||\mathbf{y}(\mathbf{x}_k, \mathbf{u}_k) - \hat{\mathbf{y}}_k||^2_{W_\mathrm{y}} + \sum_{k=0}^{N_p-1} ||\mathbf{x}_k - \hat{\mathbf{x}}_k||^2_{W_\mathrm{x}} + ||\mathbf{u}_k - \hat{\mathbf{u}}_k||_{W_\mathrm{u}} + ||\boldsymbol{\zeta}_k - \hat{\boldsymbol{\zeta}}_k||_{W_\zeta}\right] \tag{5.4}$$

where hats refer to reference signals to be tracked. It is important to underline the meaning of every variable composing (5.4) :

- $\hat{\mathbf{y}}$ represents the output signals trajectories to be tracked, i.e. the linear accelerations and the angular velocities that the driver would perceive at the center of his head in a real car;

- $\hat{\mathbf{x}}$ plays the role of a washout filter. In fact, valuable motion cueing algorithms have to consider a washout action to cope with the limited platform workspace. MPC includes an alternative to the classical, filter based procedure by setting constant zero reference signals for the position and velocities of the center of the hexapod top disk $\mathcal{O}_P$. This realization of washout is easy to handle and to modify, allowing a considerable exploitation of the platform movements.

- $\hat{\mathbf{u}}$ equal to the zero vector aims to minimize the input signals, i.e. the accelerations of the hexapod top disk, in order to provide sustainable control signals from the actuators.

- $\boldsymbol{\zeta} = [\zeta_l \quad \zeta_v \quad \zeta_a]^T$ is the vector containing the slack variables. The reference signal $\hat{\boldsymbol{\zeta}}$ is constant and equal to $[0 \quad 0 \quad 0]^T$, since slack variables have to be as smaller as possible to ensure the platform does not violate its limits.

- W_y, W_x, W_u, W_ζ are symmetric positive-definite weighting matrices. The weights represent the tuning parameters of the motion cueing algorithm and are manipulated to reach the desired performance in the reproduction of the accelerations and velocities. Their main advantage is the fact that they are associated to physical quantities of the system, and so are easy to interpret also by non-expert users.

6 Results

In this chapter, experimental results will be presented, with the aim to verify the main features of the motion cueing algorithm based on MPC. Later, considerations about computational time will be made, and different receding horizon strategies will be discussed.
In all the following simulations, the reference signal of the linear output acceleration along the x axis is given by the second derivative of a sine wave, with magnitude equal to 16 m and frequency equal to $\frac{1}{16}$Hz.
Simulations are run on PC Intel(R) Core(TM) i7 -7700HQ @ 2.80GHz CPU.

6.1 Importance of the prediction horizon length to stabilize the algorithm

The length of the prediction window directly affects the control performance, so N_p becomes a tunable parameter that can be varied in order to obtain the desired tracking performance.
At the end of this chapter, a discussion about the role of the prediction horizon length to best exploit the available workspace will be made. In the following, N_p is treated as a tunable parameter that affects the stability of the MPC. Since this algorithm suffers of instability issues, it is necessary to choose a minimum value of N_p which stabilizes the problem.
To illustrate how the value of N_p affects the control performance, two simulations are made, using for the first one $N_p = 20$, and for the second one $N_p = 40$. The weighting matrices are set equal to:

$$W_y = diag\left(\begin{bmatrix}10 & 10 & 10 & 10 & 10 & 10\end{bmatrix}\right) \tag{6.1}$$

$$W_x = diag\left(\begin{bmatrix}0.01 & 0.01 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 1 & 1 & 1\end{bmatrix}\right) \tag{6.2}$$

$$W_u = diag\left(\begin{bmatrix}0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001\end{bmatrix}\right) \tag{6.3}$$

$$W_\zeta = diag\left(\begin{bmatrix}1000 & 1000 & 1000\end{bmatrix}\right) \tag{6.4}$$

where $diag(\text{vector})$ creates a square matrix with the elements of the vector on the main diagonal. For reasons of convenience, this set of matrices is referred with W_1.
The comparison between the resulting linear acceleration at the center of the driver head, performed along the x axis, in the case with $N_p = 20$ and in the case between $N_p = 40$, is shown in Fig. 6.1. Using $N_p = 40$, the system output f_x perfectly tracks the reference signal, and in Fig.6.2, the linear displacement of the center of the top disk from its neutral position can be seen.

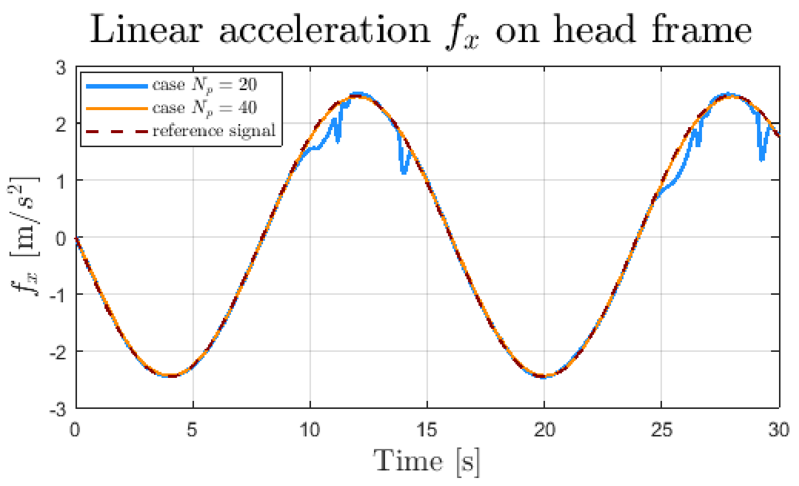

Figure 6.1: Comparison between the output tracking f_x in the case with $N_p = 20$ and in the case with $N_p = 40$. The set of matrices W_1 is used.

