Università degli Studi di Padova

Department of Information Engineering

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Master Thesis in Control System Engineering

Nonlinear model predictive control optimization for autonomous mobile robots

Supervisor
Prof. Angelo Cenedese

Master Candidate
Riccardo Lorigiola

October 20, 2022

Academic Year 2021/2022

Abstract

In the Agent-Target Coordination field, one of the most researched area is the coordination of a group of heterogeneous mobile agents in order to accomplish advanced tasks. Thanks to the accessibility and the improvement of Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV) over the last decades, their use in exploration, research and industrial cooperative applications is increasing. However, solving challenging tasks, such as search&rescue and environmental monitoring, demands the application of control laws that require high performance computational systems. Despite the components miniaturization, the complexity of developing light-weight but performing processors has lead the growth of cloud-computing.

In this thesis, it is addressed the problem of driving a UGV to follow an UAV in order to set up a landing scenario, while dealing with computational resources allocation. Specifically, the target trajectory is not know in advance by the agent and the only source of information concerning the poses and velocities of both vehicles comes from the camera attached to the external computational node. To solve this problem, it is proposed a cascade of control techniques based on the Model Predictive Control (MPC) and Gaussian Process Regression (GPR) approaches. The Model Predictive Control controller is devoted to solving the Agent-Target Coordination problem by driving the UGV under the aerial vehicle. The GPR module, instead, is dedicated to predicting the computational effort of the controller, to providing the MPC control invariant N and to allocating the computation of the Model Predictive Control solution locally or on the external node.

Simulation results on Matlab are presented in order to illustrate and validate the proposed approach.

Keywords: Nonlinear Model Predictive Control, Optimization, Mobile Robots

Sommario

Una delle aree di maggior sviluppo e innovazione nell'ambito della robotica mobile risulta essere il coordinamento degli agenti mobili eterogenei. Questi vengono impiegati sia in ambito cooperativo industriale che in ambito tecnico e scientifico e sono destinati a svolgere operazioni da semplici a molto complesse. Fanno parte di questi particolari agenti mobili i veicoli terrestri (UGV) e i veicoli aerei (UAV) senza equipaggiamento che per eseguire compiti elaborati, quali search&rescue o monitoraggio ambientale, necessitano di complesse e articolate leggi di controllo che a loro volta richiedono sofisticati processori in grado di fornire elevate performance computazionali. La miniaturizzazione dei processori non sempre riesce a supportare la potenza di calcolo necessaria per gestire tali controllori, pertanto, la nuova frontiera risulta essere il cloud-computing.

Questo lavoro di tesi ha lo scopo di progettare un controllore di un UGV in grado di seguire un UAV anche durante eventuali manovre di atterraggio. Le manovre di inseguimento dell'agente di terra devono essere eseguite mentre il suo processore gestisce anche l'allocazione delle risorse computazionali localmente o in un nodo esterno al sistema. Si è supposto che la traiettoria dell'UAV non risulti nota a priori e che gli unici dati di cui si è potuto disporre risultino forniti dalla telecamera collegata al nodo computazionale esterno. Il problema è stato affrontato proponendo l'implementazione di un controllore a cascata basato sul Model Predictive Control (MPC) e sul Gaussian Process Regression (GPR). Il controllore MPC è stato programmato per coordinare l'UGV al fine di posizionarlo al di sotto dell'UAV. Il modulo GPR è invece strutturato per prevedere il costo computazionale del controllore MPC e allocare il calcolo della soluzione dell'MPC localmente o sul nodo esterno. Tutti i risultati ottenuti nelle diverse fasi di studio del sistema sono stati simulati in ambiente Matlab e gli stessi validano l'architettura di controllo costruita in funzione delle ipotesi effettuate.

Keywords: Nonlinear Model Predictive Control, Optimization, Mobile Robots

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 State of the art

Over the last few years, the use of Autonomous Mobile Robots (AMRs) is significantly increasing in the industrial, educational and research field. They are used in a wide variety of environments and applications, including space and seafloor explorations, packages delivery and environmental monitoring. Among the AMR theme, the cooperation of heterogeneous mobile robots system is one of the topics that met the largest interest. In fact, the complexity and variety of advanced modern tasks requires the application of mixed mobile agents sharing the same global goal but with dedicated sub-tasks. Therefore, it is necessary to resort to a control law that can manage simultaneously the robots goal and the cooperation requirements. One of the most used approach to solve these problems is the *Model Predictive Control* (MPC), also referred to as *Receding Horizon Control* or *Moving Horizon Optimal Control*.

The *Model Predictive Control* is a control algorithm that, given the model of the system, a predicting horizon and an optimization tool, it drives the system in order to minimize a provided cost function. This controller has achieved enormous success thanks to the simplicity whereby the system constraints can be included in the controller formulation. In many cases, the adoption of a linear-MPC may not be enough to satisfy the process requirements due to the intrinsic nonlinearity of the kinematic and the dynamic of systems, sensors and actuators. It is therefore necessary to resort to the *Nonlinear Model Predictive Control* or NMPC, which embeds in the model those nonlinearities. On the other hand, by integrating in the model the nonlinear system behaviour, the optimal solution can no longer be found in close form, requiring the use of numerical methodologies for solving the NMPC problem. In general, these numerical techniques requires high computational capabilities in order to be solved within an acceptable time limit [1, 2, 3, 4].

During these years, despite the increasing number of embedded systems with high computational capability, the use of cloud computing is one of the most prevailing trending [5]. In general, cloud computing is the on-demand availability of computer system resources, mainly data storage and computing power, without direct management by the user. Such method allows low-performance systems to solve challenging tasks by allocating the problem on external systems and by retrieving and applying the generated solution. In addition, this technique allows to assign repetitive tasks to external systems and to fetch the information when needed, without occupying the internal computational resources.

This thesis focuses on the agent-target cooperation between a Differential Drive Robot and a quadrotor. The agent has to follow the target in order to set up a landing scenario while dealing with the NMPC problem allocation between the internal agent processor and the external computational node. In addition, there is no communication between the two mobile robots so the quadrotor trajectory is not known in advance by the UGV. The only source of information about agent and target poses and velocities are the data processed by the external node, which is provided with a camera seeing the whole scenario.

1.2 Thesis structure

The chapters are organized as follows:

- In Chapter 2 are provided the preliminary notions about the three and two dimensional pose of a rigid body and the discussion about holonomic and non-holonomic constrains.

- Chapter 3 is dedicated to the description of agent and target models. Both the unicycle and DDR kinematic models are derived and then deepen the wheels and motors dynamic model. It is also introduced the target model and the necessary conversion to project the quadrotor measures into the unicycle model.

- In Chapter 4 is presented and discussed the external computational node. Then are formalized the assumptions concerning this system.

- Chapter 5 formalizes the *Agent-Target Optimized Coordination* problem by defining the metrics and the objectives of the controller. It is successively introduced the definitions of optimal control problem and model predictive control.

- Chapter 6 provides the tools to optimally discretize a dynamic system using *Runge-Kutta* integration method. It also defines the NMPC cost function and architecture.

- Chapter 7 presents the final controller that solves both the Agent-Target Coordination and the Controller Optimization requirements. It presents the Gaussian Process Regression method and its application in order to solve the Controller Optimization requirement. It is then presented the whole controller scheme.

- Chapter 8 reports the results obtained during the simulation of the whole system.

- Chapter 9 summarizes the obtained results and suggests future work topics.

1.3 Notation

In this thesis, vectors in $\mathbb{R}^m$ and matrices in $\mathbb{R}^{m\times n}$ are indicated respectively with bold lowercase and bold uppercase letters. $\mathbf{0}_{m\times n} \in \mathbb{R}^{m\times n}$ and $\mathbf{0}_m \in \mathbb{R}^m$ denote respectively the null $(m \times n)$ matrix and m-dimensional column vector, $\boldsymbol{I}_n \in \mathbb{R}^{n\times n}$ denotes the identity square matrix. $SO(n)$ denotes the special orthogonal group of dimension n, while S^n denotes the n-sphere space.

Chapter 2

Preliminary

2.1 Pose of a rigid body

A rigid body is entirely described in space by its position and orientation, or in short by its pose, with respect to a reference frame. Let $\mathfrak{F}_W$ be the orthonormal reference frame and x, y and z be the unit vectors of the frame axes. In order to describe the rigid body pose, it is convenient to consider the orthonormal frame attached to the body $\mathfrak{F}_B$ and express its position and orientation with respect to $\mathfrak{F}_W$. Therefore, it is possible to define $\boldsymbol{p}$ as the vector connecting frames origins and $\boldsymbol{R}$ as the rotation matrix of $\mathfrak{F}_B$ with respect to $\mathfrak{F}_W$. The columns of $\boldsymbol{R}$ represent the coordinates of the $\mathfrak{F}_B$ axis x', y' and z' with respect the body frame axis x, y and z.

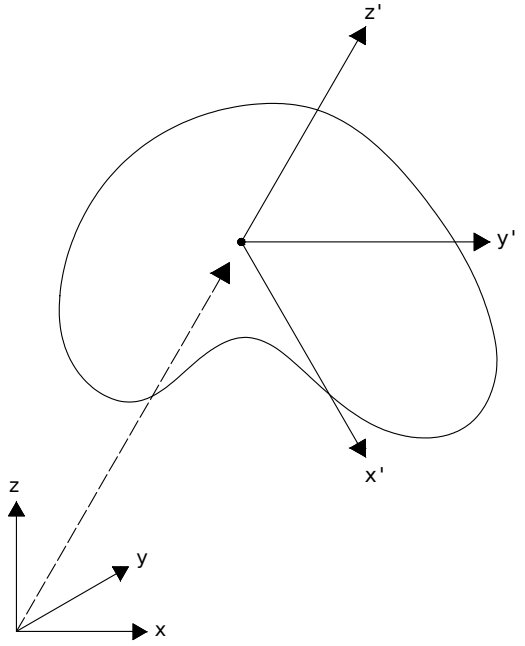


Figure 2.1: Position and orientation of a rigid body

Before proceeding, it is worth mentioning some properties of the rotation matrices. $\boldsymbol{R}$ is an *orthogonal* matrix and therefore

$$\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}_3 \iff \boldsymbol{R}^T = \boldsymbol{R}^{-1} \tag{2.1.1}$$

which leads to the fact that its transpose is equal to its inverse. It is also possible to note that if $\det(\boldsymbol{R}) = 1$, the frame is right-handed while, if $\det(\boldsymbol{R}) = -1$ the frame is left-handed. The rotation matrix defined above belongs to the special orthogonal group $\mathbb{SO}(m)$ of the real $(m \times m)$ matrices. For more details see book [6].

It is possible to attribute a geometrical meaning to the rotation matrix, namely $\boldsymbol{R}$ describes the rotation around an axis in space needed to align the axes of the reference frame with the corresponding axes of the body frame. In fact, a point P in space can be represented as $\boldsymbol{p}$ in $\mathfrak{F}_W$ or as $\boldsymbol{p}'$ in $\mathfrak{F}_{W'}$. It follows that

$$\boldsymbol{p} = \boldsymbol{R}\boldsymbol{p}' \quad \iff \quad \boldsymbol{p}' = \boldsymbol{R}^T\boldsymbol{p} \tag{2.1.2}$$

It is possible also to derive a composition rule for consecutive rotations of the same vector. Let $\mathfrak{F}_0$, $\mathfrak{F}_1$ and $\mathfrak{F}_2$ be three reference frame with common origin O and define $\boldsymbol{R}^i_j$ as the rotation matrix of frame i with respect to j. It follows that the generic point P can be expressed as

$$\begin{cases} \boldsymbol{p}^0 = \boldsymbol{R}^0_1\,\boldsymbol{p}^1 \\ \boldsymbol{p}^1 = \boldsymbol{R}^1_2\,\boldsymbol{p}^2 \end{cases} \quad \Longrightarrow \quad \boldsymbol{p}^0 = \boldsymbol{R}^0_1\,\boldsymbol{R}^1_2\,\boldsymbol{p}^2 = \boldsymbol{R}^0_2\,\boldsymbol{p}^2 \tag{2.1.3}$$

where the superscript on positions and matrices denotes the frame in which elements are expressed. Using this reasoning, it is possible to prove that the composition of rotations is not commutative

2.1.1 Euler angles

In a three dimensional environment, the rotation matrix $\boldsymbol{R}$ describes the rotation of a frame using nine elements, though are not independent. By exploiting the orthogonality property of $\boldsymbol{R}$, the elements are related by six constraints that yield this representation a redundant description. In fact, a minimal representation of the special orthogonal group $\mathbb{SO}(m)$ requires $m(m-1)/2$ parameters. Therefore, three parameters are needed to parameterize $\mathbb{SO}(3)$, while only one is needed for $\mathbb{SO}(2)$ (planar rotation). It follows that a minimal representation of the orientation of a rigid body in a three dimensional environment can be obtained by using a set of three angles.

Theorem 2.1.1 (Euler's rotation theorem) *A generic rotation matrix* $\boldsymbol{R} \in \mathbb{SO}(3)$ *can be obtained by composing a suitable sequence of three elementary rotations while guaranteeing that two successive rotations are not made about parallel axes.*

Using the *Euler's rotation* theorem, it is possible to create 12 distinct set of rotations around the coordinate frame axes, called *Euler angles* triplets. In the following, the ZYX angles set or *Roll-Pitch-Yaw* angles set is used, namely $\boldsymbol{\Phi} = [\varphi, \vartheta, \psi]^T$.

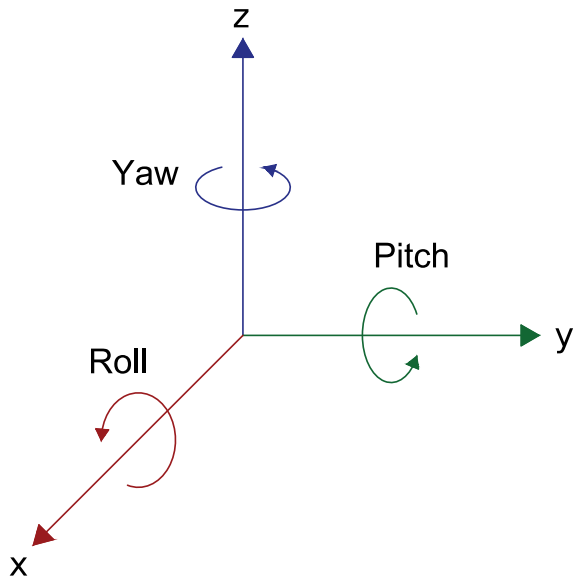


Figure 2.2: *Roll-Pitch-Yaw* angles set representation

The rotation adopting *Roll-Pitch-Yaw* angles can be obtained by rotating the frame around axis x by the angle φ (roll), the rotating around axis y by ϑ (pitch) and at last around axis z by ψ (yaw). Using the composition rule for consecutive rotations in Equation 2.1.3 and defining the three elementary rotation as

$$\boldsymbol{R}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix} \tag{2.1.4a}$$

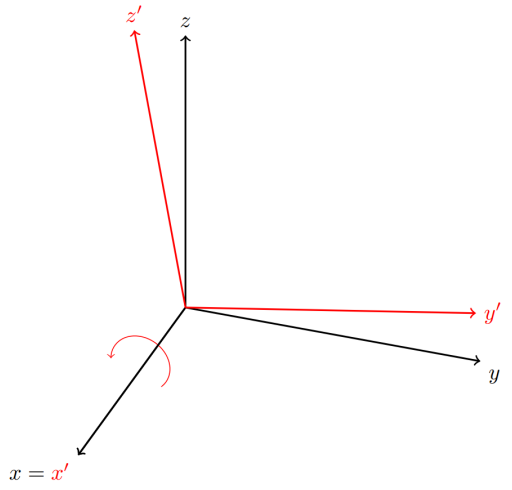
$$\boldsymbol{R}_y(\vartheta) = \begin{bmatrix} \cos\vartheta & 0 & \sin\vartheta \\ 0 & 1 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta \end{bmatrix} \tag{2.1.4b}$$

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1.4c}$$

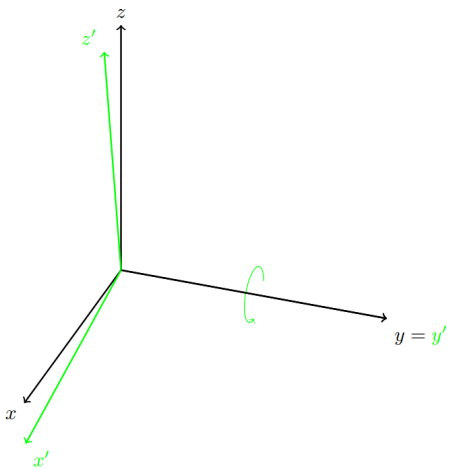
it is possible to compute $\boldsymbol{R}(\boldsymbol{\Phi})$ as

$$\begin{aligned}\boldsymbol{R}(\boldsymbol{\Phi}) &= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\vartheta)\boldsymbol{R}_x(\varphi) \\ &= \begin{bmatrix} \mathrm{c}_\psi \mathrm{c}_\vartheta & -\mathrm{s}_\psi \mathrm{c}_\varphi + \mathrm{c}_\psi \mathrm{s}_\vartheta \mathrm{s}_\varphi & \mathrm{s}_\psi \mathrm{s}_\varphi + \mathrm{c}_\psi \mathrm{s}_\vartheta \mathrm{c}_\varphi \\ \mathrm{s}_\psi \mathrm{c}_\vartheta & \mathrm{c}_\psi \mathrm{c}_\varphi + \mathrm{s}_\psi \mathrm{s}_\vartheta \mathrm{s}_\varphi & -\mathrm{c}_\psi \mathrm{s}_\varphi + \mathrm{s}_\psi \mathrm{s}_\vartheta \mathrm{c}_\varphi \\ -\mathrm{s}_\vartheta & \mathrm{c}_\vartheta \mathrm{s}_\varphi & \mathrm{c}_\vartheta \mathrm{c}_\varphi \end{bmatrix}\end{aligned} \tag{2.1.5}$$

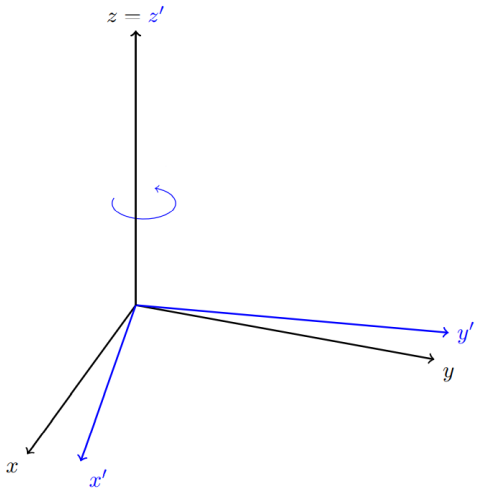
where s_α and c_α are respectively the sine and cosine of angle α.



(a) x axis rotation (roll)



(b) y axis rotation (pitch)



(c) z axis rotation (yaw)

Figure 2.3: Elementary rotations around the coordinate axis

2.2 Holonomic and non-holonomic constraints

In order to derive a mobile robot kinematic and dynamic equations, it is relevant to point out and analyze the presence of *holonomic* and/or *non-holonomic* constraints. Such constraints can limit the admissible configuration space and/or how the system can reach the configurations.

Definition 2.2.1 *Given a dynamical system of equation* $\ddot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}, t)$*, or equivalently* $\dot{\boldsymbol{x}} = \boldsymbol{f}_x(\boldsymbol{x}, \boldsymbol{u}, t)$*, it is said to be subjected to a constraints if there is a function defined over* $\boldsymbol{q}$ *and* $\boldsymbol{u}$ *of the form*

$$\psi(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}, t) \geq 0 \tag{2.2.1}$$

In this wide variety of constraints, the state constraints can be divided into two group

$$\psi(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \geq 0 \qquad (\textit{ non-holonomic constraint })$$

$$\psi(\boldsymbol{q}, t) \geq 0 \qquad (\textit{ holonomic constraint })$$

The *non-holonomic* constraints are also referred as *kinematic* constraints and as can be seen from from the equation, they can not be integrated. On the other hand, the holonomic constraints can be integrated. From a practical point of view, *non-holonomic* constraints impose that a combination of states and velocities are not accessible, constraining the system kinematic and dynamic but leaving unchanged the configuration space. Instead, the *holonomic* ones reduce the configuration space, making some states inaccessible.

A very useful and straightforward example is the disk rolling on a plane without slipping on the horizontal plane, while keeping the plane that contains it (sagittal plane) in the vertical direction (Figure 2.4).

By defining its generalized coordinates, namely the Cartesian coordinates x and y and its orientation ϑ with respect to a fixed reference frame $\mathfrak{F}_W$, it is possible to define the disk configuration vector $\boldsymbol{q} = [x, y, \vartheta]^T$. The disk pure rolling condition imposes that the velocity along the direction perpendicular to the sagittal plane has to be zero. Therefore, by using some simple geometric considerations, this constraint imposes that

$$\dot{x}\sin\vartheta - \dot{y}\cos\vartheta = 0 \quad \rightarrow \quad \begin{bmatrix}\sin\vartheta & -\cos\vartheta & 0\end{bmatrix}\dot{\boldsymbol{q}} = 0 \tag{2.2.2}$$

The constraint in Equation 2.2.2 is *non-holonomic* since it leaves unchanged the configuration space but imposes that some combinations of configurations and

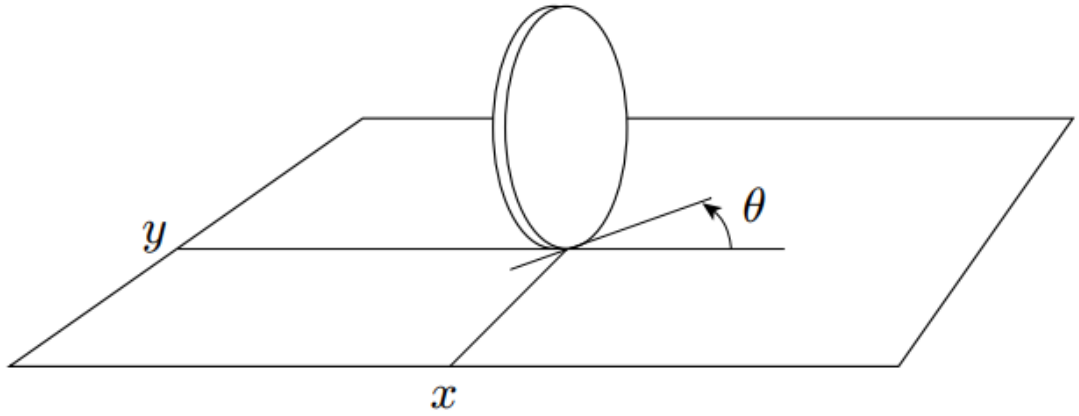


Figure 2.4: Non-holonomic constraint example
Rolling disk on a plane

velocities are locally not admissible. For more details and examples on *holonomic* and *non-holonomic* constraints, see book [6].

Chapter 3

Agent and Target Models

3.1 Agent Model

This section presents an exhaustive description of the agent dynamic. The agent taken into account is an Unmanned Ground Vehicle or UGV.

Figure 3.1: Example of UGV
Differential Drive Robot configuration

Despite moving in a three-dimensional environment, the examined agent has to deal only with three Degrees of Freedom (2 translational DoFs and 1 rotational DoF) since the remaining three DoFs are constrained by the terrain wherein it is moving. Therefore, by knowing the topography, UGVs can be idealized as they are moving on planar surfaces.

This project covers a particular configuration of UGV namely the Differential Drive Robot or DDR. The structure of Differential Drive Robots consists on two

independent drivable wheels with the same rotation axis. In this thesis, it is considered a DDR with also some free castor and/or spherical wheels which are used for physical stability purpose. It is worth noticing that castor and spherical wheels do not add motion constraints and hence their presence can be omitted in the system model.

In the next sections, the generic UGV model is first derived and successively used as basis for the DDR model.

3.1.1 UGV Model

In order to better comprehend the DDR system and to build the simplified target model, it is mandatory to derive the UGV model. In this thesis, the generic UGV is modelled as unicycle, which is a ground vehicle with a single orientable wheel, as shown in Figure 3.2.

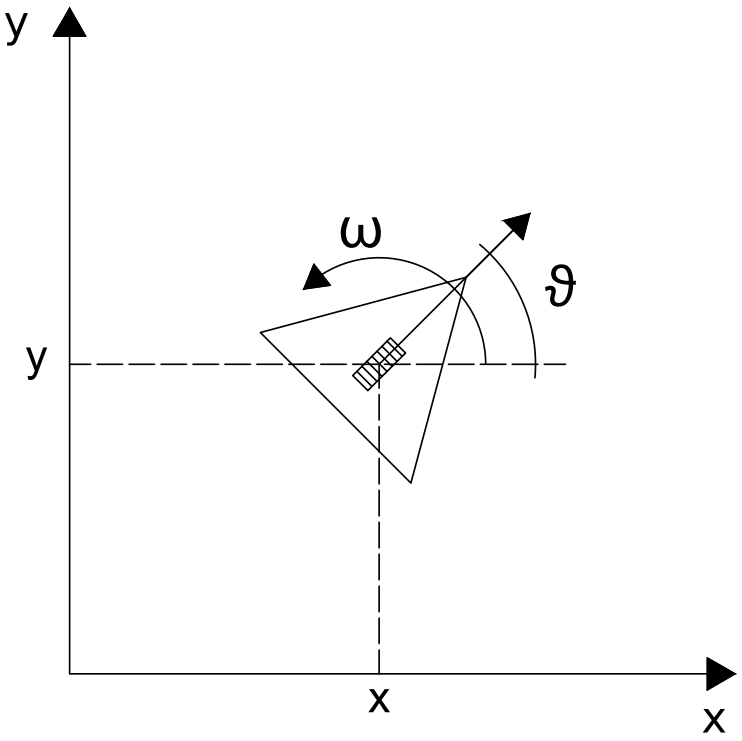


Figure 3.2: UGV unicycle model

Let $\mathfrak{F}_W$ denote the world frame and $\mathfrak{F}_B$ the body-fixed frame, with origin in O_B. In addition, let $\boldsymbol{p} \in \mathbb{R}^2$ be the position of 0_B in $\mathfrak{F}_W$ and $\vartheta \in S^1$be the orientation of the body frame with respect to the world frame. It is therefore possible to define the system state $\boldsymbol{x} = [\boldsymbol{p}, \vartheta]^T = [x, y, \vartheta]^T \in \mathbb{R}^2 \times S^1$ which, accordingly to Section 2.1, represents the pose of the rigid body on planar surface. In addition, it is possible to denote with $v \in \mathbb{R}$ the linear velocity of O_B in the inertial world frame and $\omega \in \mathbb{R}$ the angular velocity of $\mathfrak{F}_B$ with respect to $\mathfrak{F}_W$.

By using Figure 3.2 and some geometric considerations, it is possible to derive the unicycle kinematic model, i.e.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \end{bmatrix} = \begin{bmatrix} \cos\vartheta & 0 \\ \sin\vartheta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.1.1}$$

The previously defined system has no state dynamic and it is only drive by the two inputs v and ω. Moreover, due to the presence of $\sin\vartheta$ and $\cos\vartheta$ inside the input matrix, the system is non-linear.

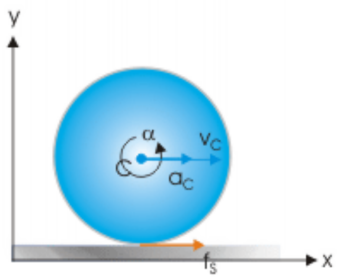
3.1.2 DDR Model

Before going into the Differential Drive Robot model details, the rolling disk example presented in Section 2.2 has to be discussed in more details. During the disk *non-holonomic* constraint derivation, the no-slip assumption is made. This assumption imposes that a wheels has to move with a pure rolling motion, or equivalently without losing grip with the contact surface. This implies that there is a linear direct map between linear and angular wheel velocities, i.e.

$$v = r\,\omega \quad \text{with } v = \sqrt{\dot{x} + \dot{y}} \tag{3.1.2}$$

where r is the wheel radius and ω is its angular velocity.

In general, considering a drivable wheel, its spinning rate ω is not directly imposed but indeed an external torque is applied in order to make it spin. Therefore, given the model in Figure 3.3, the applied torque τ and the mass m, inertia I and radius r of the wheel, it is possible to derive the following dynamic equations



$$f = m\,a_c \quad \tau - rf = I\,\frac{a_c}{r} \tag{3.1.3}$$

$$f = \frac{\tau}{r\left(1 + \frac{I}{m\,r^2}\right)} \tag{3.1.4}$$

Figure 3.3: Pure rolling model

where f is the friction force between the wheel and the surface.

Equation 3.1.4 shows the necessary friction force that allows a torque τ to drive the wheel. To satisfy the no-slip constraint, it is hence necessary that

$$\tau_{max} \leq \mu_s m g\, r \left(1 + \frac{I}{m\, r^2}\right) \tag{3.1.5}$$

where μ_s is the static friction coefficient and g is the gravitational acceleration.

If this constraint is not satisfied, the linear map in Equation 3.1.2 is no more valid and the model has to be completely rewritten. For this reason, Equation 3.1.5 and hence the no-slip constraint are assumed always satisfied.

Thanks to this assumption, the Differential Drive Robot model can thereby be split in two: the motion kinematic model and the wheel dynamic model. The first part considers how the wheels angular velocities modify the agent position and orientation while the second describes the electro-mechanical system model and the implemented low level controller.

3.1.2.1 DDR Kinematic Model

Given the UGV unicycle kinematic model, it is possible to expand it to the Differential Drive Robot kinematic one by using some geometric considerations.

