UNIVERSITAT POLITÈCNICA DE CATALUNYA

Degree program

INDUSTRIAL TECHNOLOGY ENGINEERING

Degree Thesis

DYNAMIC MODELLING AND VELOCITY CONTROL OF A TWO-WHEELED INVERTED PENDULUM ROBOT

Eduard Godayol Carceller

Supervisors:
Lluís Ros Giralt
Enric Celaya Llover

Tutor:
Maria Alba Pérez Gracia

June 2022

Dynamic Modelling and Velocity Control of a Two-wheeled Inverted Pendulum Robot

by Eduard Godayol Carceller

This work has been presented at the Universitat Politècnica de Catalunya to obtain the Degree in Industrial Engineering

ETSEIB degree program:
Degree in Industrial Technology Engineering

This work has been done in:
Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Supervisors:
Lluís Ros Giralt
Enric Celaya Llover

Tutor:
Maria Alba Pérez Gracia

Dynamic Modelling and Velocity Control of a Two-wheeled Inverted Pendulum Robot

Eduard Godayol Carceller

Abstract

With the advancement of Industry 4.0, mobile robots are being applied to more and more tasks, in areas such as exploring unfamiliar environments, inspecting and monitoring infrastructure, finding and rescuing people, or transporting and handling loads, among others. In this project we will focus on the modeling and control of two-wheeled inverted pendulum robots. Although they must be actively stabilized to prevent them from tipping over, these systems have several advantages over stable robots with more wheels: they can rotate around a point without moving, compensate external force disturbances that would tip over a conventional robot, and achieve taller and slimmer geometries while being stable. Along the project we will see how the kinematic and dynamic models for a *twinbot* (two-wheeled inverted pendulum robot) are obtained, we will go over the design of a control system, that, using the dynamic model, stabilizes the robot in the upright position along a real time defined trajectory, and we will end up validating the robustness of this control by applying force disturbances while the robot is trying to follow a defined trajectory.

Acknowledgments

First of all, and most importantly, I would like to thank professors Lluís Ros and Enric Celaya for their big support and dedication to the project, they have opened me the doors of a new universe of control theory that I would have had a lot of trouble understanding on my own. They have given me their time and knowledge, that without them, facing this project would have probably been unaffordable.

I want to thank the Institut de Robòtica i Informàtica Industrial (IRI) for welcoming me into their offices, and providing me the facilities I needed to develop the project. I really appreciate the effort of Maria Alba Pérez Gracia for accepting the role of the director of the project, and finally, I want to thank my family and friends for their infinite support throughout the project.

Contents

1 Introduction

1.1 Motivation

With the advancement of Industry 4.0, mobile robots are being applied to more and more tasks, in areas such as exploring remote environments, inspecting infrastructure, finding and rescuing people, or transporting and handling loads, among others [1]. Various solutions have been proposed for the locomotion of these robots based on the use of legs or wheels [2, 3]. While legged robots are capable of overcoming complex obstacles, they are also more difficult to design and control, and tend to have higher energy consumption. Wheeled robots, on the other hand, tend to be more energy efficient, and usually have a simpler mechanical structure and dynamic model, making them more attractive in a variety of situations [4].

In this project we are focused on the modelling and control of two-wheeled inverted pendulum robots (Fig. 1.1). These robots, which we call *twinbots* for short, use two coaxial drive wheels located on either side of a central body, or chassis. Their center of mass is located above the wheel's axis (either due to the arrangement of their parts, or because of an added load that is being transported), causing a natural tendency of falling. Although a twinbot must be actively stabilized to prevent it from tipping over, these systems have several advantages over stable robots with more wheels [2]. First of all, their simple design makes them very maneuverable, as they can rotate around a point without moving, and the fact of being actively stabilized allows them to compensate for external force disturbances that would tip over a conventional robot. Secondly, twinbots can achieve taller and slimmer geometries and still be stable, which makes them suitable for carrying loads stacked in narrow aisles, as well as allowing the movement through tight corridors or hard-to-reach corners. Furthermore, as these robots can tilt forward or backwards, they are also able to maintain stability on slopes and even to accelerate quickly without tipping over, making them more versatile and faster compared to regular mobile platforms.

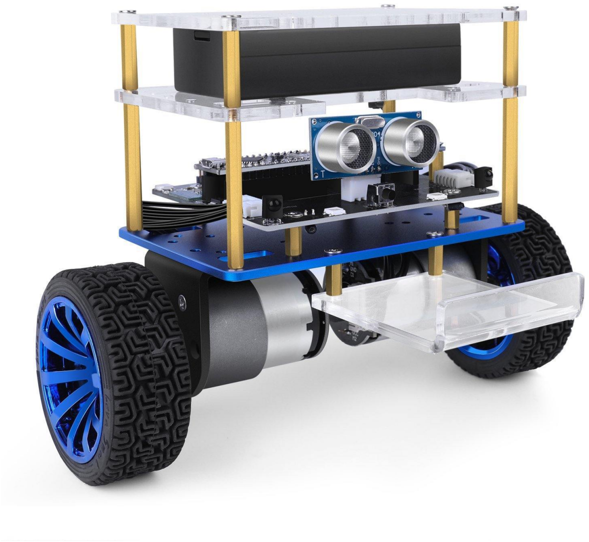


Figure 1.1: A two-wheeled inverted pendulum robot. Picture courtesy of Elegoo.

The aim of this work is to design a control system that allows the execution of trajectories of constant velocity with these robots, specified by means of linear and angular speed commands of the chassis. This system must calculate the motor actions required to meet these speeds at all times, while ensuring the robot stays close to the upright position, and making sure it returns to that position in the event of any deviations due to unforeseen disturbances or orientation errors.

There are two clear applications of this control system. Firstly, it can serve as a control module for twinbots with autonomous navigation capability (Fig. 1.2, left). These robots use a pre-built map of the environment to generate a collision-free path that takes them to a desired position. During the execution of this path, the navigation system is constantly generating corrective commands for the speed of advance, either to reduce the risk of collision with obstacles already foreseen, or to negotiate with any new obstacles that may appear near the robot. In this context, the control system of this project allows a precise fulfillment of the aforementioned commands. Another clear application of the control system is to manually pilot a robot: a human generates speed commands with the help of an input device (such as a joystick, a keyboard, or a mouse) and the robot tries to follow them closely. The piloting can be remote, in which case we will have to communicate with the robot to receive feedback from what it sees, or sitting on the robot itself, in which case we can use the robot as a personal transportation system (Fig. 1.2, right). Some brands are beginning to propose this concept in fact, as an alternative to the Segway concept, in which the user needs to be standing up in order to gain control.

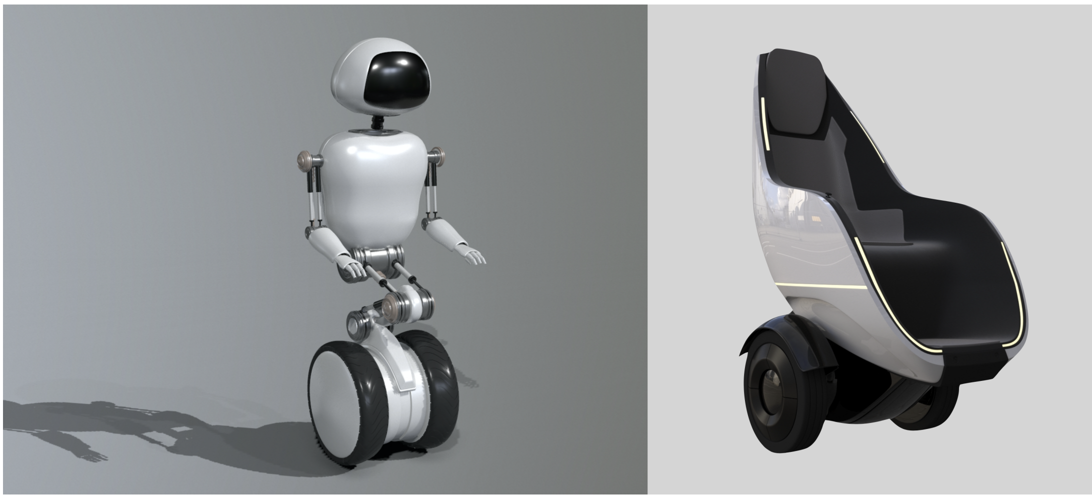


Figure 1.2: Conceptual designs for a two-wheeled autonomous robot (left) and a wheelchair controlled with a joystick (right). The joystick is very small, located on the right hand side of the chair. Pictures courtesy of Turbosquid and Segway, respectively.

1.2 Project objectives

Specifically, this project pursues three objectives:

1. To obtain the kinematic and dynamic models of a twinbot. The kinematic model must take into account the non-slip condition imposed by the contacts of the wheels with the ground, and must serve as a basis for a later development of the dynamic model of the robot. This second model should be sufficiently general, incorporating aspects such as a generic mass distribution for the main body of the robot, the friction on the wheel's axes, or perturbation forces that might be applied to the robot.

2. To design a control system that, using the above dynamic model, stabilizes a twinbot in an upright position along a constant velocity trajectory, specified by means of linear and angular velocity setpoints for the chassis.

3. To validate the robustness of the control system in the event of non-modeled force disturbances, or deviations of the robot with respect to the vertical position, either with the robot stationary or in motion, while following a given trajectory.

1.3 Scope of the project

Usually, the development of system modelling and control methods is approached in two stages. In a first stage, such methods are designed and programmed, and their performance is tested in simulation. Then, in a second stage, the methods are implemented and validated in a real system. In this project we have focused on covering all tasks of the first stage, leaving those of the second for future work. Therefore, this project has been devoted to:

1. Obtain the dynamic model of the Twinbot using the laws of Mechanics.
2. Design a control law to stabilize the system along a desired trajectory.
3. Validate the identification and control methods in simulation.

The following implementation aspects are thus left for a future project:

1. The construction of a physical protoype of the robot.
2. Identify its parameters.
3. Hardware implementation and validation.
4. The development of control interfaces for its sensors and actuators.
5. The experimental validation of the methods proposed in this project.

1.4 Document structure

The rest of the document is structured as follows:

- In Chapter 2 we explain how the kinematic model of a twinbot has been obtained. This model describes the feasible motions of the robot without considering their generating forces. The resulting equations are necessary to later obtain the dynamic model of the robot.
- In Chapter 3 we obtain the dynamic model of a twinbot. By this model we mean the equations of motion that relate the accelerations of the robot to its motor torques. In this chapter we introduce two working models, with their respective pros and cons, and how to move from one to the other.

- In Chapter 4 we introduce the optimal control theory that we use to stabilize the robot at a certain configuration, and around a given trajectory. We review both the finite and the infinite horizon LQR controllers that will be used for this purpose.
- In Chapter 5 we deal with the controller design. Here we design a control loop algorithm that allows us to stabilize a twinbot along a constant velocity trajectory, determined by the linear and angular speeds of the chassis. Moreover we will explain how the control gains can be adjusted, and we will validate the controller in simulation.
- Finally, in Chapter 6 we provide the main conclusions of this project, and list several points for further attention.

The document has two appendices providing the MATLAB programs implementing our algorithms and simulated experiments, and the project budget.

1.5 Notation

Throughout the document we will denote scalar magnitudes with normal font, and vector ones with bold font. Vectors and matrices will be denoted with lowercase and uppercase letters, respectively. Thus, x will denote a scalar, $\boldsymbol{x}$ a vector, and $\boldsymbol{X}$ a matrix. When $\boldsymbol{x}$ is a vector that appears in a mathematical operation, it will be assumed to be a column vector.

When we have to make the components of a vector explicit, we will usually write them in columns and in square brackets.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}. \tag{1.1}$$

Sometimes, however, we will also use $\boldsymbol{x} = (x_1, \ldots, x_n)$ for compactness, which we will assume to be equivalent to Eq. (1.1).

To ease the explanations, we will use underbraces to define new symbols or make clarifications. For example, when we write

$$\underbrace{\text{expression}}_{\text{symbol}}$$

we mean that, thereafter, the specified symbol will denote the underbraced expression.

In long vectors or matrices, we sometimes use s_α and c_α as a shorthand for the sine and cosine of α.

2
Kinematic model

In this chapter we develop the kinematic model of a twinbot. We start by defining the different reference frames and vector bases that we have set, and the configuration and state coordinates of the robot. Then we cover the different constraints imposed by the rolling contacts of the robot with the ground. Finally, we introduce the configuration and state spaces, and show that the system is nonholonomic and underactuated.

2.1 Reference frames and vector bases

A twinbot can be viewed as a differential drive robot whose chassis is allowed to rotate around the wheels' axis. The wheels are powered by electric motors mounted underneath the chassis (Fig. 2.1).

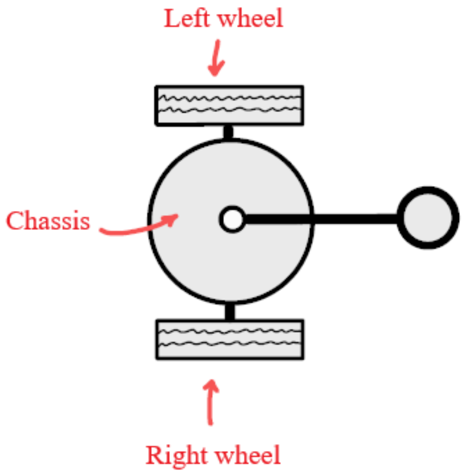


Figure 2.1: Kinematic structure of a twinbot.

The structure of the robot is simple but effective. With proper maneuvering, the robot can reach any desired position and orientation in the plane. However, since its center of gravity is above the wheels axis, the robot may tip over if proper control torques are not applied. The challenge is to move the robot forward (even while turning) in a stable manner. To obtain the robot's model, we use the reference frames seen in Fig. 2.2.

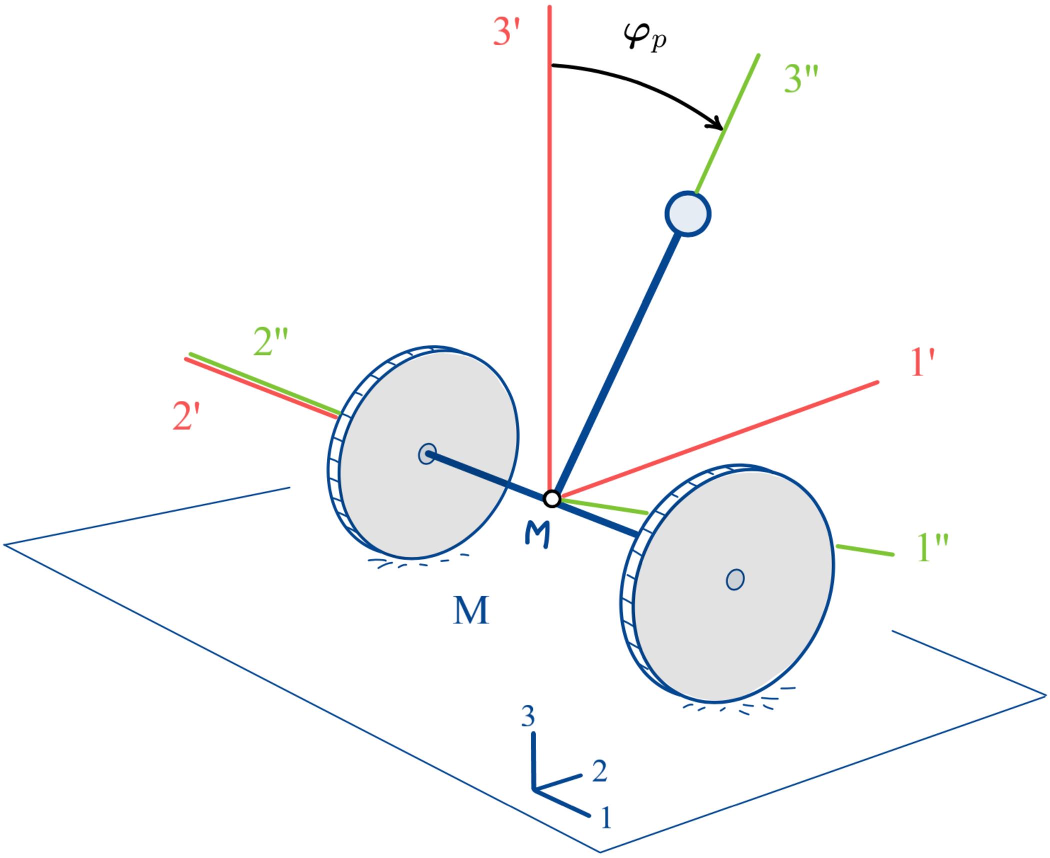


Figure 2.2: Reference frames of a twinbot.

The blue frame is the **absolute frame** fixed to the ground, with the axes 1, 2, and 3. The red frame is centered at the midpoint M of the wheels' axis. Axes 1', 2', and 3' correspond to the heading direction of the robot, the wheels' axis, and the vertical direction through M, respectively. We call this frame the **robot frame**. The green frame is fixed to the robot chassis, and is rotated an angle φ_p with respect to the robot frame. We call this frame the **chassis frame**. The three frames are sometimes referred to as $\mathcal{F}, \mathcal{F}'$, and $\mathcal{F}''$, respectively.

Each frame has a vector basis attached to it, whose unit vectors are directed along the frame axes. The three bases will be referred to as:

- $B = \{1, 2, 3\} \longleftarrow$ Fixed to the absolute frame
- $B' = \{1', 2', 3'\} \longleftarrow$ Fixed to the robot frame
- $B'' = \{1'', 2'', 3''\} \longleftarrow$ Fixed to the chassis frame

2.2 Configuration and state coordinates

The robot configuration can be described by means of six coordinates: the position (x, y) of M in the absolute frame, the orientation angle θ of the platform (between axes 1' and 1), the pitch angle φ_p, and the angles of the left and right wheels, φ_l and φ_r, relative to the chassis. The robot configuration is thus given by

$$\boldsymbol{q} = (x, y, \theta, \varphi_p, \varphi_l, \varphi_r),$$

and the time derivative of $\boldsymbol{q}$ provides the robot velocity. Therefore, the robot state is given by

$$\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}}).$$

It will also be useful to consider an internal configuration vector

$$\boldsymbol{\varphi} = (\varphi_p, \varphi_l, \varphi_r)$$

and its associate internal state

$$\boldsymbol{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}).$$

2.3 Rolling contact constraints

We next see that the coordinates of $\boldsymbol{x}$ are not independent. The rolling contacts of the wheels impose a kinematic constraint of the form

$$\boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} = 0, \tag{2.1}$$

where $\boldsymbol{J}(\boldsymbol{q})$ is a 3×6 Jacobian matrix. We will next obtain this Jacobian. In our derivations we use $\boldsymbol{v}(Q)$ to denote the velocity of a point Q of the robot. The basis in which $\boldsymbol{v}(Q)$ is expressed will be mentioned explicitly, or understood by context.