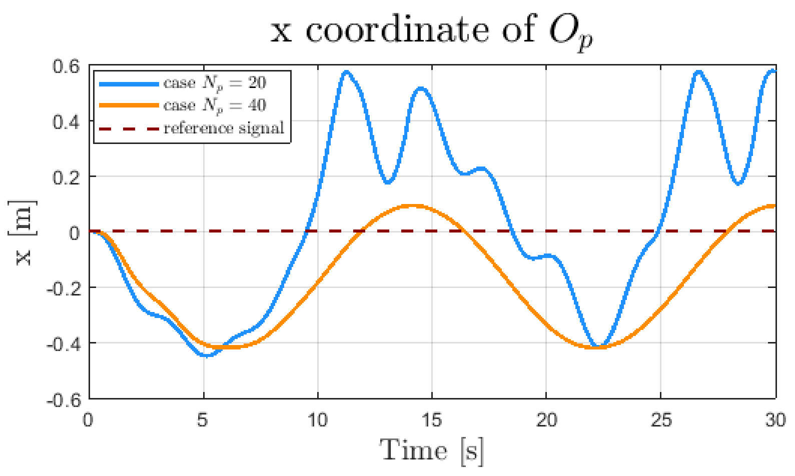

Figure 6.2: Comparison between the linear displacement along the x axis of O_p in the case with $N_p = 20$ and in the case with $N_p = 40$. The set of matrices W_1 is used.

O_p draws a smooth trajectory, without sudden movements. On the contrary, the output resulting from the test with $N_p = 20$ diverges from the reference signal on several points. This is due to the fact that, how can be observed in Fig.6.2, when actuators overcome their limits, the platform is forced to unexpected and sharp maneuvers to get away from the limits and move towards the neutral position.

Fig.6.3 and Fig.6.4 compare slack variables in the two examined cases. The plot shows the behaviour of the hexapod legs: as can be seen, using $N_p = 20$ both constraints on legs lengths and velocities are violated, denoting the instability of the algorithm. In the case with $N_p = 40$, instead, thanks to the wider prediction horizon, all the slack variables are equal to zero, i.e. no constraints violation occurs.

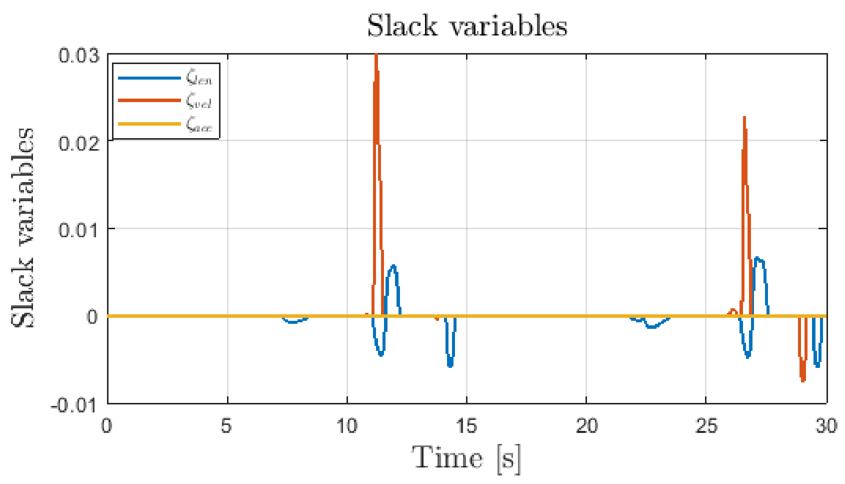

Figure 6.3: Slack variables on constraints, case with W_1 and $N_p = 20$

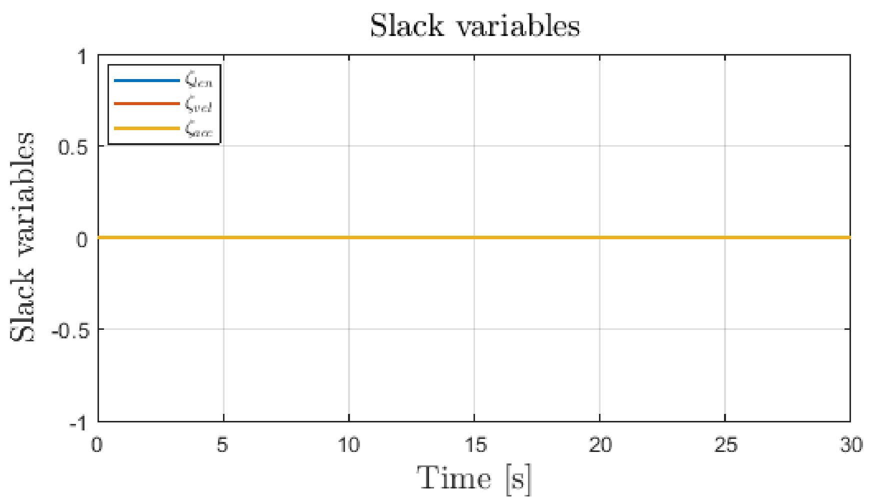

Figure 6.4: Slack variables on constraints, case with W_1 and $N_p = 40$

This fact is confirmed by Fig.6.5 and Fig.6.6, which depict the extension and the reduction of the hexapod legs lengths. In the first case, the trend is unruly and unpredictable, and actuators saturate at the time when slack variables associated to legs lengths are different from zero, while in the second case the trend is almost sinusoidal and symmetric, denoting a smoother motion within the limits.