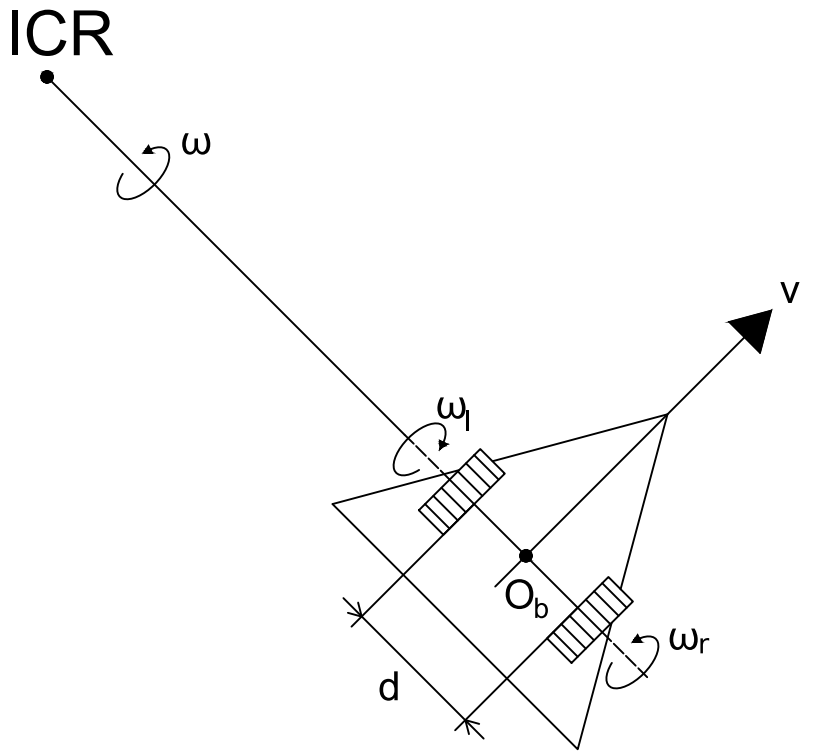


Figure 3.4: Differential Drive Robot model scheme

Considering the middle point between the tho wheels the body frame center O_B, defining r and d ad the wheels radius and distance and ω_r and ω_l as the right and left wheel angular velocities, it is possible to derive the relation between ω_r and ω_l and v and ω of the unicycle model i.e.

$$\begin{cases} v = r\,\frac{\omega_r+\omega_l}{2} \\ \omega = \frac{r}{d}\,(\omega_r - \omega_l) \end{cases} \quad \longleftrightarrow \quad \begin{cases} \omega_r = \frac{v}{r} + \frac{d\,\omega}{2r} \\ \omega_l = \frac{v}{r} - \frac{d\,\omega}{2r} \end{cases} \tag{3.1.6}$$

By combining Equation 3.1.1 and 3.1.6, it is possible to obtain the Differential Drive Robot kinematic model.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \end{bmatrix} = r \begin{bmatrix} \frac{\cos\vartheta}{2} & \frac{\cos\vartheta}{2} \\ \frac{\sin\vartheta}{2} & \frac{\sin\vartheta}{2} \\ \frac{1}{d} & -\frac{1}{d} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \tag{3.1.7}$$

3.1.2.2 DDR wheels dynamic model

To generate an accurate model of the agent, it is necessary to model also the wheels dynamic. The DDR has two independently drivable wheels, connected via gearboxes to two DC motors. In this thesis, it is considered also the presence of two rotary encoders, which measure the wheels angular positions. In the following, the model of a single wheel is considered since, despite the different parameters, the design of the two wheel systems is the same.

Figure 3.5 presents the wheel model scheme: the DC motor is powered by a voltage v and generates a torque that makes the wheel spin.

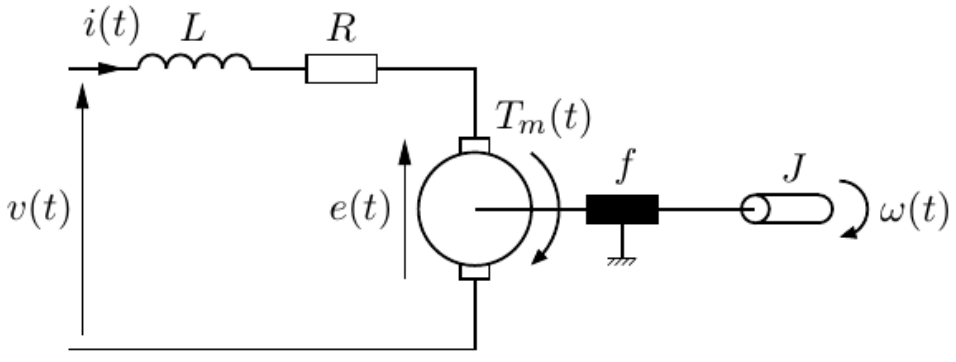


Figure 3.5: Wheels electro-mechanical scheme

From a practical viewpoint, the DC motor is generally controlled via PWM (Pulse Width Modulation) using a H-bridge system, allowing also the rotation of the wheel in both direction. Assuming an ideal PWM control (no input voltage oscillation), the dynamic equations of the wheel system can be written as

$$\begin{aligned} v(t) &= L\,\frac{\mathrm{d}i(t)}{\mathrm{d}t} + R\,i(t) + K\Phi\,\omega(t) \\ K\Phi\,i(t) &= J\,\frac{\mathrm{d}\omega(t)}{\mathrm{d}t} + b\,\omega(t) \end{aligned} \tag{3.1.8}$$

In order to reach a desired wheel speed set-points, two independent proportional-integral feedback control loops (PI) are designed and implemented. This controller scheme has been chosen since it is simple to implement in a low-level controller and it is robust to model parameters change and environment disturbances.

Parameter	Description	SI
R	armature resistance	Ω
L	armature inductance	H
$K\Phi$	motor constant	$\frac{Nm}{\sqrt{W}}$
J	system inertia	$kg\ m^2$
b	viscous friction	$\frac{Nm}{rad/s}$
i	armature current	A
v	input voltage	V

Table 3.1: DC motor electro-mechanical parameters

Assuming that the wheel angular velocity is available, thus making the feedback action possible, the whole wheel dynamic is given by

$$\begin{aligned} v(t) &= L\,\frac{\mathrm{d}i(t)}{\mathrm{d}t} + R\,i(t) + K\Phi\,\omega(t) \\ K\Phi\,i(t) &= J\,\frac{\mathrm{d}\omega(t)}{\mathrm{d}t} + b\,\omega(t) \\ v(t) &= K_p\,e(t) + K_i\int_0^t e(\tau)\,d\tau \\ e(t) &= \omega_d(t) - \omega(t) \end{aligned} \tag{3.1.9}$$

where ω_d is the desired wheel angular velocity, K_p and K_i are respectively the proportional and integral gains. The stability of the PI control action and the availability of the wheel angular velocities are proven and discussed in Appendices A.1.1 and A.1.2.

3.1.2.3 DDR complete dynamic model

By the combination of the models derived in Equations A.1.4 and 3.1.7, it is possible to expressthe whole DDR dynamic model in a state space representation, i.e.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \\ \dot{i}_r \\ \dot{\omega}_r \\ \dot{e}_{i,r} \\ \dot{i}_l \\ \dot{\omega}_l \\ \dot{e}_{i,l} \end{bmatrix} = \left[\begin{array}{c|ccc|ccc} & 0 & \frac{r}{2}\cos\vartheta & 0 & 0 & \frac{r}{2}\cos\vartheta & 0 \\ \mathbf{0}_{3\times3} & 0 & \frac{r}{2}\sin\vartheta & 0 & 0 & \frac{r}{2}\sin\vartheta & 0 \\ & 0 & \frac{r}{d} & 0 & 0 & -\frac{r}{d} & 0 \\ \hline & -\frac{R_r}{L_r} & -\frac{K\Phi_r+K_{p,r}}{L_r} & \frac{K_{i,r}}{L_r} & & & \\ \mathbf{0}_{3\times3} & \frac{K\Phi_r}{J_r} & -\frac{b_r}{J_r} & 0 & \multicolumn{3}{c}{\mathbf{0}_{3\times3}} \\ & 0 & -1 & 0 & & & \\ \hline & & & & -\frac{R_r}{L_r} & -\frac{K\Phi_l+K_{p,l}}{L_l} & \frac{K_{i,l}}{L_l} \\ \mathbf{0}_{3\times3} & \multicolumn{3}{c}{\mathbf{0}_{3\times3}} & \frac{K\Phi_l}{J_l} & -\frac{b_l}{J_l} & 0 \\ & & & & 0 & -1 & 0 \end{array}\right] \begin{bmatrix} x \\ \dot{y} \\ \vartheta \\ i_r \\ \omega_r \\ e_{i,r} \\ i_l \\ \omega_l \\ e_{i,l} \end{bmatrix} + \left[\begin{array}{c|c} & \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} \\ & \\ \hline K_{p,r} & \\ 0 & \mathbf{0}_{3\times1} \\ 1 & \\ \hline & K_{p,l} \\ \mathbf{0}_{3\times1} & 0 \\ & 1 \end{array}\right] \begin{bmatrix} \omega_{d,r} \\ \omega_{d,l} \end{bmatrix} \tag{3.1.10}$$

where the subscript r and l identifies respectively the right and left wheels.

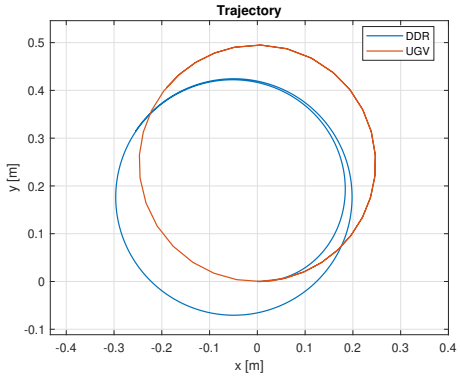
It is possible to notice that the state matrix is block partitioned: each low level controller modifies only the dynamic of the corresponding wheel, without interfering with the other. This feature is useful to tune properly the controllers gains in order to adapt them to different wheels dynamic internal parameters.

Figure 3.6 validates the derived model, the effectiveness of the low level PI controller and compares the unicycle performance to the DDR ones. The parameters and gains used in this comparison are reported in Table 3.2.

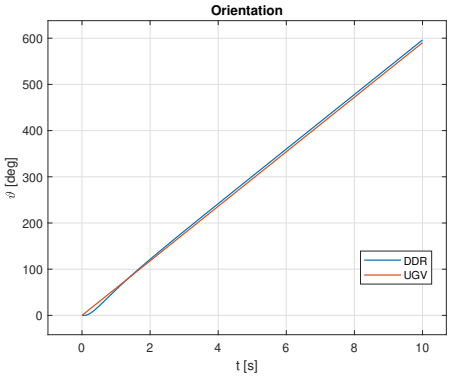
	R	L	$K\Phi$	b	J	K_p	K_i	w_d	r	d
	$[\Omega]$	$[mH]$	$[\frac{Nm}{\sqrt{W}}]$	$[\frac{\mu Nm}{rad/s}]$	$[g\,m^2]$	$[-]$	$[-]$	$[rad/s]$	$[m]$	$[m]$
Right	2	30	0.5	0.1	10	0.1	2	10	0.034	0.165
Left	1.8	20	0.6	0.2	8	0.2	1	5	0.034	0.165

Table 3.2: Unicycle - DDR comparison simulation parameters

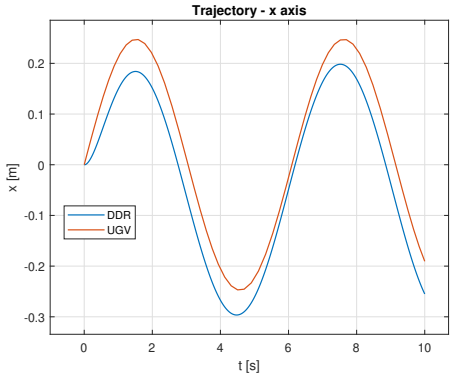
It is possible to notice a difference between the unicycle and the DDR due to the DDR trajectories. Despite the fact that at steady state the two models both draw a circle, the center of rotation is modified due to the transitory behaviour of the DDR wheels as shown in Figure 3.6a. This behaviour can be clearly seen in Figures 3.6c and 3.6d: both the sine waves have the same amplitude and frequency but the DDR one is translated and delayed.



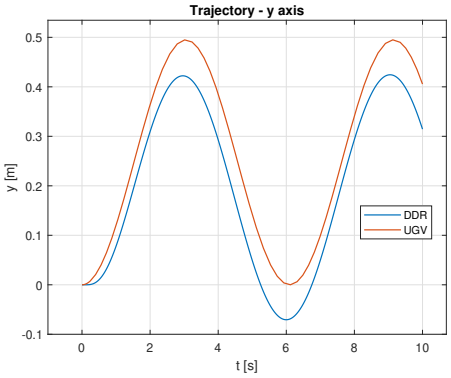
(a) Trajectory



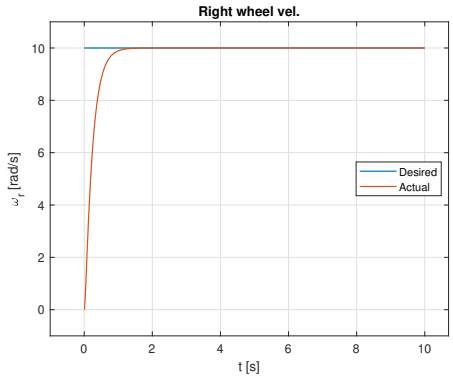
(b) Orientation



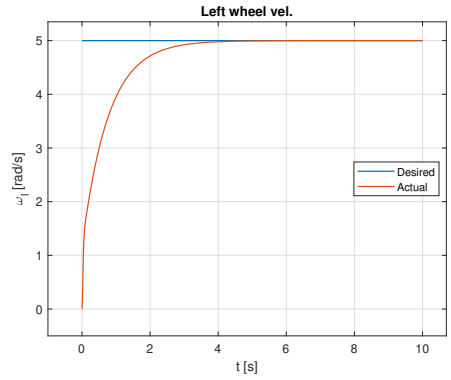
(c) Trajectory - x axis



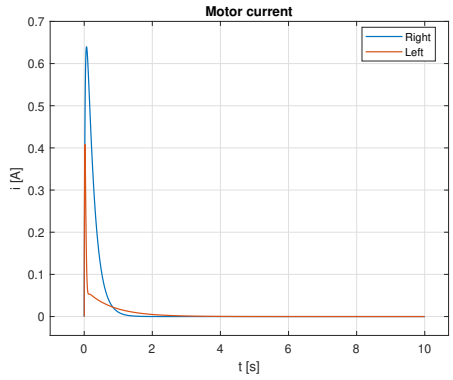
(d) Trajectory - y axis



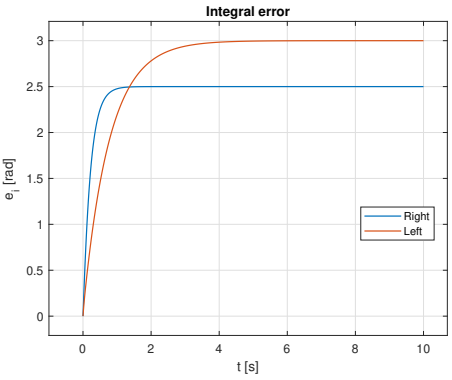
(e) Right wheel angular velocity



(f) Left wheel angular velocity



(g) Motor current



(h) Integral error

Figure 3.6: DDR - unicycle comparison

3.2 Target Model

Tho goal of this section is to understand the dynamic and to provide a suitable model of the quadrotor, i.e. the Unmanned Aerial Vehicle (UAV) with four propellers used as target in this thesis. An example of this vehicle is shown in Figure 3.7.

Figure 3.7: Example of quadrotor

The quadrotor moves in a three-dimensional environment without constraints and thus it has to cope with 6 DoFs (3 translational DoFs and 3 rotational DoFs), having only 4 control inputs. It is therefore evident its under-actuated nature and consequently it can not command an instantaneous acceleration in an arbitrary direction. However, this limitation restricts only the way the quadrotor can reach a desired configuration.

To derive the dynamic model of an aircraft, two prevailing methods are applied: *Euler-Lagrange* formalism and *Newton-Euler* formalism. Both methods provide a consistent description of the system dynamic, though, only the second formalism is presented and use to derive the quadrotor model.

Let $\mathfrak{F}_W$ be the inertial world frame and $\mathfrak{F}_B$ be the body frame of the quadrotor, which its origin O_B is at the center of mass (CoM) of the aircraft. In addition, let $\boldsymbol{p} \in \mathbb{R}^3$ be the position of O_B in $\mathfrak{F}_W$ frame and the rotation matrix $\boldsymbol{R} \in \mathbb{SO}(3)$ be the orientation of $\mathfrak{F}_B$ with respect to the inertial frame. It is possible to describe the full pose of the aircraft in $\mathfrak{F}_W$ by using the pair $(\boldsymbol{p}, \boldsymbol{R}) \in \mathbb{R}^3 \times \mathbb{SO}(3)$. By defining $\boldsymbol{v} \in \mathbb{R}^3$ as the linear velocity of O_B in the inertial frame and $\boldsymbol{\omega} \in \mathbb{R}^3$ as the angular velocity of $\mathfrak{F}_B$ with respect to $\mathfrak{F}_W$, it is possible to derive the kinematic of the quadrotor, namely

$$\begin{aligned} \dot{\boldsymbol{p}} &= \boldsymbol{v} \\ \dot{\boldsymbol{R}} &= \boldsymbol{R}\,[\boldsymbol{\omega}]_\times \end{aligned} \tag{3.2.1}$$

where the last equation is derived in Appendix A.2.1.

In order to model also the dynamic of the aircraft, the forces and torques generated by each propeller are considered in the motion equations. Each propeller rotates around $\boldsymbol{e}_{z_i} \in \mathbb{R}^3$ axis with an angular velocity $\omega_i \in \mathbb{R}$, $i = 1, \dots, 4$. If the propeller spins clockwise (CW) with respect its axis, then its angular velocity in $\mathfrak{F}_W$ is $-\omega_i \boldsymbol{e}_{z_i}$, otherwise it is $\omega_i \boldsymbol{e}_{z_i}$.

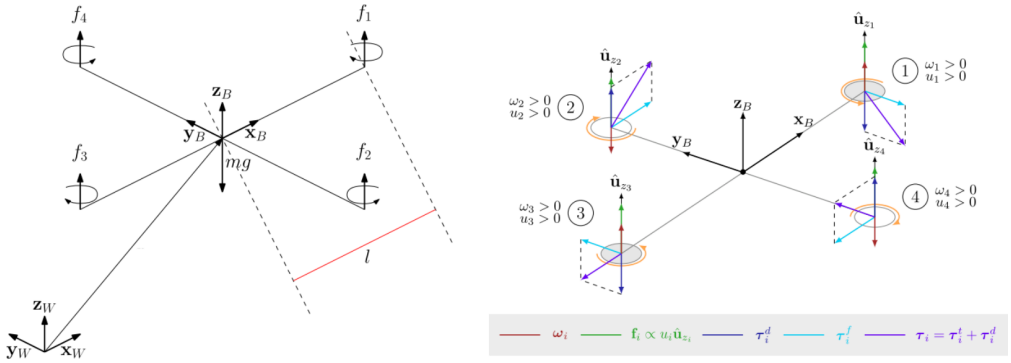


Figure 3.8: Quadrotor model with highlighted forces and torques

Defining $\Omega_i = \omega_i |\omega_i| \in \mathbb{R}$ as the propeller control input, the propeller thrust force $\boldsymbol{f}_i \in \mathbb{R}^3$ is equal to

$$\boldsymbol{f}_i = c_{f_i} \Omega_i \boldsymbol{e}_{z_i} \tag{3.2.2}$$

where $c_{f_i} \in \mathbb{R}^+$ is the propeller trust coefficient constant parameter. As consequence of the thrust force, a torque $\boldsymbol{\tau}_i^t \in \mathbb{R}^3$ associated to each propeller is generated, i.e.

$$\boldsymbol{\tau}_i^t = \boldsymbol{p}_i \times \boldsymbol{f}_i = c_{f_i} \Omega_i \, \boldsymbol{p}_i \times \boldsymbol{e}_{z_i} \tag{3.2.3}$$

where $\boldsymbol{p}_i$ is the distance of the propeller to the CoM of the aircraft. Moreover, the propeller blades generate a drag torque $\boldsymbol{\tau}_i^d \in \mathbb{R}^3$ that has opposite direction compared to the propeller angular velocity and is given by

$$\boldsymbol{\tau}_i^d = c_{\tau_i} \Omega_i \boldsymbol{e}_{z_i} \tag{3.2.4}$$

where $c_{\tau_i} \in \mathbb{R}$ is the propeller drag coefficient which is positive if the propeller spins clockwise and negative otherwise. Combining all the contributions of each propeller (Equations 3.2.2 - 3.2.4), it is possible to obtain the total force and torque acting on the quadrotor CoM, expressed in body frame $\mathfrak{F}_B$, i.e.

$$\boldsymbol{f}_c = \sum_{i=1}^{4} \boldsymbol{f}_i = \sum_{i=1}^{4} c_{f_i} \Omega_i \boldsymbol{e}_{z_i} \tag{3.2.5}$$

$$\boldsymbol{\tau}_c = \sum_{i=1}^{4} \boldsymbol{\tau}_i^t + \boldsymbol{\tau}_i^d = \sum_{i=1}^{4} (c_{f_i} \, \boldsymbol{p}_i \times \boldsymbol{e}_{z_i} + c_{\tau_i} \boldsymbol{e}_{z_i}) \Omega_i \tag{3.2.6}$$

In the final dynamic model, $\boldsymbol{f}_c$ and $\boldsymbol{\tau}_c$ are referred as control variables even if these two quantities are related to the propeller angular velocities $\boldsymbol{\Omega} = [\Omega_1 \quad \Omega_4]^T$ by the wrench map

$$\boldsymbol{f}_c = \boldsymbol{F}\boldsymbol{\Omega} \qquad \boldsymbol{\tau}_c = \boldsymbol{M}\boldsymbol{\Omega} \tag{3.2.7}$$

By denoting l as the aircraft arm length, the matrices $\boldsymbol{F}, \boldsymbol{M} \in \mathbb{R}^{3\times4}$ are given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_{f_1} & c_{f_2} & c_{f_3} & c_{f_4} \end{bmatrix} \qquad \boldsymbol{M} = \begin{bmatrix} 0 & l\,c_{f_2} & 0 & -l\,c_{f_4} \\ -l\,c_{f_1} & 0 & l\,c_{f_3} & 0 \\ -|c_{\tau_1}| & -|c_{\tau_2}| & -|c_{\tau_3}| & -|c_{\tau_4}| \end{bmatrix} \tag{3.2.8}$$

In this thesis, minor perturbation caused by aerodynamic and gyroscopic effects due to rotors and flapping are neglected. Therefore, combining the *Newton-Euler* dynamic equations and the kinematic equations (Equation 3.2.1), the model of the simplified quadrotor can be described by

$$\dot{\boldsymbol{p}} = \boldsymbol{v} \tag{3.2.9a}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}\,[\boldsymbol{\omega}]_\times \tag{3.2.9b}$$

$$\dot{\boldsymbol{v}} = -g\,\boldsymbol{e}_3 + m^{-1}\,\boldsymbol{R}\boldsymbol{f}_c \tag{3.2.9c}$$

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}(-\omega \times \boldsymbol{J}\omega + \boldsymbol{\tau}_c) \tag{3.2.9d}$$

where $g > 0$, $m > 0$ and $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ are respectively the gravitational acceleration, the aircraft mass and its positive definite inertia matrix while $\boldsymbol{R}$ is the rotation matrix embedding the aircraft attitude. In Equation 3.2.9d, the term $-\omega \times \boldsymbol{J}\omega$ has been added to consider the gyroscopic effect caused by the rotation of the whole platform. For a more accurate quadrotor model, see [7, 8].

3.2.1 Quadrotor Projection

The developed model is useful to understand how the target behaves but is does not provide directly useful information for a *Agent-Target Cooperation* scenario. In particular, comparing the angles representing the rotation around the world frame z axis, it is possible to notice that the quadrotor yaw angle ψ does not provide any information about the quadrotor motion direction, as opposed to the DDR ϑ one. Therefore, it is necessary to project the target state into the simplified agent one, i.e. project the target into the unicycles "space".

Given the quadrotor state $\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{\Phi}, \boldsymbol{v}, \boldsymbol{\omega}]^T$ and defining the target unicycle state $\boldsymbol{x}_t = [x_t, y_t, \vartheta_t]^T$ and input $\boldsymbol{u}_t = [v_t, \omega_t]^T$, it is possible to obtain the following equations

$$x_t = \boldsymbol{p}_x \tag{3.2.10a}$$

$$y_t = \boldsymbol{p}_y \tag{3.2.10b}$$

$$v_t = \sqrt{\boldsymbol{v}_x^2 + \boldsymbol{v}_y^2} \tag{3.2.10c}$$

$$\vartheta_t = \text{atan2}(\boldsymbol{v}_y, \boldsymbol{v}_x) \tag{3.2.10d}$$

The previously presented equations represent the projection of the quadrotor position and velocity on the xy-plane. Moreover, the last equation imposes the *non-holonomic* constraints to the target unicycle model. By rewriting Equation 3.2.10d as

$$\frac{\boldsymbol{v}_y}{\boldsymbol{v}_x} = \tan\vartheta_t \;\rightarrow\; \boldsymbol{v}_x \sin\vartheta_t - \boldsymbol{v}_y \cos\vartheta_t = 0 \tag{3.2.11}$$

it is possible to notice that the similarity with the *non-holonomic* rolling disk constraint in Equation 2.2.2.

The last parameter needed to represent the quadrotor as an unicycle is the angular velocity ω_t. By time differentiation of the ϑ_d angle, ω_t can be expressed as

$$\omega_t = \frac{\mathrm{d}\vartheta_t}{\mathrm{d}t} = \frac{\mathrm{d}[\text{atan2}(\boldsymbol{v}_y(t), \boldsymbol{v}_x(t))]}{\mathrm{d}t} = \frac{\dot{\boldsymbol{v}}_y \boldsymbol{v}_x - \boldsymbol{v}_y \dot{\boldsymbol{v}}_x}{\boldsymbol{v}_x^2 + \boldsymbol{v}_y^2} \tag{3.2.12}$$

Despite such quantity can be computed, it is necessary to have the knowledge of $\dot{\boldsymbol{v}}$ and thus of the quadrotor input (Equation 3.2.9c). In this thesis, however, the target trajectory is not know in advance, so the quadrotor input is not available and it is necessary to resort to a different way to compute ω_t. Due to the lack of further information, it is assumed that it is possible to recover such quantity by discrete time differentiation of ϑ_t.

Figures 3.12 and 3.13 show the evolution of the quadrotor model using the parameters reported in Table 3.3. In order to generate the trajectory, a position and yaw angle controller has been implemented with gains as in Table 3.4. All the controller details are reported in Appendix A.2.2.

Moreover, it is possible to observe how Figures 3.12b and 3.12h proves the independence of the quadrotor trajectory from its yaw angle ψ.