The robot pose is given by the position (x, y) of point M, and by the θ angle. The half-length of the wheels axis is d. Also, C_r and C_l denote the centers of the right and left wheels, and P_r and P_l are the contact points of such wheels with the ground (Fig. 2.3). The two wheels have the same radius r.

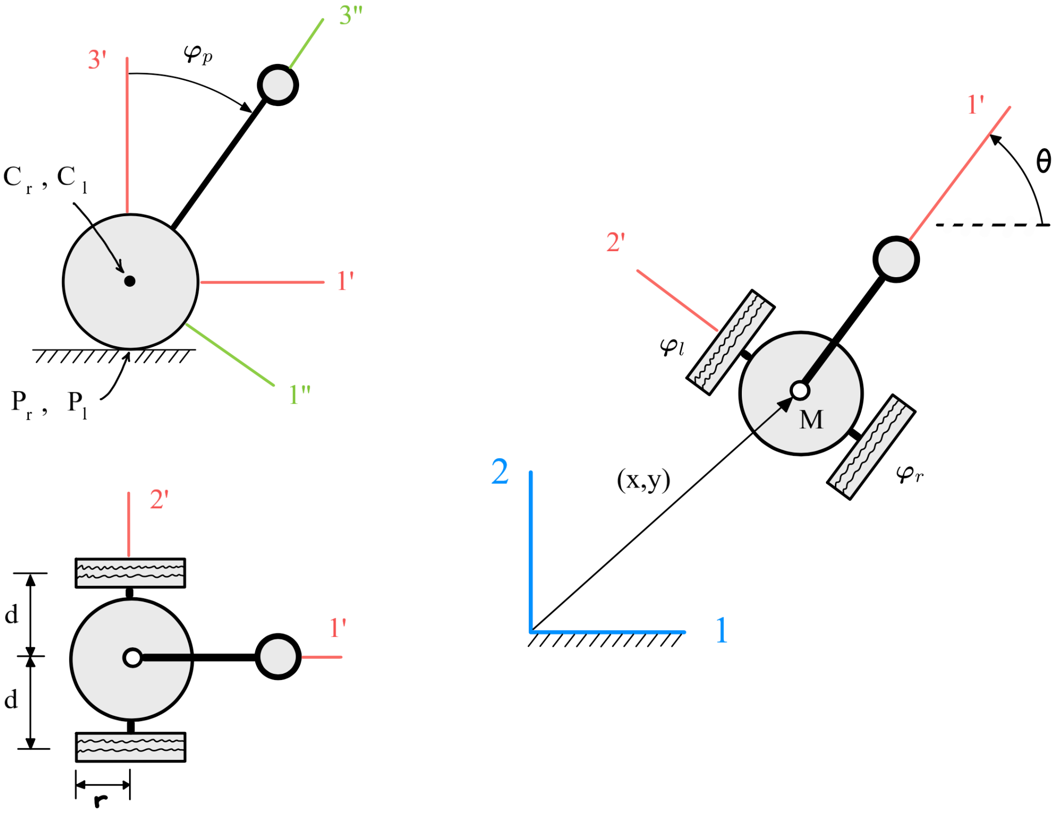


Figure 2.3: Main notation for a twinbot.

The rolling contact constraints of the chassis can be found by computing the velocities $\boldsymbol{v}(P_r)$ and $\boldsymbol{v}(P_l)$ in terms of $\dot{x}$, $\dot{y}$, $\dot{\theta}$, $\dot{\varphi}_p$, $\dot{\varphi}_l$ and $\dot{\varphi}_r$, and forcing these velocities to be zero (as the wheels do not slide when placed on the ground).

To obtain $\boldsymbol{v}(P_r)$ and $\boldsymbol{v}(P_l)$, note first that the velocity of M can only be directed along axis 1', since lateral slipping is forbidden under perfect rolling. Therefore in $B' = \{1', 2', 3'\}$, $\boldsymbol{v}(M)$ takes the form

$$\boldsymbol{v}(M) = \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix}.$$

Also note that, if $\boldsymbol{\omega}_{\text{robot}}$ is the absolute angular velocity of the robot frame, the velocities of the centers of the two wheels are

$$\boldsymbol{v}(C_r) = \boldsymbol{v}(M) + \boldsymbol{\omega}_{robot} \times \boldsymbol{MC_r} = \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ -d \\ 0 \end{bmatrix} = \begin{bmatrix} v + d\,\dot{\theta} \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{v}(C_l) = \boldsymbol{v}(M) + \boldsymbol{\omega}_{robot} \times \boldsymbol{MC_l} = \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} v - d\,\dot{\theta} \\ 0 \\ 0 \end{bmatrix}.$$

Therefore, the velocities of the ground contact points are given by

$$\begin{aligned} \boldsymbol{v}(P_r) &= \boldsymbol{v}(C_r) + \boldsymbol{\omega}_{\text{wheel}} \times \boldsymbol{C_r P_r} = \\ &= \begin{bmatrix} v + d\,\dot{\theta} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\varphi}_p + \dot{\varphi}_r \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ -r \end{bmatrix} = \begin{bmatrix} v + d\,\dot{\theta} - r\,(\dot{\varphi}_p + \dot{\varphi}_r) \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \boldsymbol{v}(P_l) &= \boldsymbol{v}(C_l) + \boldsymbol{\omega}_{\text{wheel}} \times \boldsymbol{C_l P_l} = \\ &= \begin{bmatrix} v - d\,\dot{\theta} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\varphi}_p + \dot{\varphi}_l \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ -r \end{bmatrix} = \begin{bmatrix} v - d\,\dot{\theta} - r\,(\dot{\varphi}_p + \dot{\varphi}_l) \\ 0 \\ 0 \end{bmatrix}. \end{aligned}$$

Since P_r and P_l do not slip, $\boldsymbol{v}(P_r)$ and $\boldsymbol{v}(P_l)$ must be zero, which gives the two fundamental constraints of the robot:

$$\begin{aligned} v - r\,(\dot{\varphi}_p + \dot{\varphi}_r) + d\,\dot{\theta} &= 0 \\ v - r\,(\dot{\varphi}_l + \dot{\varphi}_p) - d\,\dot{\theta} &= 0. \end{aligned}$$

Solving these equations for v and $\dot{\theta}$, we obtain

$$v = \frac{r\,(\dot{\varphi}_l + 2\,\dot{\varphi}_p + \dot{\varphi}_r)}{2}, \tag{2.2}$$

$$\dot{\theta} = -\frac{r\,(\dot{\varphi}_l - \dot{\varphi}_r)}{2\,d}. \tag{2.3}$$

For later use, we can also solve the equations for $\dot{\varphi}_l$ and $\dot{\varphi}_r$, which gives

$$\dot{\varphi}_r = \frac{v + d\,\dot{\theta} - r\,\dot{\varphi}_p}{r},$$

$$\dot{\varphi}_l = -\frac{d\,\dot{\theta} - v + r\,\dot{\varphi}_p}{r}.$$

From Figs 2.2 and 2.3 we see that the velocity of point M in basis B={1,2,3} is

$$\boldsymbol{v}(M) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix},$$

and its components are:

$$\begin{cases} \dot{x} = v \cdot \cos\theta \\ \dot{y} = v \cdot \sin\theta. \end{cases}$$

By substituting Eq. (2.2) in the previous two equations, and also considering Eq. (2.3), we obtain the system of equations:

$$\begin{cases} \dot{x} = \cos(\theta) \left(\frac{r\,\dot{\varphi}_l}{2} + r\,\dot{\varphi}_p + \frac{r\,\dot{\varphi}_r}{2} \right) \\ \dot{y} = \sin(\theta) \left(\frac{r\,\dot{\varphi}_l}{2} + r\,\dot{\varphi}_p + \frac{r\,\dot{\varphi}_r}{2} \right) \\ \dot{\theta} = -\frac{r\,(\dot{\varphi}_l - \dot{\varphi}_r)}{2\,d} \end{cases} \tag{2.4}$$

This system expresses the rolling contact constraints of the robot. For compactness, we write it as

$$\dot{\boldsymbol{p}} = \boldsymbol{N}(\boldsymbol{q}) \cdot \dot{\boldsymbol{\varphi}},$$

where $\boldsymbol{p} = (x, y, z)$, $\boldsymbol{\varphi} = (\varphi_p, \varphi_l, \varphi_r)$, and $\boldsymbol{N}(\boldsymbol{q})$ is the 3×3 matrix

$$\boldsymbol{N}(\boldsymbol{q}) = \begin{bmatrix} r\,\cos(\theta) & \frac{r\,\cos(\theta)}{2} & \frac{r\,\cos(\theta)}{2} \\ r\,\sin(\theta) & \frac{r\,\sin(\theta)}{2} & \frac{r\,\sin(\theta)}{2} \\ 0 & -\frac{r}{2\,d} & \frac{r}{2\,d} \end{bmatrix}. \tag{2.5}$$

The equation $\dot{\boldsymbol{p}} = \boldsymbol{N}(\boldsymbol{q}) \cdot \dot{\boldsymbol{\varphi}}$ can be written as

$$\begin{bmatrix} \boldsymbol{I}_3 & -\boldsymbol{N} \end{bmatrix} \cdot \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{\varphi}} \end{bmatrix} = \boldsymbol{0} \rightarrow \begin{bmatrix} \boldsymbol{I}_3 & -\boldsymbol{N} \end{bmatrix} \cdot \dot{\boldsymbol{q}} = \boldsymbol{0}.$$

Thus, the constraint Jacobian $J(q)$ in Eq. (2.1) is

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{I}_3 & -\boldsymbol{N} \end{bmatrix}.$$

2.4 Holonomic and nonholonomic constraints

Recall that the last equation of the system $\dot{\boldsymbol{p}} = \boldsymbol{N}(\boldsymbol{q}) \cdot \dot{\boldsymbol{\varphi}}$ (Eq. (2.4)) was

$$\dot{\theta} = \frac{r}{2d}\,(\dot{\varphi}_r - \dot{\varphi}_l).$$

This equation is integrable, and yields the holonomic constraint [5]

$$\phi(\boldsymbol{q}) = 0,$$

where

$$\begin{aligned} \phi(\boldsymbol{q}) &= \theta - \tfrac{r}{2d}(\varphi_r - \varphi_l) - K_0 \\ K_0 &= \theta_0 - \tfrac{r}{2d}\,(\varphi_{r,0} - \varphi_{l,0}). \end{aligned}$$

In sum, the robot is subject to three nonholonomic and one holonomic scalar constraints:

$$\boldsymbol{F}(\boldsymbol{x}) \equiv \left\{ \begin{array}{l} \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} = \boldsymbol{0} \\ \phi(\boldsymbol{q}) = 0 \end{array} \right.$$

2.5 The configuration and state spaces

The configuration space of the robot (C-space) is the set

$$\mathcal{C} = \{\boldsymbol{q} : \phi(\boldsymbol{q}) = \boldsymbol{0}\}.$$

The state space $\mathcal{X}$ of the robot, in turn, is defined by the 4 constraints introduced in the previous section:

$$\mathcal{X} = \{\boldsymbol{x} : \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}\}.$$

Taking into account that $\boldsymbol{q}$ is a 6-dimensional vector, and that it is subject to the scalar constraint $\phi(\boldsymbol{q}) = 0$, $\mathcal{C}$ is a 5-dimensional manifold. This manifold is smooth because $\boldsymbol{\phi_q}$ is full rank:

$$\boldsymbol{\phi_q}(\boldsymbol{q}) = \frac{\partial \phi(\boldsymbol{q})}{\partial \boldsymbol{q}} = \left[\begin{array}{cccccc} 0 & 0 & 1 & 0 & \frac{r}{2d} & -\frac{r}{2d} \end{array} \right].$$

Note that the space of feasible accelerations must be three dimensional since $\nu = \{\dot{\boldsymbol{q}} : \boldsymbol{J} \cdot \dot{\boldsymbol{q}} = 0\}$ is three-dimensional. Since the robot only has two motors, it will only be able to command a 2-dimensional space of accelerations. That is, the robot not only is nonholonomic, it is also underactuated [6].

3
Dynamic model

In this chapter we obtain the dynamic model of a twinbot, which provides the relationship between the accelerations of the system and the torques applied by its motors. Since we are working with a non-holonomic system, the model will be derived using Lagrange's equation with multipliers [7, 8]. The solutions obtained with this model will be used to achieve a precise control of the robot that takes into account all the dynamic parameters. Moreover, we will introduce two different working models. The $\boldsymbol{x}$-model encompasses all the $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ coordinates, while the $\boldsymbol{y}$-model is more simplified and only takes into account the $(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$ ones. Finaly we will cover the odometry equations that will allow us to move between the $\boldsymbol{x}$ and $\boldsymbol{y}$ models; and we will make a particularization of the dynamic model using a concrete geometry for a twinbot.

3.1 Lagrange's equation of motion

Recall that the Lagrange equation of a non-holonomic robot takes the form

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{J}(\boldsymbol{q})^{\top}\,\boldsymbol{\lambda} = \boldsymbol{Q}_a(\boldsymbol{u}) + \boldsymbol{Q}_f + \boldsymbol{Q}_d, \tag{3.1}$$

where:

- $\boldsymbol{q}$ is the configuration vector (of size $n_q = 6$ in our case).
- $\boldsymbol{u} = (\tau_l, \tau_r)$ is the vector of motor torques (the left and right wheel torques).
- $\boldsymbol{M}\,(\boldsymbol{q})$ is the mass matrix of the unconstrained system (positive-definite of size $n_q \times n_q$).
- $\boldsymbol{C}\,(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the generalized Coriolis and centrifugal force matrix.
- $\boldsymbol{G}\,(\boldsymbol{q})$ is the generalized gravity force.
- $\boldsymbol{J}(\boldsymbol{q})$ is the constraint Jacobian.
- $\boldsymbol{\lambda}$ is a vector of Lagrange mulitpliers.
- $\boldsymbol{Q}_a(\boldsymbol{u})$ is the generalized force of actuation, which can be written in the form $E(\boldsymbol{q}) \cdot u$.
- $\boldsymbol{Q}_f$ is the generalised force modelling all friction forces in the system.

- $\boldsymbol{Q}_d$ is a generalised force modelling all disturbance forces considered, which may be added to test disturbance rejection when a feedback controller is used.

Although a twinbot moves on flat terrain, for different configurations of the φ_p angle, the height of its center of gravity changes. For this reason the potential energy is not constant, therefore $\boldsymbol{G}(\boldsymbol{q})$ will not be 0. Our task is to obtain the values of all the matrices needed to calculate the equation of motion of the robot: $\boldsymbol{M}(\boldsymbol{q})$, $\boldsymbol{C}(q, \dot{\boldsymbol{q}})$, $\boldsymbol{G}(\boldsymbol{q})$, $\boldsymbol{Q}_a(\boldsymbol{u})$, $\boldsymbol{Q}_f$, and $\boldsymbol{Q}_d$.

3.1.1 Mass matrix

Recall that the robot configuration is given by

$$\boldsymbol{q} = (x, y, \theta, \varphi_p, \varphi_l, \varphi_r).$$

To find the mass matrix $\boldsymbol{M}(\boldsymbol{q})$ we need to write the kinetic energy T of the robot as a function of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ and then express it as:

$$T(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}\, \dot{\boldsymbol{q}}^\top \cdot \boldsymbol{M}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}}.$$

We have to compute the translational and rotational kinetic energies of all bodies in a twinbot and add them all to obtain T. In computing such energies we must view the robot as a kinematic tree whose configuration is given by $\boldsymbol{q}$.

The expressions of the translational kinetic energies for the chassis, the left wheel, and the right wheel are:

$$\begin{aligned} T_{tra,chas} &= \frac{1}{2}\, \boldsymbol{v}(G)^\top\, m_c\, \boldsymbol{v}(G), \\ T_{tra,l} &= \frac{1}{2}\, \boldsymbol{v}(C_l)^\top\, m_w\, \boldsymbol{v}(C_l), \\ T_{tra,r} &= \frac{1}{2} \boldsymbol{v}(C_r)^\top\, m_w\, \boldsymbol{v}(C_r). \end{aligned} \tag{3.2}$$

The only parameter left to compute the previous equations is the absolute velocity of the center of gravity G of the chassis, which is given by

$$\boldsymbol{v}(G) = \boldsymbol{v}(M) + \boldsymbol{\omega}_{\text{chassis}} \times \boldsymbol{MG}.$$

In the basis B' we have

$$\boldsymbol{v}(G) = \underbrace{\begin{bmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top}_{\mathbf{Rot_{B\,to\,B'}}} \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\varphi}_p \\ \dot{\theta} \end{bmatrix} \times \underbrace{\begin{bmatrix} c_{\varphi_p} & 0 & s_{\varphi_p} \\ 0 & 1 & 0 \\ -s_{\varphi_p} & 0 & c_{\varphi_p} \end{bmatrix}}_{\mathbf{Rot_{B''\,to\,B'}}} \begin{bmatrix} x_G \\ y_G \\ z_G \end{bmatrix},$$

where (x_G, y_G, z_G) is the vector $\boldsymbol{MG}$ in the chassis reference (B"). If we develop the previous equation, we see that

$$\boldsymbol{v}(G) = \begin{bmatrix} \dot{\varphi}_p \left(z_G \cos(\varphi_p) - x_G \sin(\varphi_p)\right) - \dot{\theta}\, y_G + \dot{x} \cos(\theta) + \dot{y} \sin(\theta) \\ \dot{\theta} \left(x_G \cos(\varphi_p) + z_G \sin(\varphi_p)\right) + \dot{y} \cos(\theta) - \dot{x} \sin(\theta) \\ -\dot{\varphi}_p \left(x_G \cos(\varphi_p) + z_G \sin(\varphi_p)\right) \end{bmatrix}.$$

Once $\boldsymbol{v}(G)$ is determined, we can apply the Eq. (3.2) to obtain the translational kinetic energies of the chassis and the two wheels (Eq. (3.3)).