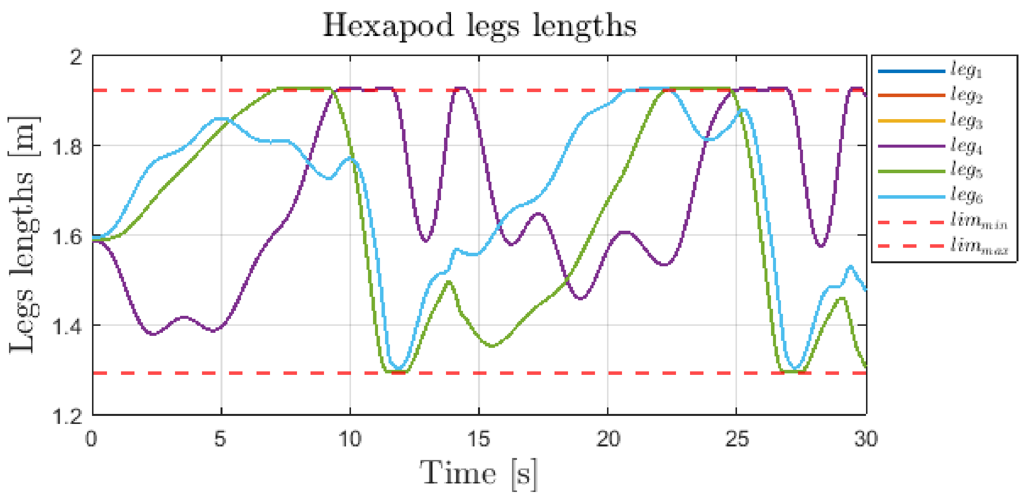

Figure 6.5: Hexapod legs lengths, case with W_1 and $N_p = 20$

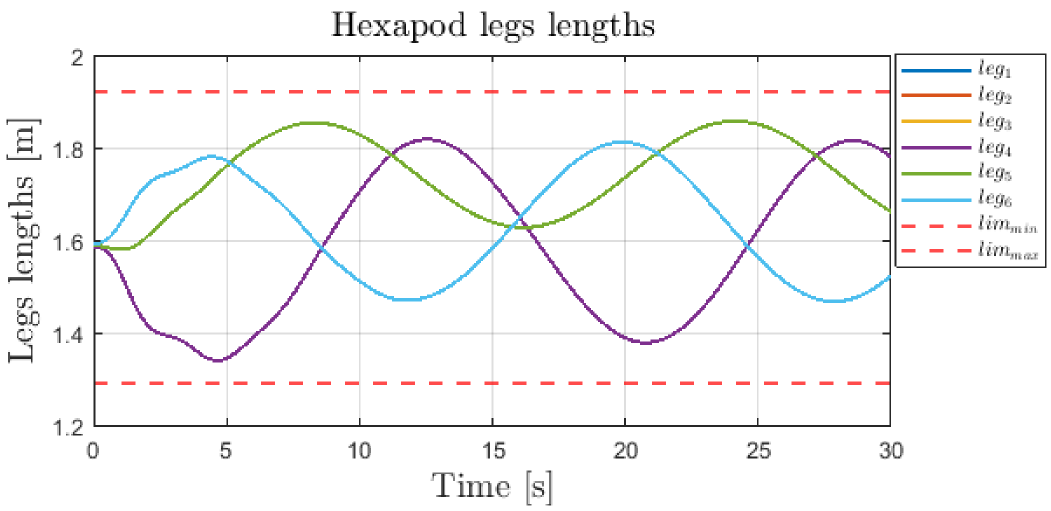

Figure 6.6: Hexapod legs lengths, case with W_1 and $N_p = 40$

From now on, the length of the prediction window will be kept equal to 40.

6.2 Washout and tilt coordination effects

To provide a demonstration of the *washout* and *tilt coordination* effects, the previous case with W_1 is compared with an example in which a new set of matrices, named W_2, is used. It is given by:

$$W_{\mathrm{y}} = diag\left(\begin{bmatrix}10 & 10 & 10 & 10 & 10 & 10\end{bmatrix}\right) \tag{6.5}$$

$$W_{\mathrm{x}} = diag\left(\begin{bmatrix}10 & 10 & 100 & 0.1 & 0.1 & 0.1 & 10 & 10 & 10 & 0.1 & 0.1 & 0.1\end{bmatrix}\right) \tag{6.6}$$

$$W_{\mathrm{u}} = diag\left(\begin{bmatrix}0.001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001\end{bmatrix}\right) \tag{6.7}$$

$$W_{\zeta} = diag\left(\begin{bmatrix}1000 & 1000 & 1000\end{bmatrix}\right) \tag{6.8}$$

Here, the weights associated to the linear components of the platform pose have been increased, while those relative to the angular components of the platform pose and of the platform velocity have been decreased. The purpose is to force the platform to stay close to its neutral position, exploiting the gravity effects to replicate the desired outputs by tilting the top disk.
As can be seen observing Fig.6.7, the resulting output linear accelerations along the x axis are basically the same in the case with W_1 and in the case with W_2. The fundamental difference is the way in which the outputs are produced.

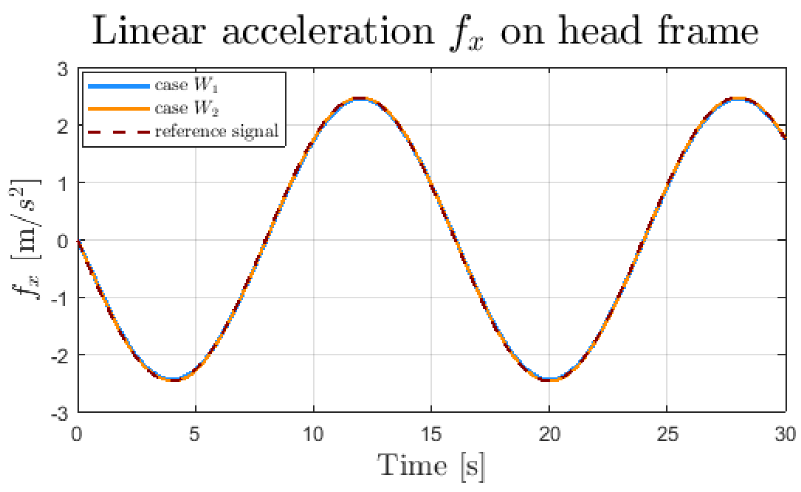

Figure 6.7: Comparison between the output tracking f_x in the case with W_1 and in the case with W_2. N_p = 40 is used.

Comparing the linear displacement along the x axis of O_p, depicted in Fig.6.8, it is clear how in the case with W_2 it is much lower than in the case with W_1. This comparison clearly shows the washout effect: using the set of weighting matrices W_2, the platform is repeatedly brought back to its equilibrium position and all the motions are performed maintaining the platform close to the origin, thus better exploiting the available workspace.