Therefore, given the quadrotor state, it is possible to compute the quadrotor projection orientation ϑ_t as in Equation 3.2.10d, resulting in Figure 3.9a. Due to

the discontinuity of the atan2 function, the graph presents jumps and oscillations. Despite solving the discontinuity problem (Figure 3.9b), the resulting graph still has some oscillation, probably caused by the fluctuation of the quadrotor velocities around zero. Hence, by applying the following equation

$$\vartheta_t(k) = \begin{cases} \vartheta_t(k-1) & \text{if } \boldsymbol{v}_x, \boldsymbol{v}_y \leq \varepsilon_v \\ \text{atan2}(\boldsymbol{v}_y(k), \boldsymbol{v}_x(k)) & \text{otherwise} \end{cases} \tag{3.2.13}$$

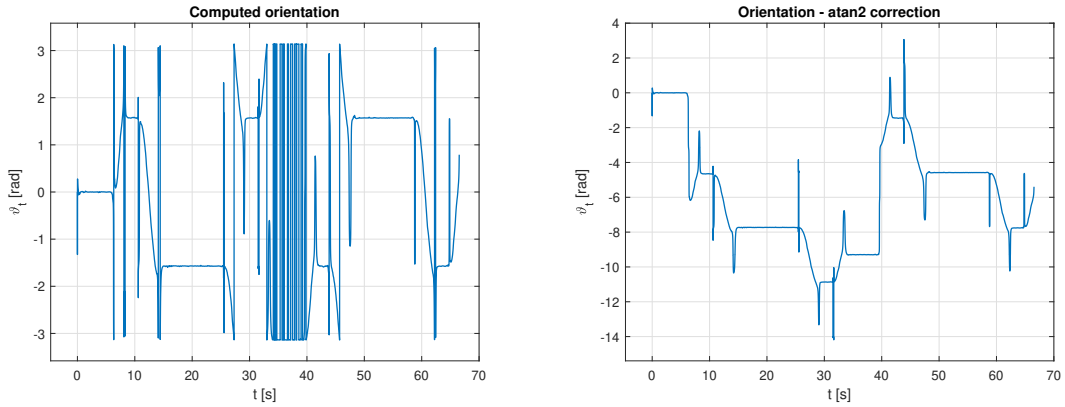
and applying a Butterworth filter, it is possible to obtain ϑ_t and ω_t as in Figure 3.10 and 3.11. Due to the discrete derivative, ω_t presents some noise but it is still reliable.

m $[kg]$	J_x $[kg\,m^2]$	J_y $[kg\,m^2]$	J_z $[kg\,m^2]$	l $[m]$	c_f	c_τ
1.5	0.029125	0.029125	0.055225	0.2555	5.84μ	0.06

Table 3.3: Quadrotor parameters

$\boldsymbol{K}_{pp}$	$\boldsymbol{K}_{ip}$	$\boldsymbol{K}_{dp}$	$\boldsymbol{K}_{da}$	$\boldsymbol{K}_{pa}$
$\begin{bmatrix} 50 & & \\ & 50 & \\ & & 50 \end{bmatrix}$	$\begin{bmatrix} 0.1 & & \\ & 0.1 & \\ & & 0.5 \end{bmatrix}$	$\begin{bmatrix} 15 & & \\ & 15 & \\ & & 10 \end{bmatrix}$	$\begin{bmatrix} 150 & & \\ & 150 & \\ & & 50 \end{bmatrix}$	$\begin{bmatrix} 15 & & \\ & 15 & \\ & & 10 \end{bmatrix}$

Table 3.4: Position and yaw angle controller parameters



(a) atan2 function output

(b) atan2 discontinuity correction

Figure 3.9: Quadrotor projection - Orientation ϑ_t

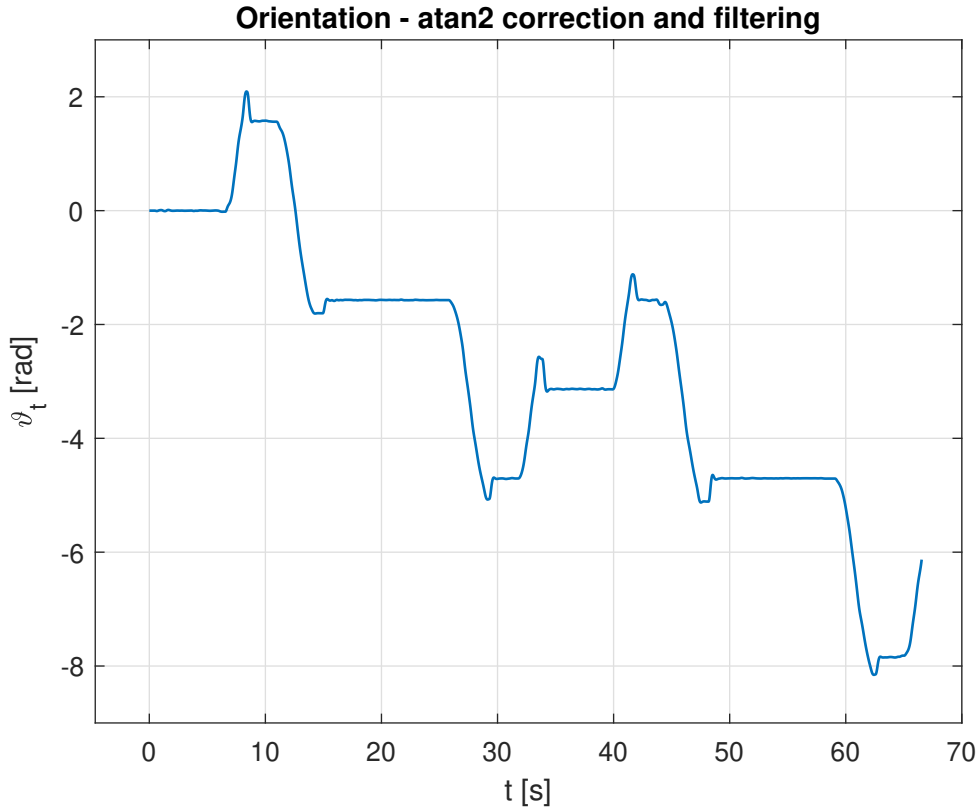


Figure 3.10: Quadrotor projection - Orientation ϑ_t
atan2 discontinuity correction and filtering

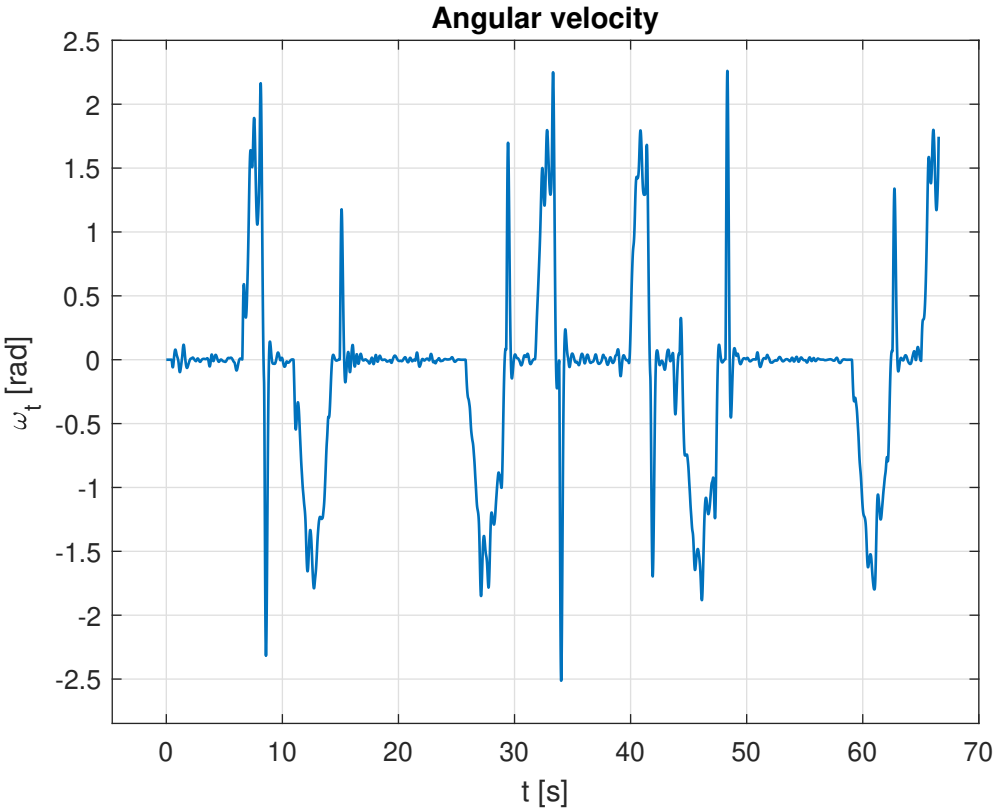
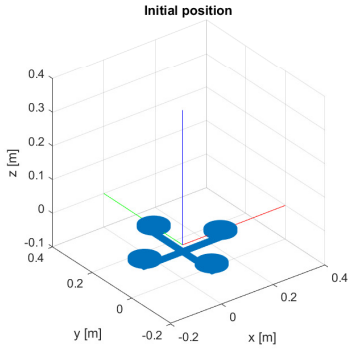
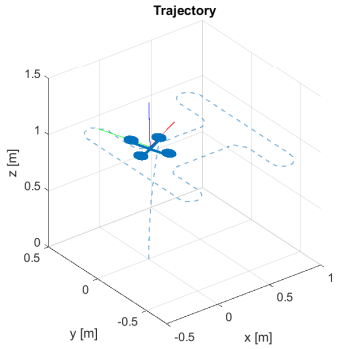


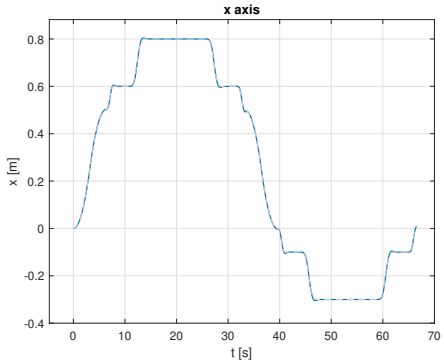
Figure 3.11: Quadrotor projection - Angular velocity ω_t



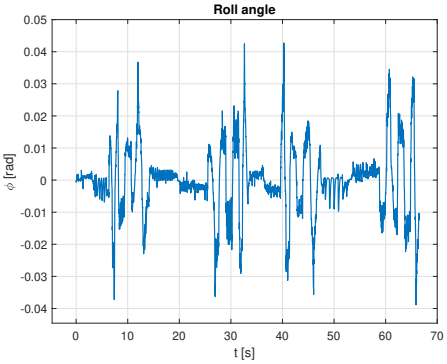
(a) Initial conditions



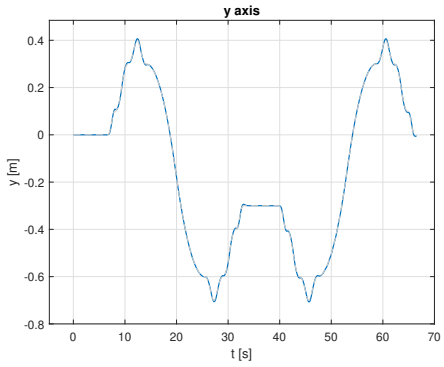
(b) Trajectory



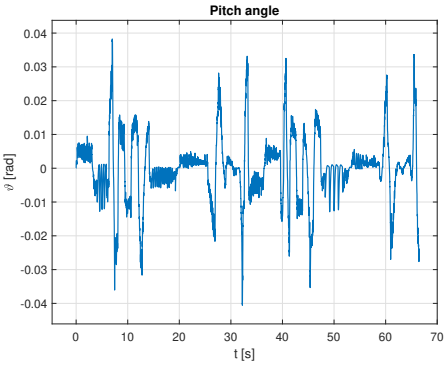
(c) Trajectory - x axis



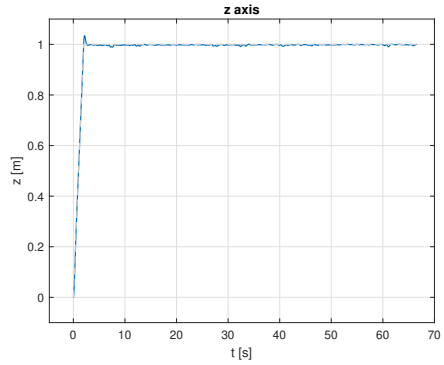
(d) Attitude - roll angle



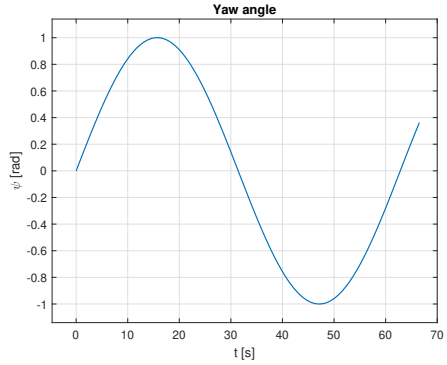
(e) Trajectory - y axis



(f) Attitude - pitch angle

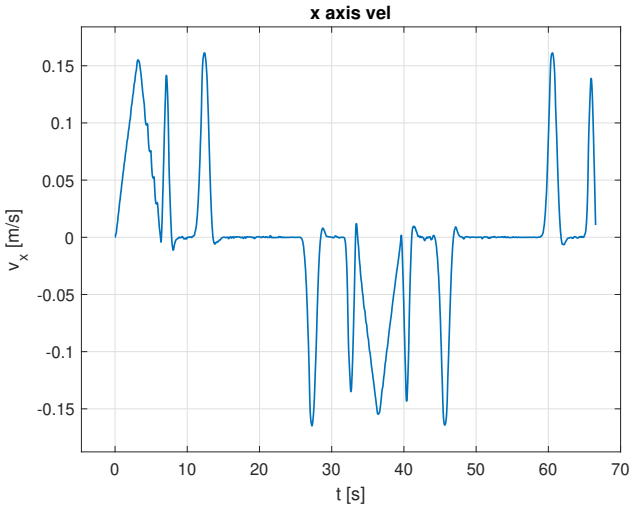


(g) Trajectory - z axis

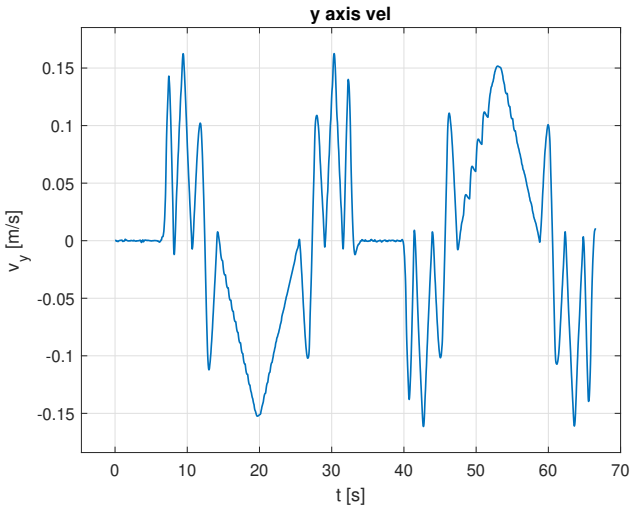


(h) Attitude - yaw angle

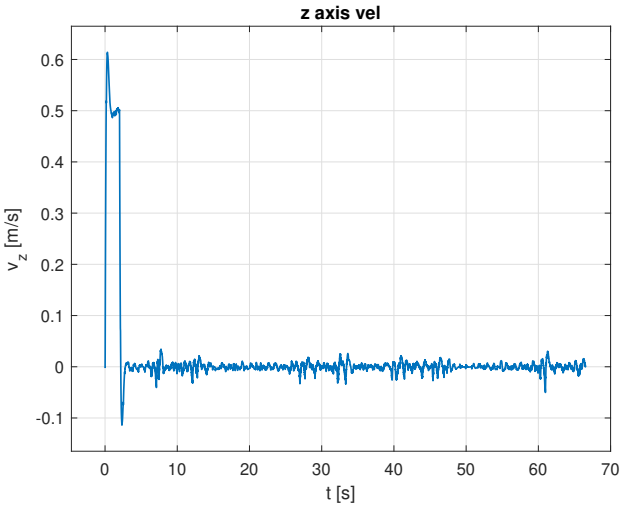
Figure 3.12: Quadrotor simulation - positions and orientations



(a) Linear velocity - x axis

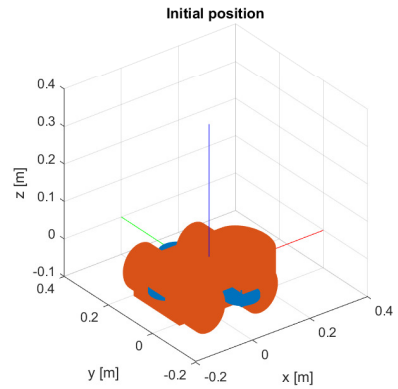


(b) Linear velocity - y axis

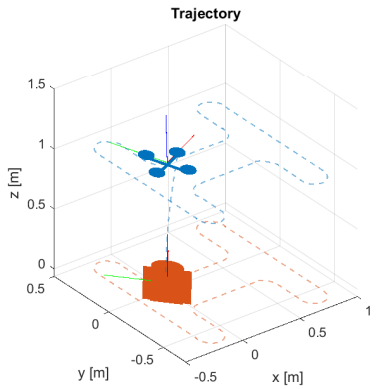


(c) Linear velocity - z axis

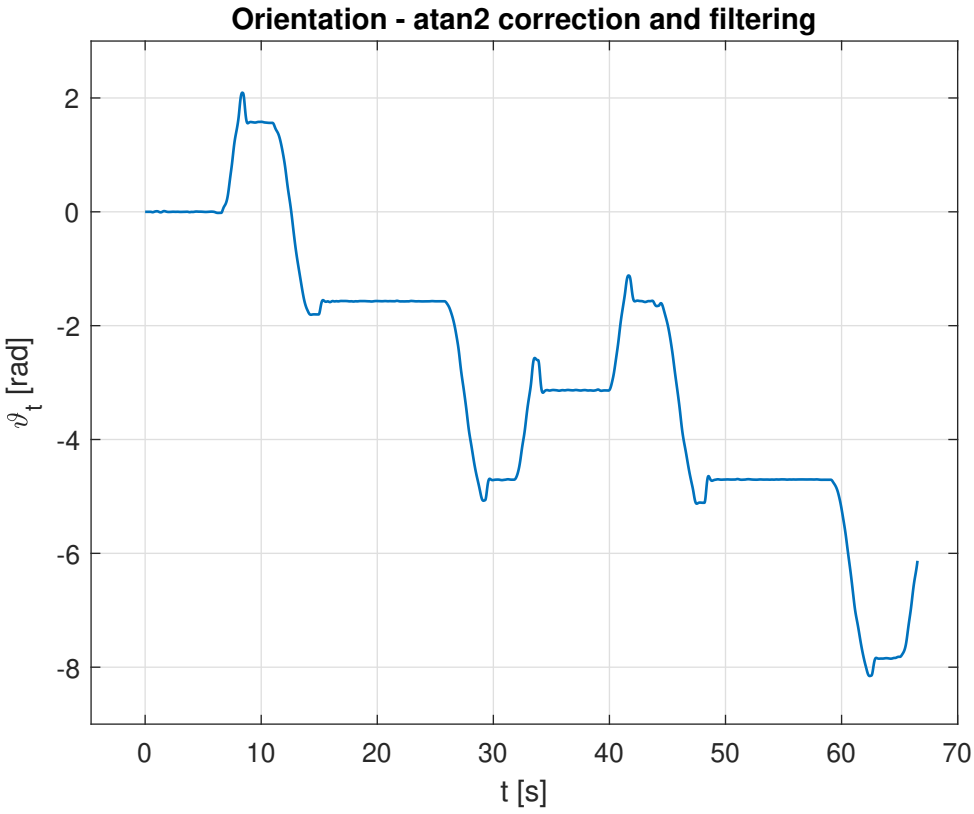
Figure 3.13: Quadrotor simulation - linear velocity



(a) Initial conditions

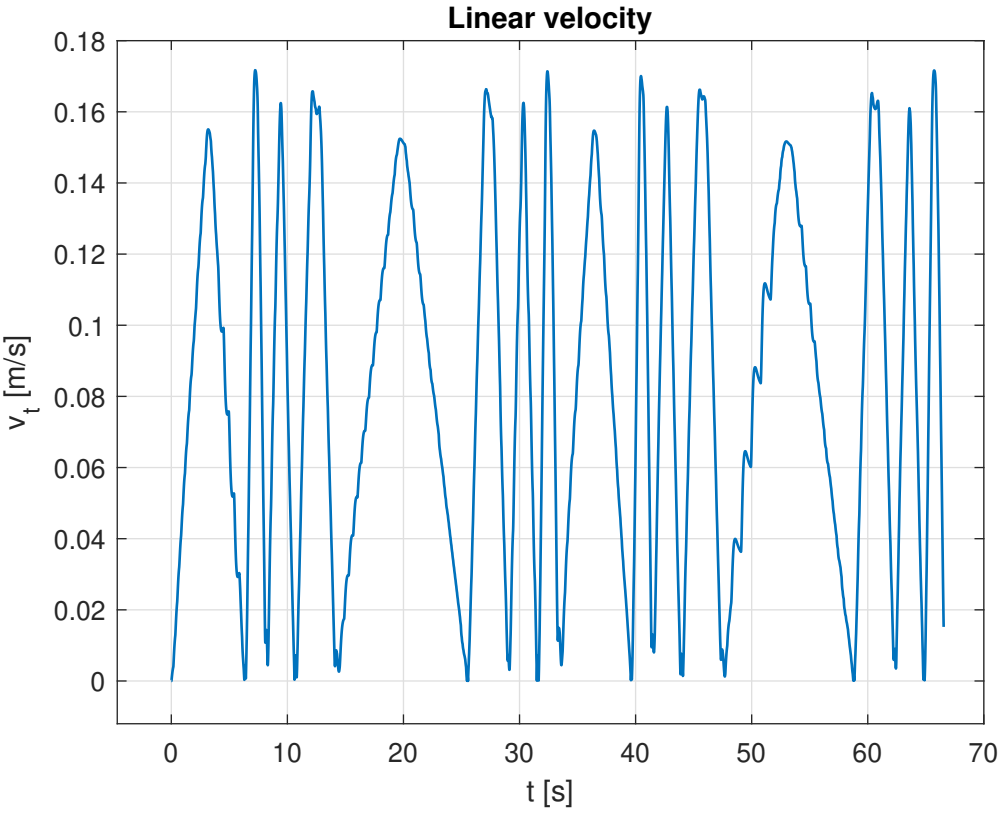


(b) Trajectory

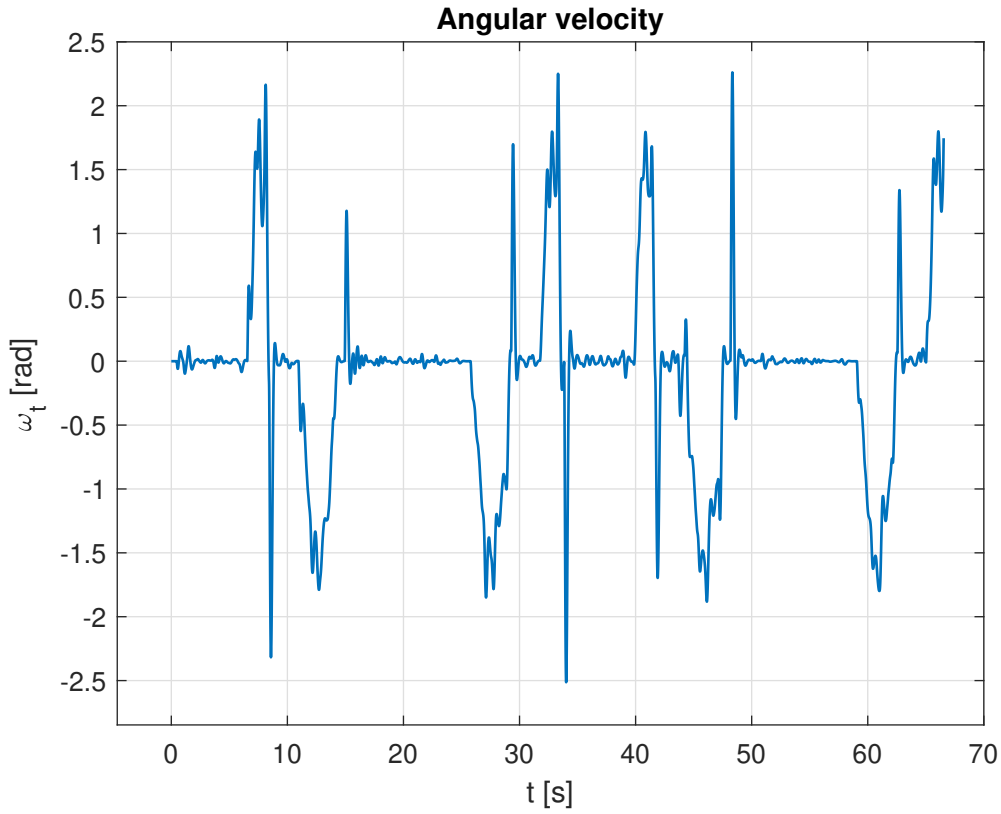


(c) Orientation ϑ_t

Figure 3.14: Quadrotor projection - position and orientation



(a) Linear velocity v_t



(b) Angular velocity ω_t

Figure 3.15: Quadrotor projection - Velocities

Chapter 4

Computation node

The use of external computational nodes for practical application is growing due to the increasing power and computational demanding of ever new controller.

In this thesis, such node is used to provide computational power on demand and the agent and target measures. To obtain information on the agent and target systems, such node is provided with a fixed camera which sees the whole scenario (Figure 4.1). By applying some calibration, detection and appropriate filtering techniques, it is possible to retrieve not only the poses (positions and orientations) but also the linear velocities of the agent and target in the camera frame $\mathfrak{F}_C$.

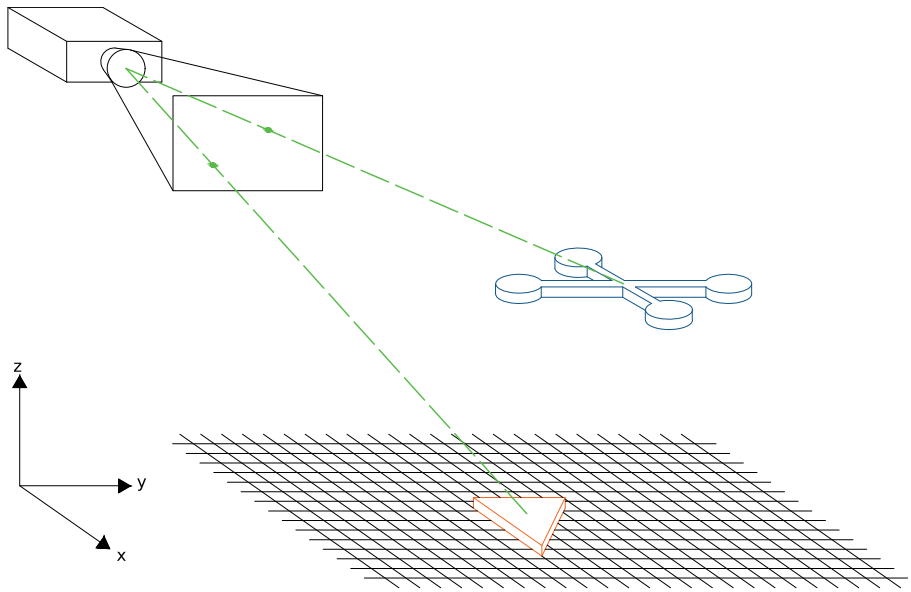


Figure 4.1: Example of camera placement

A very effective, reliable and light system to compute the pose of elements in a camera field of view is the AprilTag detection system. AprilTags are a visual fiducials artificial landmarks designed to be easily to recognized; they are based on a two dimensional QR code but with far smaller data payload (between 4 to

12 bits). The AprilTag system is implemented in C without external dependencies and carefully designed to be portable to real-time embedded devices. Using the provided detection software, it is possible to obtain both position and orientation with respect to the camera frame while identifying the tag. An example of its application can be seen in Figure 4.2. For more details see [9].

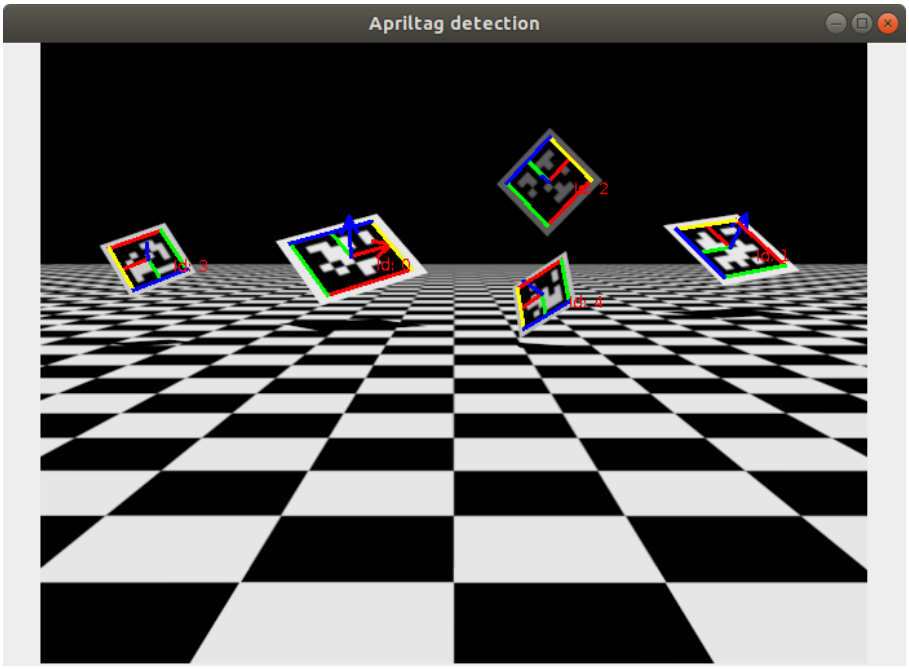


Figure 4.2: AprilTag pose detection example

Combining such detection system with an Extended Kalman Filter (EKF), it is possible also to obtain the object linear velocity $\boldsymbol{v} \in \mathbb{R}^3$ for both the agent and the target.

In order to communicate with the agent, a wireless channel is used. In this work, such channel is idealized, namely it does not introduce communication and package loss latencies.

Chapter 5

Problem formulation and Model Predictive Control

5.1 Problem formulation

The aim of this section is to introduce the system optimization problem and to define the assumption used to design the controller. This work considers a UGV as controlled agent and an Unmanned Aerial Vehicle (UAV) as target. The agent is modelled as Differential Drive Robot (DDR), with dynamics as in Section 3.1, whereas the target has model as presented in Section 3.2.1.

The purpose of the agent is twofold:

- **Agent-Target Coordination**: track the target movement and position itself under the quadrotor, in order to be prepared for a landing scenario;
- **Controller Optimization**: adapt the control system to the target dynamics, balancing between the performance and the computational resources employed

Assuming that the agent is moving on a planar surface ($z = 0$), it is possible to define $\boldsymbol{p} = [x,\, y]^T$ and ϑ as its position and orientation of the body frame $\mathfrak{F}_B$ with respect to the inertial reference frame $\mathfrak{F}_W$. In addition, defining $\boldsymbol{\Omega}_k = [i_k,\, \omega_k,\, e_{ik}]^T$ as the agent wheel-"k" state ($k \in r, l$), it is possible to denote with $\boldsymbol{\xi} \in \mathbb{R}^9$ the state of the agent system, i.e.

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{p} \\ \vartheta \\ \boldsymbol{\Omega}_r \\ \boldsymbol{\Omega}_l \end{bmatrix} \tag{5.1.1}$$

Furthermore, by defining the agent control input as the right and left desired wheels angular velocities $\boldsymbol{\omega}_d = [\omega_{dr}, \omega_{dl}]^T$, the Differential Driver Robot can be described by the following non-linear dynamic system

$$\dot{\boldsymbol{\xi}}(t) = \boldsymbol{f}(\boldsymbol{\xi}(t), \boldsymbol{\omega}_d(t)) \tag{5.1.2}$$

where $\boldsymbol{f}(\cdot, \cdot)$ is derived in Equation 5.1.2.

Similar reasoning can be applied to the quadrotor projection variables, now referred as target quantities. Defining $\boldsymbol{\xi}_t = [x_t, y_t, \vartheta_t]^T$ and $\boldsymbol{u}_t = [v_t, \omega_t]^T$ as the target state and input variables, it is possible to denote the target non-linear dynamic equation, namely

$$\dot{\boldsymbol{\xi}}_t(t) = \boldsymbol{f}_u(\boldsymbol{\xi}_t(t), \boldsymbol{u}_d(t)) \tag{5.1.3}$$

where $\boldsymbol{f}_u(\cdot, \cdot)$ is derived in Equation 3.1.1.

In order to pass the correct information to the controller, it is necessary to estimate the system states.

As previously presented in Chapter 4, the external computation node provides the pose and linear velocities of both agent and target. In this thesis, the agent linear velocity measures are unused but it could be integrated in the model in order to improve the agent model parameters and wheels speed measures.

To complete the agent state, it is necessary to estimate the wheels states, namely $[i_r, \omega_r, e_{ir}, i_l, \omega_l, e_{il}]^T$. Using the internal wheels position measures (Section 3.1.2.2) and the e_{ir} and e_{il} exact knowledge provided by the low level controllers, it is possible to estimate the whole wheels state, as discussed in Appendix A.1.2. At last, it is possible to compute the target angular velocity as presented in Section 3.2.1.

Therefore, it is possible to reconstruct both the whole agent and target states, i.e.

$$\tilde{\boldsymbol{\xi}} = \begin{bmatrix} \tilde{x}_t \\ \tilde{y}_t \\ \tilde{\vartheta} \\ \tilde{i}_r \\ \tilde{\omega}_r \\ \tilde{e}_{ir} \\ \tilde{i}_l \\ \tilde{\omega}_l \\ \tilde{e}_{il} \end{bmatrix} \qquad \tilde{\boldsymbol{\xi}}_t = \begin{bmatrix} \tilde{x}_t \\ \tilde{y}_t \\ \tilde{\vartheta}_t \\ \tilde{v}_t \\ \tilde{\omega}_t \end{bmatrix} \tag{5.1.4}$$

Having identified the agent and target states estimation assumptions, it is possible to define the Differential Drive Robot requirement for the *Agent-Target Optimization Coordination* (ATOC) problem. In order to define the problem, the following assumption have been made:

1. the target velocities do not exceed the actuation capability of the agent;
2. the quadrotor projection model is suitable to represent the actual quadrotor trajectory;
3. the computation node provides the agent and target measures continuously, without losing any of them;
4. the exchange of data between the computation node and the agent takes place via an ideal channel (no latency);

The ATOC problem can be then divided into two sub-problems: *Agent-Target Coordination* (ATC) and *Controller Optimization* (CO)- It is possible to independently define their definitions and requirements and later merge the solutions to obtain the *Agent-Target Optimized Coordination* solution.