$$\begin{aligned} T_{tra,chas} = &\frac{m_c \left(\dot{\theta} \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right) + \dot{y}\, c_\theta - \dot{x}\, s_\theta\right)^2}{2} \\ &+ \frac{m_c \left(\dot{\varphi}_p \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) - \dot{\theta}\, y_G + \dot{x}\, c_\theta + \dot{y}\, s_\theta\right)^2}{2} \\ &+ \frac{m_c\, \dot{\varphi}_p^2 \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right)^2}{2} \end{aligned} \tag{3.3}$$

$$T_{tra,l} = \frac{m_w \left(v - d\,\dot{\theta}\right)^2}{2}$$

$$T_{tra,r} = \frac{m_w \left(v + d\,\dot{\theta}\right)^2}{2}$$

The expressions of the rotational kinetic energies in the basis B' are:

$$\begin{aligned} T_{rot,chas} &= \frac{1}{2}\, \boldsymbol{\omega}_{chas}^\top\, \boldsymbol{I}_{chas,B'}\, \boldsymbol{\omega}_{chas} \\ T_{rot,l} &= \frac{1}{2}\, \boldsymbol{\omega}_l^\top\, \boldsymbol{I}_l\, \boldsymbol{\omega}_l \\ T_{rot,r} &= \frac{1}{2}\, \boldsymbol{\omega}_r^\top\, \boldsymbol{I}_r\, \boldsymbol{\omega}_r, \end{aligned} \tag{3.4}$$

where the inertia matrices ($\boldsymbol{I}_{chas,B'}$, $\boldsymbol{I}_l$, $\boldsymbol{I}_r$) are:

$$\boldsymbol{I}_{chas,B''} = \begin{bmatrix} I_{c11} & I_{c12} & I_{c13} \\ I_{c12} & I_{c22} & I_{c23} \\ I_{c13} & I_{c23} & I_{c33} \end{bmatrix}$$

$$\boldsymbol{S} = \underbrace{\begin{bmatrix} c_{\varphi_p} & 0 & s_{\varphi_p} \\ 0 & 1 & 0 \\ -s_{\varphi_p} & 0 & c_{\varphi_p} \end{bmatrix}}_{\mathbf{Rot}_{\mathbf{B''\,to\,B'}}} \tag{3.5}$$

$$\boldsymbol{I}_{chas,B'} = \boldsymbol{S}\,\boldsymbol{I}_{\boldsymbol{chas_{B''}}}\,\boldsymbol{S}^{\top}$$

$$\boldsymbol{I_l} = \boldsymbol{I_r} = \begin{bmatrix} I_t & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_t \end{bmatrix},$$

and the angular speed values of the three bodies are:

$$\boldsymbol{\omega}_{chas} = \begin{bmatrix} 0 & \dot{\varphi}_p & \dot{\theta} \end{bmatrix}^{\top}$$

$$\boldsymbol{\omega}_l = \begin{bmatrix} 0 & \dot{\varphi}_p + \dot{\varphi}_l & \dot{\theta} \end{bmatrix}^{\top} \tag{3.6}$$

$$\boldsymbol{\omega}_r = \begin{bmatrix} 0 & \dot{\varphi}_p + \dot{\varphi}_r & \dot{\theta} \end{bmatrix}^{\top}.$$

Combining Eqs. (3.4 - 3.6), we get the values of the three rotational kinetic energies:

$$\begin{aligned} T_{rot,chas} = {} & \dot{\theta} \left(\frac{\dot{\varphi}_p \left(I_{c23}\, c_{\varphi_p} - I_{c12}\, s_{\varphi_p} \right)}{2} \right) \\ & + \left(\frac{\dot{\theta}^2 \left(c_{\varphi_p} \left(I_{c33}\, c_{\varphi_p} - I_{c13}\, s_{\varphi_p} \right) - s_{\varphi_p} \left(I_{c13}\, c_{\varphi_p} - I_{c11}\, s_{\varphi_p} \right) \right)}{2} \right) \\ & + \dot{\varphi}_p \left(\frac{I_{c22}\, \dot{\varphi}_p}{2} + \frac{\dot{\theta} \left(I_{c23}\, c_{\varphi_p} - I_{c12}\, s_{\varphi_p} \right)}{2} \right) \end{aligned} \tag{3.7}$$

$$T_{rot,l} = \frac{I_t\, \dot{\theta}^2}{2} + I_a \left(\frac{\dot{\varphi}_l}{2} + \frac{\dot{\varphi}_p}{2} \right) \left(\dot{\varphi}_l + \dot{\varphi}_p \right)$$

$$T_{rot,r} = \frac{I_t\, \dot{\theta}^2}{2} + I_a \left(\frac{\dot{\varphi}_p}{2} + \frac{\dot{\varphi}_r}{2} \right) \left(\dot{\varphi}_p + \dot{\varphi}_r \right)$$

The global kinetic energy is the sum of all the kinetic energies calculated in the Eqs. (3.3) and (3.7):

$$T = T_{tra,chas} + T_{tra,l} + T_{tra,r} + T_{rot,chas} + T_{rot,l} + T_{rot,r}$$

Since T is a quadratic form, the Hessian of T gives the desired mass matrix:

$$
\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix}
m_c\, {c_\theta}^2 + m_c\, {s_\theta}^2 & 0 & \begin{matrix} -m_c\, y_G\, c_\theta \\ -m_c\, s_\theta\, (x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}) \end{matrix} & m_c\, c_\theta, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) & 0 & 0 \\
0 & m_c\, {c_\theta}^2 + m_c\, {s_\theta}^2 & \begin{matrix} m_c\, c_\theta, \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right) \\ -m_c\, y_G\, s_\theta \end{matrix} & m_c\, s_\theta\, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) & 0 & 0 \\
\begin{matrix} -m_c\, y_G\, c_\theta \\ -m_c\, s_\theta\, \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right) \end{matrix} & \begin{matrix} m_c\, c_\theta, \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right) \\ -m_c\, y_G\, s_\theta \end{matrix} & \begin{matrix} 2\, I_t \\ +m_c\, \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right)^2 \\ +2\, d^2\, m_w + m_c\, {y_G}^2 \\ +c_{\varphi_p}\, \left(I_{c33}\, c_{\varphi_p} - I_{c13}\, s_{\varphi_p}\right) \\ -s_{\varphi_p}\, \left(I_{c13}\, c_{\varphi_p} - I_{c11}\, s_{\varphi_p}\right) \end{matrix} & \begin{matrix} I_{c23}\, c_{\varphi_p} \\ -I_{c12}\, s_{\varphi_p} \\ -m_c\, y_G\, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) \end{matrix} & 0 & 0 \\
m_c\, c_\theta, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) & m_c\, s_\theta\, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) & \begin{matrix} I_{c23}\, c_{\varphi_p} \\ -I_{c12}\, s_{\varphi_p} \\ -m_c\, y_G\, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right) \end{matrix} & \begin{matrix} 2\, I_a + I_{c22} \\ +m_c\, \left(x_G\, c_{\varphi_p} + z_G\, s_{\varphi_p}\right)^2 \\ +m_c\, \left(z_G\, c_{\varphi_p} - x_G\, s_{\varphi_p}\right)^2 \end{matrix} & I_a & I_a \\
0 & 0 & 0 & I_a & I_a & 0 \\
0 & 0 & 0 & I_a & 0 & I_a
\end{bmatrix}
$$

3.1.2 Coriolis matrix

Recall that the (i, j) elements of $\boldsymbol{C}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)$ are given by the following formula [9]:

$$\boldsymbol{C}_{ij} = \frac{1}{2}\sum_{\mathrm{k}=1}^{\mathrm{n}}\left(\frac{\partial \boldsymbol{M}_{ij}}{\partial \boldsymbol{q}_{\mathrm{k}}} + \frac{\partial \boldsymbol{M}_{ik}}{\partial \boldsymbol{q}_{\mathrm{j}}} - \frac{\partial \boldsymbol{M}_{kj}}{\partial \boldsymbol{q}_{\mathrm{i}}}\right)\dot{\boldsymbol{q}}_{\mathrm{k}}$$

The resulting Coriolis matrix is:

$$\boldsymbol{C}(\boldsymbol{q}) = \begin{bmatrix}
0 & 0 & \begin{array}{c} \frac{\dot{\theta}\left(2\,m_c\,y_G\,s_\theta - 2\,m_c\,c_\theta\left(x_G\,c_{\varphi_p} + z_G\,s_{\varphi_p}\right)\right)}{2} \\ -m_c\,\dot{\varphi}_p\,s_\theta\left(z_G\,c_{\varphi_p} - x_G\,s_{\varphi_p}\right) \end{array} & \begin{array}{c} -m_c\,\dot{\varphi}_p\,c_\theta\left(x_G\,c_{\varphi_p} + z_G\,s_{\varphi_p}\right) \\ -m_c\,\dot{\theta}\,s_\theta\left(z_G\,c_{\varphi_p} - x_G\,s_{\varphi_p}\right) \end{array} & 0 & 0 \\
0 & 0 & \begin{array}{c} m_c\,\dot{\varphi}_p\,c_\theta\left(z_G\,c_{\varphi_p} - x_G\,s_{\varphi_p}\right) \\ -\frac{\dot{\theta}\left(2\,m_c\,y_G\,c_\theta + 2\,m_c\,s_\theta\left(x_G\,c_{\varphi_p} + z_G\,s_{\varphi_p}\right)\right)}{2} \end{array} & \begin{array}{c} m_c\,\dot{\theta}\,c_\theta\left(z_G\,c_{\varphi_p} - x_G\,s_{\varphi_p}\right) \\ -m_c\,\dot{\varphi}_p\,s_\theta\left(x_G\,c_{\varphi_p} + z_G\,s_{\varphi_p}\right) \end{array} & 0 & 0 \\
0 & 0 & \begin{array}{c} -\frac{\dot{\varphi}_p\left(m_c\,s_{\varphi_p}\,{x_G}^2 - 2\,m_c\,c_{\varphi_p}\,x_G\,z_G\right)}{2} \\ +\frac{\dot{\varphi}_p\left(m_c\,s_{\varphi_p}\,{z_G}^2 - 2\,I_{c13}\,c_{\varphi_p}\right)}{2} \\ +\frac{\dot{\varphi}_p\left(+I_{c11}\,s_{\varphi_p} - I_{c33}\,s_{\varphi_p}\right)}{2} \end{array} & \begin{array}{c} -\frac{m_c\,\dot{\theta}\,s_{\varphi_p}\,{x_G}^2}{2} \\ +m_c\,\dot{\theta}\,c_{\varphi_p}\,x_G\,z_G \\ +m_c\,\dot{\varphi}_p\,y_G\,c_{\varphi_p}\,x_G \\ +\frac{m_c\,\dot{\theta}\,s_{\varphi_p}\,{z_G}^2}{2} \\ +m_c\,\dot{\varphi}_p\,y_G\,s_{\varphi_p}\,z_G \\ -I_{c23}\,\dot{\varphi}_p\,s_{\varphi_p} - I_{c13}\,\dot{\theta}\,c_{\varphi_p} \\ +\frac{I_{c11}\,\dot{\theta}\,s_{\varphi_p}}{2} - \frac{I_{c33}\,\dot{\theta}\,s_{\varphi_p}}{2} \\ -I_{c12}\,\dot{\varphi}_p\,c_{\varphi_p} \end{array} & 0 & 0 \\
0 & 0 & \begin{array}{c} \frac{\dot{\theta}\left(m_c\,s_{\varphi_p}\,{x_G}^2 - 2\,m_c\,c_{\varphi_p}\,x_G\,z_G\right)}{2} \\ -\frac{\dot{\theta}\left(m_c\,s_{\varphi_p}\,{z_G}^2 - 2\,I_{c13}\,c_{\varphi_p}\right)}{2} \\ -\frac{\dot{\theta}\left(+I_{c11}\,s_{\varphi_p} - I_{c33}\,s_{\varphi_p}\right)}{2} \end{array} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

3.1.3 Generalized gravity force

To find the generalized gravity force, we need the gravitational potential energy of the robot. We assume that a mass has zero potential energy when it is located in the plane through M that is parallel to ground ($U(\boldsymbol{q}) = 0$ when $\varphi_p = 90°$). Thus, the wheels have always zero potential energy as their center of gravity is aligned with M, and we only have to count the potential energy of the chassis. In view of Fig. 3.1, this energy can be computed as follows using basis B''.

$$U(\boldsymbol{q}) = m_c \cdot \boldsymbol{GM}^\top \cdot \boldsymbol{g} = m_c \cdot \begin{bmatrix} -x_G & -y_G & -z_G \end{bmatrix} \cdot \begin{bmatrix} g \sin\varphi_p \\ 0 \\ -g\cos\varphi_p \end{bmatrix},$$

where $\boldsymbol{g}$ is the acceleration of gravity vector, g is its magnitude, and (x_G, y_G, z_G) are the coordinates of G in the chassis frame (green). Therefore:

$$U = g\, m_c\, z_G\, \cos(\varphi_p) - g\, m_c\, x_G\, \sin(\varphi_p)\,.$$

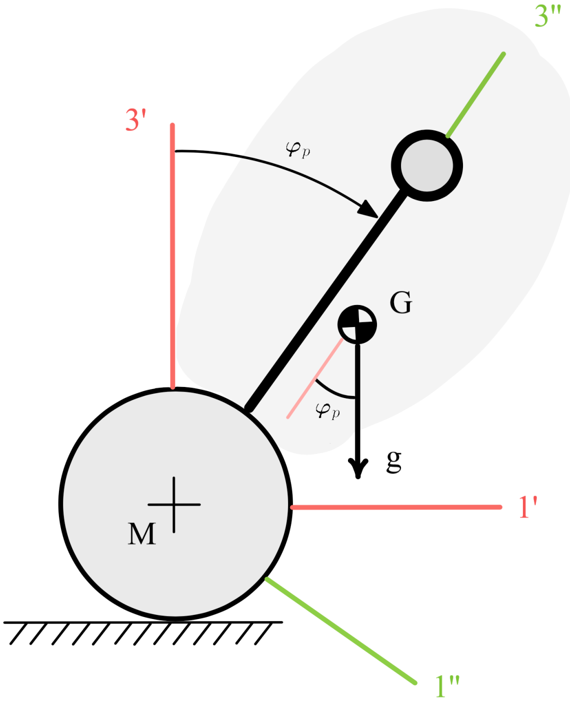


Figure 3.1: Geometry of the gravity force.

The generalized gravity force is $\boldsymbol{G}(\boldsymbol{q}) = \frac{\partial U(\boldsymbol{q})}{\partial \boldsymbol{q}}$. Therefore, we have

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g\, m_c\, x_G\, \cos(\varphi_p) - g\, m_c\, z_G\, \sin(\varphi_p) \\ 0 \\ 0 \end{bmatrix}.$$

3.1.4 Generalized force of actuation

Our vector $\boldsymbol{u}$ of motor torques is defined as

$$\boldsymbol{u} = \begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix},$$

where each τ represents the torque of the corresponding wheel. Each of the torques in $\boldsymbol{u}$ acts directly on a q_i coordinate and, therefore, the generalized force of actuation is

$$\boldsymbol{Q}_a = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \tau_l \\ \tau_r \end{bmatrix}. \tag{3.8}$$

To rewrite this force in the form $\boldsymbol{Q}_a = \boldsymbol{E} \cdot \boldsymbol{u}$ we only have to define

$$\boldsymbol{E} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{3.9}$$

3.1.5 Generalized force of friction

In this project we only consider the friction forces of the wheels' rotation. The expression of the generalized friction force is given by

$$\boldsymbol{Q}_f = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -b\,\dot{\varphi}_l \\ -b\,\dot{\varphi}_r \end{bmatrix}. \tag{3.10}$$

3.1.6 Generalized disturbance force

This disturbance will not be used to design the controller. Rather, it will serve to test the controller under unmodelled situations. For example, when there is some wind or a colision between the chassis and an object or person, or when the ground offers some rolling resistance. For the moment we just model a disturbance force $\boldsymbol{F_d} = [F_{dx}, F_{dy}, F_{dz}]$ applied at G, which would allow the modeling of the first two examples, but would not include the case of a change of the rolling resistance.

$$\boldsymbol{Q}_d = \begin{bmatrix} F_{dx}\,\cos(\theta) - F_{dy}\,\sin(\theta) \\ F_{dy}\,\cos(\theta) + F_{dx}\,\sin(\theta) \\ F_{dy}\,(x_G\,\cos(\varphi_p) + z_G\,\sin(\varphi_p)) - F_{dx}\,y_G \\ F_{dx}\,(z_G\,\cos(\varphi_p) - x_G\,\sin(\varphi_p)) - F_{dz}\,(x_G\,\cos(\varphi_p) + z_G\,\sin(\varphi_p)) \\ 0 \\ 0 \end{bmatrix},$$

where

$$\boldsymbol{F_d} \cdot \boldsymbol{v}_G = \boldsymbol{Q}_d \cdot \dot{\boldsymbol{q}}.$$

3.2 The x model

We now wish to obtain the robot model in the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$. This requires writing $\ddot{\boldsymbol{q}}$ as a function of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\boldsymbol{u}$, to then convert the 2^{nd} order ODE into a 1^{st} order one.