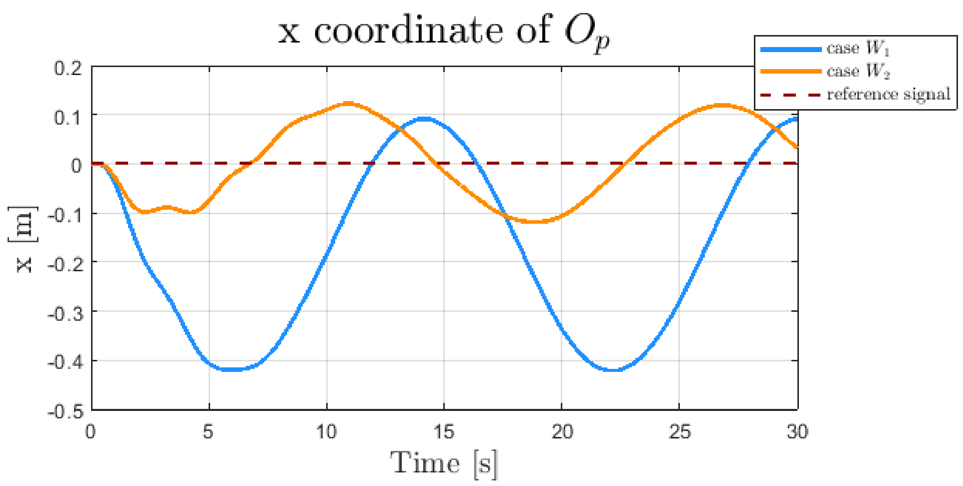

Figure 6.8: Comparison between the linear displacement along the x axis of O_p in the case with W_1 and in the case with W_2. $N_p = 40$ is used.

The difference between the pitch angle shown in Fig.6.9 is not very noticeable and, for the sake of clarity, the precise values are reported in Tab.(6.1). The two signals are almost the same because the adopted reference signal implies a considerable linear acceleration. To track the output signal, in both cases, the simulator is required to utilize the pitch angle, otherwise, using the only linear displacement, it would overcome the limits.

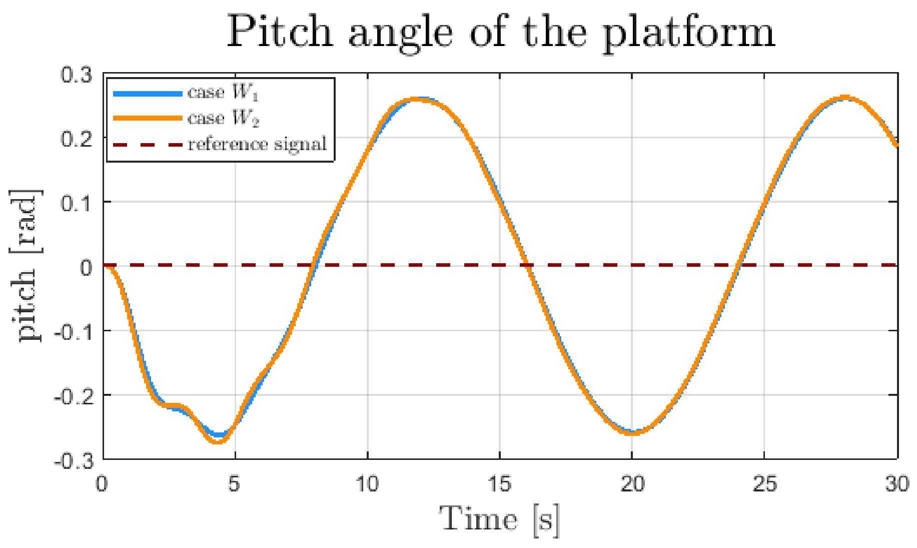

Figure 6.9: Comparison between the platform pitch angle in the case with W_1 and in the case with W_2. $N_p = 40$ is used.

In conclusion, Fig.6.10 proves that the legs extension is lower than the case given by W_1.

	Case with set W_1	Case with set W_2
min value of x_{O_p} [m]	-0.4217	-0.1188
max value of x_{O_p} [m]	0.0916	0.1227
min value of pitch$_{O_p}$ [rad]	-0.2644	-0.2761
max value of pitch$_{O_p}$ [rad]	0.2595	0.2607

Table 6.1: Comparison between the values of the x-coordinate and pitch angle of the center of the top disk O_p when using the set of matrices W_1 and W_2.

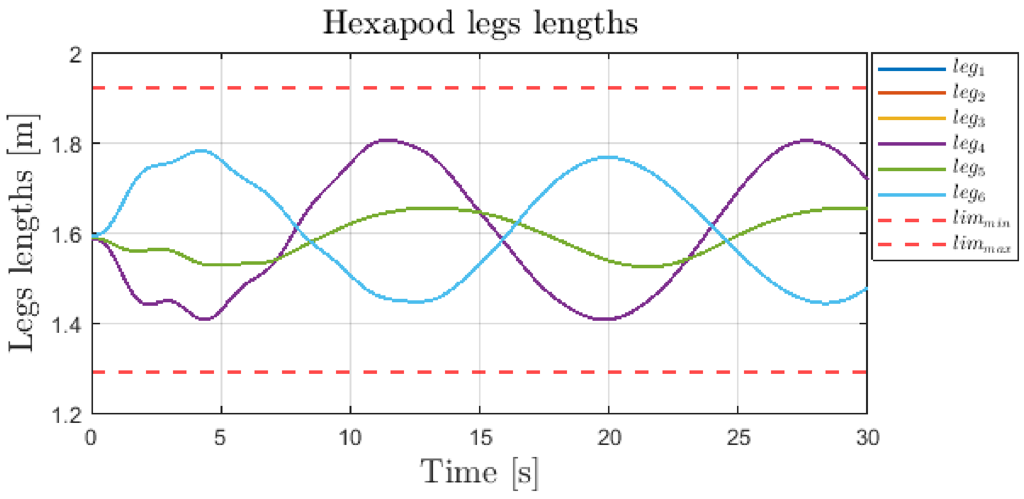

Figure 6.10: Hexapod legs lengths, case with W_2 and $N = 40$

Examining the above simulations, the main features of a motion cueing algorithm have been tested.

Now, to improve the algorithm performances and use it in real-time applications, considerations about computational time will be made.