Problem 5.1.1 (Agent-Target Coordination) *Given the agent state* $\boldsymbol{\xi}$ *as in Equation 5.1.1, the target pose* $\boldsymbol{\xi}_t = [\boldsymbol{p}_t,\ \vartheta_t]^T = [x_t,\ y_t,\ \vartheta_t]$ *as in Equation 3.2.10, the controller aims to drive the Differential Drive Robot in order to*

$$\lim_{t\to\infty} \|\boldsymbol{p}_t(t) - \boldsymbol{p}(t)\|^2 = 0 \tag{5.1.5a}$$

$$\lim_{t\to\infty} |\vartheta_t(t) - \vartheta(t)|^2 = k\pi \qquad \text{with } k \in \mathbb{N} \tag{5.1.5b}$$

The first equation imposes that the agent position corresponds exactly to the target one, while the second one requires that the DDR heading direction is aligned with the target one, without taking into account its direction.

In order to define the *Controller Optimization* problem, it is necessary to make some further consideration. In a real world application, time between the controller input reception and output response can not be overlooked, resulting in a delay in the actuation of the system. Such delay is generally negligible with respect to the system evolution and/or the controller sample time although, in case of complex control algorithm, it has to be taken into account. In this work, the time needed to compute the solution is taken into account merely during the controller choice.

Furthermore, despite using an ideal channel to communicate with the external computation node, the controller penalizes the use of this source to solve high demanding tasks, in order to mimic a real behaviour.

Given these consideration, it is possible to define the *Controller Optimization* problem.

Problem 5.1.2 (Controller Optimization) *Let $\mathcal{C}$ be a set of n controllers solving the same problem and $\boldsymbol{u}_i$ and T_i be the i-th controller solution and the needed computational time. In addition, let c_{ext} be a fixed cost representing the use of the external computation node. The controller aims to choose the best controller which minimizes the following cost function*

$$n^* = \underset{i \in 1,\dots,n}{\operatorname{argmin}} (\boldsymbol{u}_i - \bar{\boldsymbol{u}})^T \boldsymbol{C}_u (\boldsymbol{u}_i - \bar{\boldsymbol{u}}) + c_T T_i^2 + J_{ext} \tag{5.1.6}$$

where $\boldsymbol{C}_u$ is a weighting cost matrix, $\bar{\boldsymbol{u}}$ is the best solution and J_{ext} is defined as

$$J_{ext} = \begin{cases} 0 & \text{if } T_i < T_{ext} \\ c_{ext} & \text{otherwise} \end{cases} \tag{5.1.7}$$

By tuning the cost parameters and solving the previously defined problem, it is possible to obtain a trade off between an optimal control input, which leads to the best system performance, the computational time needed and the use of the external resource.

5.2 Optimal Control Problem

In order to satisfy every increasing control requirements, advanced control techniques have been studied and implemented. To deal with more and more demanding scenario, optimal control theory is being developed and it is becoming increasingly used. In fact, optimal control theory aims to find an appropriate control action for dynamical systems in order to optimize a cost function. Due to its general theory, this method can handle multiple objective despite they might concern different fields. An example is the autonomous drive vehicle problem: the vehicle may have to minimize the fuel consumption and the travelled distance while respecting the road code and avoiding other vehicles, bicycles and pedestrians.

In this work, the agent has to solve simultaneously several task: it has to position itself under a moving target while aligning itself with the motion direction. Such objectives concerns different fields and might not be compatible; it is thus necessary to introduce a prioritization mechanism. Control strategies based on the solution of optimization problem appear to be best suited to solve such problems.

Reformulating the requirements as cost function and applying optimal control theory, it is possible to obtain an input that guarantees the best system performances. In this thesis, the *Optimal Control* problem is solved by a *Model Predictive Control* (MPC) scheme.

5.2.1 General description

At first, let's introduce the concept of *Infinite horizon optimal control problem*, considering the dynamic continuous time system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{5.2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state and $\boldsymbol{u} \in \mathbb{R}^m$ is the system input. The goal of a model predictive control is to find $\boldsymbol{u}(t)$ for $t \in [0, +\infty)$ that minimizes the cost function J_∞ given the initial condition $\boldsymbol{x}(0) = \tilde{\boldsymbol{x}}_0$, i.e.

$$\min_{\boldsymbol{u}(\cdot)} \; J_\infty(\boldsymbol{x}, \boldsymbol{u}) = \int_0^\infty V(\boldsymbol{x}(t), \boldsymbol{u}(t))\, dt \tag{5.2.2}$$

where $V(\boldsymbol{x}, \boldsymbol{u})$ is function that associates a cost for each state and input of the system. Such problem has not a generic close form solution, but it is possible to prove that if $V(\cdot)$ is positive definite and both $\boldsymbol{f}(\cdot, \cdot)$ and $V(\cdot)$ are enough regular, then exists $\boldsymbol{u}^*(t)$ with $t \in [0, +\infty)$ that minimizes Equation 5.2.2 and stabilizes the system in Equation 5.2.1 in a neighbourhood of $\tilde{\boldsymbol{x}}_0$ [10].

In particular, if we consider the following minimization problem

$$\min_{\boldsymbol{u}(\cdot)} \; J_\infty(\boldsymbol{x}, \boldsymbol{u}) = \int_0^\infty \boldsymbol{x}^T(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^T(t)\boldsymbol{R}\boldsymbol{u}(t)\, dt \tag{5.2.3}$$

$$\text{subject to} \quad \begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) & (\textit{ model dynamics }) \\ \boldsymbol{x}(0) = \tilde{\boldsymbol{x}}_0 & (\textit{ initial condition }) \end{cases}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive semi-definite weight matrices ($\boldsymbol{Q}$, $\boldsymbol{R} \geq 0$), the solution is guaranteed and it is obtained in closed form as stated by Theorem 5.2.1.

Theorem 5.2.1 (Solution of the infinite horizon LQ optimal control problem) *Given the optimization problem as stated in Equation 5.2.3 and let $\boldsymbol{Q}^{\frac{1}{2}}$ be a matrix such that $\boldsymbol{Q} = (\boldsymbol{Q}^{\frac{1}{2}})^T\boldsymbol{Q}^{\frac{1}{2}}$. It holds that:*

1. *the Algebraic Riccati Equation (ARE)*

$$\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{Q} = 0 \tag{5.2.4}$$

has a unique positive semi-definite solution $\boldsymbol{P}_\infty$*. Furthermore, if* $(\boldsymbol{A}, \boldsymbol{Q}^{\frac{1}{2}})$ *is observable,* $\boldsymbol{P}_\infty$ *is positive definite;*

2. *the state feedback control law*

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t) \quad with \quad \boldsymbol{K} = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}_\infty \tag{5.2.5}$$

minimize the infinite horizon quadratic cost function and makes the closed-loop system asymptotically stable

if and only if $(\boldsymbol{A}, \boldsymbol{B}$ *is stabilizable and* $(\boldsymbol{A}, \boldsymbol{Q}^{\frac{1}{2}})$ *is detectable.*

Equation 5.2.3 and Theorem 5.2.1 take into account only linear systems and linear-quadratic cost functions.

In general, when $V(\cdot)$ and $\boldsymbol{f}(\cdot,\cdot)$ assume a generic form, the optimal input that minimize Equation 5.2.2 can not be found in close form. Moreover, the problem is not suitable to be solved numerically due to the infinite space dimension of the solution and the infinite interval of interest.

5.2.2 Model Predictive Control

To solve numerically a generic optimal control problem, an approximate numerical solution is taken into account by using some assumptions. The first assumption is to formulate the problem in a discrete time frame. Despite most of the systems have a continuous dynamic model, the controller is generally implemented digitally and thus discrete dynamic models are preferred.

Hence, the system model in Equation 5.2.1 has to be discretized using proper techniques and sample time T_s, in order to obtain

$$\boldsymbol{x}(k+1) = \boldsymbol{f}_k(\boldsymbol{x}(k), \boldsymbol{u}(k)) \tag{5.2.6}$$

where, as result of a notation abuse it holds that $k = hT_s$ $h \in \mathbb{N}$.

The second assumption is to restrict the infinite time horizon to a finite time interval, namely $[0, T]$, which due to the discrete time nature of the new system can be brought back to a finite set of time samples of length N, referred to as *control horizon*.

The optimal problem in Equation 5.2.3 is then reformulated as follow.

Problem 5.2.1 *Find* $\boldsymbol{u}^*(0) \ldots \boldsymbol{u}^*(N-1)$ *that minimizes the following optimization problem*

$$\min_{\boldsymbol{u}(\cdot)} J = \sum_{k=0}^{N-1} V(\boldsymbol{x}(k), \boldsymbol{u}(k) + V(\boldsymbol{x}(N)) \tag{5.2.7}$$

$$\textit{subject to} \quad \begin{aligned} &\boldsymbol{x}(k+1) = \boldsymbol{f}_k(\boldsymbol{x}(k), \boldsymbol{u}(k)) && (\textit{ model dynamics }) \\ &\boldsymbol{x}(0) = \tilde{\boldsymbol{x}}_0 && (\textit{ initial condition }) \\ &\boldsymbol{x}(k) \in \mathcal{X} \quad \forall\, k \in [0, N] && (\textit{ state constraint }) \\ &\boldsymbol{u}(k) \in \mathcal{U} \quad \forall\, k \in [0, N-1] && (\textit{ input constrain }) \end{aligned}$$

where $J(\cdot)$ *is the cost function,* $\boldsymbol{x}(\cdot)$ *and* $\boldsymbol{u}(\cdot)$ *are the discrete system state and input,* $\mathcal{X}$ *and* $\mathcal{U}$ *are respectively the state and input spaces.*

It is possible to notice that state and input constraints are embedded in the problem formulation, since the state and the input are imposed in their respective sets $\mathcal{X}$ and $\mathcal{U}$.

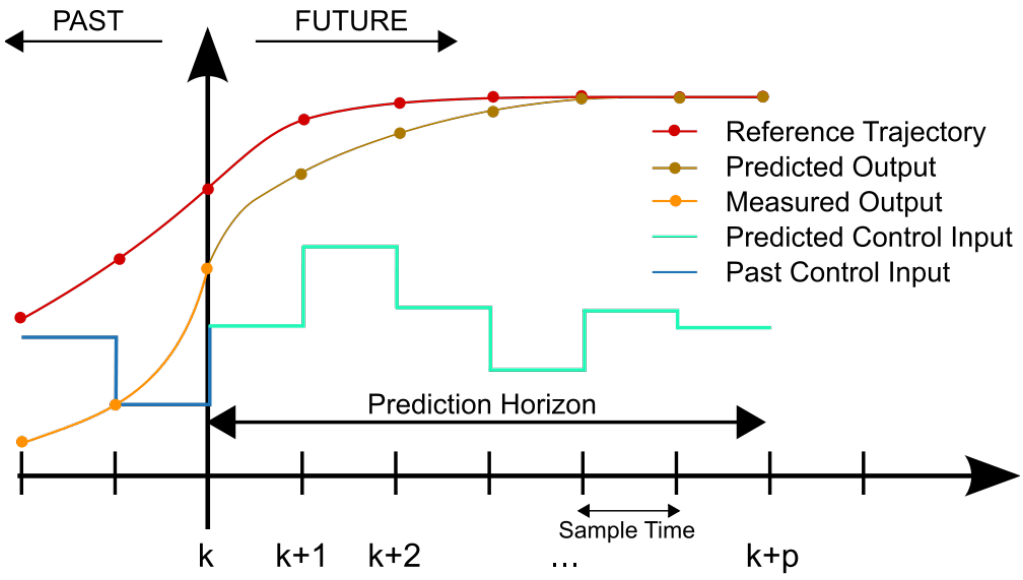


Figure 5.1: Visual representation of the MPC concept

The **Model Predictive Control** is an approach based on the iterative solution of the *Optimal Control Problem* (OCP) (Problem 5.2.1) at each time instant t.

The cost function takes into account the system evolution over the control horizon and minimizes it in order to obtain a series of input $\boldsymbol{u}^*(0) \ldots \boldsymbol{u}^*(N-1)$ that leads to the minimum cost.

In order to achieve a close-loop control, only the first input $\boldsymbol{u}^*(0)$ is applied enabling the system evolution and the construction of a new control problem. Figure 5.1 visually shows the main idea behind the MPC concept.

Chapter 6

NMPC controller definition

This chapter presents the Nonlinear Model Predictive Control (NMPC) scheme used to control the Differential Drive Robot described in Equations 5.1.1 and 5.1.2. The aim is to provide the tools and the arguments to build the different cost functions, to define the input space, to describe the agent update and to finally present the whole controller scheme.

6.1 Runge-Kutta Integration

As stated in Section 5.2.2, it is necessary to convert the system continuous dynamic model in a discrete one in order to be able to apply the NMPC scheme. Since the agent and taget models are strongly non-linear, classic discretization techniques can not be applied due to the high accumulation of approximation errors. Therefore, the *Runge-Kutta* integration method is used. This technique provides a high accurate numerical approximation of a function without using its own high order derivative.

Given the first-order initial-value problem

$$\begin{cases} \dot{y} = f(t,\, y) \quad a \leq t \leq b \\ y(a) = y_0 \end{cases} \tag{6.1.1}$$

and dividing the interval $[a, b]$ into N sub-intervals $[t_n, t_{n+1}]$ with $n = 0, 1, \ldots, N-1$, it is possible to integrate the function by applying the *mean value theorem* for integral and to obtain that

$$\begin{aligned} y(t_{n+1}) - y(t_n) &= \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))\, d\tau = h f(\varepsilon, y(\varepsilon)) \\ \Rightarrow \quad y(t_{n+1}) &= y(t_n) + h f(\varepsilon, y(\varepsilon)) \end{aligned} \tag{6.1.2}$$

where $h = t_{n+1} - t_n$ and $\varepsilon \in [t_n,\, t_{n+1}]$.

Approximating $f(\varepsilon,y(\varepsilon))$ with the linear combination of $f(\varepsilon_1,y(\varepsilon_1))$, $f(\varepsilon_2,y(\varepsilon_2))$, $\ldots$, $f(\varepsilon_m,y(\varepsilon_m))$ on the interval $[t_n, t_{n+1}]$, it is possible to define the general form of the *Runge-Kutta* method i.e.

$$y_{n+1} = y_n + h \sum_{i=1}^{m} c_i\, f(\varepsilon_i, y(\varepsilon_i)) \tag{6.1.3}$$

In this thesis, the *four-stage Runge-Kutta* method has been used, namely

$$\begin{cases} y_{n+1} & = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 & = h\, f(t_n, y_n) \\ k_2 & = h\, f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ k_3 & = h\, f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\ k_4 & = h\, f(t_n + h, y_n + k_3) \end{cases} \tag{6.1.4}$$

Using Equation 6.1.4, it is possible to discretize Equations 5.1.2 and obtain

$$\boldsymbol{\xi}(k+1) = \boldsymbol{f}_k(\boldsymbol{\xi}(k), \boldsymbol{\omega}_d(k)) \tag{6.1.5}$$

The same reasoning can be applied to the unicycle Equation 3.1.1 in order to discretize the target model.

However, the approximation is accurate only if some requirements are satisfied. Let's consider a linear dynamic system of equation

$$\dot{y} = \lambda y \quad \text{with} \quad y(0) = y_0 (\neq 0),\ \lambda \in \mathbb{R}^- \tag{6.1.6}$$

with trivial convergent solution $y(t) = e^{\lambda t}$. Given such result, it is relevant to study under what conditions on the step size h the *four-stage Runge-Kutta* method reproduces this behaviour. Applying Equation 6.1.6 to Equation 6.1.4 yields

$$y_{n+1} = \left(1 + \bar{h}\frac{1}{2}\bar{h}^2 + \frac{1}{6}\bar{h}^3 + \frac{1}{24}\bar{h}^4\right) y_n \quad \text{with } n > 0,\ \bar{h} = h\lambda \tag{6.1.7}$$

and therefore

$$y_n = \left(1 + \bar{h} + \frac{1}{2}\bar{h}^2 + \frac{1}{6}\bar{h}^3 + \frac{1}{24}\bar{h}^4\right)^n y_0 \tag{6.1.8}$$

The previously defined discrete time system is stable if and only if

$$\left|1 + \bar{h} + \frac{\bar{h}^2}{2} + \frac{\bar{h}^3}{6} + \frac{\bar{h}^4}{24}\right| < 1 \Rightarrow \bar{h} \in (-2.78,\, 0) \tag{6.1.9}$$

and hence the *four-stage Runge-Kutta* method reproduces the exponential behaviour if and only if $\bar{h} \in (-2.78, 0)$ [11].

Figure 6.1 proves the convergence interval of *four-stage Runge-Kutta* method.

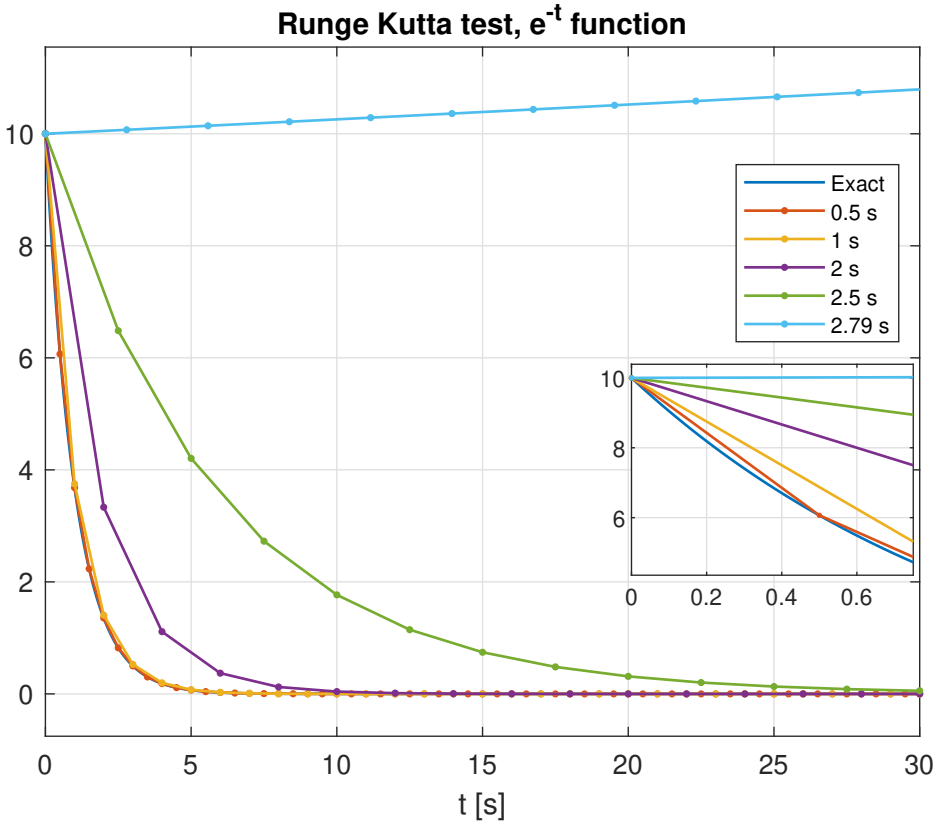


Figure 6.1: *four-stage Runge-Kutta* method test
$\dot{y} = -y$ function approximation

6.1.1 Application of Runge-Kutta Integration

In order to correctly discretize the models in Equations 5.1.2 and 5.1.3, it is necessary to identify their sample time convergence intervals for the *four-stage Runeg-Kutta* method. By computing the eigenvalues, it is possible to find a sample upper bound, i.e.

$$T_s < T_{max} = \frac{2.78}{max(\lambda)} \tag{6.1.10}$$

which has to be satisfied by the controller sample time T_s.

In general, this requirement is not met. In a widely variety of controlled systems, a cascade control architecture is used. This architecture allows the control of slow but complex task by dealing with relatively fast low level processes: moving from the top layer (high level control) to the bottom one (low level control) controller complexity and sample time are reduced.

An example of this phenomena is the agent controller scheme taken into account. The controller provides, with its own sample rate, the desired angular wheel speed $\boldsymbol{\omega}_d$ in order to change the DDR posture, while the internal PI controllers, with an higher sample rate, has to match the desired references.

Therefore, this sample time difference has to be taken into account during the discretization procedure.

Given a generic continuous time system $\dot{\boldsymbol{\xi}} = f(\boldsymbol{\xi}, \boldsymbol{u})$ and defining T_s as the high level controller sample time which does not satisfy Equation 6.1.10, the objective is to develop an approximation method to approximate $\boldsymbol{\xi}(t + T_s)$ given $\boldsymbol{\xi}(t)$ and $\boldsymbol{u}(t)$.

Let $T_{c,1}$, $T_{c,2}$, ..., $T_{c,j}$ be a set of j controller step size which satisfies Equation 6.1.10 and $\boldsymbol{f}_k(\boldsymbol{\xi}, \boldsymbol{u}, T)$ be the *four-stage Runge-Kutta* discretization with sample time T, it is possible to express $\boldsymbol{\xi}(t + T_s)$ as

$$\boldsymbol{\xi}(t + T_s) = \boldsymbol{\xi}_j \tag{6.1.11}$$

where

$$\begin{aligned} \boldsymbol{\xi}_i &= \boldsymbol{f}_k(\boldsymbol{\xi}_{i-1}, \boldsymbol{u}, T_{c,i}) \quad \text{with } i = 1, \ldots, j \\ \boldsymbol{\xi}_0 &= \boldsymbol{\xi}(t), \qquad \sum_{i=1}^{j} T_{c,i} = T_s \end{aligned} \tag{6.1.12}$$

Using these method, it is possible to simulate a T_s step time without losing accuracy. It is noteworthy that the $T_{c,i}$ can be different from the actual low level controller sample time. The pseudocode of this discretization method is reported in Algorithm 1, while a possible result is shown in Figure 6.2.

This method certainly allows a correct discretization procedure despite high sample time, at the expense of the model complexity.

Algorithm 1 Accurate simulation using *four-stage Runge-Kutta* discretization

1: $\boldsymbol{\xi} \leftarrow$ system state at time t ($\boldsymbol{\xi}(t)$)
2: $\boldsymbol{u} \leftarrow$ system input
3: $T_{c,1}$, $T_{c,2}$, ..., $T_{c,j} \leftarrow$ low level controller step
4: **for** $i \leftarrow 1$ to j **do**
5: $\quad \boldsymbol{\xi} \leftarrow \boldsymbol{f}_k(\boldsymbol{\xi}, \boldsymbol{u}, T_{c,i})$
6: **end for**
7: $\boldsymbol{\xi} \rightarrow$ system state at time $t + T_s$ ($\boldsymbol{\xi}(t + T_s)$)

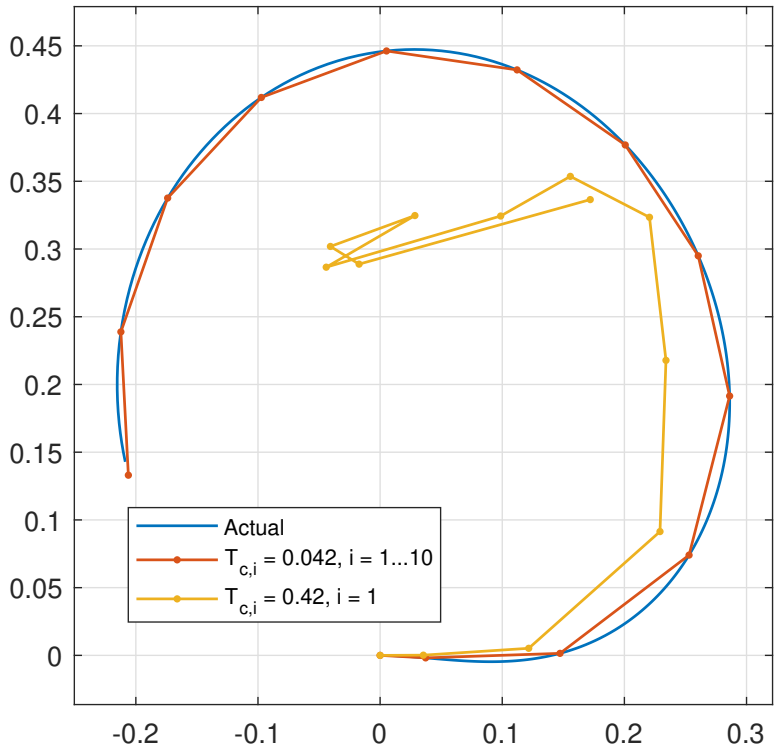


Figure 6.2: DDR simulation - $T_s = 0.42s$

6.2 NMPC Agent Controller

The NMPC controller aims to coordinate the Differential Drive Robot with the target by solving Problem 5.1.1. Therefore, to generate the NMPC cost function, it is necessary to translate the requirements in appropriate cost factors.

The optimization problem can be formulated as

$$\boldsymbol{\omega}_d^* = \underset{\hat{\boldsymbol{\omega}}_d(\cdot)}{\operatorname{argmin}} \quad J = J_d + J_\vartheta + J_{v\omega} + J_{\omega_d} \tag{6.2.1a}$$

$$\text{subject to} \quad \hat{\boldsymbol{\xi}}(k+1) = \boldsymbol{f}_k(\hat{\boldsymbol{\xi}}(k), \hat{\boldsymbol{\omega}}_d(k)) \qquad (\textit{model dynamics}) \tag{6.2.1b}$$

$$\hat{\boldsymbol{\xi}}(0) = \tilde{\boldsymbol{\xi}}(t) \qquad (\textit{initial condition}) \tag{6.2.1c}$$

$$\hat{\boldsymbol{\omega}}_d(k) \in \mathcal{U} \qquad (\textit{input constraint}) \tag{6.2.1d}$$

where J_i are the different cost factors, $\hat{\boldsymbol{\xi}}$ and $\hat{\boldsymbol{u}}$ are the simulated state and input used in the minimization procedure while $\tilde{\boldsymbol{\xi}}$ is the system state measures. In addition, with a slight abuse of notation, the model dynamic updating equation (Equation 6.2.1b) uses the method presented in Section 6.1.1.

6.3 Definition of the cost factors

The NMPC cost function is composed by four independent costs with the task of minimizing a particular sub-problem.

The first term J_d is the straight forward conversion of the Problem 5.1.1 position requirement, i.e.

$$J_d = \sum_{k=0}^{N-1} c_d \cdot \|\hat{\boldsymbol{p}}_t(k) - \hat{\boldsymbol{p}}(k)\|^2 = \sum_{k=0}^{N-1} c_d \cdot \|\hat{\boldsymbol{e}}_d(k)\|^2 \tag{6.3.1}$$

where c_d is the distance cost tuning factor and $\boldsymbol{p}_t$ is the target position. It is possible to notice that the defined cost function is convex, semi-definite positive and has minimum in zero, which is one of the problem requirements.

The second term J_ϑ tries to fulfill the Problem 5.1.1 orientation requirement. Given the unitary heading vectors $\boldsymbol{n}$ of the agent and $\boldsymbol{n}_t$ of target as in Figure 6.3, the dot product can be used to measure their line-up, i.e.

$$\boldsymbol{n}_t \cdot \boldsymbol{n} = \|\boldsymbol{n}_t\|\|\boldsymbol{n}\| \cos\left(\hat{\vartheta}_t - \hat{\vartheta}\right) = cos(\hat{\vartheta}_t - \hat{\vartheta}) = cos(\hat{e}_\vartheta) \in [-1, 1] \tag{6.3.2}$$

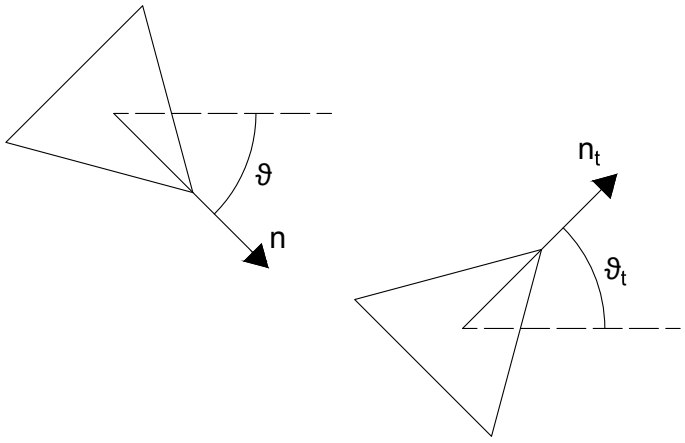


Figure 6.3: Generic iteration of the optimization algorithm

Therefore, a good candidate for the orientation alignment cost function is

$$J_\vartheta = \sum_{k=0}^{N-1} c_\vartheta \cdot (1 - \cos^2(\hat{e}_\vartheta(k))) = \sum_{k=0}^{N-1} c_\vartheta \cdot e_a(k) \tag{6.3.3}$$

where c_ϑ is the alignment cost tuning factor. It is trivial to notice that the function has minimum in zero when $\vartheta_t - \vartheta = k\pi$ with $k \in \mathbb{Z}$, as requested by the problem formulation.

Clearly, the distance and orientation errors are estimated quantities since they depend on the projection in the interval N of the agent and target states. The agent dynamic can be estimated by applying recursively Equation 6.2.1b from the initial conditions.

On the other hand, it is possible to obtain the target state projection by applying recursively the *four-stage Runge-Kutta* discretization of Equation 5.1.3.

The cost function term $J_{v\omega}$ tries to minimize the mismatch between the target and agent linear and angular velocities. Since the agent and target models are different, in order to compare those quantities, it is necessary to convert the wheels angular velocities ω_r and ω_l into v and ω by using Equation 3.1.6. Therefore, the cost function $J_{v\omega}$ can be expressed as

$$\begin{aligned} J_{v\omega} &= \sum_{k=0}^{N-1} \hat{\boldsymbol{e}}_{v\omega}^T(k)\, \boldsymbol{C}_{v\omega}\, \hat{\boldsymbol{e}}_{v\omega}(k) = \sum_{k=0}^{N-1} \|\hat{\boldsymbol{e}}_{v\omega}(k)\|^2_{\boldsymbol{C}_{v\omega}} \\ \boldsymbol{e}_{v\omega}(k) &= [v_t,\, \omega_t]^T - \text{DDR2uni}([\hat{\omega}_r(k),\, \hat{\omega}_l(k)]^T) \end{aligned} \tag{6.3.4}$$

where $\boldsymbol{C}_{v\omega}$ is the velocities tuning matrix and DDR2uni is the conversion function, embedding the DDR wheels radius r and distance d parameters.