Recall that Lagrange's equation is

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} + \boldsymbol{C}\,\dot{\boldsymbol{q}} + \boldsymbol{G} + \boldsymbol{J}^{\top}\,\boldsymbol{\lambda} = \boldsymbol{Q}_a\ +\ \boldsymbol{Q}_f$$

where we have omitted the dependencies on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ for simplicity. Since we are working with a system of 6 equations in 9 unknown variables (6 coordinates in $\ddot{\boldsymbol{q}}$ and 3 in $\boldsymbol{\lambda}$) we need 3 additional equations to determine $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\lambda}$ for a given $\boldsymbol{u}$. These can be obtained by taking the time derivative of the kinematic constraint

$$\boldsymbol{J} \cdot \dot{\boldsymbol{q}} = \boldsymbol{0},$$

which gives

$$\boldsymbol{J}\, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\, \dot{\boldsymbol{q}} = 0,$$

or equivalently,

$$\boldsymbol{J}\, \ddot{\boldsymbol{q}} = -\dot{\boldsymbol{J}}\, \dot{\boldsymbol{q}}. \tag{3.11}$$

Equation (3.11) is called the acceleration constraint of the robot. In order to get the $\dot{\boldsymbol{J}}$ value we need to go back to the expression

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{I_3} & -\boldsymbol{N} \end{bmatrix}.$$

If we take the time derivative of the previous equation, we get

$$\dot{\boldsymbol{J}} = \begin{bmatrix} \boldsymbol{0_3} & -\dot{\boldsymbol{N}} \end{bmatrix},$$

where $\dot{\boldsymbol{N}}$ is the time derivative of $\boldsymbol{N}$ (Eq. (2.5)):

$$\dot{\boldsymbol{N}} = \begin{bmatrix} -r\,\dot{\theta}\,\sin(\theta) & -\frac{r\,\dot{\theta}\,\sin(\theta)}{2} & -\frac{r\,\dot{\theta}\,\sin(\theta)}{2} \\ r\,\dot{\theta}\,\cos(\theta) & \frac{r\,\dot{\theta}\,\cos(\theta)}{2} & \frac{r\,\dot{\theta}\,\cos(\theta)}{2} \\ 0 & 0 & 0 \end{bmatrix}.$$

Therefore, $\dot{\boldsymbol{J}}$ is:

$$\dot{\boldsymbol{J}} = \begin{bmatrix} 0 & 0 & 0 & r\,\dot{\theta}\,\sin(\theta) & \frac{r\,\dot{\theta}\,\sin(\theta)}{2} & \frac{r\,\dot{\theta}\,\sin(\theta)}{2} \\ 0 & 0 & 0 & -r\,\dot{\theta}\,\cos(\theta) & -\frac{r\,\dot{\theta}\,\cos(\theta)}{2} & -\frac{r\,\dot{\theta}\,\cos(\theta)}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The system formed by Lagrange's equation and the acceleration constraint is

$$\begin{cases} \boldsymbol{M}\, \ddot{\boldsymbol{q}} + \boldsymbol{C}\, \dot{\boldsymbol{q}} + \boldsymbol{J}^\top\, \boldsymbol{\lambda} = \boldsymbol{E}\, \boldsymbol{u} \\ \boldsymbol{J}\, \ddot{\boldsymbol{q}} = -\dot{\boldsymbol{J}}\, \dot{\boldsymbol{q}} \end{cases}$$

which can be written in matrix form as

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E}\, \boldsymbol{u} - \boldsymbol{C}\, \dot{\boldsymbol{q}} \\ -\dot{\boldsymbol{J}}\, \dot{\boldsymbol{q}} \end{bmatrix}.$$

The matrix on the LHS is called the **extended mass matrix**. This matrix is invertible, since $\boldsymbol{M}$ is positive-definite and $\boldsymbol{J}$ is full row rank. Thus we can write

$$\begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{E}\,\boldsymbol{u} - \boldsymbol{C}\,\dot{\boldsymbol{q}} \\ -\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}} \end{bmatrix}.$$

Therefore

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{I}_6 & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{E}\,\boldsymbol{u} - \boldsymbol{C}\,\dot{\boldsymbol{q}} \\ -\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}} \end{bmatrix}.$$

Finally, by considering the trivial equation $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}$ in conjunction with the earlier one we arrive at

$$\begin{bmatrix} \dot{\boldsymbol{q}} \\ \ddot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{q}} \\ \begin{bmatrix} \boldsymbol{I}_6 & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{E}\,\boldsymbol{u} - \boldsymbol{C}\,\dot{\boldsymbol{q}} \\ -\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}} \end{bmatrix} \end{bmatrix},$$

which, if we let $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$, gives the robot model in the state space form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}).$$

We refer to this equation as the $\boldsymbol{x}$ model of the robot.

A drawback of this model is that the $\boldsymbol{x}$ variables are not independent (they must fulfill $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$). Therefore:

- The numerical integration of $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ easily generates errors pulling $\boldsymbol{x}(t)$ away from the state space $\mathcal{X}$. In practice, this translates into sliding of the robot on the ground (both lateral and longitudinal).
- The equation cannot be used to design a common controller to stabilize a given trajectory, as such controllers typically assume the $\boldsymbol{x}$ variables to be independent.

We next derive an alternative model in independent coordinates, which overcomes these limitations in the case of stabilizing the constant velocity trajectories for the internal state $\boldsymbol{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$.

3.3 The y model

We start again from Lagrange's equation of the x-model

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} + \boldsymbol{C}\,\dot{\boldsymbol{q}} + \boldsymbol{G} + \boldsymbol{J}^\top\,\boldsymbol{\lambda} = \boldsymbol{Q}_a(u) + \boldsymbol{Q}_f + \boldsymbol{Q}_d,$$

and consider this parameterization of the feasible velocities $\dot{\boldsymbol{q}}$

$$\begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{\varphi}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{N} \\ \boldsymbol{I_3} \end{bmatrix} \cdot \dot{\boldsymbol{\varphi}},$$

which we write as

$$\dot{\boldsymbol{q}} = \boldsymbol{\Delta} \cdot \dot{\boldsymbol{\varphi}},$$

where $\boldsymbol{\Delta} = \begin{bmatrix} \boldsymbol{N} \\ \boldsymbol{I_3} \end{bmatrix}$.

If we multiply Lagrange's equation by $\boldsymbol{\Delta}^\top$ we get

$$\boldsymbol{\Delta}^\top \boldsymbol{M}\, \ddot{\boldsymbol{q}} + \boldsymbol{\Delta}^\top \boldsymbol{C}\, \dot{\boldsymbol{q}} + \boldsymbol{\Delta}^\top \boldsymbol{G} + (\boldsymbol{J}\boldsymbol{\Delta})^\top \boldsymbol{\lambda} = \boldsymbol{\Delta}^\top \boldsymbol{Q}_a(\boldsymbol{u}) + \boldsymbol{\Delta}^\top \boldsymbol{Q}_f + \boldsymbol{\Delta}^\top \boldsymbol{Q}_d.$$

Since the columns of $\boldsymbol{\Delta}$ are feasible velocities, and the rows of $\boldsymbol{J}$ are orthogonal to such velocities (by virtue of $\boldsymbol{J} \cdot \dot{\boldsymbol{q}} = \boldsymbol{0}$), $\boldsymbol{J}\boldsymbol{\Delta}$ must be $\boldsymbol{0}$. Therefore, the equation simplifies to

$$\boldsymbol{\Delta}^\top \boldsymbol{M}\, \ddot{\boldsymbol{q}} + \boldsymbol{\Delta}^\top \boldsymbol{C}\, \dot{\boldsymbol{q}} + \boldsymbol{\Delta}^\top \boldsymbol{G} = \boldsymbol{T_a}(\boldsymbol{u}) + \boldsymbol{T_f} + \boldsymbol{T_d},$$

where $\boldsymbol{T_a}(\boldsymbol{u}) = \boldsymbol{\Delta}^\top \boldsymbol{Q}_a(\boldsymbol{u})$, $\boldsymbol{T_f} = \boldsymbol{\Delta}^\top \boldsymbol{Q}_f$, and $\boldsymbol{T_d} = \boldsymbol{\Delta}^\top \boldsymbol{Q}_d$. Furthermore, if we apply the substitutions

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{\Delta} \cdot \dot{\boldsymbol{\varphi}} \\ \ddot{\boldsymbol{q}} &= \boldsymbol{\Delta} \cdot \ddot{\boldsymbol{\varphi}} + \dot{\boldsymbol{\Delta}} \cdot \dot{\boldsymbol{\varphi}}, \end{aligned}$$

we obtain

$$\bar{\boldsymbol{M}}\, \ddot{\boldsymbol{\varphi}} + \bar{\boldsymbol{C}}\, \dot{\boldsymbol{\varphi}} + \bar{\boldsymbol{G}} = \boldsymbol{T_a}(\boldsymbol{u}) + \boldsymbol{T_f} + \boldsymbol{T_d}, \tag{3.12}$$

where

$$\begin{aligned} \bar{\boldsymbol{M}} &= \boldsymbol{\Delta}^\top \boldsymbol{M}\, \boldsymbol{\Delta} \\ \bar{\boldsymbol{C}} &= \boldsymbol{\Delta}^\top (\boldsymbol{M}\, \dot{\boldsymbol{\Delta}} + \boldsymbol{C}\, \boldsymbol{\Delta}) \\ \bar{\boldsymbol{G}} &= \boldsymbol{\Delta}^\top \boldsymbol{G}. \end{aligned}$$

Equation (3.12) is the equation of motion in $\boldsymbol{\varphi}$ coordinates. This ODE is remarkable, as all of its terms depend only on $\boldsymbol{\varphi}$ or $\dot{\boldsymbol{\varphi}}$ (see our calculations below). In fact, they only depend on $(\varphi_p, \dot{\varphi}_p, \dot{\varphi}_l, \dot{\varphi}_r)$. To convert the ODE into first order form (Eq. (3.13)), we isolate $\ddot{\boldsymbol{\varphi}}$ and add the trivial equation $\dot{\boldsymbol{\varphi}} = \dot{\boldsymbol{\varphi}}$.

$$\begin{cases} \dot{\boldsymbol{\varphi}} = \dot{\boldsymbol{\varphi}} \\ \ddot{\boldsymbol{\varphi}} = \bar{\boldsymbol{M}}^{-1}(\boldsymbol{T_a}(\boldsymbol{u}) + \boldsymbol{T_f} + \boldsymbol{T_d} - \bar{\boldsymbol{C}}\, \dot{\boldsymbol{\varphi}} - \bar{\boldsymbol{G}}) \end{cases} \tag{3.13}$$

If we define $\boldsymbol{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$ and represent the RHS of the previous system by $\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{u})$, we have

$$\dot{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{u}).$$

We refer to $\dot{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{u})$ as the $\boldsymbol{y}$ model of the robot, and to $\boldsymbol{y}$ as the "internal state". The $\boldsymbol{y}$ model has these advantages:

- It is sufficient to predict the trajectory $\boldsymbol{\varphi}(t)$ for a given input $\boldsymbol{u}(\boldsymbol{t})$.
- Since the $\boldsymbol{\varphi}$ coordinates are independent, the result will also be compliant with $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$.
- The model allows the design of a feedback controller to stabilize the robot in the upright position, or moving with constant speeds v and $\dot{\theta}$, using proprioceptive feedback only.
- Since the terms of the model depend only on $(\varphi_p, \dot{\varphi}_p, \dot{\varphi}_l, \dot{\varphi}_r)$, the linearized model will not depend on (φ_l, φ_r). In other words, the system linearization about a state $\boldsymbol{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$ stays constant as long as φ_p and $\dot{\boldsymbol{\varphi}}$ do not change.

3.4 Odometry

Suppose we have simulated the robot using the $\boldsymbol{y}$ model $\dot{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{u})$. We then have $(\boldsymbol{\varphi}(t), \dot{\boldsymbol{\varphi}}(t))$, but in some situations we would like to recover the full configuration $(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))$. The angle $\theta(t)$ is easy to obtain, since using the holonomic constraint we have:

$$\theta(t) = \frac{r}{2d}(\varphi_r(t) + \varphi_l(t)) + K_0.$$

To recover $\boldsymbol{r}(t) = (x(t), y(t))$, we can use the first two equations of $\dot{\boldsymbol{p}} = \boldsymbol{N}(\theta) \cdot \dot{\boldsymbol{\varphi}}$, which we write as

$$\dot{\boldsymbol{r}} = \boldsymbol{V}(\theta) \cdot \dot{\boldsymbol{\varphi}},$$

where $\boldsymbol{V}(\theta)$ is the matrix defined by the first two rows of $\boldsymbol{N}(\theta)$. Therefore:

$$\boldsymbol{r}(t) = \boldsymbol{r}(0) + \int_0^t \boldsymbol{V}(\theta(t)) \cdot \dot{\boldsymbol{\varphi}}(t) \cdot dt.$$

This integral will be computed numerically using Matlab's *integral* function in our implementation (Chapter 5). This function admits vector-valued integrands like $\boldsymbol{V}(\theta(t)) \cdot \boldsymbol{\varphi}(t)$.

If the values $\boldsymbol{r}_k = \boldsymbol{r}(t_k)$ have to be computed on-line from a sequence of values $\theta_k = \theta(t_k)$, $\dot{\boldsymbol{\varphi}}_k = \dot{\boldsymbol{\varphi}}(t_k)$, for $k = 0, 1, 2, \ldots$; we can use the following recurrence, which approximates the integral from t_k to t_{k+1} by using the trapezoidal rule:

$$\begin{cases} \boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \dfrac{h}{2}\left(\boldsymbol{V}(\theta_k)\cdot\dot{\boldsymbol{\varphi}}_k + \boldsymbol{V}(\theta_{k+1})\cdot\dot{\boldsymbol{\varphi}}_{k+1}\right) \\ \boldsymbol{r}_0 = \boldsymbol{r}(0) \end{cases}$$

where

$$h = t_{k+1} - t_k.$$

3.5 Particularized y model

In order to have a working model to validate our controller, the $\boldsymbol{y}$ model has been particularized assuming the chassis has a symmetric design in which:

- G is located on axis 3", so $x_G = y_G = 0$.
- The chassis inertia tensor at G is diagonal.

Although real life robots are not that perfect, most of the designs made are not very far from being symmetric. For these reasons our calculations will not be very distant from reality, and small variations can always be compensated by a well designed controller.

The geometric parameters we choose are inspired by the Loomo robot made by Segway [10]. We used this robot as a reference as it was a small robot that came with the same movement mechanism that we are developing, and matched with the idea that we had. With this in mind, and assuming the wheels and the chassis are cylinders of uniform mass distribution, the resulting dimensions of our particular twinbot are specified in Table 3.1.

If we now substitute the values of the Table 3.1 into the dynamic matrices, we will have the final system we will be working with. The notation $\bar{\boldsymbol{M}}_{twinbot}$, $\bar{\boldsymbol{C}}_{twinbot}$, $\bar{\boldsymbol{G}}_{twinbot}$, $\bar{\boldsymbol{T}}_{\boldsymbol{f}\,twinbot}$, $\bar{\boldsymbol{T}}_{\boldsymbol{d}twinbot}$ means that we are using the $\boldsymbol{y}$-model with the substituted parameters of our twinbot. We can see the results in the following equations:

$$\bar{\boldsymbol{M}}_{twinbot} = \begin{bmatrix} 0.8775\,\cos(\varphi_p) + 1.507 & 0.2194\,\cos(\varphi_p) + 0.2138 & 0.2194\,\cos(\varphi_p) + 0.2138 \\ 0.2194\,\cos(\varphi_p) + 0.2138 & 0.2415 - 0.02361\,\cos(2.0\,\varphi_p) & 0.02361\,\cos(2.0\,\varphi_p) - 0.02779 \\ 0.2194\,\cos(\varphi_p) + 0.2138 & 0.02361\,\cos(2.0\,\varphi_p) - 0.02779 & 0.2415 - 0.02361\,\cos(2.0\,\varphi_p) \end{bmatrix}.$$

$$\bar{\boldsymbol{C}}_{twinbot} =$$

$$\begin{bmatrix} -0.4388\,\dot{\varphi}_p\,s_{\varphi_p} & \begin{array}{c} -0.02361\,s_{2\,\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \\ -0.03148\,s_{\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \end{array} & \begin{array}{c} 0.02361\,s_{2\,\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \\ +0.03148\,s_{\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \end{array} \\ \begin{array}{c} 0.03148\,s_{\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \\ -0.2194\,\dot{\varphi}_p\,s_{\varphi_p} \\ +0.02361\,s_{2\,\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \end{array} & 0.02361\,\dot{\varphi}_p\,s_{2\,\varphi_p} & \begin{array}{c} 0.03148\,s_{\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \\ -0.02361\,\dot{\varphi}_p\,s_{2\,\varphi_p} \end{array} \\ \begin{array}{c} -0.03148\,s_{\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \\ -0.219375\,\dot{\varphi}_p\,s_{\varphi_p} \\ -0.02361\,s_{2\,\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \end{array} & \begin{array}{c} -0.03148\,s_{\varphi_p}\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) \\ -0.02361\,\dot{\varphi}_p\,s_{2\,\varphi_p} \end{array} & 0.02361\,\dot{\varphi}_p\,s_{2\,\varphi_p} \end{bmatrix}.$$

$$\bar{\boldsymbol{G}}_{twinbot} = \begin{bmatrix} -28.67\,\sin\left(\varphi_p\right) \\ 0 \\ 0 \end{bmatrix}$$

$$\bar{\boldsymbol{T}}_{f_{twinbot}} = \begin{bmatrix} 0 \\ -0.1\,\dot{\varphi}_l \\ -0.1\,\dot{\varphi}_r \end{bmatrix}.$$

$$\bar{\boldsymbol{T}}_{d_{twinbot}} = \begin{bmatrix} F_{dx}\,r - F_{dz}\,z_G\,\sin\left(\varphi_p\right) + F_{dx}\,z_G\,\cos\left(\varphi_p\right) \\ \frac{r\left(F_{dx}\,d - F_{dy}\,z_G\,\sin(\varphi_p)\right)}{2\,d} \\ \frac{r\left(F_{dx}\,d + F_{dy}\,z_G\,\sin(\varphi_p)\right)}{2\,d} \end{bmatrix}.$$

As the Force vector ($\boldsymbol{F} = [F_{dx},\, F_{dy},\, F_{dz}]$) is expressed in the B' reference system, and the force applied has been defined in the direction of the movement, $\boldsymbol{F} = [F_{dx},\, 0,\, 0]$. Then, $\bar{\boldsymbol{T}}_{d_{twinbot}}$ should be modeled as:

$$\bar{\boldsymbol{T}}_{d_{twinbot}} = \begin{bmatrix} r\,F_{dx} + z_G\,F_{dx}\,\cos(\varphi_p) \\ \left(\frac{r}{2}\right)F_{dx} \\ \left(\frac{r}{2}\right)F_{dx} \end{bmatrix} = \begin{bmatrix} 0.15\,F_{dx} + 0.225\,F_{dx}\,\cos(\varphi_p) \\ 0.075\,F_{dx} \\ 0.075\,F_{dx} \end{bmatrix}.$$

Table 3.1: Nominal parameters of the robot assumed

Symbol	Meaning	Value	Unit
g	Acceleration of gravity	9.81	$\mathrm{m\ s^{-2}}$
x_G	x coord of G	0	m
y_G	y coord of G	0	m
z_G	z coord of G	0.225	m
d	Semiaxis length	0.280	m
r_w	Wheel radius	0.150	m
h_w	Wheel with	0.050	m
m_w	Wheel mass	3	kg
I_a	Wheel axial inertia moment	0.150	$\mathrm{kg\ m^2}$
I_t	Wheel twisting inertia moment	0.150	$\mathrm{kg\ m^2}$
r_c	Chassis radius	0.180	m
h_c	Chassis height	0.550	m
m_c	Chassis mass	13	kg
I_{c11}	Chassis inertia tensor 11 element	$m_c\ (r_c^2/2 + h_c^2/12)$	$\mathrm{kg\ m^2}$
I_{c12}	Chassis inertia tensor 12 element	0	$\mathrm{kg\ m^2}$
I_{c13}	Chassis inertia tensor 13 element	0	$\mathrm{kg\ m^2}$
I_{c22}	Chassis inertia tensor 22 element	$m_c\ r_c^2$	$\mathrm{kg\ m^2}$
I_{c23}	Chassis inertia tensor 23 element	0	$\mathrm{kg\ m^2}$
I_{c33}	Chassis inertia tensor 33 element	I_{c11}	$\mathrm{kg\ m^2}$
b	Viscous friction coeff. wheels	0.150	$\mathrm{kg\ m^2\ s^{-1}}$
$\tau_{max,w}$	Maximum torque applicable by a wheel motor	10	Nm

4
Optimal Control Theory

In this chapter we provide the optimal control theory required in order to stabilize a twinbot around a constant speed trajectory. The objective is that the robot does not fall on the ground, while trying to follow the different linear and angular velocities given in real time.