6.3 Computational time

Real-time implementation is a non-trivial task, since it implies fast dynamics which requires small computation time. Hence, first of all it is desirable to investigate how the explicitly declared constraints affect resolution time.
Using $N_p = 40$ and the set of matrices W_1, three different scenarios are considered, whose related average computation time are reported in Tab.6.2. In the first case, constraints on legs lengths, velocities and accelerations are considered, in the second case constraints on legs lengths and velocities are taken into account, while in the third one only constraints on legs lengths are assumed. The substantial difference is due to the high complexity of the constraints mathematical expression, which increase the resolution time. Once an appropriate value of N_p and a good set of matrices are chosen, it has been seen that the most restrictive constraints are those relative

to legs lengths, while the others are largely avoided. Hence, constraints on legs velocities and accelerations will be omitted in the following simulations, thus decreasing the computation time.

	Average computational time [ms]
Constraints on legs lengths, velocities and accelerations	6.6089
Constraints on legs lengths and velocities	4.4161
Constraints on legs lengths	2.861

Table 6.2: Variation of the average computational time in relation to the declaration of constraints.

The use of valid reference signals, i.e. non constant references, requires a wide prediction horizon, which results in a large number of samples: for examples, a prediction of 10 seconds at 100 Hz means $N_p = 1000$, and the increase of N_p makes the problem very hard to be handled. In conclusion, N_p has to be chosen as small as possible to provide sustainable resolution time, while ensuring the correct exploitation of the workspace. To satisfy this requirement, two different strategies are considered: *move blocking* and *non-uniform grid*.

The main computational effort of NMPC depends on the dimension of the optimal control problem (OCP), that has to be iteratively solved on-line. Since the dimension of the OCP increases proportionally with the length of the prediction horizon, strategies to reduce the number of discretisation nodes, and consequently the dimension of the OCP, are needed.
Move blocking and non-uniform grid are two widely used methods for this purpose. The idea behind the two strategies is to divide the prediction horizon into M non-equidistant shooting intervals, typically more dense at the beginning of the prediction horizon and more sparse at the end of the horizon. Subsequently, in non-uniform grid iteration, the state and control trajectories are parameterised into M intervals, whereas, in move blocking iteration, the state is parameterised into N_p intervals, while control is parameterised into M shooting intervals [14]. These differences result in important aspects, such as:

- MB provides an accurate state trajectory, since for state discretisation it uses the same grid as the uniform grid MPC. However, the calculation of trajectories and constraints on each shooting point comes at the cost of a considerable computational time;
- Non-uniform grid schemes employs a more coarse grid, resulting in less accurate predictive trajectory. This results in a larger possibility of violating constraint, since they are fulfilled only at the shooting nodes. The advantage is that computational time is considerably reduced and longer predictions are allowed.

However, simulations have revealed that, with the increasing of the prediction horizon length, the differences between the two methods are attenuated, i.e. the results regarding the linear acceleration of O_p and the tracking of the output f_x are almost the same, as can be seen in Fig. 6.11 and in Fig. 6.12, respectively.

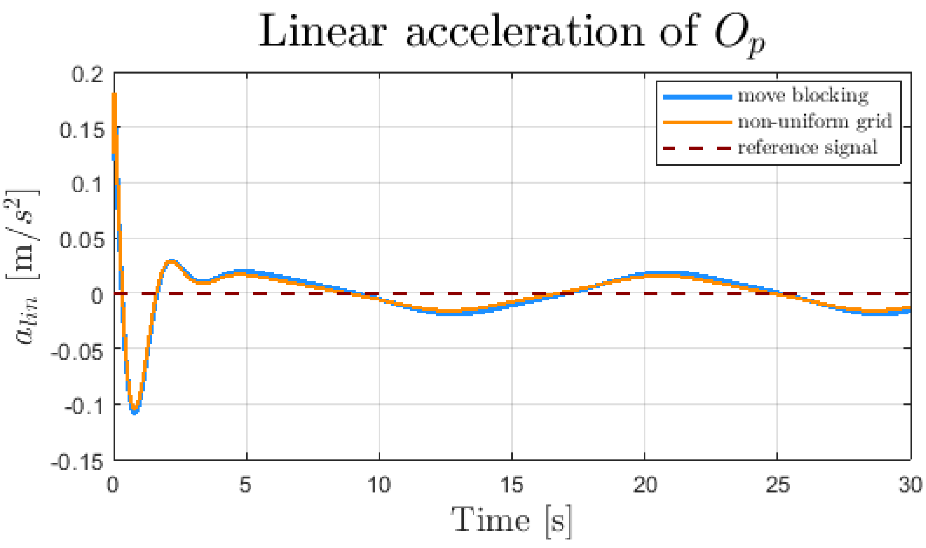

Figure 6.11: Comparison between the linear acceleration along the x axis of O_p when using the move blocking and the non-uniform grid techniques.

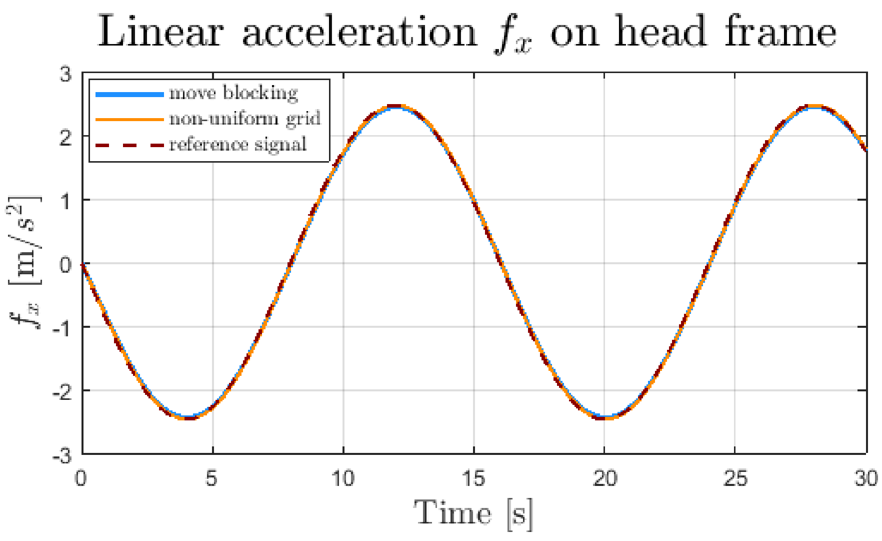

Figure 6.12: Comparison between the output tracking f_x when using the move blocking and the non-uniform grid techniques.