The last term J_{ω_d} contributes to the stability of the Differential Drive Robot input. As reported in Equation 6.2.1d, the input has to belong to $\mathcal{U} = \{\boldsymbol{\omega}_d \in \mathbb{R}^2 \ : \omega_{d,min} \leq \omega_d \leq \omega_{d,max}\}$. However, there are no constraints which impose the absence of input oscillations which might damaging the actuators. For this reason, the J_{ω_d} penalizing factor is added to the NMPC cost function, i.e.

$$J_{\omega_d} = \sum_{k=0}^{N-1} \|\hat{\boldsymbol{\omega}}_d(k)\|^2_R \tag{6.3.5}$$

Having defined the structure of the cost function and all its term, it is possible to summarize the NMPC problem as

$$\begin{aligned} \boldsymbol{\omega}_d^* = \underset{\hat{\boldsymbol{\omega}}_d(\cdot)}{\text{argmin}} \quad & \sum_{k=0}^{N-1} c_d \cdot \|\hat{\boldsymbol{e}}_d(k)\|^2 + c_\vartheta \cdot e_a(k) + \|\hat{\boldsymbol{e}}_{v\omega}(k)\|^2_{\boldsymbol{C}_{v\omega}} + \|\hat{\boldsymbol{\omega}}_d(k)\|^2_R \\ \text{subject to} \quad & \hat{\boldsymbol{\xi}}(k+1) = \boldsymbol{f}_k(\hat{\boldsymbol{\xi}}(k), \hat{\boldsymbol{\omega}}_d(k)) \\ & \hat{\boldsymbol{\xi}}(0) = \tilde{\boldsymbol{\xi}}(t) \\ & \hat{\boldsymbol{\xi}}_t(k+1) = \boldsymbol{f}_{k,uni}(\hat{\boldsymbol{\xi}}_t(k), [\tilde{v}_t(t), \tilde{\omega}_t(t)]^T) \\ & \hat{\boldsymbol{\xi}}_t(0) = \tilde{\boldsymbol{\xi}}_t(t) = [\tilde{\boldsymbol{p}}_t(t),\ \tilde{\vartheta}_t(t)]^T \\ & \hat{\boldsymbol{\omega}}_d(k) \in \mathcal{U} \end{aligned} \tag{6.3.6}$$

6.3.1 Controller architecture

The controller scheme is shown in Figure 6.4 below. The NMPC controller receives as input the agent state $\tilde{\boldsymbol{\xi}}(t)$, the target state $\tilde{\boldsymbol{\xi}}_t(t)$, the number of control invariant N and the dedicated algorithm parameters. Given these information, the controller solves the optimization problem presented in Equation 6.3.6. As stated in Section 5.2.1, only the first input of the optimal input sequence is applied, i.e. $\boldsymbol{\omega}_d(t) = \boldsymbol{\omega}_d^*(0)$.

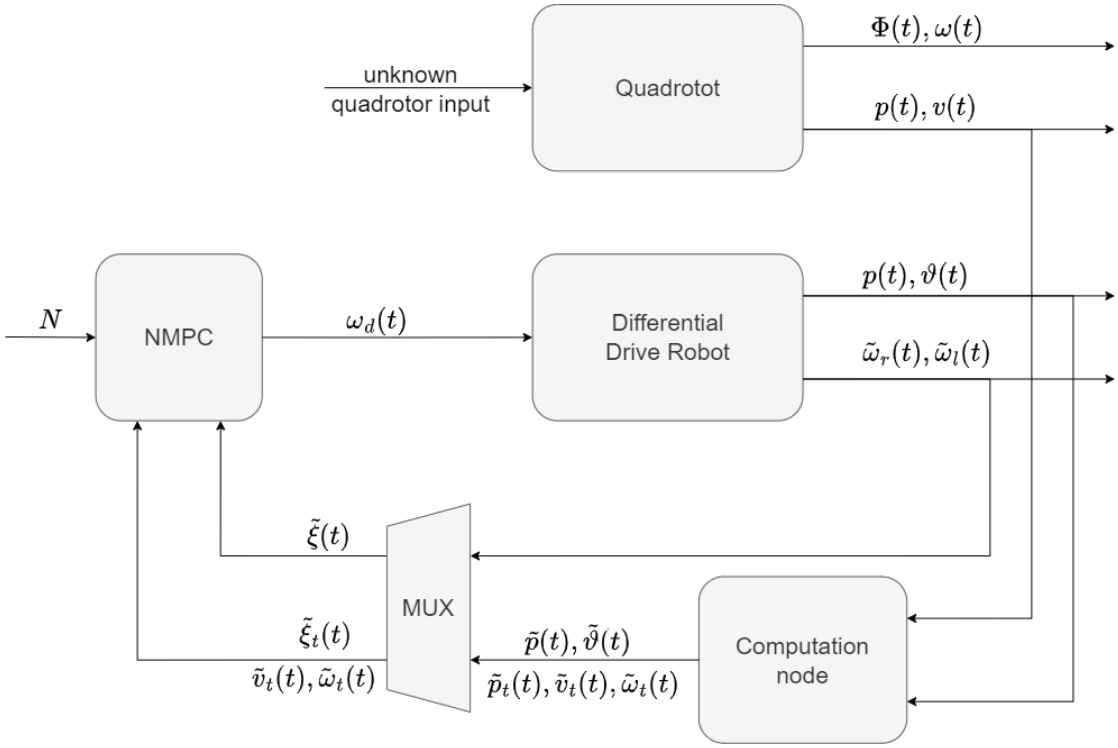


Figure 6.4: NMPC controller architecture

Chapter 7

DDR control

This Chapter is devoted the creation of the Differential Drive Robot controller that solves both *Agent-Target Coordination* and *Controller Optimization* problems. It is necessary to implement an additional controller that addresses Problem 5.1.2 since the previously defined NMPC scheme deals only with Problem 5.1.1. In the following sections are presented the tools and arguments used to implement this controller. At the end, the whole system scheme is shown, highlighting its cascade architecture.

7.1 Controller Optimization

Before solving the *Controller Optimization* problem, it is necessary to address which controllers have been used. Referring to the definition of Problem 5.1.2, these controllers have to solve the same problem and to return the same type of output $\boldsymbol{u}_i$.

Good candidates are the NMPC controllers with different control invariant index: they take as input the same quantities to compute the optimal solution, considering different time invariant $N_1 < N_2 < \cdots < N_n$.

Despite using the same initial conditions, changing the time horizon generally influences the outcome: increasing the number of control invariant N allows the algorithm to see the overall system evolution and to adapt to it at best, typically at the expense of the computational effort.

In this thesis, it is considered the following set of time invariant N

$$N_1 = 10 \quad N_2 = 30 \quad N_3 = 60 \quad N_4 = 100 \tag{7.1.1}$$

In order to solve Problem 5.1.2, the last element to define is the best solution $\bar{\boldsymbol{u}}$. Since the target trajectory is not known in advance, it is not possible to compute the best input to adjust the agent system in relation to the overall quadrotor motion. By resorting to the assumption that the unicycle model is a suitable representation of the quadrotor motion, it is possible to define the approximated best solution $\tilde{\bar{\boldsymbol{u}}}^*$ as

$$\tilde{\bar{\boldsymbol{u}}}^* = \underset{\hat{\boldsymbol{\omega}}_d(\cdot)}{\operatorname{argmin}} \sum_{k=0}^{N_{max}-1} J, \qquad N_{max} = \max_{i \in \{1,2,\dots,n\}} N_i \tag{7.1.2}$$

where the cost function is defined as in Equation 6.3.6.

Figures 7.1, 7.2 and 7.3 show three NMPC solutions with the same cost function definition and weight factors but with different initial conditions. It is possible to notice three different behaviours:

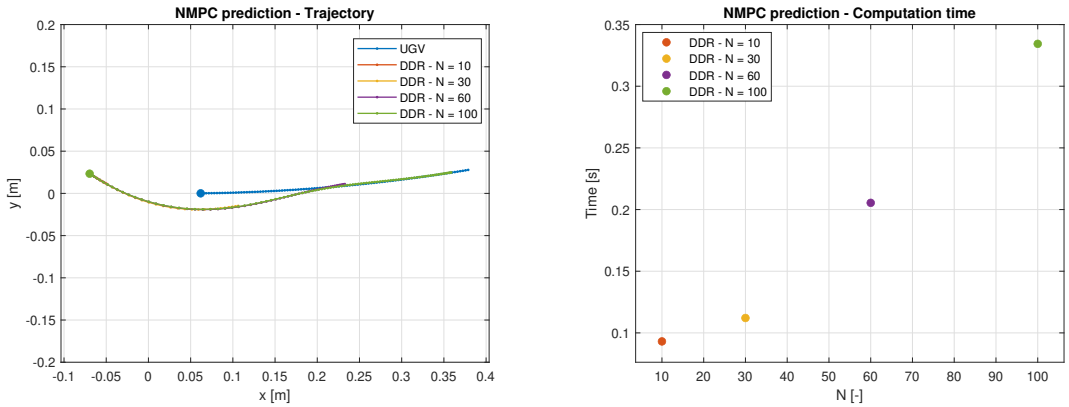
Fig. 7.1: all the controllers drive the DDR in the same way;

Fig. 7.2: there are two control strategies, performed by $N = 10, 30$ and $N = 60, 100$;

Fig. 7.3: $N = 10, 30, 60$ make the agent execute the same trajectory, which differs greatly from the $N = 100$ one.

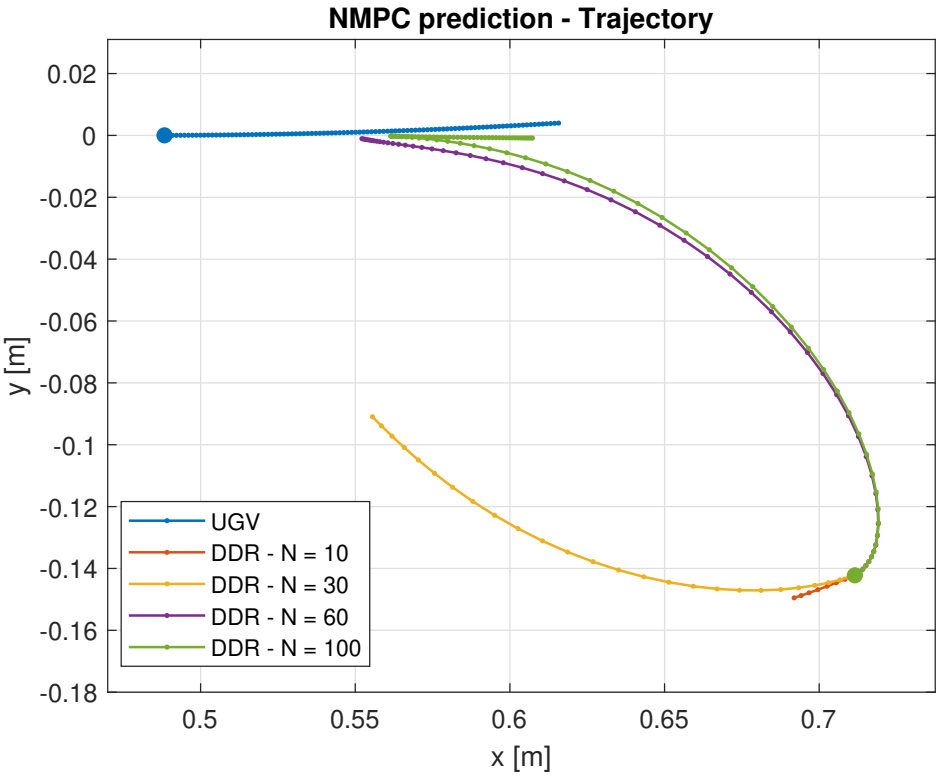
In all the examples, the solution computation time increases as the control invariant N raises.

It is therefore possible to understand the relevance of solving the *Controller Optimization* problem: when the performances of all the controllers are alike, it is convenient to resort to the less computationally demanding one (Figure 7.1), while sometimes it is needed to resort to a controller with higher control invariant to achieve the desired system requirements (Figure 7.3).

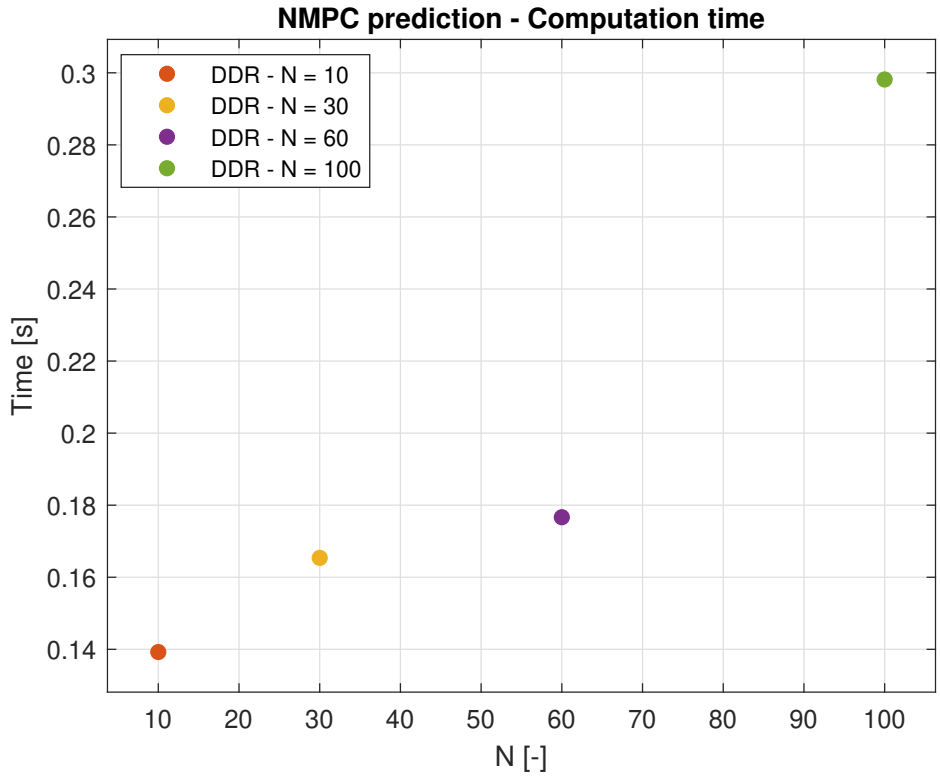


(a) Trajectory prediction (b) Solution computational time

Figure 7.1: Example of NMPC solution
Similar controller behaviours

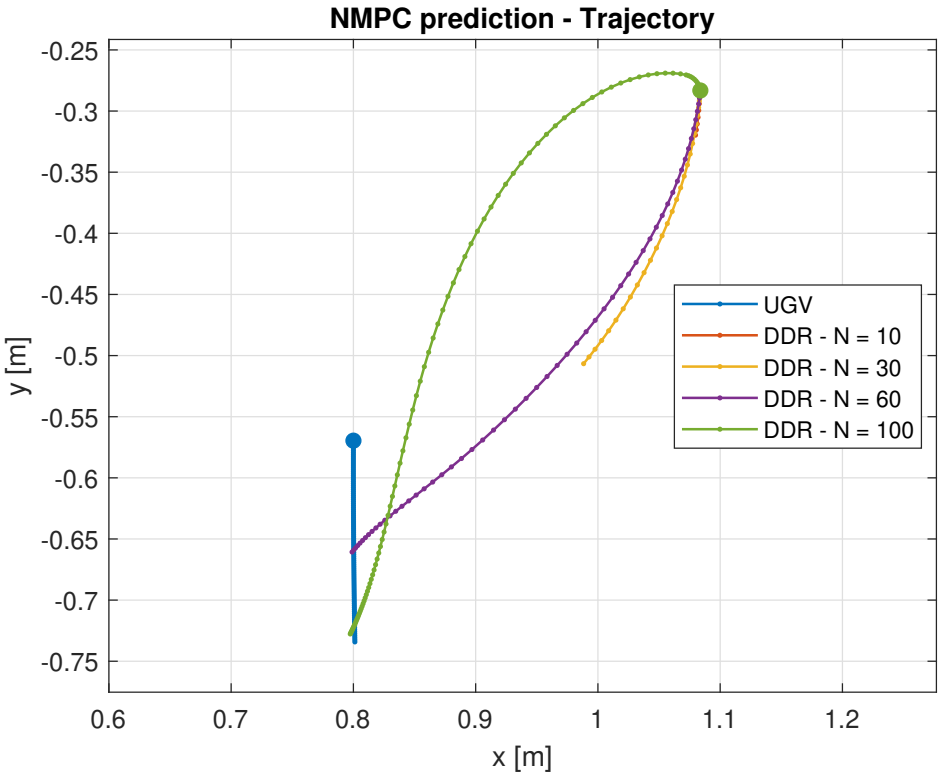


(a) Trajectory prediction

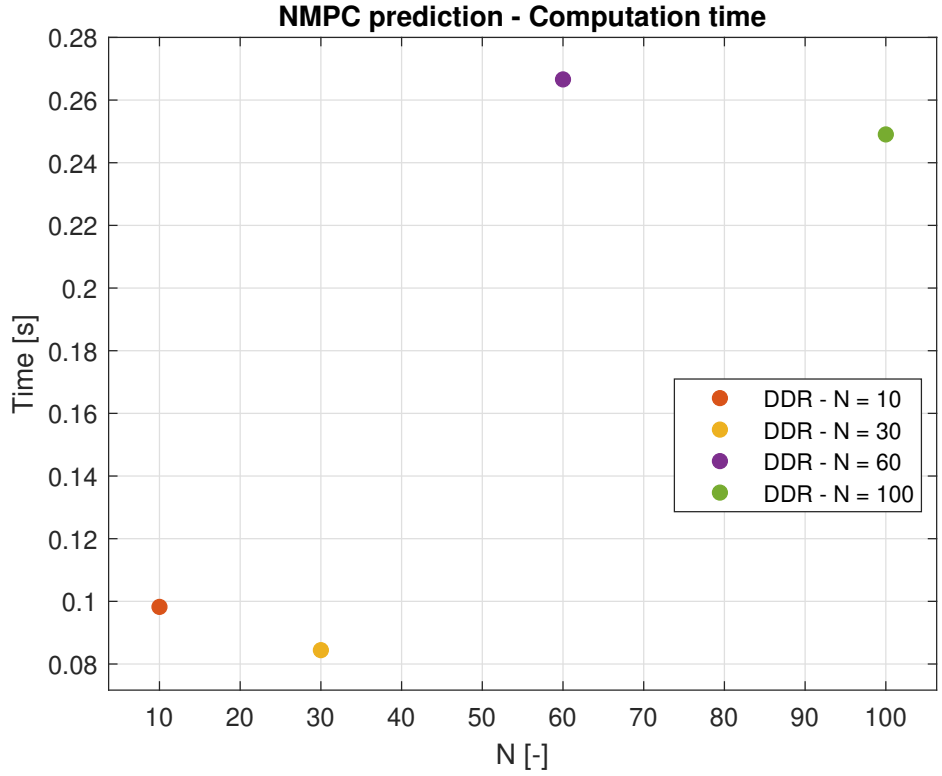


(b) Solution computational time

Figure 7.2: Example of NMPC solution
Two control strategies



(a) Trajectory prediction



(b) Solution computational time

Figure 7.3: Example of NMPC solution
Difference between $N = 10, 30, 60$ and $N = 100$ solutions

7.1.1 Controller Computational Time and Input Predictor

Determining the NMPC controllers solutions might be inefficient both from an energy and computational point of view, especially if, at the end, only one solution is adopted. Therefore, it is necessary to implement a predictor of the controllers computational time and outcome.

7.1.1.1 Gaussian Process Regression

Building a mathematical model to describe a system might be challenging due to the limited knowledge of the underlying system, complex non-linearities or the existence of unknown exogeneous phenomena impacting the input-output response. Therefore, it is relevant to develop methods for system identification using only batches of input-output data.

A possible solution is to apply the Gaussian Process theory to the regression problem. The *Gaussian Process Regression* (GPR) is a method for non-linear regression which assumes that the target function can be modelled as Gaussian Process. This methodology can be divided in two step: generate the prior and incorporate the process observation. The prior is a Gaussian vector of the form

$$\boldsymbol{f}_* \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*)) \tag{7.1.3}$$

where $\boldsymbol{X}_*$ are a set of points to predict and $\boldsymbol{K}(\cdot,\cdot)$ is the covariance matrix. Such probabilistic function represents the prior knowledge before making any observations of the actual process and it depends only on the form of the imposed covariance matrix.

By defining $\boldsymbol{f}$ and $\boldsymbol{X}$ as the process observation and training points, it is possible to express the joint distribution of $[\boldsymbol{f},\ \boldsymbol{f}_*]^T$ as

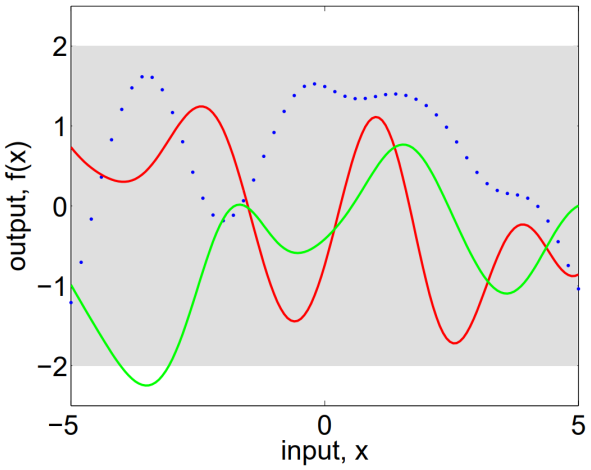
$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) \end{bmatrix}\right) \tag{7.1.4}$$

and, by using the condition distribution theory for Gaussian random vectors, it is possible obtain

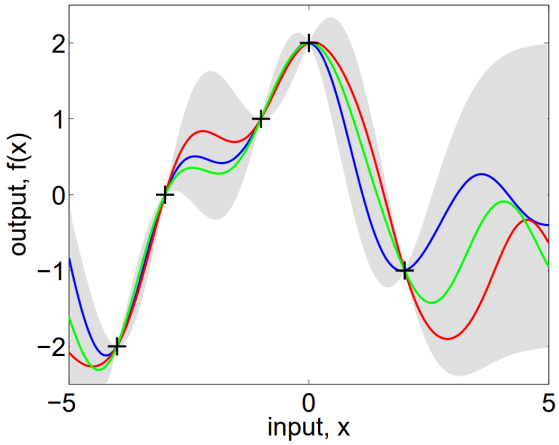
$$\boldsymbol{f}_* | \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})^{-1}\boldsymbol{f},$$
$$\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)) \tag{7.1.5}$$

The function values $\boldsymbol{f}_*$, which correspond to prediction input $\boldsymbol{X}_*$, can be sampled from the joint posterior distribution of Equation 7.1.5 by evaluating its mean vector and covariance matrix.

An example of the whole process is shown in Figure 7.4. Figure 7.4a shows different samples from the prior $\boldsymbol{f}_*$ distribution, highlighting also the 95% confidence interval; Figure 7.4b presents the posterior samples conditioned by the five "+" observations. It is possible to notice that the confidence interval shrinks around the observed points and widens the farther away from these points, returning as in the prior.



(a) Prior



(b) Posterior

Figure 7.4: GPR prior and posterior example

During the Gaussian Process Regression formulation, the choice of the covariance function $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}')$ plays a fundamental role in the predictor efficacy, since it encodes the assumption about the objective unknown function. In this thesis, the *Square Exponential* (SE) kernel is use, i.e.

$$cov(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x}')) = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}') = exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}'\|^2\right) \tag{7.1.6}$$

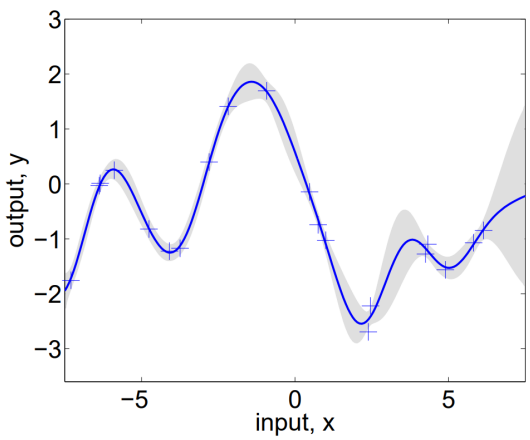
From the previous equation, it is possible to notice that the covariance of the

output is written as function of the inputs and that it encodes a distance similarity metrics, due to the use of the square norm. Beside being one of the most used covariance function within the kernel machines field, it has some relevant properties. It is infinitely differentiable, which implies that the GP is very smooth and it can be expressed as a linear combination of an infinite number of Gaussian-shaped basis function.

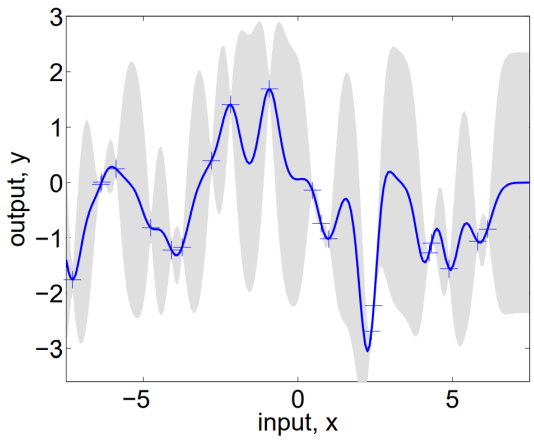
In order to adapt to different system, typically some hyperparameters are introduced in the SE kernel function. For example, considering a one dimensional SE covariance function, the kernel can be expressed as

$$k(x_p, x_q) = \sigma_f^2 \, exp\left(-\frac{1}{2l^2}(x_p - x_q)^2\right) + \sigma_n^2 \delta_{pq} \tag{7.1.7}$$

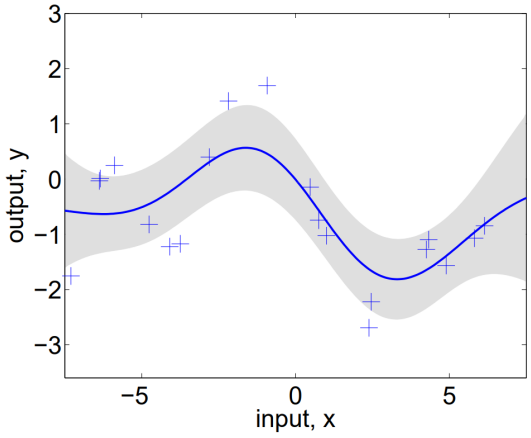
where l is the length-scale, σ_f^2 is the signal variance and σ_n^2 is the noise variance.



(a) $(l, \sigma_f, \sigma_n) = (1, 1, 0.1)$



(b) $(l, \sigma_f, \sigma_n) = (0.3, 1.08, 5 \cdot 10^{-5})$



(c) $(l, \sigma_f, \sigma_n) = (3, 1.16, 0.89)$

Figure 7.5: GPR hyperparameters examples
Samples from GP with $(l, \sigma_f, \sigma_n) = (1, 1, 0.1)$

The relevance of an accurate tuning of these hyperparameters is presented in the Figure 7.5. The "+" data are sampled from a GP with $(l, \sigma_f, \sigma_n) = (1, 1, 0.1)$. Figure 7.5a shows the results of training a Gaussian Process Regressor with the same hyperparameters. Instead, Figures 7.5b and 7.5c presents the results obtained by training it with $(l, \sigma_f, \sigma_n) = (0.3, 1.08, 5 \cdot 10^{-5})$ and $(l, \sigma_f, \sigma_n) = (3, 1.16, 0.89)$. Therefore, it is necessary to resort to some methods, such as the marginal likelihood, to score the various model and to take the best. A more comprehensive description of the Square Exponential and of other kernel functions can be found in [12].

In this thesis, the `fitrgp` [13] Matlab function has been utilized to train the GPR and to tune the hyperparameters. It is worth to point out that it takes as output a one-dimensional response vector `Y_train` and therefore it is necessary to generate a save a different GPR model for each predictor computational time and outputs.

```
GPmodel = fitrgp(X_train, Y_train, ...
        'BasisFunction', 'none', ...
        'KernelFunction','ardsquaredexponential', ...
        'Optimizer', 'fmincon');
saveLearnerForCoder(GPmodel);
```

Code 7.1: GPR training Matlab code

`KernelFunction` and `Optimizer` specifies the kernel function and how the hyperparameters are tuned. In this case, a Squared Exponential kernel with separate length per predictor is utilized as covariance function and it is supposed the presence of a constant offset in the function. The GPR hyperparameters are found by minimizing a constrained nonlinear optimization of the marginal likelihood.

7.1.1.2 Generating Training Dataset

In order to train the Gaussian Process Regressor, it is needed a training dataset. Since the controller takes as input only the agent state $\tilde{\boldsymbol{\xi}}(t)$ and the target state $\tilde{\boldsymbol{\xi}}_t(t)$ and input $\boldsymbol{u}_t(t)$, it is not essential to generate a quadrotor trajectory and simulate the whole system. Therefore, it is possible to generate randomly some hypothetical configurations and to feed them to the NMPC controllers.

Algorithm 2 shows the code used to obtain the training samples for the GPR of the NMPC controllers computational time and outputs. Each scenario is left to evolve for some steps before generating another one: this method allows the generation of more truthful data as well as increasing the training samples.

Algorithm 2 Generating training set for GPR

```
iter ← number of example to simulate
iter_e ← iterations dedicated to system evolution
for i ← 1 to iter do
    generate randomly ξ(0), v_t and ω_t
    for j ← 1 to 4 do
        for k ← 1 to iter_e do
            ω*_{d,j} ← solve 6.3.6 with N = N_j
            save ξ(k), ξ_t(k), ω*_{d,j}, v_t, ω_t and computation time T_{j,k}
            evolve DDR model using ω*_{d,j}
            evolve UGV model using v_t and ω_t
        end for
        reset DDR and UGV initial state
    end for
end for
```

To generate the training initial states $\boldsymbol{\xi}(0)$, v_t and ω_t, the following assumptions and considerations are made:

a. it is not necessary to generate both the agent and target pose; it is sufficient to impose the target on the world frame $\mathfrak{F}_W$ origin with zero orientation and to generate randomly the agent pose;

b. the target linear and angular velocities are randomly generated but do not exceed the DDR limits;

c. the wheels currents is sampled from a Gaussian distribution with mean equals to

$$i_i = \omega_i \frac{b_i}{K\Phi_i} \quad i \in r, l \tag{7.1.8}$$

which represents the current needed to balances the friction torque;

d. the integral term e_i is set randomly.