4.1 The finite horizon LQR controller

Let us consider a generic linear time-varying (LTV) system as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\,\boldsymbol{x}(t) + \boldsymbol{B}(t)\,\boldsymbol{u}(t)$$

which we wish to stabilize at $\boldsymbol{x} = \boldsymbol{0}$, under $\boldsymbol{u} = \boldsymbol{0}$, during T seconds. To this end we need to find a feedback law $\boldsymbol{u}(t) = \Pi(\boldsymbol{x}(t))$ such that all solution trajectories $\boldsymbol{x}(t)$ of the ODE

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\,\boldsymbol{x}(t) + \boldsymbol{B}(t)\,\Pi(\boldsymbol{x})$$

stay close to $\boldsymbol{x} = \boldsymbol{0}$, $\boldsymbol{u} = \boldsymbol{0}$ for $t \in [0, T]$. There are many such laws and we will choose one that is optimal in a specific sense.

According to optimal control theory, if we wish that $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t) = \Pi(\boldsymbol{x}(t))$ minimize

$$\boldsymbol{J}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \boldsymbol{x}(T)^T\,\boldsymbol{Q_f}\,\boldsymbol{x}(T) + \int_0^T (\boldsymbol{x}^T\,\boldsymbol{Q}\,\boldsymbol{x} + \boldsymbol{u}^T\,\boldsymbol{R}\,\boldsymbol{u})dt,$$

then, the control law takes the form

$$\boldsymbol{u}(t) = -\boldsymbol{K}(t)\,\boldsymbol{x}(t),$$

where

$$\boldsymbol{J}(t) = -\boldsymbol{R}^{-1}\,\boldsymbol{B}(t)^T\,\boldsymbol{S}(t),$$

and $\boldsymbol{S}(t)$ is the solution for $t \in [0, T]$ of the Ricatti equation

$$\dot{\boldsymbol{S}} = -\boldsymbol{S}\,\boldsymbol{A} - \boldsymbol{A}^T\,\boldsymbol{S} + \boldsymbol{S}\,\boldsymbol{B}\,\boldsymbol{R}^{-1}\,\boldsymbol{B}^T\,\boldsymbol{S} - \boldsymbol{Q} \tag{4.1}$$

that satisfies the final condition

$$\boldsymbol{S}(t) = \boldsymbol{Q}_f. \tag{4.2}$$

Some considerations must be taken into account about the finite horizon LQR controller:

- The previous theory assumes that $\boldsymbol{x}$ and $\boldsymbol{u}$ are not subject to any constraints, so the coordinates in $\boldsymbol{x}$ and $\boldsymbol{u}$ must be independent and unbounded.

- To implement the previous control law, $\boldsymbol{S}(t)$ must be computed a priori via solving the Ricatti equation numerically from $\boldsymbol{S}(T) = \boldsymbol{Q}_f$. This can be done using the *ode45* command that Matlab offers. For its implementation, it is necessary to have the values of $\boldsymbol{S}(t)$ and $\boldsymbol{K}(t) = \boldsymbol{R}^{-1}\,\boldsymbol{B}(t)^T\,\boldsymbol{S}(t)$ stored for all $t \in [0, T]$ (or sufficient discrete values).

- $\dot{\boldsymbol{S}}(t)$ is symmetric for all t, which implies that $\boldsymbol{S}(t)$ must be symmetric for all t too. It can also be shown that $\boldsymbol{S}(t)$ is positive semidefinite. Tedrake shows how these properties can be preserved by using the factorization $\boldsymbol{S}(t) = \boldsymbol{P}(t)\,\boldsymbol{P}(t)^\top$ and an alternative form of Eqs. (4.1) and (4.2) [6] . An alternative is to only integrate the upper $(n^2 - n)/(2 + n)$ components of $\boldsymbol{S}(t)$.

4.2 The infinite horizon LQR controller

There is a case in which the gain matrix $\boldsymbol{K}(t)$ is constant $\forall t$. This occurs when the following conditions are all met [11]:

- $\boldsymbol{A}$ and $\boldsymbol{B}$ are constant $\forall t$, so the system is linear time-invariant (LTI).

- $\boldsymbol{Q}$ and $\boldsymbol{R}$ are also constant $\forall t$.

- The system $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\,\boldsymbol{x}(t) + \boldsymbol{B}\,\boldsymbol{u}(t)$ is controllable.

- $\boldsymbol{Q}_f = \mathbf{0}$ (there is no final state penalty)

- $T = \infty$, so we consider an infinite-horizon problem.

In this situation, since the Ricatti equation is integrated backwards in time, the solution $\boldsymbol{S}(t)$ can be expected to approach a constant matrix $\boldsymbol{S}$ for t near zero. Accordingly $\dot{\boldsymbol{S}}(t)$ approaches zero, so the limiting matrix $\boldsymbol{S}$ is the solution of the algebraic Ricatti equation:

$$\mathbf{0} = -\boldsymbol{S}\,\boldsymbol{A} - \boldsymbol{A}^T\,\boldsymbol{S} + \boldsymbol{S}\,\boldsymbol{B}\,\boldsymbol{R}^{-1}\,\boldsymbol{B}^T\,\boldsymbol{S} - \boldsymbol{Q}. \tag{4.3}$$

In this case, the optimal control torque is simply

$$\boldsymbol{u}(t) = -\boldsymbol{R}^{-1}\,\boldsymbol{B}^T\,\boldsymbol{S}\cdot\boldsymbol{x}(t) = -\boldsymbol{K}\cdot\boldsymbol{x}(t),$$

where $\boldsymbol{K}$ is a constant matrix. Thus, if our system is LTI and we do not have a requirement to bring it to $\boldsymbol{x} = \mathbf{0}$ in some specific time T, this law is preferable because it is easier to implement as the gains won't be changing through time. In this case, for the calculations used in the forward sections, the matrix $\boldsymbol{K}$ can be computed in Matlab using the command lqr [11].

4.3 Stabilization of a given trajectory

Once the controller to be used is defined, we are ready to stabilize a twinbot along a given trajectory $(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t))$, that satisfies

$$\dot{\boldsymbol{x}}_0 = \boldsymbol{f}(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)),$$

where $\boldsymbol{f}$ is the dynamic function previously seen in Eq. (3.2).

That is, we aim the reference trajectory to be an attractor of the closed-loop system. In this manner, if we start the robot at $(\boldsymbol{x}_0(0), \boldsymbol{u}_0(0))$, it will follow the trajectory; and if it suffers some disturbance that deviates it from $(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t))$, it will return back to it. We wish the closed-loop system to have a similar behaviour as the representation in Fig. 4.1.

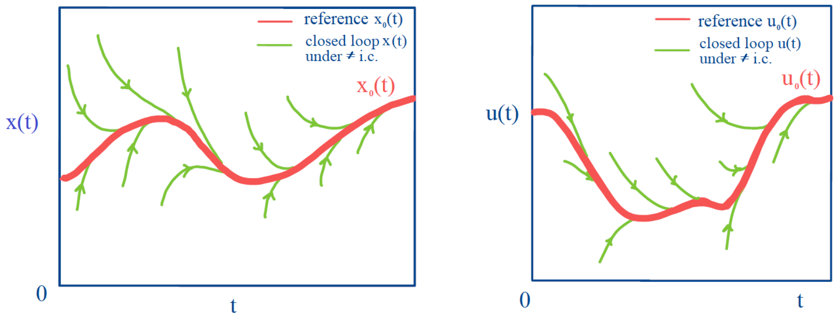


Figure 4.1: Ideal closed-loop behaviour.

To construct the controller, let us consider the time varying linearization of the system at $(\boldsymbol{x}_0(0), \boldsymbol{u}_0(0))$:

$$\dot{\boldsymbol{x}} = \underbrace{\boldsymbol{f}(\boldsymbol{x}_0(0), \boldsymbol{u}_0(0))}_{\dot{\boldsymbol{x}}_0} + \underbrace{\left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_0 \\ \boldsymbol{u}=\boldsymbol{u}_0}}}_{\boldsymbol{A}} (\boldsymbol{x} - \boldsymbol{x}_0) + \underbrace{\left.\frac{\partial f}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_0 \\ \boldsymbol{u}=\boldsymbol{u}_0}}}_{\boldsymbol{B}} (\boldsymbol{u} - \boldsymbol{u}_0). \tag{4.4}$$

If now we apply the change of variables

$$\begin{aligned} \overline{\boldsymbol{x}} &= (\boldsymbol{x} - \boldsymbol{x}_0) \Rightarrow \dot{\overline{\boldsymbol{x}}} = \dot{\boldsymbol{x}}, \\ \overline{\boldsymbol{u}} &= (\boldsymbol{u} - \boldsymbol{u}_0), \end{aligned}$$

then Eq. (4.4) looks like

$$\dot{\overline{\boldsymbol{x}}} = \boldsymbol{A}\,\overline{\boldsymbol{x}} + \boldsymbol{B}\,\overline{\boldsymbol{u}}. \tag{4.5}$$

Formally, Eq. (4.5) is a linear time-varying system. This is because $\boldsymbol{A}$ and $\boldsymbol{B}$ are evaluated at $(\boldsymbol{x}_0, \boldsymbol{u}_0)$, which are not constant values, therefore $\boldsymbol{A}$ and $\boldsymbol{B}$ are time-variant. This way, generally we would have to use a finite-horizon LQR controller to stabilize this system at $(\overline{\boldsymbol{x}} = 0, \overline{\boldsymbol{u}} = 0)$, which means stabilizing at $(\boldsymbol{x} = \boldsymbol{x}_0, \boldsymbol{u} = \boldsymbol{u}_0)$:

$$\begin{aligned} \overline{\boldsymbol{x}} = \boldsymbol{0} \Rightarrow (\boldsymbol{x} - \boldsymbol{x}_0(t)) = \boldsymbol{0} \Rightarrow \boldsymbol{x} = \boldsymbol{x}_0(t), \\ \overline{\boldsymbol{u}} = \boldsymbol{0} \Rightarrow (\boldsymbol{u} - \boldsymbol{u}_0(t)) = \boldsymbol{0} \Rightarrow \boldsymbol{u} = \boldsymbol{u}_0(t). \end{aligned}$$

The feedback law would be, as before

$$\overline{\boldsymbol{u}} = -\boldsymbol{K}\,\overline{\boldsymbol{x}} \Rightarrow \boldsymbol{u}(t) = \boldsymbol{u}_0(t) - \boldsymbol{K}(t)\,(\boldsymbol{x} - \boldsymbol{x}_0(t)),$$

where $\boldsymbol{K}$ is time-dependent because $\boldsymbol{K} = \boldsymbol{R}^{-1}\,\boldsymbol{B}^\top\,\boldsymbol{S}$, and both $\boldsymbol{B}$ and $\boldsymbol{S}$ (Eq. (4.3)) vary through time. Thus, $\boldsymbol{u}$ is time-dependent too.

However, if for some reason $\boldsymbol{A}$ and $\boldsymbol{B}$ happen to be constant along the trajectory, then we can implement the ∞-horizon LQR controller, for which $\boldsymbol{S}$ is constant (and so is $\boldsymbol{K} = \boldsymbol{R}^{-1}\,\boldsymbol{B}^\top\,\boldsymbol{S}$).

In our twinbot we will see that the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices for the $\boldsymbol{y}$-model are constant along trajectories that keep

$$\left.\begin{aligned} v &= constant \\ \dot{\theta} &= constant \\ \varphi_p &= constant \end{aligned}\right\} \forall t.$$

For these trajectories we will be able to design an ∞-horizon LQR controller.

5 Controller design

In this chapter we will apply the theory from the previous chapter to design a control law that is able to stabilize a twinbot along a constant velocity trajectory, determined by the desired linear and angular velocities for the chassis (v and $\dot{\theta}$). Lastly we will provide the simulation results to analize the behavior of a twinbot while working with the control system.

5.1 A feedback law for a twinbot

The $\boldsymbol{y}$-model of a twinbot ($\dot{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{u})$) describes the evolution of the internal state $\boldsymbol{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$. In this project, we are going to use this model to design a feedback law that keeps the robot moving under constant velocities v and $\dot{\theta}$, while keeping $\varphi_p(t) = 0 \; \forall t$. This corresponds to asking the robot to move under the following angular velocities for all t:

$$\begin{cases} \dot{\varphi}_p = 0 \\ \dot{\varphi}_l = \dfrac{v - d\,\dot{\theta}}{r} \\ \dot{\varphi}_r = \dfrac{v + d\,\dot{\theta}}{r} \end{cases}$$

where v, $\dot{\theta}$ are the input variables, and d, r are geometric parameters previously defined in Table 3.1. Assuming $\varphi_l(0) = \varphi_r(0) = 0$, the angles $\varphi_l(t)$ and $\varphi_r(t)$ will be

$$\varphi_l(t) = \tfrac{v - d\,\dot{\theta}}{r}\, t\,, \quad \varphi_r(t) = \tfrac{v + d\,\dot{\theta}}{r}\, t.$$

In sum, the internal state trajectory to be followed is:

$$\boldsymbol{y}_0 = \frac{1}{r}\begin{bmatrix} 0 \\ (v - d\,\dot{\theta})\,t \\ (v + d\,\dot{\theta})\,t \\ 0 \\ v - d\,\dot{\theta} \\ v + d\,\dot{\theta} \end{bmatrix}.$$

Moreover, the torques $\boldsymbol{u} = \begin{bmatrix} \tau_{l_0} \\ \tau_{r_0} \end{bmatrix}$ that allow the robot to maintain this trajectory can be obtained by substituting

$$\varphi_p = 0\,, \quad \dot{\varphi}_p = 0\,, \quad \dot{\varphi}_l = \tfrac{v - d\,\dot{\theta}}{r}\,, \quad \dot{\varphi}_r = \tfrac{v + d\,\dot{\theta}}{r}\,, \quad \ddot{\varphi}_p = \ddot{\varphi}_l = \ddot{\varphi}_r = 0$$

in the equation of motion (3.12) and solving for $\boldsymbol{T_a}(\boldsymbol{u})$. For the specific case of $x_G = 0$, $y_G = 0$ and $I_c(1,3) = 0$, we get:

$$\boldsymbol{T_a}(\boldsymbol{u}) = \begin{bmatrix} 0 \\ \frac{b\,(v - d\,\dot{\theta})}{r} \\ \frac{b\,(v + d\,\dot{\theta})}{r} \end{bmatrix},$$

so the required wheel torques are

$$\boldsymbol{u}_0 = \begin{bmatrix} \tau_{l_0}(t) \\ \tau_{r_0}(t) \end{bmatrix} = \begin{bmatrix} \frac{b\,(v - d\,\dot{\theta})}{r} \\ \frac{b\,(v + d\,\dot{\theta})}{r} \end{bmatrix}.$$

In our case, $\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{u})$ only depends on

$$\varphi_p\,, \quad \dot{\varphi}_p\,, \quad \dot{\varphi}_l\,, \quad \dot{\varphi}_r\,, \quad \tau_l\,, \quad \tau_r \;\; .$$

Since these values remain constant along $\boldsymbol{y}_0(t)$, the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$

$$\boldsymbol{A} = \frac{\partial g}{\partial \boldsymbol{y}}\bigg|_{\substack{\boldsymbol{y}=\boldsymbol{y}_0(t) \\ \boldsymbol{u}=\boldsymbol{u}_0(t)}} \qquad \boldsymbol{B} = \frac{\partial g}{\partial \boldsymbol{u}}\bigg|_{\substack{\boldsymbol{y}=\boldsymbol{y}_0(t) \\ \boldsymbol{u}=\boldsymbol{u}_0(t)}} \tag{5.1}$$

will be constant along $(\boldsymbol{y}_0, \boldsymbol{u}_0)$. In other words, the linearization of the system along the desired trajectory $\boldsymbol{y}_0(t)$ will be an LTI system of the form

$$\dot{\bar{\boldsymbol{y}}} = \boldsymbol{A}\,\bar{\boldsymbol{y}} + \boldsymbol{B}\,\bar{\boldsymbol{u}},$$

where $\bar{\boldsymbol{y}}(t)$ is the state error of the system and $\bar{\boldsymbol{u}}(t)$ is the action error (5.2).

$$\begin{aligned} \bar{\boldsymbol{y}}(t) &= \boldsymbol{y}(t) - \boldsymbol{y}_0(t) \leftarrow \text{State error} \\ \bar{\boldsymbol{u}}(t) &= \boldsymbol{u}(t) - \boldsymbol{u}_0(t) \leftarrow \text{Action error.} \end{aligned} \tag{5.2}$$

In sum, as we are working with constant $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, we can stabilize the system along $\boldsymbol{y}_0(t)$ with an infinite-horizon LQR controller.

Assuming that the controller has to minimize

$$\boldsymbol{J}(\boldsymbol{y}(t), \bar{u}(t)) = \int_0^\infty [\bar{\boldsymbol{y}}(t)^\top \boldsymbol{Q}\, \bar{\boldsymbol{y}}(t) + \bar{\boldsymbol{u}}(t)^\top \boldsymbol{R}\, \bar{\boldsymbol{u}}(t)]dt,$$

the feedback law will be

$$\bar{u}(t) = -\boldsymbol{K}\, \bar{\boldsymbol{y}}(t) \rightarrow \boldsymbol{u}(t) = \boldsymbol{u}_0(t) - \boldsymbol{K}\,(\boldsymbol{y}(t) - \boldsymbol{y}_0(t)),$$

where $\boldsymbol{K} = \boldsymbol{R}^{-1}\, \boldsymbol{B}^\top\, \boldsymbol{S}$, and $\boldsymbol{S}$ is the solution of the algebraic Ricatti equation (4.3).

The closed-loop system's stability is directly related to the state error, which behaves according to the following expression.

$$\dot{\bar{\boldsymbol{y}}} = (\boldsymbol{A} - \boldsymbol{B}\,\boldsymbol{K})\,\bar{\boldsymbol{y}}.$$

This system will be stable (all trajectories $\bar{\boldsymbol{y}}$ will converge to the origin as $t \to \infty$) if, and only if, all eigenvalues of $\boldsymbol{A} - \boldsymbol{B}\,\boldsymbol{K}$ have their real parts negative.

To make sure if we are working with a stable system, we are going to use the MATLAB command lqr, which returns the eigvals of $\boldsymbol{A} - \boldsymbol{B}\,\boldsymbol{K}$, as well as the values of the matrix $\boldsymbol{K}$.

5.2 The control loop algorithm

In this section we will go over the control loop algorithm that allowed us to move a twinbot while keeping it in an stable position.