Since the aim of this thesis is to operate with a prediction horizon as wide as possible, move blocking and non-uniform grid strategies are tested using $N_p = 990$ and $M = 100$, which means a prediction horizon of 9.9s. The index used for non-uniform grid and the one of input block for move blocking technique are the same, equal to:

$$\begin{aligned} I =&[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,\\ &25, 27, 30, 34, 38, 43, 48, 54, 60, 67, 74, 82, 90, 99, 108, 118, 128, 139, 150, 162,\\ &174, 186, 199, 212, 226, 240, 255, 270, 285, 300, 315, 330, 345, 360, 375, 390,\\ &405, 420, 435, 450, 465, 480, 495, 510, 525, 540, 555, 570, 585, 600, 615, 630,\\ &645, 660, 675, 690, 705, 720, 735, 750, 765, 780, 795, 810, 825, 840, 855, 870,\\ &885, 900, 915, 930, 945, 960, 975, 990]\end{aligned}$$

	Average computational time [ms]
Uniform grid	18847.8154
Non-uniform grid	60.4733
Move blocking	234.4871

Table 6.3: Average computation time using the three receding horizon schemes.

The sampling in the first 25 steps is equal to 10ms, then linearly increased to 150ms in 25 step, and 150ms for the last 50 steps.
Tab. 6.3 shows the average computation time using the three receding horizon schemes. The enormous difference between the results makes the uniform grid technique unemployable for the purpose. Thanks to the high value of N_p, the performances in the tracking output obtained by the three techniques are almost the same, so it is reasonable to adopt the non-uniform grid schemes, which provides a computation time considerably smaller than the others.
However, all the obtained times are too large for real-time implementations, which require an average computational time of 5ms. To reach a prediction horizon as wide as possible while satisfying real-time requirements, many simulations with different grid have been tested.
Using the uniform grid technique, a prediction horizon of just 0.5 seconds can be obtained. Choosing

$$I =[0, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]$$

as the index of input block for the move blocking technique, a prediction horizon of 1.5 seconds can be reached. Finally, adopting

$$\begin{aligned} I =&[0, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180,\\ &190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 310, 325, 340, 355, 370, 385, 400],\end{aligned}$$

the non-uniform grid allows to exploit a prediction horizon of 4 seconds.

Considering similar computational time, it is now interesting to investigate how the prediction horizon length can affect the algorithm performances, comparing the uniform grid technique with $N_p = 40$ and the non-uniform grid scheme which allows a prediction horizon of 4 seconds. For this purpose, a particular set of matrices W_3 has been chosen, with the aim of testing the algorithm capabilities of handle a situation in which the platform gets close to its limits. W_3 is set equal to:

$$W_{\mathrm{y}} = diag\left(\begin{bmatrix}50 & 10 & 0.000001 & 10 & 10 & 10\end{bmatrix}\right) \tag{6.9}$$

$$W_{\mathrm{x}} = diag\left(\begin{bmatrix}30 & 10 & 100000 & 1 & 5 & 1 & 10 & 10 & 10 & 1 & 10 & 1\end{bmatrix} \cdot 3\right) \tag{6.10}$$

$$W_{\mathrm{u}} = diag\left(\begin{bmatrix}0.0001 & 0.001 & 0.001 & 0.001 & 0.001 & 0.001\end{bmatrix}\right) \tag{6.11}$$

$$W_{\varsigma} = diag\left(\begin{bmatrix}5000\end{bmatrix}\right) \tag{6.12}$$

For the next simulations, the reference signal of the linear output acceleration along the x axis is given by an initial part in which the signal is equal to zero and a second part, starting after 1 second, consisting of the second derivative of a sine wave, with magnitude equal to 0.25m and frequency equal to 0.6Hz. It is worth noticing that a prediction window of 4 s allows the algorithm to make prediction about almost an entire period of the reference trajectory.

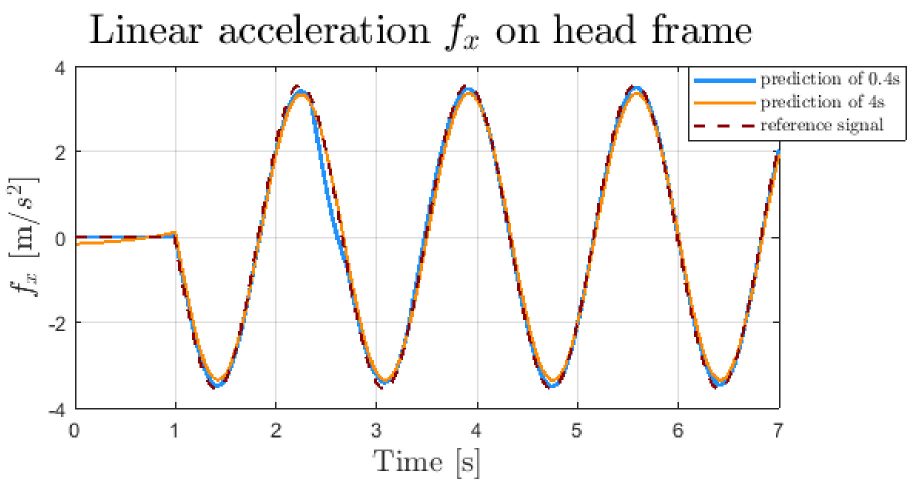

Figure 6.13: Comparison between the output tracking f_x when using a prediction horizon of $0.4s$ and $4s$. W_3 is adopted.

Fig. 6.13 depicts the tracking of the output linear acceleration along the x axis for the two cases. The most relevant difference is that the trajectory derived by the uniform grid scheme diverges from the reference signal at the times when the hexapod legs lengths violate their limits, as can be seen in Fig. 6.14. On the contrary, in the case in which the non-uniform grid technique is used, the platform performs motions within its limits, as can be verified in Fig. 6.15.