The Algorithm previously presented represents the best training samples generator but it has to provide a large amount of samples in order to sufficiently cover the possible agent-target configuration space. It is in fact possible to notice that it has to randomly generate 9 variables ($[x,\ y,\ \vartheta,\ \omega_r,\ e_{ir},\ \omega_l,\ e_{il},\ v_t,\ \omega_t]$) which have to cover the agent-target states during the simulation.

In order to assure the coverage of the target trajectory, the Algorithm is modified to generate the training samples around the desired trajectory (educated training) as shown in Algorithm 3.

Despite generating the training set around the target trajectory, the overfitting problem is avoided introducing the Gaussian noise. In addition, this "disturbance" allows also to cover some "far from trajectory" agent configurations.

Algorithm 3 Generating training set for GPR - Target trajectory coverage

$iter \leftarrow$ number of example to simulate
$iter_s \leftarrow$ number of samples for each considered target trajectory sample
$\boldsymbol{\xi}_t(k), \boldsymbol{u}_t(k) \leftarrow$ target state and input
$\boldsymbol{\sigma} \leftarrow$ samples distribution
for $i \leftarrow 1$ to $iter$ **do**
    $\boldsymbol{\omega}_{rl,t} \leftarrow$ convert $\boldsymbol{u}_t(i)$ into DDR angular wheel speed
    $\boldsymbol{i}_{rl,t} \leftarrow \omega_{rl_t} \frac{b_{rl}}{K\Phi_{rl}}$ current to balance the friction torque
    for $j \leftarrow 1$ to $iter_s$ **do**
        $\boldsymbol{\xi} \leftarrow [\boldsymbol{\xi}_t(i),\, i_{r,t},\, \omega_{r,t},\, 0,\, i_{l,t},\, \omega_{l,t},\, 0]^T + \mathcal{N}(\mathbf{0}, diag(\boldsymbol{\sigma}))$
        for $j \leftarrow 1$ to 4 **do**
            $\boldsymbol{\omega}^*_{d,j} \leftarrow$ solve 6.3.6 with $N = N_j$
            save $\boldsymbol{\xi}$, $\boldsymbol{\xi}_t(i)$, $\boldsymbol{\omega}^*_{d,j}$, $\boldsymbol{u}_t(i)$and computation time $T_{j,k}$
        end for
    end for
end for

An example of the distribution on the plane of the Algorithm 3 samples is shown in Figure 7.6. These results are obtained utilizing the parameters reported in Table 7.1.

σ_x $[m]$	σ_y $[m]$	σ_ϑ $[rad]$	σ_{i_r} $[A]$	σ_{ω_r} $[rad/s]$	$\sigma_{e_{ir}}$ $[rad]$	σ_{i_l} $[A]$	σ_{ω_l} $[rad/s]$	$\sigma_{e_{il}}$ $[rad]$
$7\,10^{-4}$	$7\,10^{-4}$	$10.1\,10^{-3}$	0.1	0.1	0.1	0.1	0.1	0.1

Table 7.1: DDR initial conditions

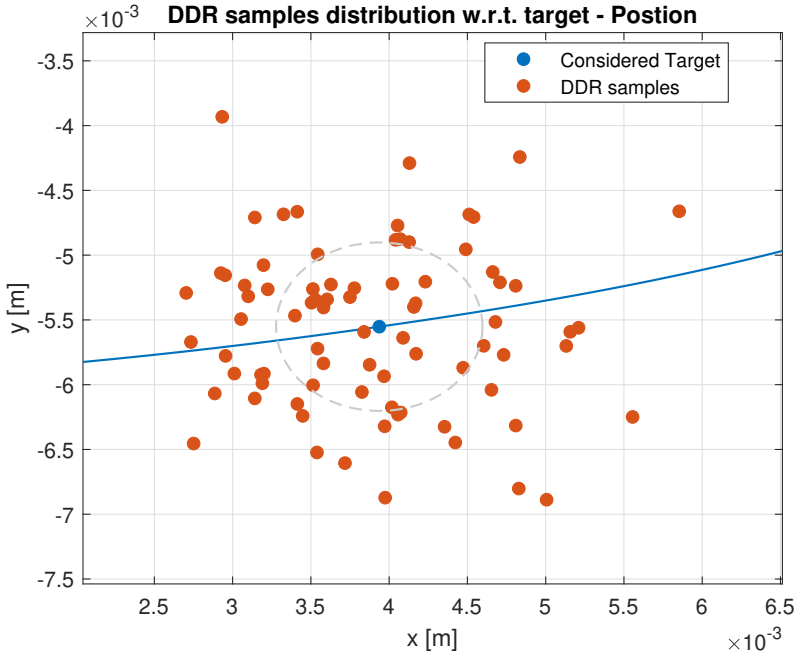


Figure 7.6: Example of training samples - $iter_s = 100$

7.1.1.3 Predictor Results

Before training the predictor, it is necessary to identify which information supply as input to the Gaussian Process Regressor. In order to generate an accurate model, these information have to represent the system status without introducing misleading details. Therefore, absolute information such as positions and orientations in the world reference frame $\mathfrak{F}_W$ have to be converted into relative measures, i.e. position and orientation relative errors. On the other hand, an excessive amount of input details will cause overfitting, namely the alignment of the predictor to the training dataset. This phenomenon makes the predictor unuseful in reference to unseen data. A good balance between model complexity and system representation can be achieved by setting as inputs $\|\boldsymbol{e}_d\|^2$, e_a, ω_r, ω_l and $\boldsymbol{u}_t$.

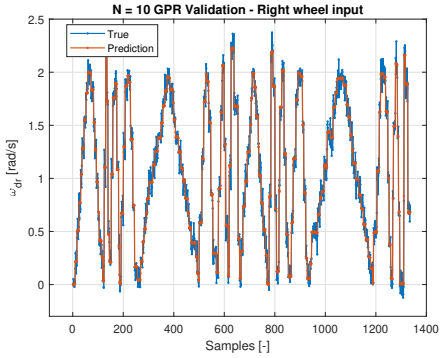
$\|\boldsymbol{e}_d\|^2$ and e_a represents the distance and the relative orientation line-up errors between the agent and the target. It is possible to notice that these quantities appear also in Equation 6.3.6 as terms of the NMPC cost function and, therefore, it is reasonable to assume their correlation with the computation time and output controller responses. ω_r, ω_l and $\boldsymbol{u}_t$ represents the DDR wheels angular velocities and the target linear and angular velocities. Such quantities are related to the vehicles motion and thus are linked to the evolution of the agent and target measures in the NMPC model updating equations.

In order to test the accuracy of the GPR, the training dataset is divided into two subsets dedicated to the actual GPR training and to validate the obtained model. The following Figures shows the validation results of the GPR model for the controller computation time and output, applying Equation 7.1.5 with $[\|\boldsymbol{e}_d\|^2, e_a, \omega_r, \omega_l, \boldsymbol{u}_t]^T$ as input vector. Since the NMPC solution depends on the DDR parameters and on the NMPC weight coefficients, Table 7.2 reports these parameters for the sake of completeness.

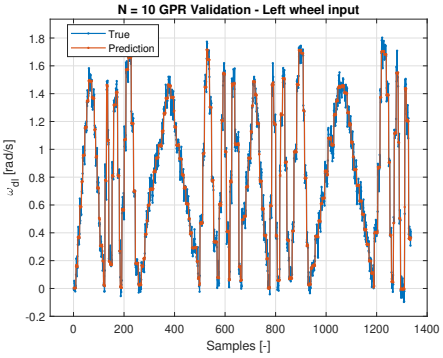
	R $[\Omega]$	L $[mH]$	$K\Phi$ $[\frac{Nm}{\sqrt{W}}]$	b $[\frac{\mu Nm}{rad/s}]$	J $[g\,m^2]$	K_p $[-]$	K_i $[-]$	r $[m]$	d $[m]$
Right	2	30	0.5	0.1	10	0.1	2	0.034	0.165
Left	1.8	20	0.6	0.2	8	0.2	1	0.034	0.165

c_d $[1/m^2]$	c_ϑ $[-]$	$\boldsymbol{C}_{v\omega}$	$\boldsymbol{R}$
$1.2\,10^4$	5	$\begin{bmatrix} 2.2\,10^{-2} & \\ & 3.1\,10^{-2} \end{bmatrix}$	$\begin{bmatrix} 5\,10^{-1} & \\ & 5\,10^{-1} \end{bmatrix}$

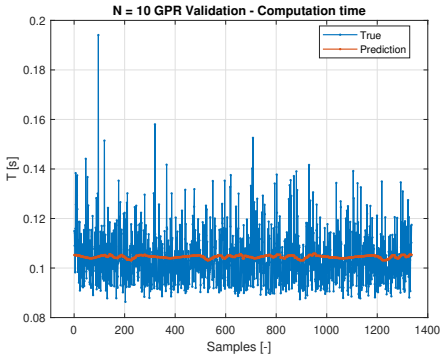
Table 7.2: DDR and NMPC parameters



(a) Right wheel angular velocity input ω_{dr}

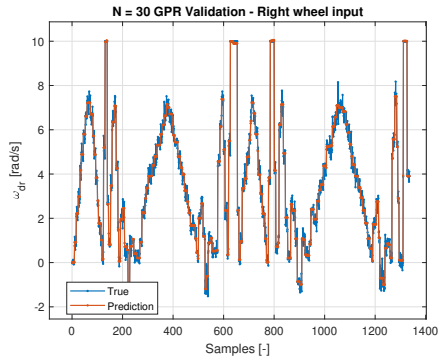


(b) Left wheel angular velocity input ω_{dl}

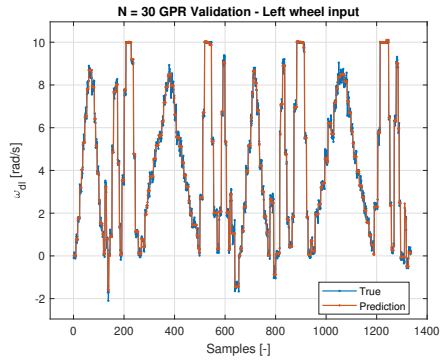


(c) Computation time T

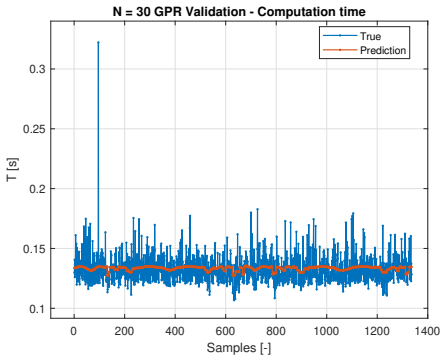
Figure 7.7: Validation $N = 10$ GPR



(a) Right wheel angular velocity input ω_{dr}

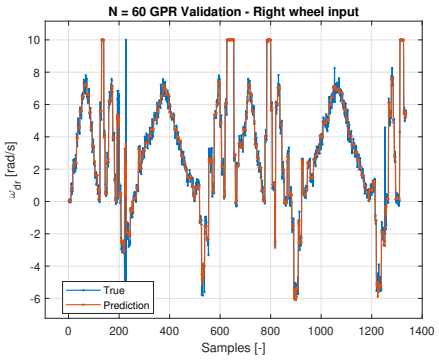


(b) Left wheel angular velocity input ω_{dl}

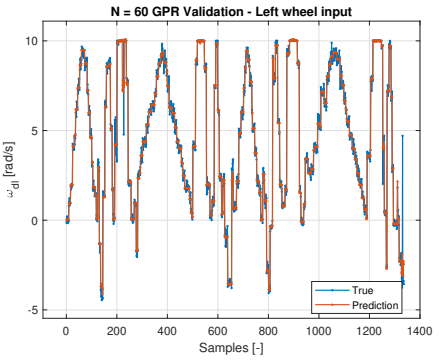


(c) Computation time T

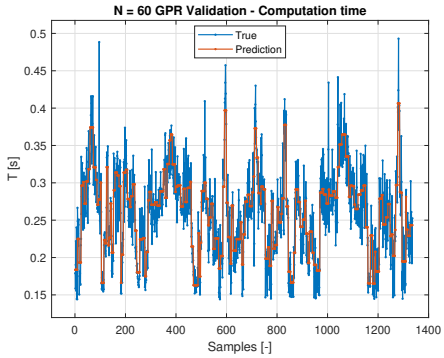
Figure 7.8: Validation $N = 30$ GPR



(a) Right wheel angular velocity input ω_{dr}

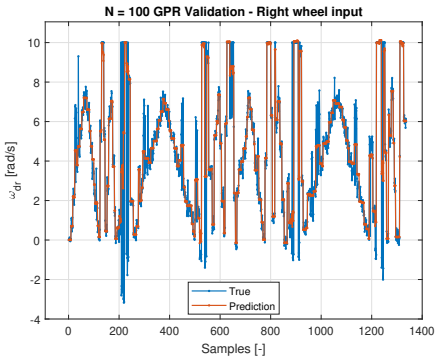


(b) Left wheel angular velocity input ω_{dl}

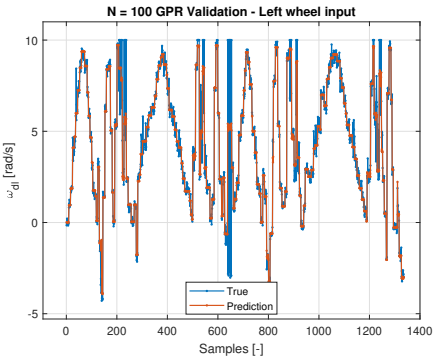


(c) Computation time T

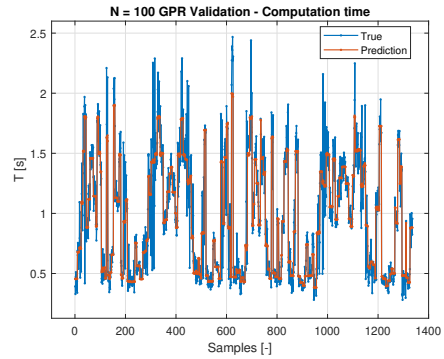
Figure 7.9: Validation $N = 60$ GPR



(a) Right wheel angular velocity input ω_{dr}



(b) Left wheel angular velocity input ω_{dl}



(c) Computation time T

Figure 7.10: Validation $N = 100$ GPR

It is possible to notice that a single set of NMPC weighting factors has been used for all the controllers. It is possible to set c_d and c_ϑ constant since all the controllers have the same task and since they modifies the importance of the Problem 5.1.1 requirements in the NMPC cost function. A similar reasoning can be applied to the input matrix $\boldsymbol{R}$. As concerns $\boldsymbol{C}_{v\omega}$, scaling this matrix as the control invariant increase implies the normalization of the velocity error factor with respect to the time horizon. This adjustment leads to the deletion of the kinematic information in the cost function. Therefore, by imposing c_d, c_ϑ and $\boldsymbol{R}$, it is necessary to set also $\boldsymbol{C}_{v\omega}$ constant.

Despite the presence of some outliers in all the graphs, it is possible to observe how the Gaussian Process Regressor performs almost perfectly in all the Figures. It is worth noticing that in Figures 7.7c and 7.8c, the GPR predict almost a constant value despite the presence of oscillation. This behaviour can be attributed to the hyperparameters estimation: the GPR has identified as noise all the oscillations in the computation time and therefore it has "filtered out" these fluctuations.

To compare the effectiveness of the prediction, it is introduced a validation metric, namely the Empirical Risk Function or Mean Square Error (MSE)

$$L_s = \frac{1}{|S|}\sum_{i=1}^{|S|}(y(i) - GPR(x(i)))^2 \tag{7.1.9}$$

where x and y are the validation set input and output, GPR is the Gaussian Process Regressor and $|S|$ is the dataset dimension.

In this thesis, the training and validation datasets are composed respectively by 12024 and 1336 samples. Table 7.3 confirms the graphical considerations.

N	$L_{s,\omega_{dr}}$	$L_{s,\omega_{dl}}$	$L_{s,T}$
10	0.0048	0.026	0.0001
30	0.0558	0.0284	0.0002
60	0.5251	0.1405	0.0007
100	2.0091	0.8054	0.0317

Table 7.3: MSE values for each time invariant N in predicting ω_{dr}, ω_{dl} and T

Taking the root of the Empirical Risk L_s, it is possible to compute the Root Mean Square Error. It provides the standard deviation of the residuals and represent how spread out the predictions are with respect to the true values.

From the previously presented results, it is possible to infer how the control invariant N is correlated to the NMPC controller output. In fact, as N increases, the

N	$\hat{\sigma}_{\omega_{dr}}$	$\hat{\sigma}_{\omega_{dl}}$	$\hat{\sigma}_T$
10	0.0693	0.0507	0.0108
30	0.2363	0.1684	0.0123
60	0.7246	0.3748	0.0274
100	1.4174	0.8974	0.1780

Table 7.4: RMSE values for each time invariant N in predicting ω_{dr}, ω_{dl} and T

standard deviation of the residuals increases, meaning that there is more variability in the data. This can be traced back to the fact that smaller variation in the NMPC initial conditions can generate unlike controller response due to the expansion of the time horizon.

7.2 Controller architecture

Figure 7.11 shows the Differential Drive Robot controller that solves both *Agent-Target Coordination* and *Controller Optimization* problems. The red box highlights the prediction module: it takes as input $\tilde{\boldsymbol{\xi}}(k)$, $\tilde{\boldsymbol{\xi}}_t(k)$ and $\tilde{\boldsymbol{u}}_t(k)$, it computes $[\|\boldsymbol{e}_d\|, e_a, \omega_r, \omega_l, \boldsymbol{u}_t]^T$ and by minimizing

$$n^* = \underset{i \in 1,\dots,n}{\operatorname{argmin}} (\boldsymbol{u}_i - \tilde{\bar{\boldsymbol{u}}})^T \boldsymbol{C}_u (\boldsymbol{u}_i - \tilde{\bar{\boldsymbol{u}}}) + c_T T_i^2 + J_{ext} \tag{7.2.1}$$

it provides as output N_n. The blue box identifies the actual DDR controller scheme while the green one identifies the computation node, as presented in Chapters 4 and 6.

It is possible to notice that the system uses a cascade architecture: the prediction module solves a high level task and provide a reference, namely N_n, to a lower level controller, dedicated to the actual system control.

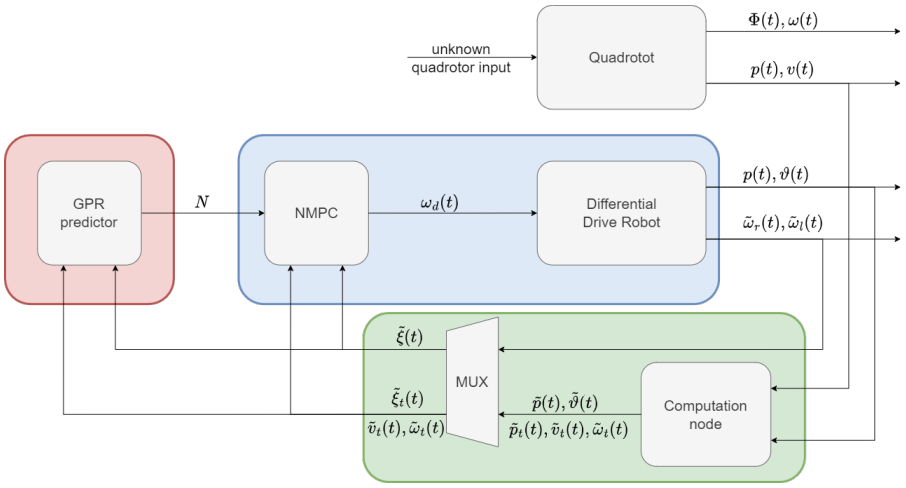


Figure 7.11: DDR controller

Chapter 8

Simulation and results

This Chapter is devoted to validate the proposed controller for the *Agent-Target Coordination* and *Controller Optimization* problems, which are presented in Chapter 5. In order to validate the designed algorithm, a Matlab simulation environment is utilized. The controller is tested using the quadrotor trajectory shown in Section 3.2.

8.1 Matlab frameworks

Before analyzing the obtained results, it is worth describing the general set up and tool used. As previously stated, all the computation and simulation are performed using Matlab framework. In order to implement the NMPC, the *CasADi* tool has been utilized.

CasADi [14] is an open-source tool for nonlinear optimization and algorithmic differentiation. It is equipped with a symbolic framework implementing forward and reverse mode of algorithmic differentiation on expression graphs in order to construct gradient, large-and-sparse Jacobian and Hessians. It also provides the tools to encapsulate the desired symbolic algorithm into Function objects, which can be exported to stand-alone C code.

In this thesis, *CasADi* has been utilized to implement the ordinary differential equation (ODE) of the unicycle and DDR models and the Nonlinear Model Predictive Control cost function. They are then converted into `mex` functions in order to have fast execution time, which arises from the advantage of compiled functions.

To solve the NMPC minimization problem, the `fmincon`[15] Matlab function is

utilized. This function finds the minimum of a problem specified as

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \qquad \text{such that} \quad \begin{cases} c(\boldsymbol{x}) \leq 0 \\ c_{eq}(\boldsymbol{x}) = 0 \\ \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b} \\ \boldsymbol{A}_{eq} \cdot \boldsymbol{x} \leq \boldsymbol{b}_{eq} \\ \boldsymbol{lb} \leq \boldsymbol{x} \leq \boldsymbol{ub} \end{cases} \tag{8.1.1}$$

where $\boldsymbol{c}(\boldsymbol{x})$ and $\boldsymbol{c}_{eq}(\boldsymbol{x})$ are functions that return vectors while $f(\boldsymbol{x})$ returns a scalar. Therefore, Equation 6.3.6 can be reformulated as

$$\begin{aligned} \boldsymbol{\omega}_d^* = \underset{\hat{\boldsymbol{\omega}}_d(\cdot)}{\operatorname{argmin}} \quad & \sum_{k=0}^{N-1} V(\hat{\boldsymbol{\xi}}(k), \hat{\boldsymbol{\xi}}_t(k), \boldsymbol{u}_t(t), \hat{\boldsymbol{\omega}}_d(k)) \\ \text{subject to} \quad & \hat{\boldsymbol{\xi}}(k+1) = \boldsymbol{f}_k(\hat{\boldsymbol{\xi}}(k), \hat{\boldsymbol{\omega}}_d(k)) \\ & \hat{\boldsymbol{\xi}}(0) = \tilde{\boldsymbol{\xi}}(t) \\ & \hat{\boldsymbol{\xi}}_t(k+1) = \boldsymbol{f}_{k,uni}(\hat{\boldsymbol{\xi}}_t(k), [\tilde{v}_t(t), \tilde{\omega}_t(t)]^T) \\ & \hat{\boldsymbol{\xi}}_t(0) = \tilde{\boldsymbol{\xi}}_t(t) = [\tilde{\boldsymbol{p}}_t(t),\ \tilde{\vartheta}_t(t)]^T \\ & \boldsymbol{\omega}_{d,min} \leq \hat{\boldsymbol{\omega}}_d(k) \leq \boldsymbol{\omega}_{d,max} \end{aligned} \tag{8.1.2}$$

where $V(\cdot, \cdot)$ is given by

$$\begin{aligned} V(\hat{\boldsymbol{\xi}}(k), \hat{\boldsymbol{\xi}}_t(k), \boldsymbol{u}_t(t), \hat{\boldsymbol{\omega}}_d(k)) = c_d \cdot \|\hat{\boldsymbol{e}}_d(k)\|^2 + \\ + c_\vartheta \cdot e_a(k) + \|\hat{\boldsymbol{e}}_{v\omega}(k)\|^2_{\boldsymbol{C}_{v\omega}} + \|\hat{\boldsymbol{\omega}}_d(k)\|^2_R \end{aligned} \tag{8.1.3}$$

All the cost function components are described in Section 6.3. Code 8.1 reports the snipped of code containing the `fmincon` usage and setting parameters.

```
option = optimoptions('fmincon', 'Display', 'none', ...
        'Algorithm', 'sqp', 'SpecifyObjectiveGradient', true);
...
U_opt = fmincon(@(U) cost_f(U), U_guess, ...
        [], [], [], [], lb, ub, [], option);
```

Code 8.1: `fmincon` usage and setting parameters

The `'Algorithm'` option sets the solver for the minimization.

In this thesis, the *Sequential Quadratic Programming* or SQP method is used. This algorithm is widely used since it satisfies bounds at all iterations and can recover from `NaN` or `Inf` results.

In addition, `cost_f` provides also the gradient of the cost function in order to improve the minimization performance. By setting `'SpecifyObjectiveGradient'` true, the `fmincon` utilizes also this information during the minimization procedure.

At last, `U_guess` is the initial point from which the algorithm starts to solve the problem. Due to the problem high complexity, the choice of this initial guess might influence the computational time needed and therefore it is always set to the input upper bound, to consistently be in the same conditions both in the training and in the test.

8.2 Results

This section is dedicated to report the results obtained for the developed controller. Below it is possible to find the parameters of the agent and target models, of the NMPC and of the Controller Optimization. In addition, Table 8.5 presents the initialization values of the DDR, indicating also the develop controller sample time T_s. Recalling Section 6.1.1, T_s has to satisfy the convergence interval otherwise it is necessary to resort to additional computation to obtain an accurate system discretization. In this simulation, to keep consistency to the real-world scenario, T_s is set to $0.04s$; this value meets the convergence interval requirements directly. At last, the trajectory preformed by the quadrotor is the same as the one shown in Section 3.2.

m	J_x	J_y	J_z	l	c_f	c_τ
$[kg]$	$[kg\,m^2]$	$[kg\,m^2]$	$[kg\,m^2]$	$[m]$		
1.5	0.029125	0.029125	0.055225	0.2555	5.84μ	0.06

(a) Model parameters

$\boldsymbol{K}_{pp}$	$\boldsymbol{K}_{ip}$	$\boldsymbol{K}_{dp}$	$\boldsymbol{K}_{da}$	$\boldsymbol{K}_{pa}$
$\begin{bmatrix} 50 & & \\ & 50 & \\ & & 50 \end{bmatrix}$	$\begin{bmatrix} 0.1 & & \\ & 0.1 & \\ & & 0.5 \end{bmatrix}$	$\begin{bmatrix} 15 & & \\ & 15 & \\ & & 10 \end{bmatrix}$	$\begin{bmatrix} 150 & & \\ & 150 & \\ & & 50 \end{bmatrix}$	$\begin{bmatrix} 15 & & \\ & 15 & \\ & & 10 \end{bmatrix}$

(b) Position and yaw angle controller parameters

Table 8.1: Quadrotor parameters

	R	L	$K\Phi$	b	J	K_p	K_i	r	d
	$[\Omega]$	$[mH]$	$[\frac{Nm}{\sqrt{W}}]$	$[\frac{\mu Nm}{rad/s}]$	$[g\,m^2]$	$[-]$	$[-]$	$[m]$	$[m]$
Right	2	30	0.5	0.1	10	0.1	2	0.034	0.165
Left	1.8	20	0.6	0.2	8	0.2	1	0.034	0.165

Table 8.2: DDR model and controller parameters

c_d	c_ϑ	$\boldsymbol{C}_{v\omega}$	$\boldsymbol{R}$
$[1/m^2]$	$[-]$		
$1.2\,10^4$	5	$\begin{bmatrix} 2.2\,10^{-2} & \\ & 3.1\,10^{-2} \end{bmatrix}$	$\begin{bmatrix} 5\,10^{-1} & \\ & 5\,10^{-1} \end{bmatrix}$

Table 8.3: NMPC cost parameters

$\boldsymbol{C}_u$	c_T	T_{ext}	c_{ext}
	$[-]$	$[s]$	$[-]$
$\begin{bmatrix} 10 & \\ & 10 \end{bmatrix}$	50	1	200

Table 8.4: Controller Optimization parameters

$\boldsymbol{p}(0)$	$\vartheta(0)$	$i_r(0)$	$\omega_r(0)$	$e_{i,r}(0)$	$i_l(0)$	$\omega_l(0)$	$e_{i,l}(0)$	ω_{min}	ω_{max}	T_s
$[m]$	$[rad]$	$[A]$	$[\frac{rad}{s}]$	$[rad]$	$[A]$	$[\frac{rad}{s}]$	$[rad]$	$[\frac{rad}{s}]$	$[\frac{rad}{s}]$	$[s]$
$[0,\,0]^T$	0	0	0	0	0	0	0	-10	10	0.04

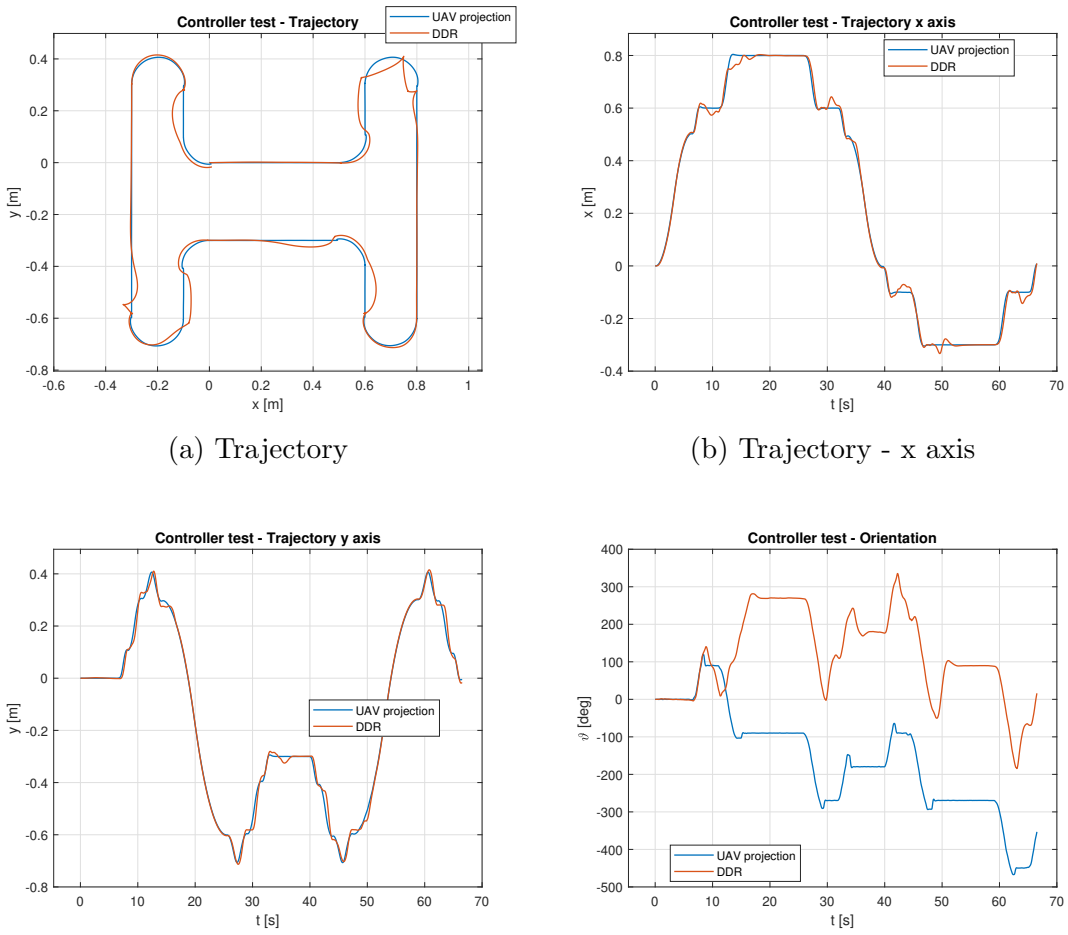
Table 8.5: DDR initial conditions

Figure 8.1 shows the effectiveness of the NMPC controller in solving the *Agent-Target Coordination* problem.