5.2.1 Initializations

First of all, we must define the initial conditions of the robot ($\boldsymbol{y}_i$) and the desired equilibrium state ($\boldsymbol{y}_0$). It is not needed to provide the $\boldsymbol{x}$ coordinates, as we will be working with the $\boldsymbol{y}$-model for the calculations. In our case we set

$$\begin{aligned} \boldsymbol{y}_i &= [\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ], \\ \boldsymbol{y}_0 &= [\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]. \end{aligned}$$

Secondly, we set the control gains of the system ($\boldsymbol{Q}$ and $\boldsymbol{R}$). To do so we use the Bryson rule [12] which gives us a reference basis from where to start adjusting the two gains. At the end there's no rule that works perfectly for all the control systems, but this one provides a starting point. Bryson proposes that the diagonal elements of the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ should be calculated with the following formula:

$$\boldsymbol{Q}_{ii} = \frac{1}{\text{maximum acceptable value of } \boldsymbol{y}(i)^2}, \quad i \in [1, 2, ..., 6]$$
$$\boldsymbol{R}_{jj} = \frac{1}{\text{maximum acceptable value of } \boldsymbol{u}(j)^2}, \quad j \in [1, 2]$$

We have set a maximum wheel's torque of $10Nm$, therefore $\boldsymbol{R}_{jj}$ should be 0.01. For $\boldsymbol{Q}_{ii}$ we start by setting them to the same value as $\boldsymbol{R}_{jj}$. Afterwards we need to start a trial-and-error iterative adjusting procedure until we get the acceptable results. In our case, all the elements of the diagonal of $\boldsymbol{Q}$ and $\boldsymbol{R}$ were set to 0.01, except for $\boldsymbol{Q}_{11}$ and the $\boldsymbol{Q}_{44}$, that correspond to φ_p and $\dot{\varphi}_p$, which are set 1000 times higher. With this modification we improve the stabilization of the chassis.

Next, we draw the initial configuration of the robot. To do it, we used the URDF technique, which allowed us to build a multi-body tree of a twinbot (Fig. 5.1).

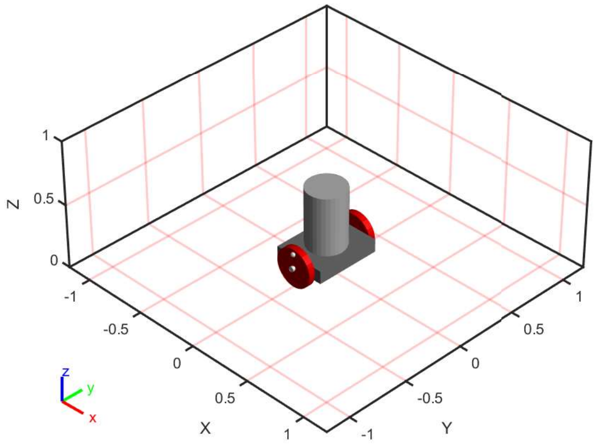


Figure 5.1: A simplified 3D model of a twinbot.

In section 3.5, we described a simplified model of the robot made by three cylinders, one for each wheel and one for the chassis. For this reason, our robot will have these three bodies. There are three main variable spatial transformations between rigid bodies that need to be given: one specifying the position of the chassis with respect to the floor, and one for each wheel specifying its position with respect to the chassis. The chassis position involves two translations (x, y), and two rotations (θ, φ_p), whereas each wheel involves only one rotation (φ_l and φ_r). Once the configuration is drawn, we can set the timer to 0 and begin the control loop.

5.2.2 Control loop

The first step to take is to read the input variables v_0 and $\dot{\theta}_0$, from an input device (a keyboard), and recalculate the $\boldsymbol{y}_0$ coordinates. The φ_{p_0} and $\dot{\varphi}_{p_0}$ values do not change, as we always want to keep the chassis as vertical as possible.

The next step is to obtain the $\boldsymbol{K}$ matrix, for which $\boldsymbol{A}$ and $\boldsymbol{B}$ are needed. The last two matrices are obtained following (5.1), and for the computation of $\boldsymbol{K}$, we use the MATLAB lqr command.

With the value of $\boldsymbol{K}$, we can finally calculate the new $\boldsymbol{y}$ coordinates. To do so, we used a function handle for the $\boldsymbol{u}$ variable, which allowed us to do the $ode45$ numerical integration. This function implements a Runge-Kutta method with a variable time step for efficient computation [13–15]. It probably was enough to use a Runge-Kutta 4, but as the computing times were not very different, we went for the most precise method.

Finally, we can recalculate the $\boldsymbol{x}$-model coordinates, using the odometry explained in section 3.4. It is important to keep in mind that for the recalculation of the $\boldsymbol{x}$-model, we need to find the new $\boldsymbol{N}$ matrix, as it is directly related to θ, which is constantly changing.

To end the loop, we draw the new configuration of the robot, using the same method explained in section 5.2.1.

The iteration time of the control loop is set to $20ms$, as most of our loops take between 10 and $15ms$ to compute. By setting the iteration time higher than the response time of most loops, we minimize the chances of over-passing the set time.

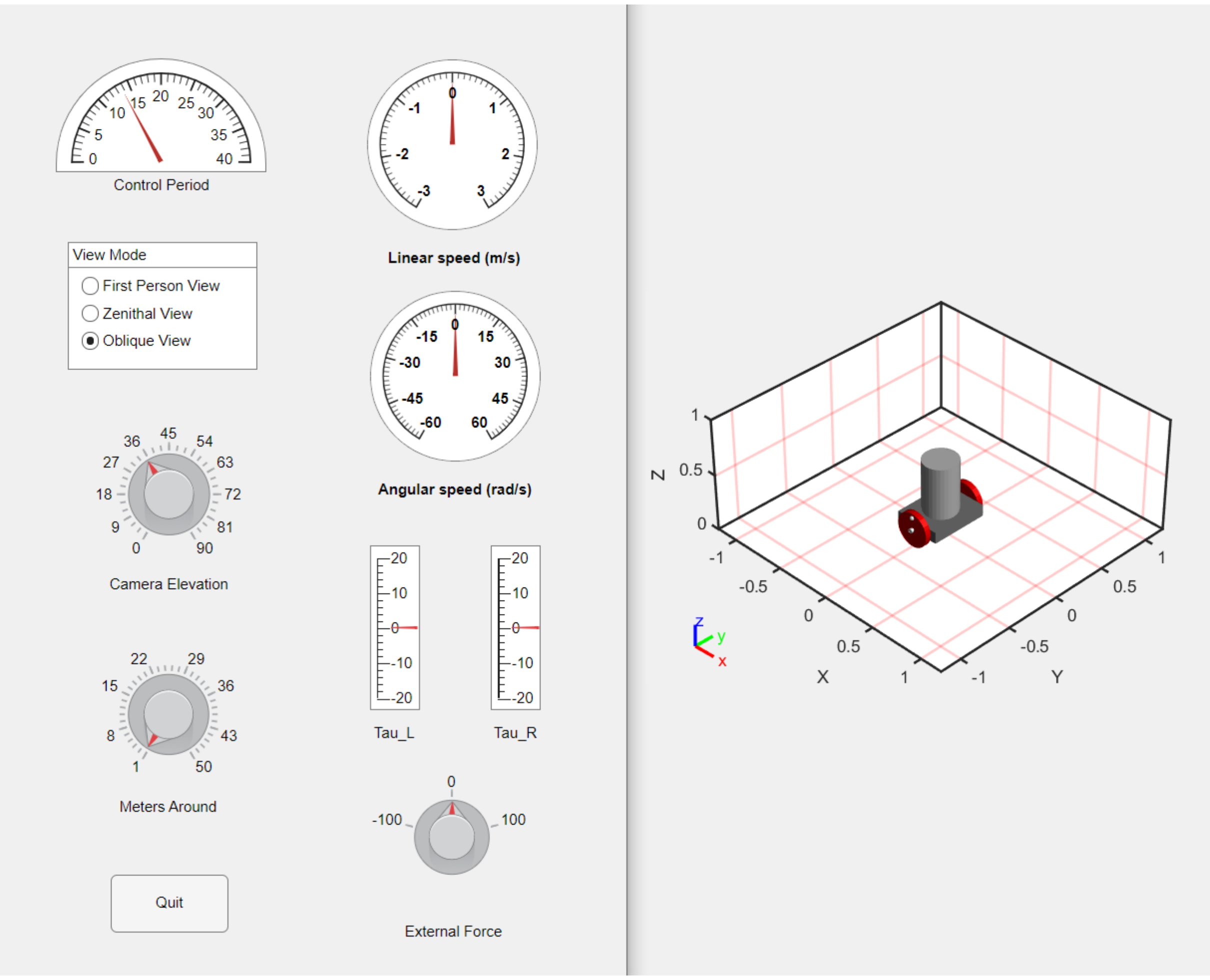
5.3 Grafical Interface

In order to develop the control loop, we used MATLAB [16]. Specifically we used an app module of the program that provides a nice interface for the user. A great advantage of this module is that, although this software is not free, once the program is finished, it can be downloaded and opened in any computer, despite not having a MATLAB license.

In our interface there is a series of buttons and visuals so that the user can tune certain visualization characteristics and can get feedback from the robot movement. These components are, starting from the top-left corner (Fig.5.2):

- A *control period semicircular gauge* that shows the control loop time (before fixing it to $20ms$). This measure is very important in order to make sure it is not over-passing the set time.

- A *view mode button* that allows the user to change the camera position. The three options are: First person view (the camera follows the robot closely), Zenithal (seen from above) and Oblique view (the camera is oriented with an azimuth angle of $45°$ [17]). The last one is the predefined view.
- A *camera elevation knob*, which only works for the oblique view, allows the user to change the elevation angle of the camera. The range is from $0°$, where you get a sideways view, to $90°$, that corresponds to a sky view.
- A *meters around knob*, which defines the camera distance from the robot. It can be set from a very close view of $1m$ to $50m$, from where the robot is hard to see.
- A *quit button*. This feature is linked to the ESC key and is used to close the app. If the button or the keyboard key is pressed once, the simulation stops and the plots show up, when pressed again, the interface closes.
- The *linear speed circular gauge* shows the desired speed of the robot in m/s. It corresponds to the variable v_0. The control of v_0 is done with the up and down arrows of the keyboard. Every time the up arrow key is pressed, the desired velocity is increased by $0.5m/s$, while the down arrow decreases this variable by the same amount, keeping the linear velocity limited between 3 and $-3m/s$.
- The *angular speed circular gauge* shows the desired angular velocity of the robot in rad/s. It corresponds to the variable $\dot{\theta}_0$. Its control is done with the left and right arrows of the keyboard. Every time the left arrow key is pressed, the desired angular speed is increased by $15rad/s$, while the right arrow decreases by the speed the same amount, keeping the angular velocities between 3 and $-3m/s$.
- The two *linear gauges* indicate both wheel's torques, which should always be in the interval $(-10, 10)$ Nm as this is how we have defined it.
- At the bottom-right corner, there is a *knob* that shows the value of an external force applied to the center of gravity of the robot in the x direction. This knob can not be directly actuated with the mouse, but its value is set through the keyboard. Each time the F key is pressed, the force applied takes the value of $100N$ for a period of $0.5ms$, while pressing G the force value becomes $-100N$ for the same period of time.
- Finally, if the $SPACE$ key is pressed on the keyboard, the robot will converge into a stop position ($v_0 = \dot{\theta}_0 = 0$).



View Mode

○ First Person View

○ Zenithal View

● Oblique View

Figure 5.2: Graphical interface caption.

5.4 Simulation Results

To show the performance of the controller we run five experiments designed to evaluate the following aspects: its ability to recover from unbalanced configurations, its response to abrupt changes in the commanded linear and angular velocities, respectively, and its reaction in front of two different disturbance situations, one consisting in a short pulse of force and a second one in which a constant force is permanently applied.

5.4.1 Twinbot's ability to recover from unbalanced configurations

In this first test, the robot starts at rest with an angle $\varphi_p = 60°$ (Fig. 5.3) and we hope the controller will be able to stabilize it at $\varphi_p = 0°$, remembering that the maximum wheel's torque is $10Nm$.

Looking at Fig. 5.4 we can see that the robot starts by accelerating in the positive x direction, what results in an effective reduction of the φ_p deviation. However, this correction of φ_p is achieved at the cost of producing an undesired x displacement, which must be subsequently compensated. So, near $t = 0.5s$, the robot starts reducing its speed until zero (just after $t = 1s$), to finally inverting it until the initial location with $x = 0$ is reached.

Note that each acceleration in x produces an acceleration of φ_p in the opposite sense. This explains why φ_p takes negative values after about $t = 0, 5s$, since this is required to permit the acceleration in the $-x$ direction while driving φ_p towards zero. The value of φ_p becomes positive again in the last part of the maneuver, and this is necessary to allow the x velocity to decelerate from a negative value to 0.

As we see, there is a delicate interplay between x and φ_p whose precise coordination yields the convergent behavior observed. Noticeably, the $x = 0$ position is asymptotically approached, without surpassing it and without oscillations.

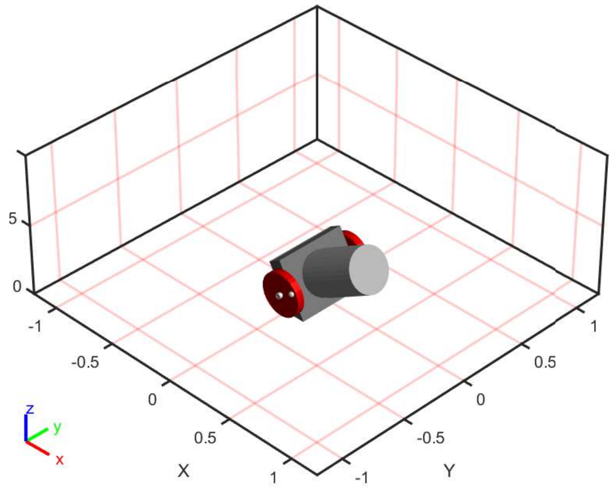


Figure 5.3: Twinbot initialized in a $\varphi_p = 60°$ configuration.

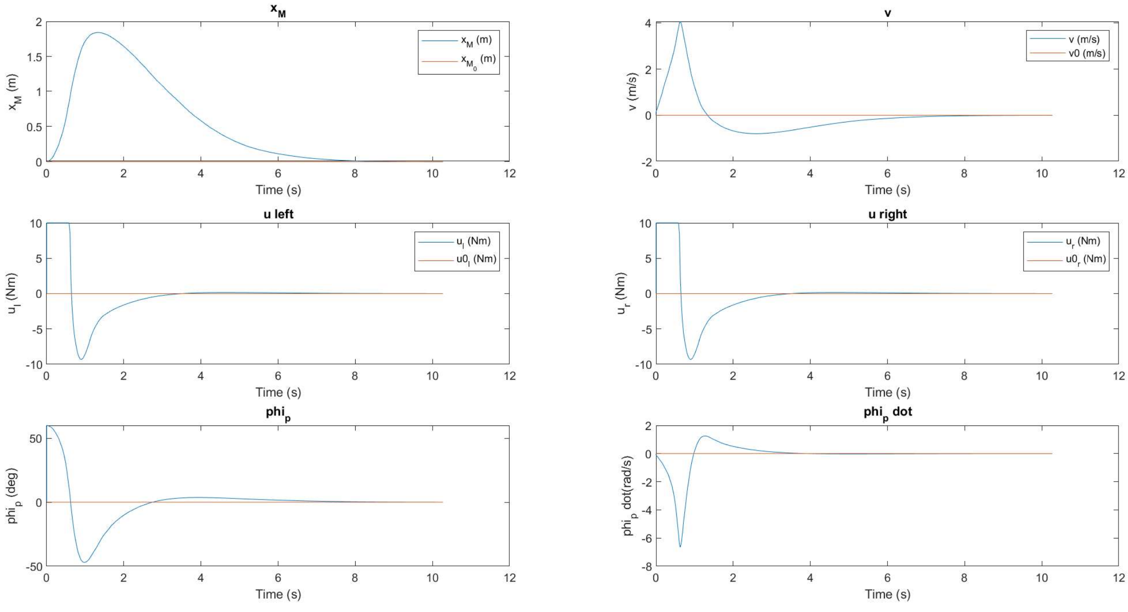


Figure 5.4: Behavior of our twinbot recovering from an unbalanced situation with $\varphi_p = 60°$.

5.4.2 Response to a v_0 step input

In this test, the robot starts at rest with $\varphi_p = 0$, and a desired velocity $v_0 = 0.5m/s$ is commanded. The first response of the controller may be surprising, since, instead of accelerating in the positive x direction to approach the required speed, the robot starts by accelerating in the opposite direction. This is the adequate behavior in this kind of robots, since, as already explained, an acceleration in the x direction produces an acceleration of φ_p in the opposite direction. Thus, it is necessary to start by setting the φ_p angle at a positive value by means of an acceleration in the $-x$ direction (Fig. 5.5). The stabilization process towards the desired trajectory is similar to that of the previous test, and we can observe how the robot trajectory converges smoothly to the desired one.

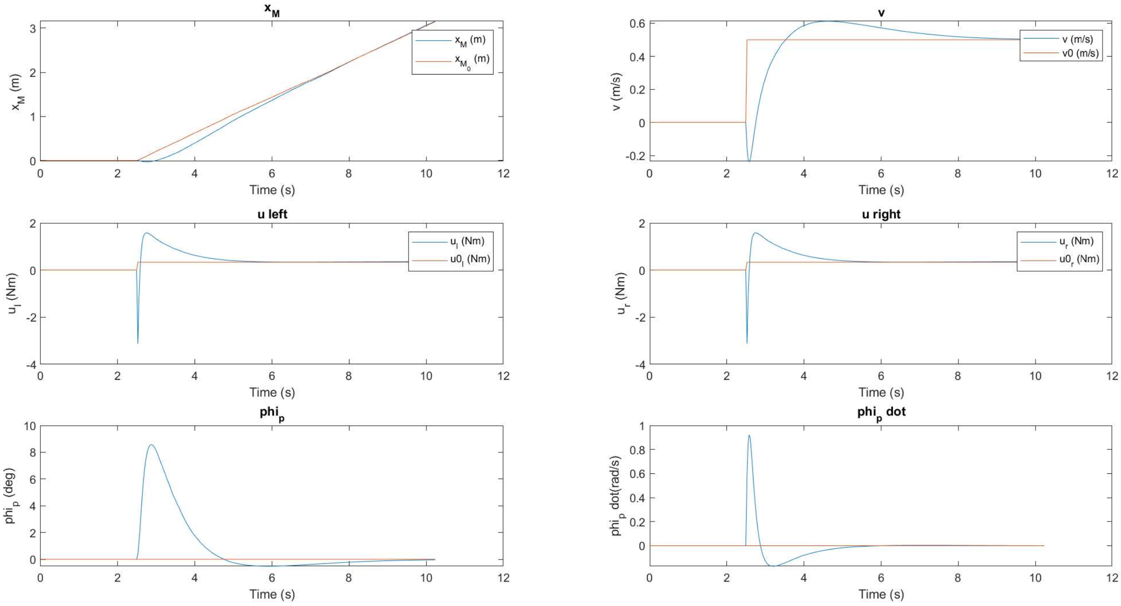


Figure 5.5: Graphics of the behavior when our twinbot is under a v_0 step input.