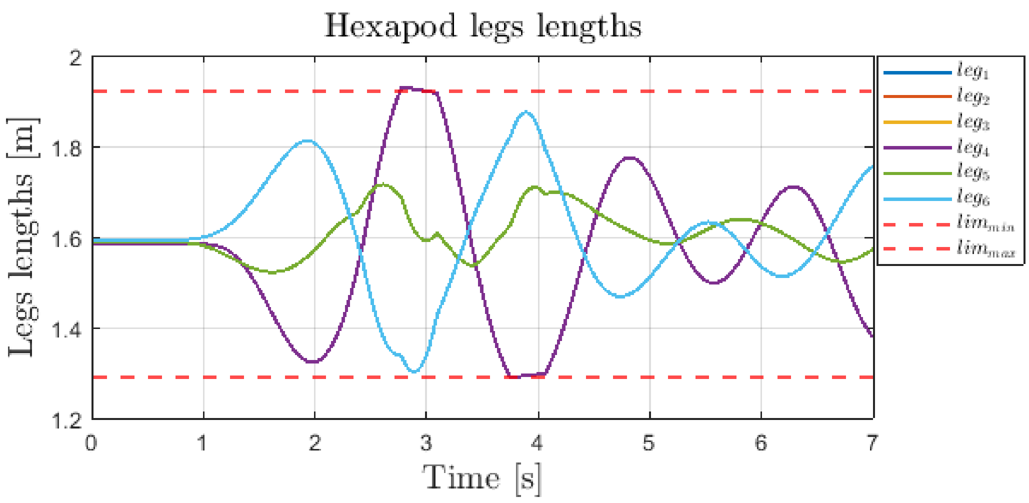

Figure 6.14: Hexapod legs lengths when using a prediction horizon of $0.4s$. W_3 is adopted.

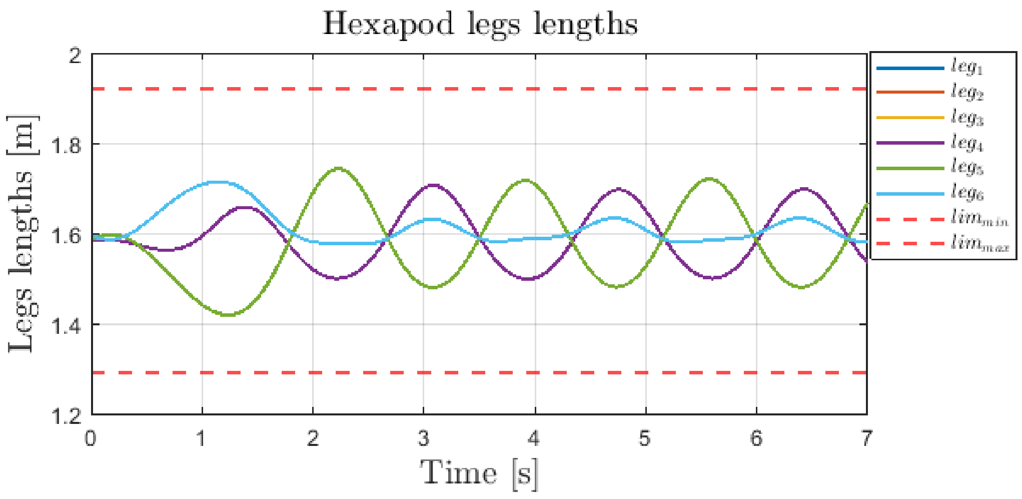

Figure 6.15: Hexapod legs lengths, when using a prediction horizon of $4s$. W_3 is adopted.

It is now important to describe how the prediction horizon length affects the platform behaviour. The linear displacement along the x axis of O_p and the platform pitch angle when using the two different strategies are reported in Fig. 6.16 and Fig. 6.17, respectively.
In the uniform grid scheme with $N_p = 40$, the platform starts moving just before the beginning of the reference sine wave, due to the limited prediction horizon length that avoids the algorithms to anticipate maneuvers. As a result, the simulator is forced to perform an initial rough maneuver to track the reference signal, resulting in the violation of the limits on legs

lengths. To compensate the irregular trajectory of O_p along the x axis, a great component of the pitch angle is exploited, in order to track the reference signal.
Very different is the platform behaviour resulting from the non-uniform grid procedure. In the initial moments, the algorithm is able to predict a wide portion of the sine wave, and this allows it to anticipate the movement. In fact, the center of the top disk O_p departs from its neutral position quite before the beginning of the sine wave, to prepare the output tracking. This initial maneuver is compensated by the use of a little pitch angle. Both movements, along the x axis and around the y axis, are generally periodic and symmetric with respect to the neutral position. The sinusoidal trajectory performed along the x axis by O_p is perfectly in phase with the pitch angle, contrary to the case with the uniform grid scheme.

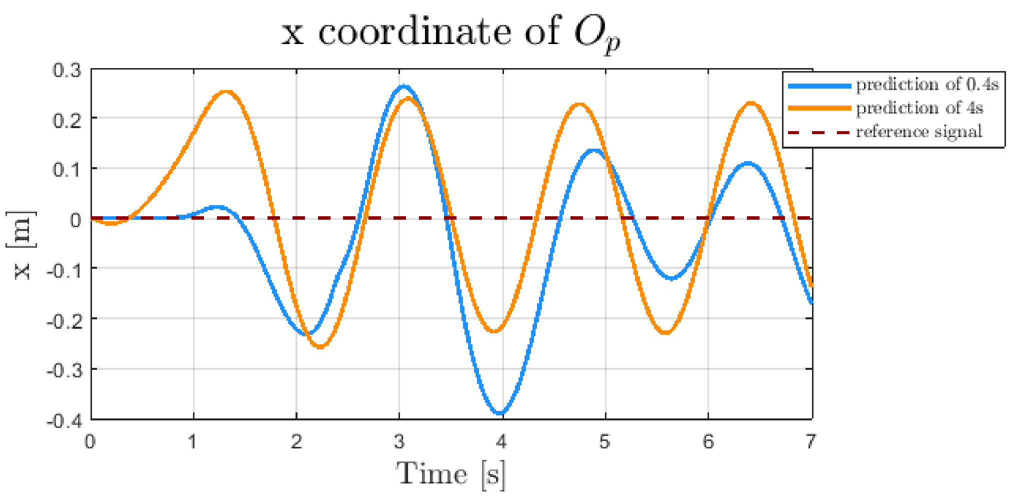

Figure 6.16: Comparison between the linear displacement along the x axis of O_p when using a prediction horizon of $0.4s$ and $4s$. W_3 is adopted.