It is possible to notice that the most difficult trajectory sections are the arc of circumference ones. This behaviour can be attributed to the linear and angular velocity profile of the target: the ratio between v_t and ω_t remains the same, fixing the Instantaneous Center of Rotation (ICR), but the target velocities are increasing. In addition, ω_t contains some computational noises which affects the controller solution.

In Figure 8.1d, it is possible to notice the UAV projection and the DDR orientation profiles. Despite ϑ and ϑ_t match almost perfectly up to 10 seconds, in all the remaining part of the simulation the orientations follow the same evolution but

without overlapping. By computing $e_\vartheta = \vartheta_t - \vartheta$, it is possible to notice that it fluctuates around 0 or 360°, i.e. the DDR makes a spin on itself and then align to the target. This behaviour is induced by the combination of the alignment and velocities cost terms in the NMPC cost function: the e_a term imposes the alignment of the DDR with the UAV projection without taking into account the direction while $e_{v\omega}$ requires the matching of the two vehicles velocities. Therefore, it follows that in general if the agent has a different direction from the target, the velocity term tries to align it.



(a) Trajectory

(b) Trajectory - x axis

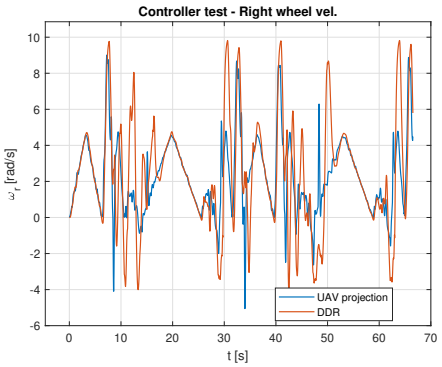
(c) Trajectory - y axis

(d) Orientation

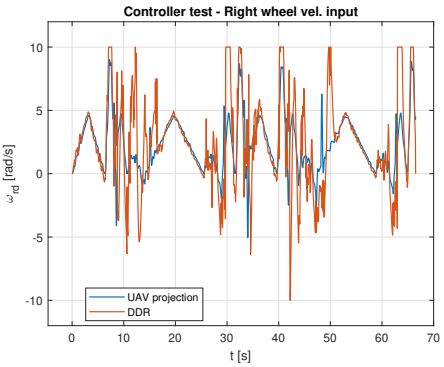
Figure 8.1: Controller simulation - pose results

Figure 8.2 presents the agent actual and desired wheels velocities. It is possible to notice that the velocities overlap during the slow evolution while oscillate when disturbances or rapid changes in the target velocity are presents.

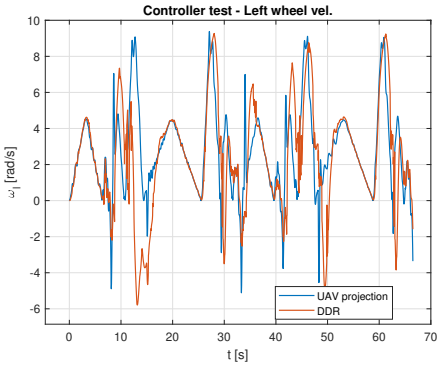
In Figure 8.3 presents the distance, alignment, linear and angular velocities errors. As designed in Section 6.3, the NMPC control the agent in order to minimize to zero these errors.



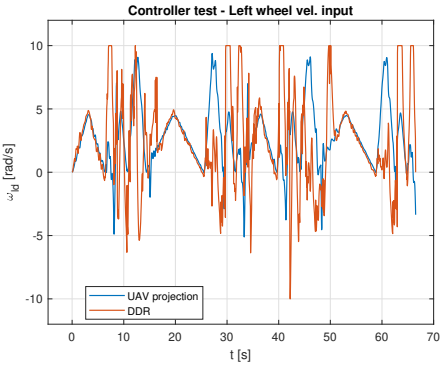
(a) Right wheel vel. ω_r



(b) Right wheel vel. input ω_{dr}

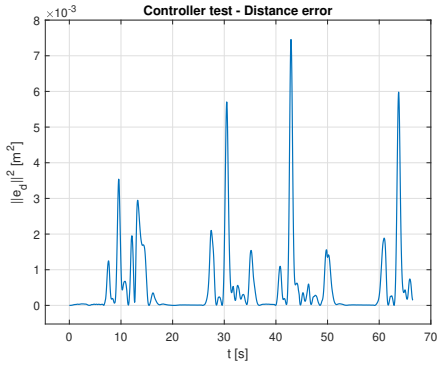


(c) Left wheel vel. ω_l

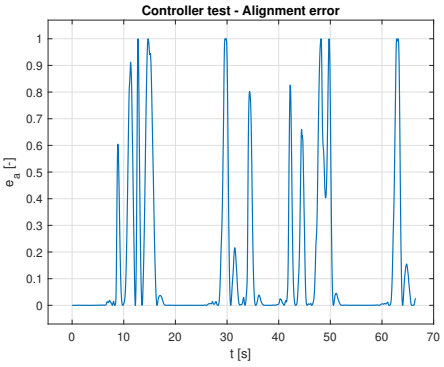


(d) Left wheel vel. input ω_{dl}

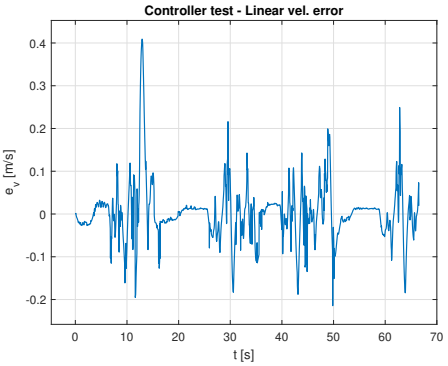
Figure 8.2: Controller simulation - wheels velocities



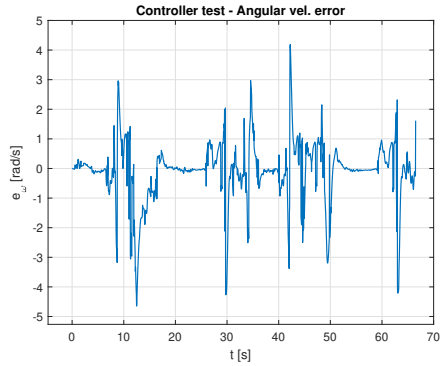
(a) Distance error $\|\boldsymbol{e}_d\|^2$



(b) Alignment error e_a



(c) Linear velocity error e_v



(d) Angular velocity error e_ω

Figure 8.3: Controller simulation - errors

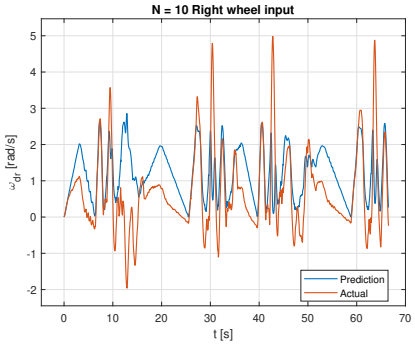
The following Figures present the comparison between the actual NMPC and the GPR prediction results. Analyzing the graphics, it is possible to state that the GPR is an effective method to predict the NMPC outputs: the regressor is able to provide reliable data which follow the actual trends as proved by Figures 8.5b, 8.6a and 8.6c.

On the other hand, Figures 8.5a, 8.5c and 8.7a show that the GPR returns zeros in some cases, despite being able to correctly predict the other controllers outputs. By analyzing the theory and the code presented in Section 7.1.1.1, it is possible to attribute this behaviour to the Gaussian Kernel prior and to the hyperparameters estimation.

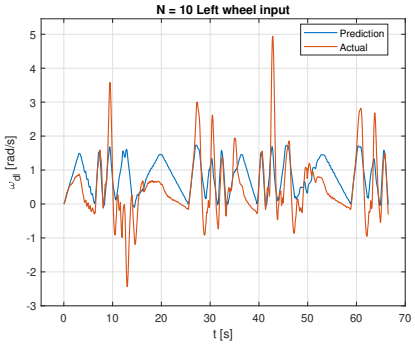
In this thesis, it is assumed to use a Square Exponential kernel with zero mean, i.e.

$$\boldsymbol{f}_* \sim \mathcal{N}(\mathbf{0}, \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*)) \tag{8.2.1}$$

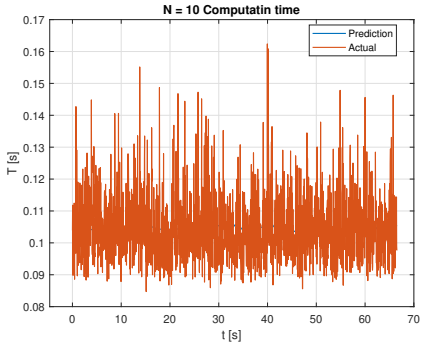
which represent the predictor function when the input data point is too distant from the training dataset values. Despite using the same training input samples for all the controllers predictor, the `figrgp` Matlab function computes independently the hyperparameters, namely the the length-scale l, the signal variance σ_f^2 and the noise variance σ_n^2. Therefore, due the optimization of these parameters, it is possible that for some predictors the input point is "near" and for other it is "far" from the training examples, due to different length scale l settings.



(a) Right wheel angular velocity input ω_{dr}

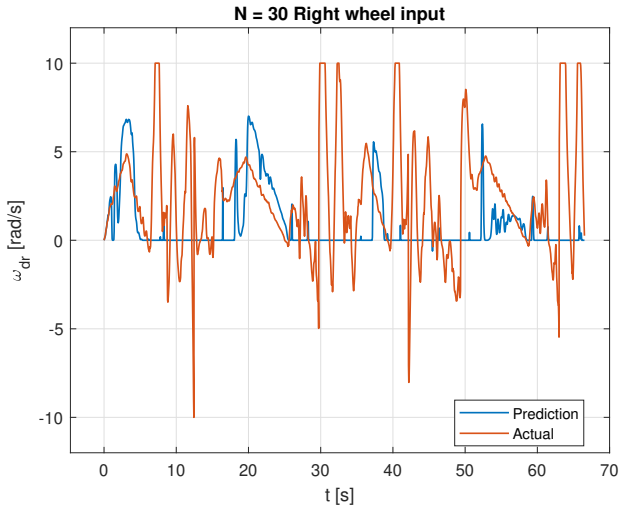


(b) Left wheel angular velocity input ω_{dl}

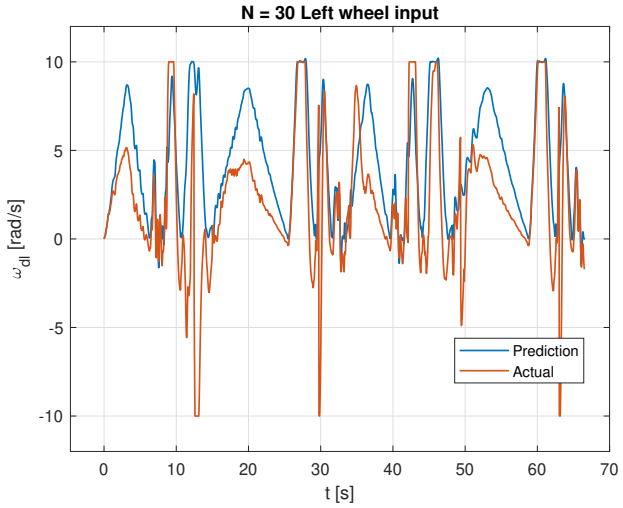


(c) Computation time T

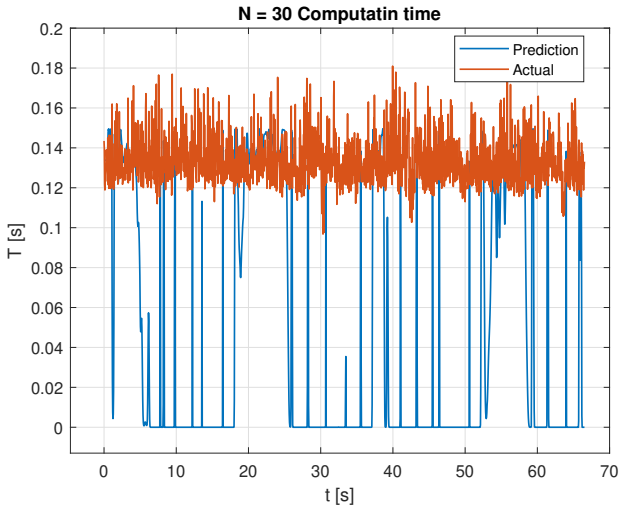
Figure 8.4: Controller simulation - Actual vs $N = 10$ GPR results



(a) Right wheel angular velocity input ω_{dr}

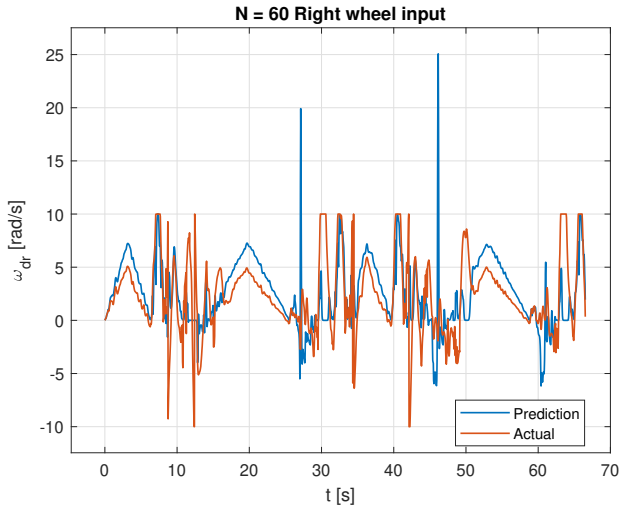


(b) Left wheel angular velocity input ω_{dl}

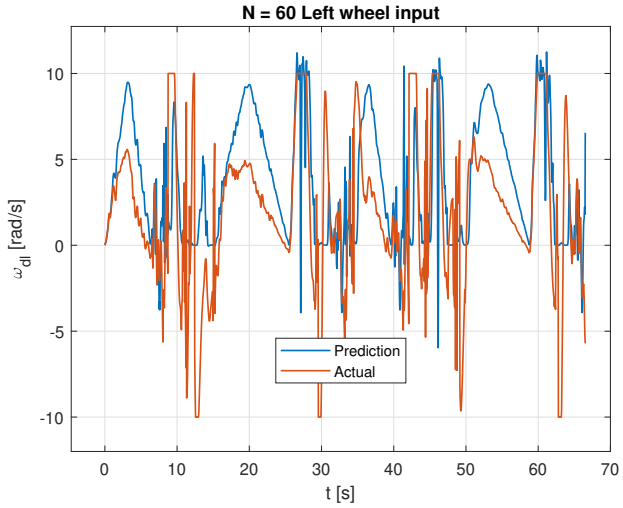


(c) Computation time T

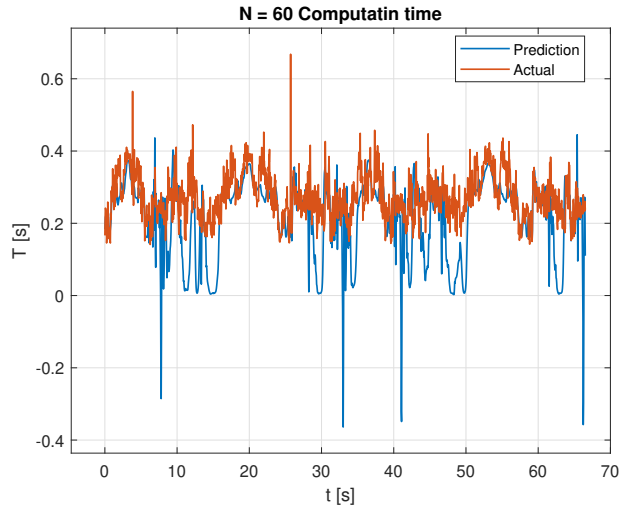
Figure 8.5: Controller simulation - Actual vs $N = 30$ GPR results



(a) Right wheel angular velocity input ω_{dr}

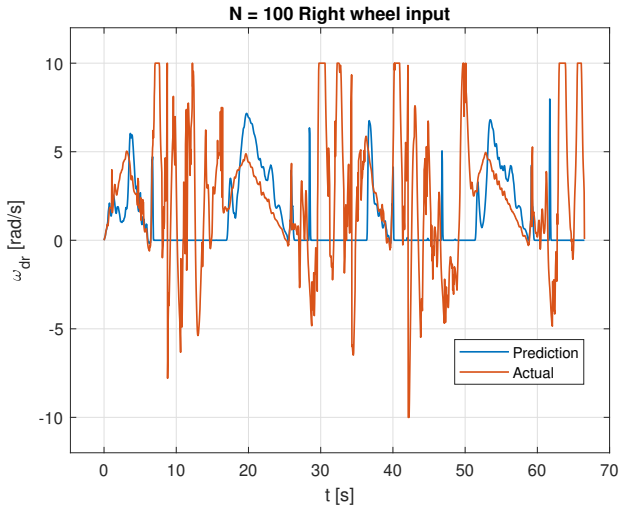


(b) Left wheel angular velocity input ω_{dl}

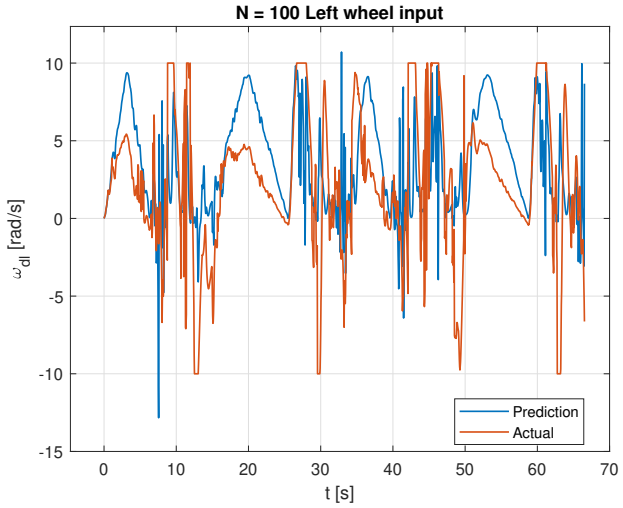


(c) Computation time T

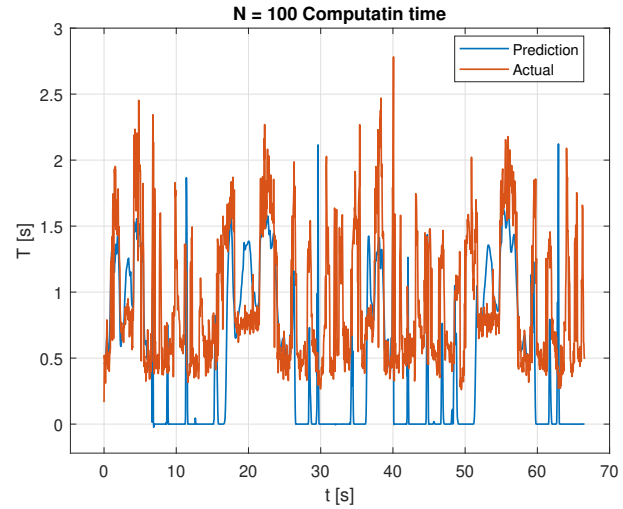
Figure 8.6: Controller simulation - Actual vs $N = 60$ GPR results



(a) Right wheel angular velocity input ω_{dr}



(b) Left wheel angular velocity input ω_{dl}



(c) Computation time T

Figure 8.7: Controller simulation - Actual vs $N = 100$ GPR results

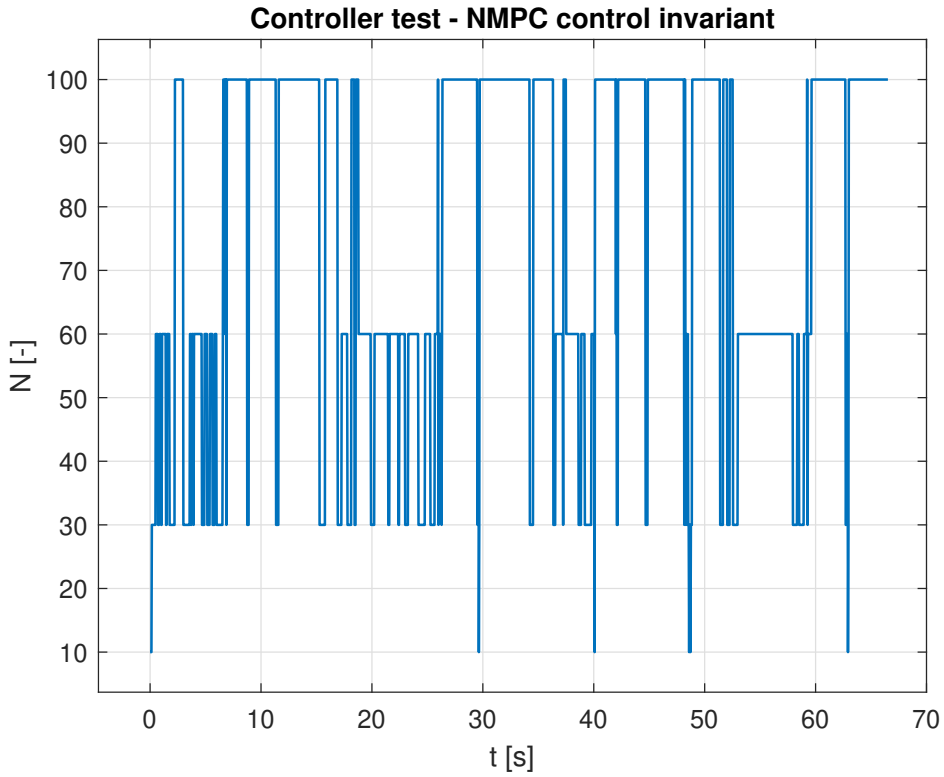


Figure 8.8: Controller simulation - Control invariant N

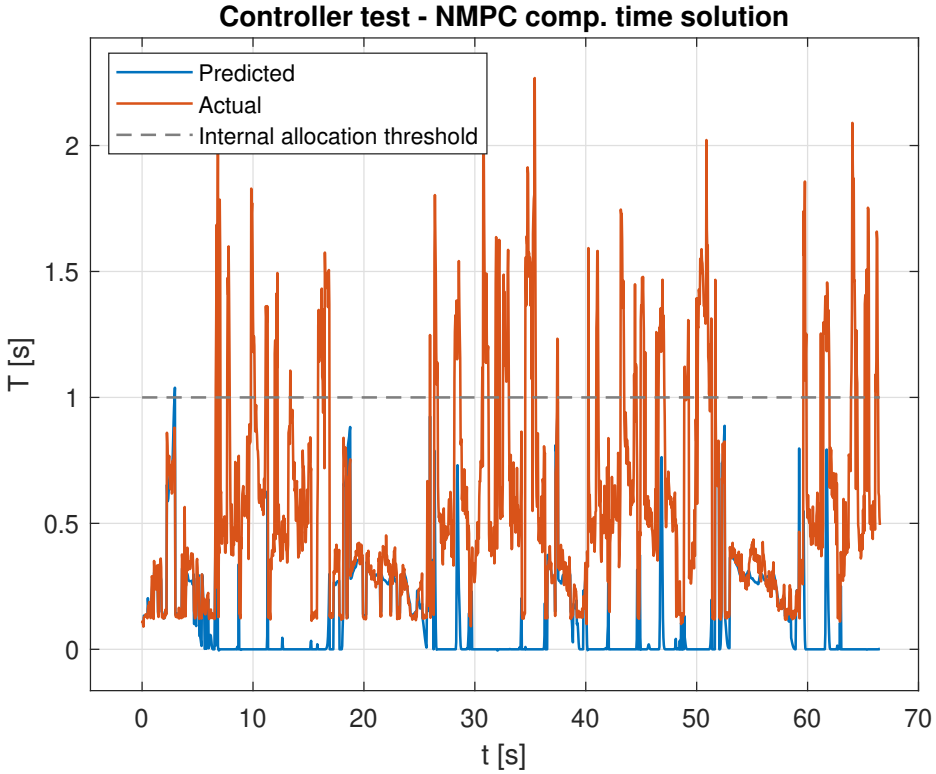


Figure 8.9: Controller simulation
NMPC computational time of the chosen control invariant N

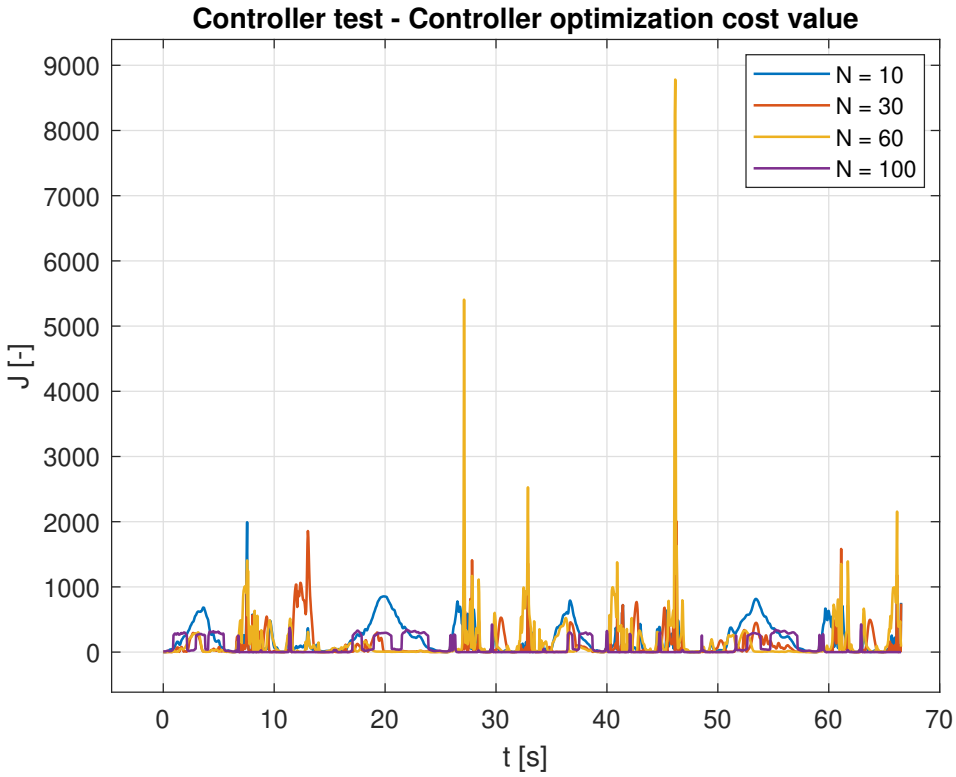


Figure 8.10: Controller simulation
Controller Optimization cost function values

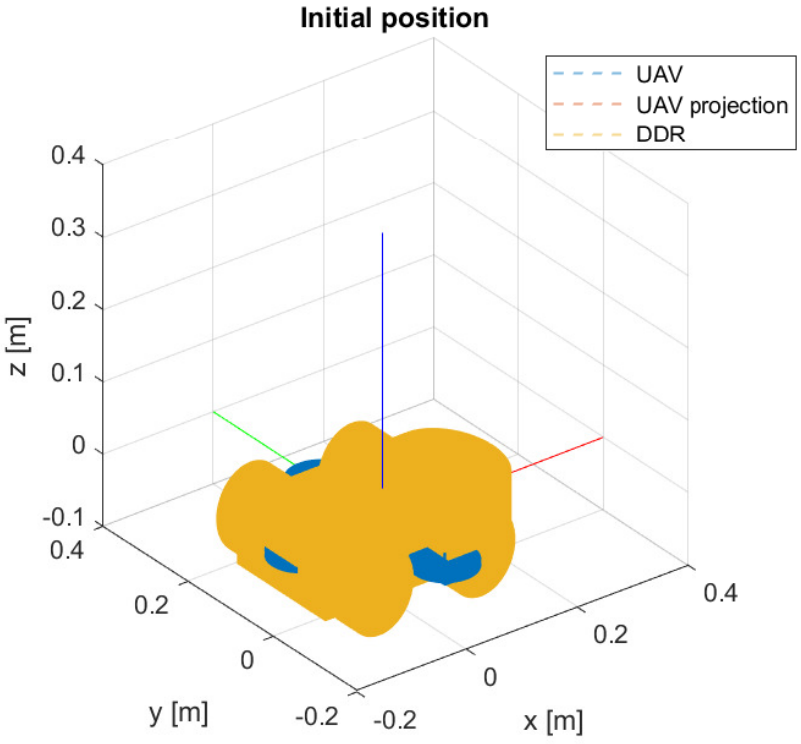


Figure 8.11: Controller simulation
Initial configuration

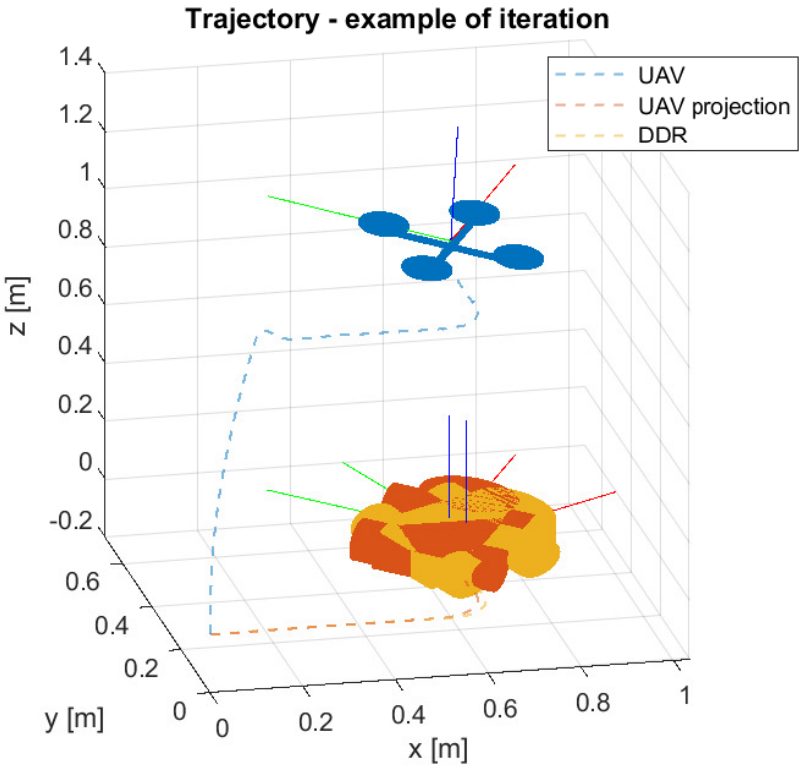


Figure 8.12: Controller simulation
Example of iteration

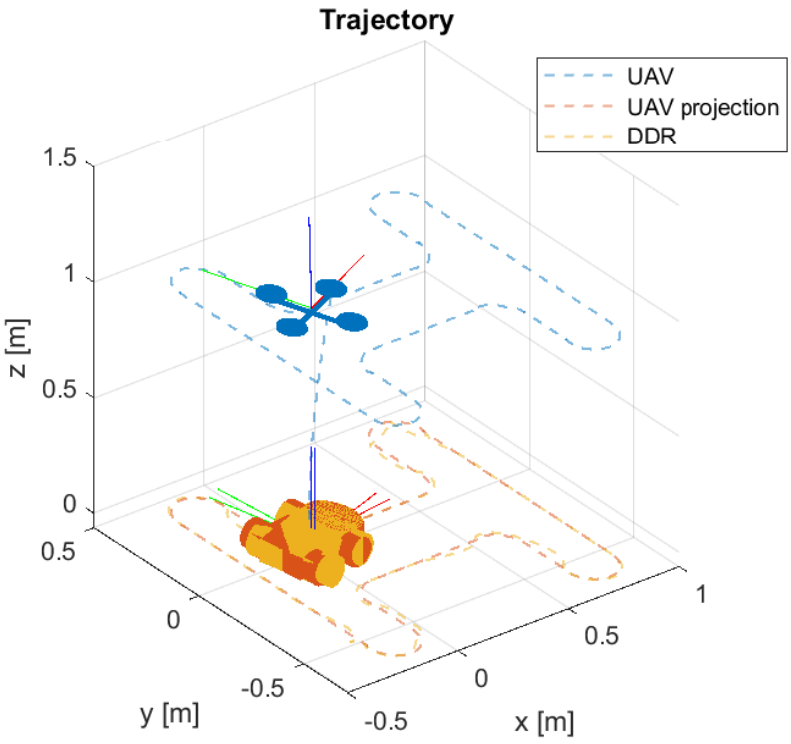


Figure 8.13: Controller simulation
Final configuration

Chapter 9

Conclusion

This thesis presents the Agent-Target Coordination of heterogeneous mobile robots and the Controller Optimization using an external computational power resource. It investigates a possible methodology to solve the problem of driving a Differential Drive Robot to follow an Unmanned Aerial Vehicle in order to set up a landing scenario, while managing the allocation of computational resources in the local processor or in the external computational node.