5.4.3 Response to a $\dot{\theta}$ step input

In this case the robot is required to turn in place with an angular velocity of $-0.25rad/s$. Since there is no need to accelerate in the x direction, the angle φ_p may stay equal to 0 at all times. Consequently, in this case, the controller produces an angular acceleration in the expected direction from the beginning (Fig. 5.6). As in the previous test, the robot is able to smoothly converge to the specified trajectory.

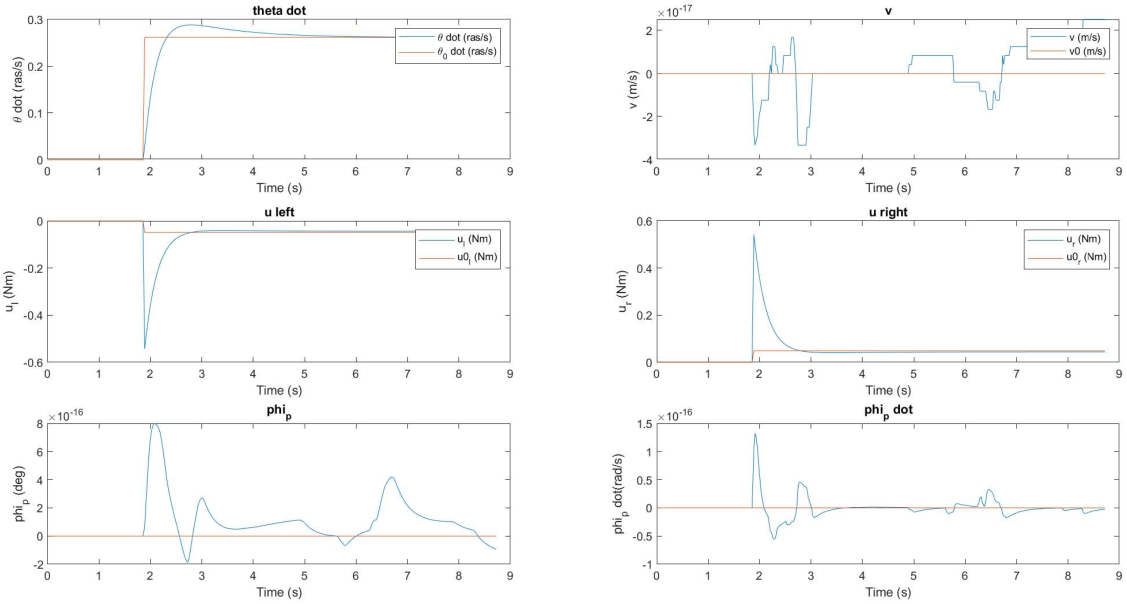


Figure 5.6: Graphics of the behavior when our twinbot is under a $\dot{\theta}$ step input.

5.4.4 Reaction to a force pulse

In this experiment, we simulate a quick collision of the robot by applying an external force along the x axis to the center of gravity (G point) for a period of time of 0.5s.

The immediate effect of the external force is a positive x displacement of the robot and a negative angular deviation of φ_p. This situation is very close to that reached in the first experiment just when φ_p takes a negative value. Once the external force disappears, the stabilization process is very similar in both tests. The robot is able to recover from the perturbed configuration following essentially the same steps (Fig. 5.7).

5.4.5 Reaction to a constant force

The last experiment consists in applying a constant force in the x direction to the center of gravity and maintain it for an indefinite period of time. In this case the robot first reacts, as in the previous experiment, by applying a torque to accelerate the robot in the same direction as that produced by the external force. This is required to reach an angle φ_p large enough to

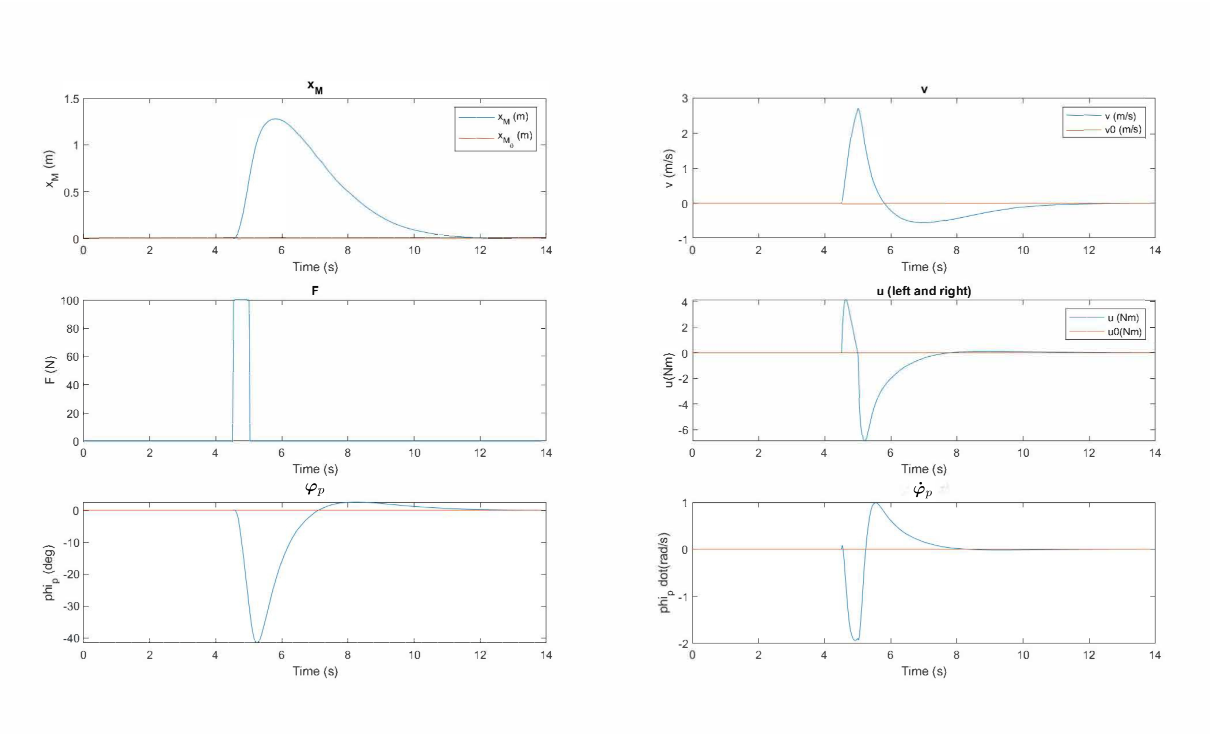


Figure 5.7: Graphics of the behavior when our twinbot is under a quick collision.

allow the effect of the gravity to compensate for the external force. The final behavior of the controller in this case is to stay in a configuration in which the angle φ_p and the torques applied to wheels exactly compensate the external force, so that the desired target state with $\varphi_p = 0$ and $\varphi_l = \varphi_r = 0$ is not reached (Fig. 5.8). This is explained noting that, if the external force is maintained, it is not possible to stay in a static configuration with the desired values of $\boldsymbol{\varphi} = 0$ and $\dot{\boldsymbol{\varphi}} = 0$. The final configuration reached by the robot is one such that a reduction of the φ_p angle would require a decrease of the torques u_l and u_r, whereas a reduction of the φ_l and φ_r angles would require an increase of the u_l and u_r torques, so that, unable to satisfy both conditions at the same time, the robot stays in a trade-off configuration in which these two requirements are mutually countered(Fig. 5.9).

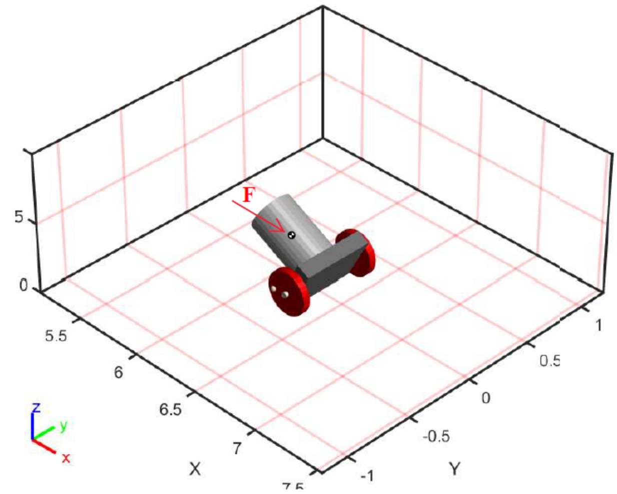


Figure 5.8: Twinbot configuration while going through a constant force.

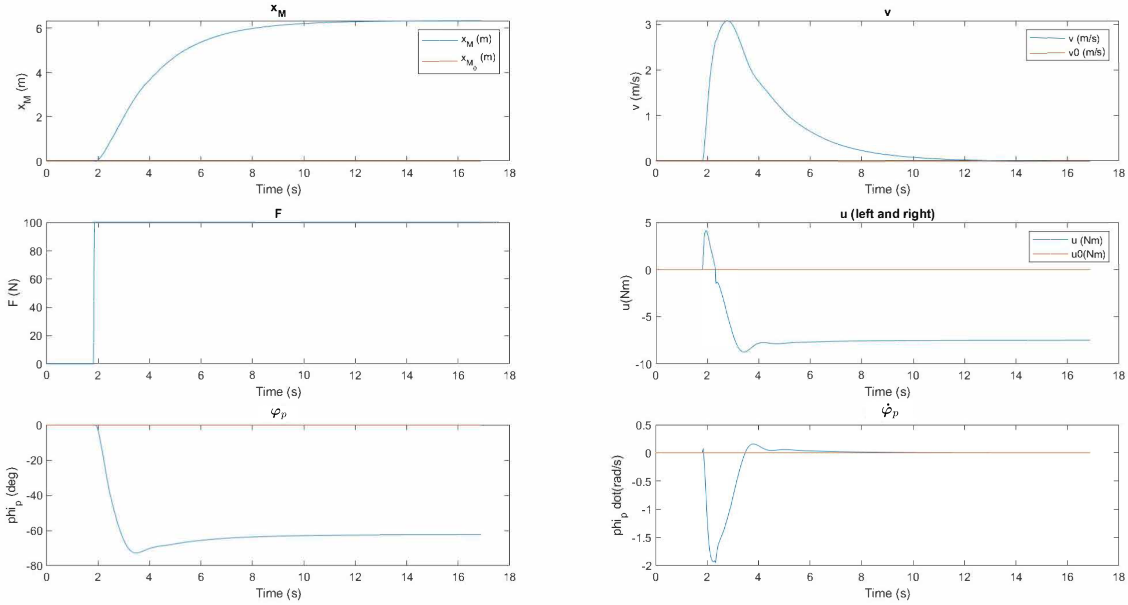


Figure 5.9: Graphics of the behavior when our twinbot is under a constant force.

6
Conclusions

The following is a summary of the main contributions of this work, and some points that could be addressed in future extensions.

6.1 Contributions

In this project we have obtained the kinematic and dynamic models for a two-wheeled inverted pendulum robot, we have designed a control system, that, using the dynamic model, stabilizes the robot in the upright position along a real time defined trajectory, and we have validated the robustness of this control by applying force disturbances while the robot was trying to keep a specific configuration.

The dynamic model takes into account the mass and inertia moments of all robot bodies, the viscous friction of the axes, and assumes arbitrary positions and dimensions for the wheels and the chassis. We started defining the model using generic coordinate systems, which includes twelve state variables. Nevertheless, this model had constraints, which would complicate the control, for this reason we developed a new reduced model that only involves six independent state variables. Finally, we made a particularization of the model with a simplified twinbot geometry, which has been essential for the control system implementation.

The controller obtained is executed in a control loop that, in every iteration, recognizes the desired configuration, compares it with the actual one, and defines the wheel's torques needed to minimize the positioning and velocity errors. The robot can be controlled by applying linear and angular velocity steps by means of a graphical interfacte that provides, among others, a real time output of the robot movement, the desired velocities, the actual motor torque of both wheels, and allows the user to move the camera settings in order to change the viewer's perspective.

Finally, a validation test was done, which provided very positive results. The robot was able to lift himself from a 60° angle, only using $10\,Nm$ torque motors; was able to follow, with no oscillations, a linear and an angular speed step, and could manage both a punctual and a constantly applied disturbance force without deviating too much from the desired trajectory.

6.2 Future work

This project is just the first step that needed to be made before having a functional robot. Right now, we have developed the control system, but if we want to materialize it, and make sure it works as it does in the simulation, there are several steps to take:

- First of all, the robot needs to be designed, and for this we need to decide what application we want to give it, how much budget we are able to spend in the process and which materials, shapes and dimensions should the robot have in order to fulfill its purpose without becoming overpriced.
- The code designed using MATLAB should be translated into C++, if we want to achieve a faster response.
- Once the robot is built, the mass, inertia, friction, and geometric parameters will need to be identified in order to particularize the dynamic model, and thus the controller.
- The capability of the control system should by widen to make it able to follow arbitrary trajectories, not restricted to those with constant speed and $\varphi_p = 0$, as has been assumed in this work.
- Furthermore, we could equip the robot with a path planner to provide it with autonomy to generate navigation plans for obstacle environments.
- Finally, the proposed controller could be programmed to become adaptive, so that the system itself were able to adjust the dynamic parameters, in case they are subject to changes, such as when depositing a load of unknown mass and inertia moment attached to the robot.

With these suggestions said, it is clear that there is still a long road to go, and a lot of work that needs to be done if we want to build our own twinbot some day.

Bibliography

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[2] R. P. M. Chan, K. A. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots," *Annual reviews in control*, vol. 37, no. 1, pp. 89–103, 2013.

[3] K. Albert, K. S. Phogat, F. Anhalt, R. N. Banavar, D. Chatterjee, and B. Lohmann, "Structure-preserving constrained optimal trajectory planning of a wheeled inverted pendulum," *IEEE Transactions on Robotics*, vol. 36, no. 3, pp. 910–923, 2020.

[4] G. Campion and W. Chung, "Wheeled robots," in *Springer Handbook of Robotics*, pp. 391–410, Springer Berlin Heidelberg, 2008.

[5] J. A. Batlle and A. B. Condomines, *Rigid Body Kinematics*. Cambridge University Press, 2020.

[6] R. Tedrake, "Underactuated Robotics. Course lecture notes. Available online through." `http://underactuated.csail.mit.edu.2018`.

[7] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[8] J. A. Batlle and A. B. Condomines, *Rigid Body Dynamics*. Cambridge University Press, 2022.

[9] R. M. Murray, Z. Li, and S. Sastry., *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[10] Segway, "Loomo specs."

[11] M. Athans and P. L. Falb, *Optimal control: an introduction to the theory and its applications*. Courier Corporation, 2013.

[12] J. P. Hespanha, "Linear systems theory," in *Linear Systems Theory*, Princeton university press, 2018.

[13] N. A. F. Senan, "A brief introduction to using ode45 in matlab," *Department of Mechanical Engineering, University of California at Berkeley*.

[14] L. F. Shampine and M. W. Reichelt, "The MATLAB ODE Suite," *Journal of Scientific Computing*, vol. 18, pp. 1–22, 1997.

[15] J. Dormand and P. Prince, "A family of embedded Runge-Kutta formulae," *Appl. Math.*, vol. 6, pp. 19–26, 1980.

[16] D. J. Higham and N. J. Higham, *MATLAB guide*. SIAM, 2016.

[17] MathWorks, “Set the viewpoints with azimuth and elevation.”

[18] Sibelga, “How much power does a computer use? and how much co2 does that represent?.”

[19] T. I. P. S. Ltd, “How much co2 does a light bulb create?.”

A
Source code

In this project, we have developed all the simulations and calculations using MATLAB. In order to obtain the kinematic and dynamic models of our twinbot we have used the symbolic toolbox that the program offers, which made the calculations possible. To make the code more visually attractive, we have used the *liveSripts* tool, which allows the inclusion of text and images interspersed with parts of the code. This file can be find in section A.1.

The graphical interface is done with the application designer that MATLAB offers, nevertheless, some of the functions are imported as external *.m* files. Finally, the URDF text file, which includes the bodies and joints of the simplified robot, is also present in this annex.

A.1 Matlab liveScript

The MATLAB *liveSript* that has been implemented for the kinematic and dynamic modeling of the robot is the following one. It is necessary to run the *liveSript* file before moving to the application, because the first one exports several matrices and parameters that are needed later on.

Dynamic models of Twinbot

Authors: Eduard Godayol and Lluís Ros, 2022

This livescript develops the **x** and **y** models of Twinbot. It computes all of their terms symbolically, and exports them for numeric evaluation. It also explains how to obtain the **z** model, which is a full model in independent coordinates.