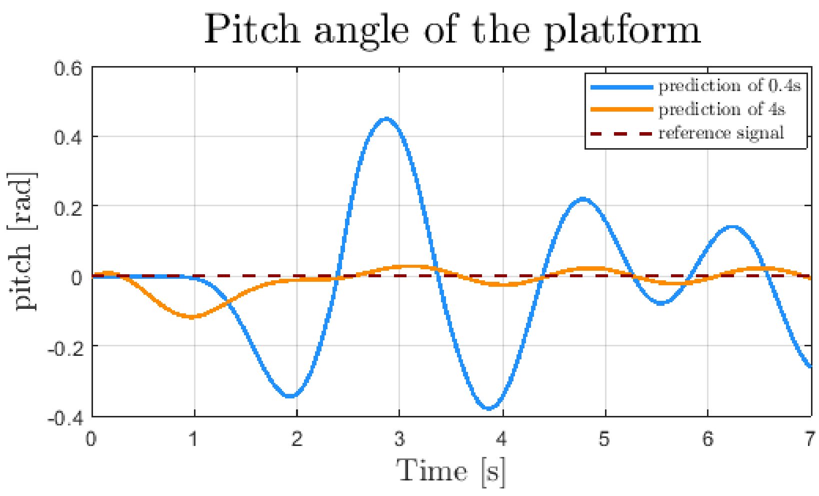

Figure 6.17: Comparison between the platform pitch angle when using a prediction horizon of $0.4s$ and $4s$. W_3 is adopted.

The previous example demonstrates the importance of the predictive technique of the Motion Cueing Algorithms based on Model Predictive Control: the knowledge of a considerable portion of the future reference trajectory allows the controller to anticipate platform maneuvers, to optimize the motion within the available workspace.

7
Discussion and future works

Thanks to its increasing importance, the dynamic driving simulator is being largely investigated, with the aim of further improve the driver's motion experience. The effectiveness of such device strictly depends, in addition to actuators characteristics, to the performance of the motion cueing algorithm, which, by the use of specific transformations, allows the reproduction of the sensation that the driver would perceive in the real car, while respecting the limited platform workspace. This thesis has presented and developed a motion cueing algorithm based on model predictive control technique, relying on the kinematic model of a hexapod Stewart platform simulator. The most important features of the algorithm have been described and proved, with particular attention to the role of the tunable parameters in the algorithm performances.
The weighting matrices associated to the terms of the cost function strongly affect the platform behaviour: the use of a relative small weights for the linear components allows the platform to perform considerable displacements within its operational space, while higher weights associated to the linear components force the platform to stay close to its neutral position, exploiting the washout and the tilt coordination effects.
To reduce the computational burden, considerations about how the constraints declaration affects the resolution time have been made, and consequently constraints on legs velocities and accelerations have been removed.
Furthermore, two different receding horizon strategies have been described, move blocking and non-uniform grid, with the aim to reduce the optimal control problem and, as a result, decrease the computational time. The main properties of the two methods have been presented, together with the maximum prediction horizon lengths they can provide.
To conclude, another tunable parameter considered is the prediction horizon length N_p. A valuable MPC-based MCA requires a prediction horizon wide enough to anticipate and prevent limits violation, thus maximizing the platform capabilities. In fact, N_p small results in

an increased probably of violating constraints, with the consequence of the infeasibility of the problem. An example to illustrate this concept has been provided, highlighting the advantage of the previously mentioned non-uniform grid strategy.
The natural development of this work is the implementation of the discussed algorithm on a real dynamic simulator with a professional driver. Other solutions to improve the computational time can be tried. Moreover, the hexapod can be equipped by a tripod to increase the number of DOF and enhance the driver's motion experience.

References

[1] J. Slob, “State-of-the-art driving simulators, a literature survey,” *DCT report*, vol. 107, 2008.

[2] D. Cleij, J. Venrooij, P. Pretto, M. Katliar, H. Bülthoff, D. Steffen, F. Hoffmeyer, and H.-P. Schoener, “Comparison between filter- and optimization-based motion cueing algorithms for driving simulation,” *Transportation Research Part F: Traffic Psychology and Behaviour*, 05 2017.

[3] N. J. Garrett and M. C. Best, “Driving simulator motion cueing algorithms – a survey of the state of the art,” 2010.

[4] A. Beghi, M. Bruschetta, and F. Maran, “A real time implementation of mpc based motion cueing strategy for driving simulators,” 12 2012, pp. 6340–6345.

[5] M. Bruschetta, C. Cenedese, and A. Beghi, “A real-time, mpc-based motion cueing algorithm with look-ahead and driver characterization,” *Transportation research part F: traffic psychology and behaviour*, vol. 61, 2019.

[6] M. Bruschetta, F. Maran, and A. Beghi, “A nonlinear, mpc-based motion cueing algorithm for a high-performance, nine-dof dynamic simulator platform,” *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 686–694, 2016.

[7] M. K. M. Olivari, F.M. Drop and H. B*ül*thoff, “Driving simulators with hexapod motion system: Adding a yaw turntable,” *in DSC 2018 Europe VR: Driving Simulation Conf.*, 2018.

[8] M. K. F.M. Drop, M. Olivari and H. B*ül*thoff, “Model predictive motion cueing: Online prediction and washout tuning,” *In Proc. Driving Simulation Conference and Exhibition (DSC)*, 2018.

[9] “Early flight simulators - a history of simulation.” [Online]. Available: https://www.historyofsimulation.com/early-flight-simulators-2/

[10] L. Wang, “Model predictive control: Design and implementation using matlab,” *Proceedings of the American Control Conference*, 2009.

[11] F. Maran, “Model-based control techniques for automotive applications.” *Doctoral thesis. University of Padova. Control System Engineering*, 2013.

[12] H. J. Ferreau, "Model predictive control algorithms for applications with millisecond timescales," *Arenberg Doctoral School of Science, Engineering & Technology, Faculty of Engineering, Department of Electrical Engineering*, 2011.

[13] "Matmpc." [Online]. Available: https://github.com/chenyutao36/MATMPC

[14] Y. Chen, N. Scarabottolo, M. Bruschetta, and A. Beghi, "An efficient move blocking strategy for multiple shooting based nonlinear model predictive control," *IET Control Theory and Applications*, 10 2019.