In the presented framework, the target trajectory is not known in advance by the agent and the only source of vehicles' poses and velocities is the camera attached to the external node. In particular, it is assumed that the measurements are provided without any delay due to communication or package losses and without any computational cost for the agent controller. On the other hand, the allocation of other processes on this external node uses some agent resources and therefore it has to be considered during the controller optimization. Despite the technological progress, this structure is becoming increasingly used as a result of the difficulties in developing performing but light-weight processors. Furthermore, in some industrial applications, it is convenient to centralize the power resources on a ground station and to mount simpler controllers on the mobile robots in order to minimize the cost and maximize the power efficiency. For these reasons, the contribution of this work is to analyze this framework and to develop an effective controller in order to deal with this setup.

In addition, this thesis validates the use of Nonlinear Model Predictive Control and Gaussian Process Regression methods to efficiently achieve the desired results. In particular, it has been proved the simplicity of integrating the system requirements into the Nonlinear Model Predictive Control objective cost function and the effectiveness of such method. Despite the difficulties in tuning the parameters, the NMPC controller has demonstrated good converging performance and system stability.

Furthermore, Section 7.1.1.1 addresses the difficulties and the benefits of using the Gaussian Process Regression to identify unknown models. Indeed, it is necessary to resort to an exhaustive training dataset and to correcly tune the hyperparameters in order to guarantee good prediction performances in the widest variety of situations. On the other hand, this method can correctly predict the unknown system outcome when the input data are reasonably close to the training ones, as shown in Chapter 8.

This thesis provides few starting points for the developing of future works:

- The controller is validated through several simulations on Matlab where the real-time constraint was not taken into account. A follow-up could study and implement the proposed solution on ROS Gazebo or another real-time simulation environment and then to test it in the laboratory.
- The computational external resource is modelled as an ideal node providing information without introducing communication delay. It could be interesting to model the communication channel, to integrate the introduced delay into the model and to investigate under which condition the system still manages to provide acceptable performances.
- During the GPR training, it has been noticed that the NMPC weight parameters affect the time needed to compute the NMPC solution despite the same initial conditions. It might be significant to investigate the correlation between the NMPC parameters and the solution computational time, in order to address the complex scenarios and hence to better generate the GPR training set.
- In this thesis, it is assumed that the quadrotor projection is suitable to represent the actual quadrotor trajectory. In case of an aerial evader, it can be easily understood that such approximation is no more reliable. One possible improvement is to study a method to identify the reliability of such assumption and to integrate the obtained information in the controller optimization choice.

Appendix A

Appendix

A.1 Agent Model

A.1.1 Stability of the PI control action

This section is dedicated to prove the stability of the PI control action presented in Section 3.1.2.2.

It is worth to rewrite Equation 3.1.9 in state space form, i.e.

$$\begin{bmatrix} \frac{\mathrm{d}i(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}\omega(t)}{\mathrm{d}t} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K\Phi}{L} \\ \frac{K\Phi}{J} & -\frac{b}{J} \end{bmatrix} \begin{bmatrix} i(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} v(t) \tag{A.1.1}$$

$$v(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau \tag{A.1.2}$$

$$e(t) = \omega_d(t) - \omega(t) \tag{A.1.3}$$

and therefore, by expanding the system state to $\boldsymbol{x} = [i,\, \omega,\, e_i]^T$ with $e_i(t) = \int_0^t e(\tau)\, d\tau$, it is possible to express the whole system as

$$\begin{bmatrix} \frac{\mathrm{d}i(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}\omega(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}e_i(t)}{\mathrm{d}t} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K\Phi+K_p}{L} & \frac{K_i}{L} \\ \frac{K\Phi}{J} & -\frac{b}{J} & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} i(t) \\ \omega(t) \\ e_i(t) \end{bmatrix} + \begin{bmatrix} \frac{Kp}{L} \\ 0 \\ 1 \end{bmatrix} \omega_d(t) = \boldsymbol{F}\boldsymbol{x}(t) + \boldsymbol{G}\omega_d(t) \tag{A.1.4}$$

To prove the system stability as K_p and K_i vary, it is possible to compute the eigenvalues of the state matrix $\boldsymbol{F}$ in Equation A.1.4. Let $\lambda \in \mathbb{C}$, the system

eigenvalues can be find by imposing

$$\det(\,\lambda \boldsymbol{I}_n - \boldsymbol{F}\,) = \begin{vmatrix} \lambda + \frac{R}{L} & \frac{K\Phi + K_p}{L} & -\frac{K_i}{L} \\ -\frac{K\Phi}{J} & \lambda + \frac{b}{J} & 0 \\ 0 & 1 & \lambda \end{vmatrix} = 0 \tag{A.1.5}$$

In order to assure the system asymptotic stability, all the eigenvalues' real part has to be negative. By resorting to the Routh-Hurwitz criteria and developing the necessary calculations, it is possible to obtain

$$\begin{aligned} \Delta(\lambda) = det(\lambda \boldsymbol{I}_n - \boldsymbol{F}) &= \lambda \left(\left(\lambda + \frac{R}{L} \right) \left(\lambda + \frac{b}{J} \right) + \frac{K\Phi(K\Phi + K_p)}{LJ} \right) + \frac{K\Phi K_i}{LJ} \\ &= \lambda^3 + \lambda^2 \left(\frac{R}{L} + \frac{b}{J} \right) + \lambda \left(\frac{Rb}{LJ} + \frac{K\Phi(K\Phi + K_p)}{LJ} \right) + \frac{K\Phi K_i}{LJ} \end{aligned} \tag{A.1.6}$$

$$\mathrm{Re}(\lambda) < 0 \mid \Delta(\lambda) = 0 \Leftrightarrow K_p > 0,\ 0 < K_i < \frac{RJ + Lb}{K\Phi} \left(\frac{Rb}{LJ} + \frac{K\Phi(K\Phi + K_p)}{LJ} \right) \tag{A.1.7}$$

Hence all the modes of the system are stable if and only id K_p and K_i satisfy the inequalities in Equation A.1.7.

In conclusion, given Equation A.1.6, it is possible to prove that the system asymptotically reaches the desired constant input reference $\omega_d(t) = k$.

$$W(s) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} (s\boldsymbol{I}_n - \boldsymbol{F})^{-1} \boldsymbol{G} = \frac{\frac{K\Phi}{JL}(K_i + sK_p)}{\Delta(s)} \tag{A.1.8}$$

$$W(s) \mid_{s=0} = 1 \Rightarrow w_d(t) = k,\ y(t) = \mathcal{L}^{-1}\left[W(s)\mathcal{L}[k]\right] = k + \underbrace{\ldots}_{\substack{\text{converging} \\ \text{modes}}} \xrightarrow{t \to \infty} k \tag{A.1.9}$$

A.1.2 Wheels State Estimation

The implemented PI feedback control loops need the knowledge of the current wheels speed angular velocities to work. However, this information is not available directly in the considered Differential Drive Robot scheme. In fact, the only source of information about the wheels states are the encoders connected to the wheels shafts, which provide their angular positions. To retrieve the wheels angular velocities information two methods can be applied: standard time derivative and state estimation.

The first method simply compute the difference between two consecutive measures of the wheel encoder and divide by the elapsed time.

$$\omega_i(k) = \frac{\vartheta(k) - \vartheta(k-1)}{T} \tag{A.1.10}$$

The so obtained measure, despite being computationally efficient, is widely affected by the time precision and by the environmental noises. In general, it is possible to improve the precision of this measure by using an embedded dedicated computation unit and by filtering the obtained measures.

A more reliable and less noise affected estimation system is the state estimation. Such method uses the whole model knowledge and can reconstruct not only the wheel angular velocity but the whole system state. Given the model in Equation 3.1.8, the applied input v and integrating the wheel angular position ϑ into the system state, it is possible to express the system as

$$\begin{gathered}\begin{bmatrix}\dot{i}\\ \dot{\vartheta}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}-\frac{R}{L} & 0 & -\frac{K\Phi}{L}\\ 0 & 0 & 1\\ \frac{K\Phi}{J} & 0 & -\frac{b}{J}\end{bmatrix}\begin{bmatrix}i\\ \vartheta\\ \omega\end{bmatrix} + \begin{bmatrix}\frac{1}{L}\\ 0\\ 0\end{bmatrix} v = \bar{\boldsymbol{F}}\bar{\boldsymbol{x}} + \bar{\boldsymbol{G}}v\\ y = \begin{bmatrix}0 & 1 & 0\end{bmatrix}\bar{\boldsymbol{x}} = \bar{\boldsymbol{H}}\bar{\boldsymbol{x}}\end{gathered} \tag{A.1.11}$$

where $\bar{\boldsymbol{x}}$ is the new state while $\bar{\boldsymbol{F}}$, $\bar{\boldsymbol{G}}$ and $\bar{\boldsymbol{H}}$ are the new state, input and output matrices.

Given the new system, in order to be able to reconstruct the state, it must be observable.

Proposition A.1.1 (Observable system [16]) *Given a n-dimensional system in the form*

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \boldsymbol{F}\boldsymbol{x}(t) + \boldsymbol{G}\boldsymbol{u}(t)\\ \boldsymbol{y}(t) &= \boldsymbol{H}\boldsymbol{x}(t)\end{aligned} \tag{A.1.12}$$

such system is said to be observable if and only if the matrix

$$\mathcal{O} = \begin{bmatrix}\boldsymbol{H}\\ \boldsymbol{H}\boldsymbol{F}\\ \vdots\\ \boldsymbol{H}\boldsymbol{F}^{n-1}\end{bmatrix} \tag{A.1.13}$$

is full rank, or equivalently if and only if $ker\mathcal{O} = \{\mathbf{0}_{\boldsymbol{n}}\}$.

Therefore, by computing the system observability matrix $\mathcal{O}$ (Equation A.1.14), it is possible to prove that the system in Equation A.1.11 is observable.

$$\mathcal{O} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ \frac{K\Phi}{J} & 0 & -\frac{b}{J} \end{bmatrix} \Rightarrow \; rank\mathcal{O} = 3 \; \Rightarrow \; ker\mathcal{O} = \{\mathbf{0}\} \tag{A.1.14}$$

By resorting to a full-order state observer, it is possible to estimate the state $\bar{\boldsymbol{x}}$ using the following state system

$$\begin{aligned} \dot{\hat{\boldsymbol{x}}} &= \bar{\boldsymbol{F}}\hat{\boldsymbol{x}} + \bar{\boldsymbol{G}}v + \boldsymbol{L}(y - \hat{y}) \\ \hat{y} &= \bar{\boldsymbol{H}}\hat{\boldsymbol{x}} \end{aligned} \tag{A.1.15}$$

where the observer gain $\boldsymbol{L} \in \mathbb{R}^{3\times 1}$ is chosen in order to ensure that $Re(eig(\bar{\boldsymbol{F}} - \boldsymbol{L}\bar{\boldsymbol{H}})) < 0$ while y and v are the input and output of the system in Equation A.1.11. It is worth to notice that the control input v is obtained by the PI low-level controller which in turn uses the wheels angular velocity estimate: such scheme is called *observer-based controller*.

A.2 Target Model

A.2.1 Time derivative of rotation matrices

This section is dedicated to derive the time derivative of a rotation matrix $\boldsymbol{R}_{W,B} \in \mathbb{SO}(3)$ representing the orientation of a frame $\mathfrak{F}_B$ attached to a rigid body with respect to an inertial frame $\mathfrak{F}_W$ when an angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ is applied. Let P be a point in the three dimensional space and let express its coordinates in the inertial frame as $\boldsymbol{p} \in \mathbb{R}^3$. Let assume that P possesses an angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$, that has axis $\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \in S^2$ passing through frame $\mathfrak{F}_W$ origin. Given these assumption, the point P is moving with uniform circular motion. Under these conditions, the time derivative of the position $\boldsymbol{p}$ is

$$\dot{\boldsymbol{p}} = \boldsymbol{\omega} \times \boldsymbol{p} \tag{A.2.1}$$

By defining $\boldsymbol{e}_i \in S^2$ as the canonical vector of $\mathfrak{F}_W$, it is possible to express the canonical vector of frame $\mathfrak{F}_B$ w.r.t. frame $\mathfrak{F}_W$ as $e_{W,i} = \boldsymbol{R}_{W,B}\,\boldsymbol{e}_i$. In addition, let $\boldsymbol{\omega}_W \in \mathbb{R}^3$ be the vector that express the angular velocity $\boldsymbol{\omega}$ in the inertial frame. By recalling the previous equation, it is possible to write

$$\dot{\boldsymbol{e}}_{W,i} = \boldsymbol{\omega}_W \times \boldsymbol{e}_{W,i} \tag{A.2.2}$$

and therefore, by recalling that $\boldsymbol{e}_{W,i}$ is the i-th column of $\boldsymbol{R}_{W,B}$ matrix and the properties of skew-symmetric matrices, it is possible to state that

$$\dot{\boldsymbol{R}}_{W,B} = [\boldsymbol{\omega}_W]_\times \boldsymbol{R}_{W,B} \tag{A.2.3}$$

The previous equation expresses the time derivative of a rotation matrix that represents the orientation of frame $\mathfrak{F}_B$ with respect to $\mathfrak{F}_W$ when the angular rate is expressed in frame $\mathfrak{F}_W$.

Considering the case where $\boldsymbol{\omega} \in \mathbb{R}^3$ is expressed in frame $\mathfrak{F}_B$, namely $\boldsymbol{\omega}_B = \boldsymbol{R}_{W,B}^T \boldsymbol{\omega}_W$, the roles of the frames are switched and thus it is possible to state that

$$\dot{\boldsymbol{R}}_{B,W} = [\hat{\boldsymbol{\omega}}_B]_\times \boldsymbol{R}_{B,W} \tag{A.2.4}$$

where $\boldsymbol{R}_{B,W} \in \mathbb{SO}^3$ is the rotation matrix representing the orientation of frame $\mathfrak{F}_W$ w.r.t. frame $\mathfrak{F}_B$ and $\hat{\boldsymbol{\omega}}_B \in \mathbb{R}^3$ is the angular velocity expressed in $\mathfrak{F}_B$. Therefore it must hold that

$$\hat{\boldsymbol{\omega}}_B = -\boldsymbol{\omega}_B = \boldsymbol{R}_{W,B}^T \boldsymbol{\omega}_W \tag{A.2.5}$$

in order to have Equations A.2.3 and A.2.4 represent the same rotation. Therefore, using the skew-symmetric matrix properties, it is possible to state that

$$\dot{\boldsymbol{R}}_{W,B} = (\dot{\boldsymbol{R}}_{B,W})^T = \boldsymbol{R}_{B,W}^T [-\boldsymbol{\omega}_B]_\times^T = \boldsymbol{R}_{W,B} [\boldsymbol{\omega}_B]_\times \tag{A.2.6}$$

A.2.2 Target position and yaw angle controller

In order to make the quadrotor follow a desired position trajectory a position and yow angle controller has been implemented. Such controller, as the name suggest, takes as input the desired position $\boldsymbol{p}_d = [x_d \quad y_d \quad z_d]^T$ and yaw angles ψ_d and generates the necessary force $\boldsymbol{f}_c$ and torque $\boldsymbol{\tau}_c$.

Given the rotation matrix as in Equation 2.1.5 and noticing that the total force acting on the quadrotor CoM is along the z axis in the body frame $\mathfrak{F}_B$, namely $\boldsymbol{f}_c = [0 \quad 0 \quad T]^T$, it is possible to rewrite the dynamic Equation 3.2.9c as

$$\dot{\boldsymbol{v}} = \ddot{\boldsymbol{p}} = \begin{bmatrix} (\mathrm{s}_\psi \mathrm{s}_\varphi + \mathrm{c}_\psi \mathrm{s}_\vartheta \mathrm{c}_\varphi)\frac{T}{m} \\ (-\mathrm{c}_\psi \mathrm{s}_\varphi + \mathrm{s}_\psi \mathrm{s}_\vartheta \mathrm{c}_\varphi)\frac{T}{m} \\ -g + (\mathrm{c}_\vartheta \mathrm{c}_\varphi)\frac{T}{m} \end{bmatrix} \tag{A.2.7}$$

Given the desired yaw orientation ψ_d and assuming ϑ and φ small, it is possible

to approximate the previous equation and obtain

$$\ddot{x} = (\mathrm{s}_{\psi_d}\varphi + \mathrm{c}_{\psi_d}\vartheta)\,\frac{T}{m} \tag{A.2.8a}$$

$$\ddot{y} = (-\mathrm{c}_{\psi_d}\varphi + \mathrm{s}_{\psi_d}\vartheta)\,\frac{T}{m} \tag{A.2.8b}$$

$$\ddot{z} = -g + \frac{T}{m} \tag{A.2.8c}$$

Assuming the quadrotor in hoovering condition, namely $\ddot{z} = 0 \Rightarrow T = mg$, it is possible to build the invertible map between $\ddot{x}$, $\ddot{y}$ and φ, ϑ, i.e.

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = g \begin{bmatrix} \mathrm{s}_{\psi_d} & \mathrm{c}_{\psi_d} \\ -\mathrm{c}_{\psi_d} & \mathrm{s}_{\psi_d} \end{bmatrix} \begin{bmatrix} \varphi \\ \vartheta \end{bmatrix}$$

$$\Downarrow \tag{A.2.9}$$

$$\begin{bmatrix} \varphi \\ \vartheta \end{bmatrix} = g^{-1} \begin{bmatrix} \mathrm{s}_{\psi_d} & -\mathrm{c}_{\psi_d} \\ \mathrm{c}_{\psi_d} & \mathrm{s}_{\psi_d} \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}$$

and therefore obtain

$$\begin{aligned} \varphi &= g^{-1}\left(\mathrm{s}_{\psi_d}\ddot{x} - \mathrm{c}_{\psi_d}\ddot{y}\right) \\ \vartheta &= g^{-1}\left(\mathrm{c}_{\psi_d}\ddot{x} + \mathrm{s}_{\psi_d}\ddot{y}\right) \end{aligned} \tag{A.2.10}$$

Using a PID feedback controller on the quadrotor position, it is possible to obtain the command quadrotor acceleration $\ddot{\boldsymbol{p}}^*$ and therefore, the desired roll and pitch angles φ^* and ϑ^*.

$$\ddot{\boldsymbol{p}}^* = \ddot{\boldsymbol{p}}_d + \boldsymbol{K}_{dp}\dot{\boldsymbol{e}}_p + \boldsymbol{K}_{pp}\boldsymbol{e}_p + \boldsymbol{K}_{ip}\int \boldsymbol{e}_p \tag{A.2.11a}$$

$$\boldsymbol{e}_p = \boldsymbol{p}_d - \boldsymbol{p} \tag{A.2.11b}$$

$$\varphi^* = g^{-1}\left(\mathrm{s}_{\psi_d}\ddot{x}^* - \mathrm{c}_{\psi_d}\ddot{y}^*\right) \tag{A.2.12a}$$

$$\vartheta^* = g^{-1}\left(\mathrm{c}_{\psi_d}\ddot{x}^* + \mathrm{s}_{\psi_d}\ddot{y}^*\right) \tag{A.2.12b}$$

where $\boldsymbol{K}_{dp}$, $\boldsymbol{K}_{pp}$ and $\boldsymbol{K}_{ip}$ are respectively the derivative, proportional and integral position matrices.

Therefore, given the desired z axis acceleration $\ddot{z}^*$ and the desired roll-pitch-yaw attitude $\boldsymbol{\alpha}_d = [\varphi^*\ \vartheta^*\ \psi_d]^T$, by resorting to an altitude-attitude controller, it

is possible to obtain the necessary trust force T and torque $\boldsymbol{\tau}$ using the following equations

$$T = \frac{m}{\mathrm{c}_\vartheta \mathrm{c}_\varphi}(g + \ddot{z}^*) \tag{A.2.13a}$$

$$\boldsymbol{\tau} = \boldsymbol{K}_{da}(\dot{\boldsymbol{\alpha}}_d - \dot{\boldsymbol{\alpha}}) + \boldsymbol{K}_{pa}(\boldsymbol{\alpha}_d - \boldsymbol{\alpha}) \tag{A.2.13b}$$

where $\boldsymbol{K}_{da}$ and $\boldsymbol{K}_{pa}$ are respectively the derivative and proportional attitude matrices.

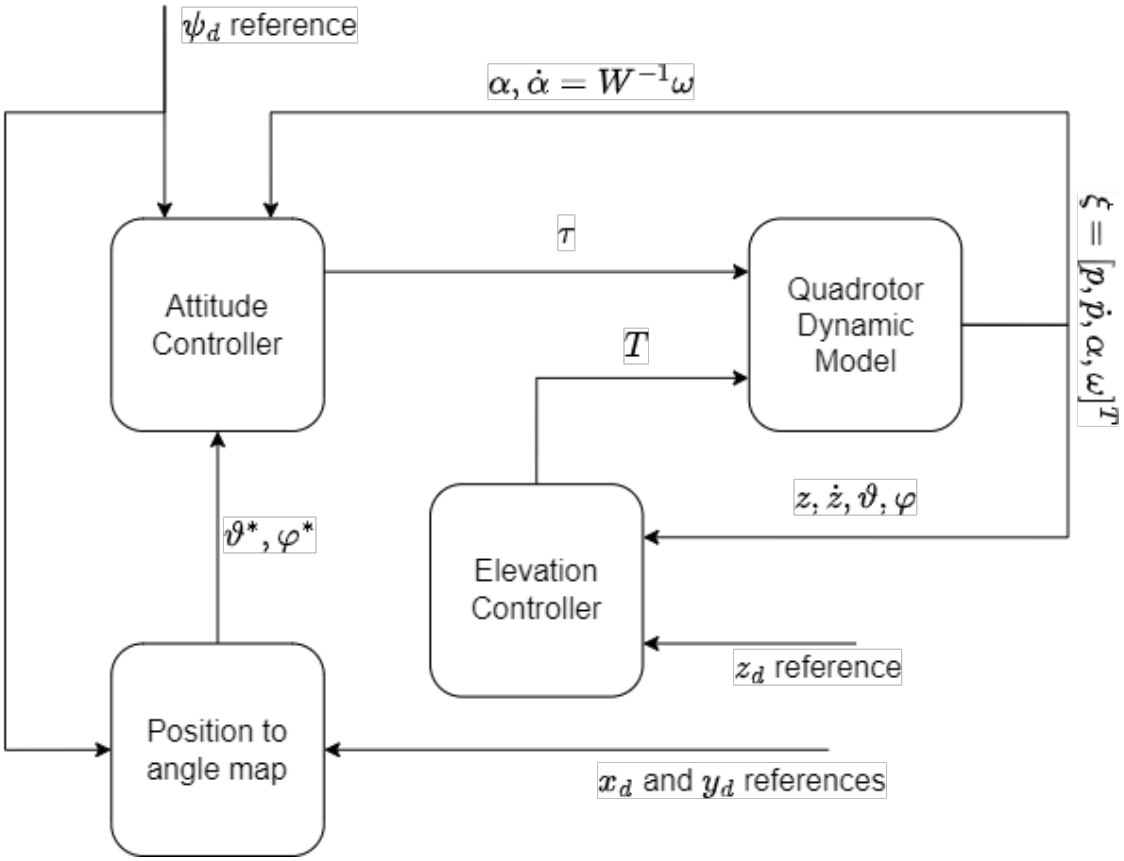


Figure A.1: Position and yaw angle controller architecture

Bibliography

[1] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range". In: *Automatica* 47.10 (2011), pp. 2279–2285. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2011.08.020. URL: https://www.sciencedirect.com/science/article/pii/S0005109811003918 (cit. on p. 1).

[2] Yutao Chen et al. "Efficient Partial Condensing Algorithms for Nonlinear Model Predictive Control with Partial Sensitivity Update". In: *IFAC-PapersOnLine* 51.20 (2018). 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018, pp. 406–411. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.11.067. URL: https://www.sciencedirect.com/science/article/pii/S2405896318327253 (cit. on p. 1).

[3] Moritz Diehl et al. "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations". In: *Journal of Process Control* 12.4 (2002), pp. 577–585. ISSN: 0959-1524. DOI: https://doi.org/10.1016/S0959-1524(01)00023-3. URL: https://www.sciencedirect.com/science/article/pii/S0959152401000233 (cit. on p. 1).

[4] Yutao Chen et al. "An Adaptive Partial Sensitivity Updating Scheme for Fast Nonlinear Model Predictive Control". In: (Aug. 2018) (cit. on p. 1).

[5] Francesco Branz et al. "1 kHz Remote Control of a Balancing Robot with Wi-Fi-in-the-Loop". In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 2614–2619. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.312. URL: https://www.sciencedirect.com/science/article/pii/S2405896320305930 (cit. on p. 2).

[6] B. Siciliano et al. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2010. URL: https://books.google.it/books?id=jPCAFmE-logC (cit. on pp. 6, 10).

[7] Shweta Gupte, Paul Infant Teenu Mohandas, and James M. Conrad. "A survey of quadrotor Unmanned Aerial Vehicles". In: *2012 Proceedings of IEEE Southeastcon*. 2012, pp. 1–6. DOI: 10.1109/SECon.2012.6196930 (cit. on p. 21).

[8] Xiaodong Zhang et al. "A Survey of Modelling and Identification of Quadrotor Robot". In: *Abstract and Applied Analysis* 2014 (Oct. 2014), pp. 1–6. DOI: 10.1155/2014/320526 (cit. on p. 21).

[9] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE International Conference on Robotics and Automation*. 2011. DOI: 10.1109/ICRA.2011.5979561 (cit. on p. 30).

[10] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN: 9780975937730. URL: https://books.google.it/books?id=MrJctAEACAAJ (cit. on p. 35).

[11] Endre Süli. "Numerical Solution of Ordinary Differential Equations". In: 2010 (cit. on p. 41).

[12] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. 2006, pp. I–XVIII, 1–248. ISBN: 026218253X (cit. on p. 54).

[13] *fitrpg MatlabWorks page*. https://it.mathworks.com/help/stats/fitrgp.html (cit. on p. 54).

[14] Joel A E Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4 (cit. on p. 63).

[15] *fmincon MatlabWorks page*. https://it.mathworks.com/help/optim/ug/fmincon.html (cit. on p. 63).

[16] Ettore Fornasini. *Appunti di Teoria dei Sistemi*. Libreria Progetto. 2015 (cit. on p. 81).