Table of Contents

Initializations

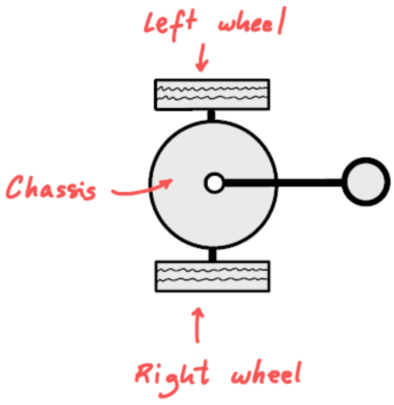
```
% Clear all variables, close all figures, and clear the command window
clearvars
close all
clc

% Start stopwatch
tic

% Display the matrices with rectangular brackets
sympref('MatrixWithSquareBrackets',true);
```

```
% Avoid Matlab's own substitution of long expressions
sympref('AbbreviateOutput',false);

% Symbolic variables
syms d r                    % Semiaxis length and wheel radius
syms x y                    % Absolute coords. of point M
syms x_dot y_dot            % Absolute velocity of point M
syms theta                  % Absolute angle between the axes 1' and 1
syms theta_dot              % Absolute angular velocity of the platform
syms varphi_p               % Pitch angle of the chassis
syms varphi_l               % Angle of left wheel relative to chassis
syms varphi_r               % Angle of right wheel relative to chassis
syms varphi_dot_p           % Time derivative of varphi_p
syms varphi_dot_l           % Time derivative of varphi_l
syms varphi_dot_r           % Time derivative of varphi_r
syms v                      % Absolute velocity of point M in basis B'
syms x_G y_G z_G            % Coords. of the chassis c.o.g. in basis B''
syms m_c                    % Mass of the chassis
syms m_w                    % Mass of a wheel
syms I_c11  I_c12  I_c13    % Entries of the chassis inertia tensor
syms I_c22  I_c23  I_c33    % Entries of the chassis inertia tensor
syms I_a                    % Axial moment of inertia of a wheel
syms I_t                    % Twisting moment of inertia of a wheel
syms b                      % Viscous friccion coeff. at the wheels axis
syms g                      % Acceleration of gravity
syms tau_l tau_r            % Torques applied to the left and right wheels
syms F_dx F_dy F_dz         % Components of disturbance force in basis B'
```

Introduction

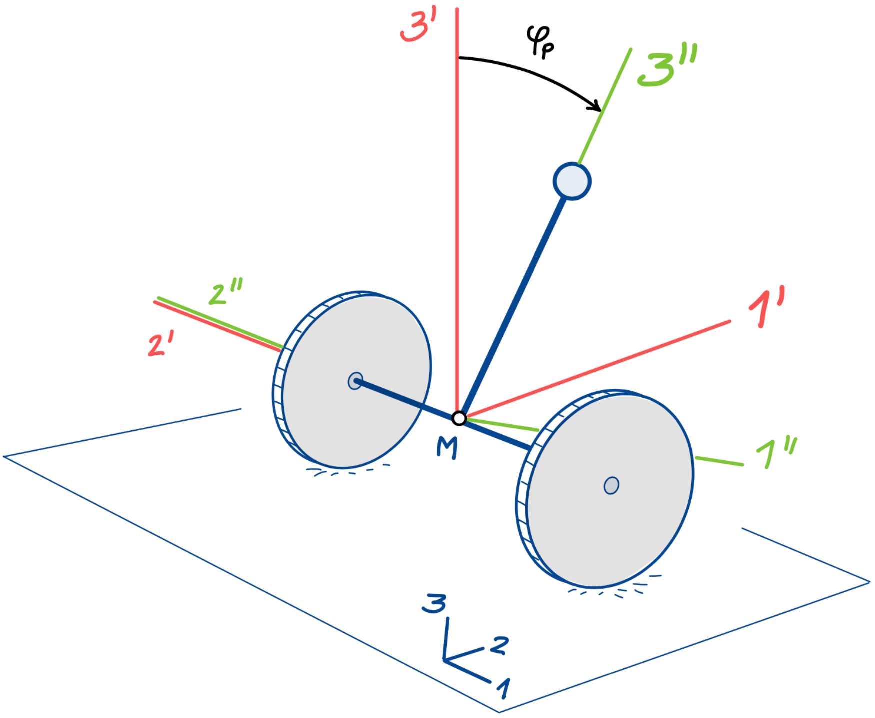
Reference frames and vector bases

Twinbot is a two-wheeled inverted pendulum robot. It can be viewed as a differential drive robot whose chassis is allowed to rotate about the wheels' axis. The wheels are powered by electric motors mounted underneath the chassis.



The structure of the robot is simple but effective. With proper maneuvering, the robot can reach any desired position and orientation in the plane. However, since its center of gravity is often above the wheels axis (specially if the robot is carrying a load) the robot may fall if proper control torques are not applied. The challenge is to move the robot forward (perhaps while turning) in a stable manner.

To obtain the robot model, we use the following reference frames:



The blue frame is the **absolute frame** fixed to the ground, with axes 1, 2, and 3. The red frame is centered at the midpoint M of the wheels' axis, and its axes 1', 2', and 3' correspond to the heading direction of the robot, the wheels axis, and the vertical direction through M, respectively. We call this frame the **robot frame**. The green frame is fixed to the robot chassis, and is rotated an angle φ_p with respect to the robot frame. We call this frame the **chassis frame**. The three frames are sometimes referred to as $\mathscr{F}$, $\mathscr{F}'$, and $\mathscr{F}''$, respectively.

Each frame has a vector basis attached to it, whose unit vectors are directed along the frame axes. The three bases will be referred to as:

- $B = \{1, 2, 3\} \longleftarrow$ Fixed to the absolute frame
- $B' = \{1', 2', 3'\} \longleftarrow$ Fixed to the robot frame
- $B'' = \{1'', 2'', 3''\}$ respectively $\longleftarrow$ Fixed to the chassis frame

Configuration and state coordinates

The robot configuration can be described by means of six coordinates: the position (x, y) of M in the absolute frame, the orientation angle θ of the platform (between axes 1' and 1), the pitch angle φ_p, and the angles of the left and right wheels, φ_l and φ_r, relative to the chassis.

The robot configuration is thus given by

$$\mathbf{q} = (x, y, \theta, \varphi_p, \varphi_l, \varphi_r)$$

and the time derivative of $\mathbf{q}$ provides the robot velocity.Therefore, the robot state is given by

$$\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$$

It will also be useful to consider an internal configuration vector

$$\boldsymbol{\varphi} = (\varphi_p, \varphi_l, \varphi_r)$$

and its associate internal state

$$\mathbf{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$$

```
% Vars. of full config. and state: q, qdot, x=(q,qdot)
qvec = [x; y; theta; varphi_p; varphi_l; varphi_r];
qdotvec = [x_dot; y_dot; theta_dot; varphi_dot_p; varphi_dot_l; varphi_dot_r];
xvec = [qvec;qdotvec];
nq = length(qvec);

% Vars. of internal config. and state: varphi, varphidot, y=(varphi,varphidot)
varphivec = [varphi_p; varphi_l; varphi_r];
varphidotvec = [varphi_dot_p; varphi_dot_l; varphi_dot_r];
yvec = [varphivec; varphidotvec];
nphi = length(varphivec);

% Action vector
uvec = [tau_l; tau_r];
nu = length(uvec);

% Pars. vector of a general design
parsvec = [d;r;x_G;y_G;z_G;m_w;I_a;I_t;m_c;I_c11;I_c12;I_c13;I_c22;I_c23;I_c33;b;g];
```

Kinematic constraints imposed by the rolling contacts

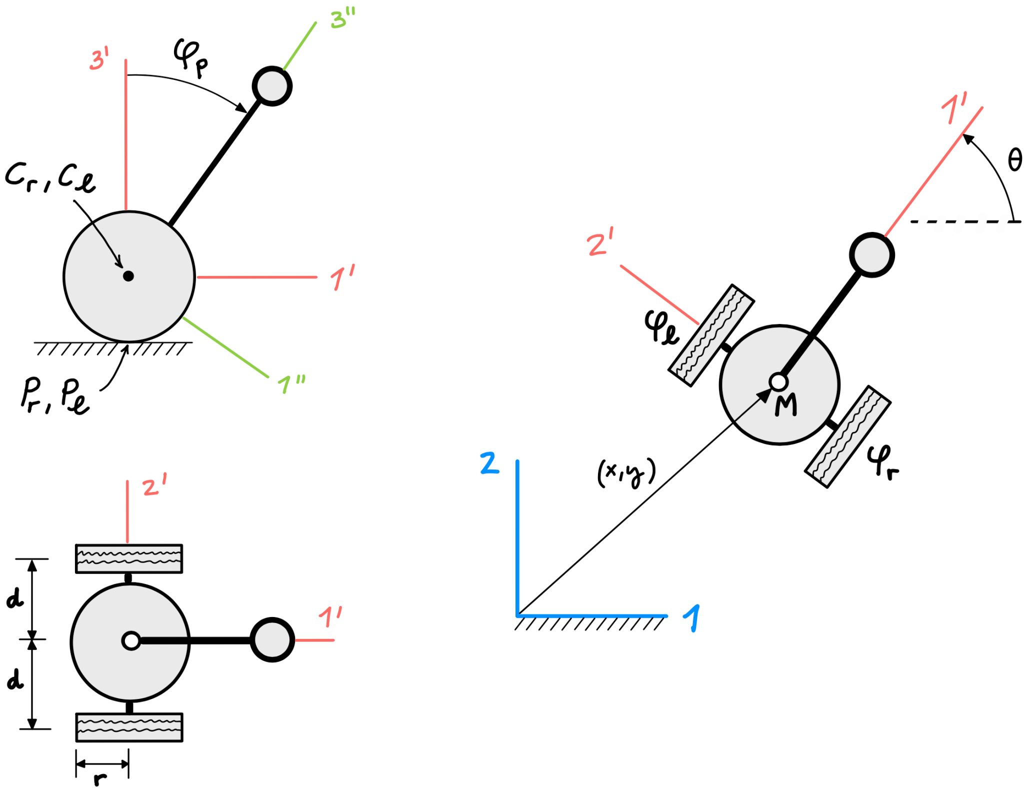
We next see that the coordinates of $\mathbf{x}$ are not independent. The rolling contacts of the wheels impose a kinematic constraint of the form

$$\mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}} = \mathbf{0}$$

where $\mathbf{J}(\mathbf{q})$ is a 3×6 Jacobian matrix. We will next obtain this Jacobian.

On our derivations we use $\mathbf{v}(Q)$ to denote the velocity of a point Q of the robot. The basis in which $\mathbf{v}(Q)$ is expressed will be mentioned explicitly, or understood by context.

We also use the following notation:



The robot pose is given by the position $(x, y)$of point M, and by the θ angle. The half-length of the wheels axis is d. Also, C_r and C_l denote the centers of the right and left wheels, and P_r and P_l are the contact points of such wheels with the ground. The two wheels have the same radius r.

The rolling contact constraints of the chassis can be found by computing the velocities of $\mathbf{v}(P_r)$ and $\mathbf{v}(P_l)$ in terms of $\dot{x}$, $\dot{y}$, $\dot{\theta}$, $\dot{\varphi_p}$, $\dot{\varphi_l}$ and $\dot{\varphi_r}$ (i.e., as if the robot were a floating kinematic tree) and forcing these velocities to be zero (as the wheels do not slide when placed on the ground).

To obtain $\mathbf{v}(P_r)$ and $\mathbf{v}(P_l)$, note first that the velocity of M can only be directed along axis 1', since lateral slipping is forbidden under perfect rolling. Therefore in $B' = \{1', 2', 3'\}$, $\mathbf{v}(M)$ takes the form

$$\mathbf{v}(M) = \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix}$$

Also note that, if $\boldsymbol{\omega}_{\text{robot}}$ is the absolute angular velocity of the robot frame, we have

$$\mathbf{v}(C_r) = \mathbf{v}(M) + \boldsymbol{\omega}_{\text{robot}} \times \overrightarrow{MC_r} = \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ -d \\ 0 \end{bmatrix} = \begin{bmatrix} v + d\ \dot{\theta} \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{v}(C_l) = \mathbf{v}(M) + \boldsymbol{\omega}_{\text{robot}} \times \overrightarrow{MC_l} = \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} v - d\ \dot{\theta} \\ 0 \\ 0 \end{bmatrix}$$

or, in Matlab:

```
v_Cr = [v;0;0] + cross( [0;0;theta_dot], [0;-d;0] );  % Velocity of Cr
v_Cl = [v;0;0] + cross( [0;0;theta_dot], [0; d;0] );  % Velocity of Cl
```

The velocities of the ground contact points are thus given by

$$\mathbf{v}(P_r) = \mathbf{v}(C_r) + \boldsymbol{\omega}_{\text{wheel}} \times \overrightarrow{C_rP_r} = \begin{bmatrix} v + d\,\dot{\theta} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\varphi}_p + \dot{\varphi}_r \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ -r \end{bmatrix} = \begin{bmatrix} v + d\,\dot{\theta} - r\,(\dot{\varphi}_p + \dot{\varphi}_r) \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{v}(P_l) = \mathbf{v}(C_l) + \boldsymbol{\omega}_{\text{wheel}} \times \overrightarrow{C_lP_l} = \begin{bmatrix} v - d\,\dot{\theta} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\varphi}_p + \dot{\varphi}_l \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ -r \end{bmatrix} = \begin{bmatrix} v - d\,\dot{\theta} - r\,(\dot{\varphi}_p + \dot{\varphi}_l) \\ 0 \\ 0 \end{bmatrix}$$

i.e.,

```
v_Pr = v_Cr + cross( [0; varphi_dot_p+varphi_dot_r; theta_dot], [0; 0; -r]);
v_Pl = v_Cl + cross( [0; varphi_dot_p+varphi_dot_l; theta_dot], [0; 0; -r]);
```

Since P_r and P_l do not slip, $\mathbf{v}(P_r)$ and $\mathbf{v}(P_l)$ must be zero, which gives the two fundamental constraints of the robot:

```
eqn1 = (v_Pr(1) == 0);
eqn2 = (v_Pl(1) == 0);
disp(eqn1); disp(eqn2);
```

$$v - r\,\left(\dot{\varphi}_p + \dot{\varphi}_r\right) + d\,\dot{\theta} = 0$$

$$v - r\,\left(\dot{\varphi}_l + \dot{\varphi}_p\right) - d\,\dot{\theta} = 0$$

It is easy to solve for v and $\dot{\theta}$ in these two equations:

```
[solved_v,solved_thetadot] = solve([eqn1,eqn2],[v,theta_dot]);

eqn3 = (v==simplify(solved_v));
eqn4 = (theta_dot==simplify(solved_thetadot));

disp(eqn3); disp(eqn4);
```

$$v = \frac{r\,\left(\dot{\varphi}_l + 2\,\dot{\varphi}_p + \dot{\varphi}_r\right)}{2}$$

$$\dot{\theta} = -\frac{r\,\left(\dot{\varphi}_l - \dot{\varphi}_r\right)}{2\,d}$$

It is also easy to solve for $\dot{\varphi}_l$ and $\dot{\varphi}_r$:

```
eqn3p = isolate(eqn1,varphi_dot_r);
```

```
eqn4p = isolate(eqn2,varphi_dot_l);
disp(eqn3p); disp(eqn4p);
```

$$\dot{\varphi}_r = \frac{v + d\,\dot{\theta} - r\,\dot{\varphi}_p}{r}$$

$$\dot{\varphi}_l = -\frac{d\,\dot{\theta} - v + r\,\dot{\varphi}_p}{r}$$

From the previous figure we see that, in basis B = {1,2,3},

$$\mathbf{v}(M) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix}.$$

We also see it must be

$$\begin{cases} \dot{x} = v \cdot \cos\theta \\ \dot{y} = v \cdot \sin\theta \end{cases}$$

By substituting $v = \frac{r\,(\dot{\varphi}_l + 2\,\dot{\varphi}_p + \dot{\varphi}_r)}{2}$ in these two equations, and also considering $\dot{\theta} = -\frac{r\,(\dot{\varphi}_l - \dot{\varphi}_r)}{2\,d}$, we obtain the system of equations:

```
eqn5 = (x_dot == solved_v * cos(theta));
eqn6 = (y_dot == solved_v * sin(theta));
eqn7 = eqn4;

disp(eqn5); disp(eqn6); disp(eqn7);
```

$$\dot{x} = \cos(\theta)\left(\frac{r\,\dot{\varphi}_l}{2} + r\,\dot{\varphi}_p + \frac{r\,\dot{\varphi}_r}{2}\right)$$

$$\dot{y} = \sin(\theta)\left(\frac{r\,\dot{\varphi}_l}{2} + r\,\dot{\varphi}_p + \frac{r\,\dot{\varphi}_r}{2}\right)$$

$$\dot{\theta} = -\frac{r\,(\dot{\varphi}_l - \dot{\varphi}_r)}{2\,d}$$

This system expresses the rolling contact constraints of the robot. For compactness, we write it as

$$\dot{\mathbf{p}} = \mathbf{N}(\mathbf{q}) \cdot \dot{\boldsymbol{\varphi}}$$

where $\mathbf{p} = (x, y, z)$, $\boldsymbol{\varphi} = (\varphi_p, \varphi_l, \varphi_r)$, and $\mathbf{N}(\mathbf{q})$ is the 3×3 matrix

```
N = equationsToMatrix([rhs(eqn5), rhs(eqn6), rhs(eqn7)],...
                         [varphi_dot_p, varphi_dot_l, varphi_dot_r])
```

```
N =
```

$$\begin{bmatrix} r\cos(\theta) & \frac{r\cos(\theta)}{2} & \frac{r\cos(\theta)}{2} \\ r\sin(\theta) & \frac{r\sin(\theta)}{2} & \frac{r\sin(\theta)}{2} \\ 0 & -\frac{r}{2\,d} & \frac{r}{2\,d} \end{bmatrix}$$

The equation $\dot{\mathbf{p}} = \mathbf{N}(\mathbf{q}) \cdot \dot{\boldsymbol{\varphi}}$ can be written as

$$\begin{bmatrix} \mathbf{I}_3 & -\mathbf{N} \end{bmatrix} \cdot \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\boldsymbol{\varphi}} \end{bmatrix} = \mathbf{0}$$

Thus, the constraint Jacobian $\mathbf{J}(\mathbf{q})$ we were searching for is

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \mathbf{I}_3 & -\mathbf{N} \end{bmatrix}$$

```
J = [eye(3), -N]
```

```
J =
```

$$\begin{bmatrix} 1 & 0 & 0 & -r\cos(\theta) & -\frac{r\cos(\theta)}{2} & -\frac{r\cos(\theta)}{2} \\ 0 & 1 & 0 & -r\sin(\theta) & -\frac{r\sin(\theta)}{2} & -\frac{r\sin(\theta)}{2} \\ 0 & 0 & 1 & 0 & \frac{r}{2\,d} & -\frac{r}{2\,d} \end{bmatrix}$$

Holonomic and nonholonomic constraints

Recall that the last equation of $\dot{\mathbf{p}} = \mathbf{N}(\mathbf{q}) \cdot \dot{\boldsymbol{\varphi}}$ was

$$\dot{\theta} = \frac{r}{2d}\,(\dot{\varphi}_r - \dot{\varphi}_l)$$

This equation is integrable, and yields the holonomic constraint

$$\phi(\mathbf{q}) = 0$$

where

$$\phi(\mathbf{q}) = \theta - \frac{r}{2d}(\varphi_r - \varphi_l) - K_0$$

$$K_0 = \theta_0 - \frac{r}{2d} \cdot (\varphi_{r,0} - \varphi_{l,0})$$

In sum, thus, the robot is subject to the 4 constraints:

$$\mathbf{F}(\mathbf{x}) \equiv \begin{cases} \mathbf{J} \cdot \dot{\mathbf{q}} = \mathbf{0} \\ \phi(\mathbf{q}) = 0 \end{cases}$$

The state space of the robot is thus the set

$\mathscr{X} = \{\mathbf{x} : \mathbf{F}(\mathbf{x}) = \mathbf{0}\}$

Lagrange's equation of motion

Recall that the equation of motion of a robot takes the form

$$\mathbf{M}(\mathbf{q})\ \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\ \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}(\mathbf{q})^\top\ \boldsymbol{\lambda} = \mathbf{Q}_a(\mathbf{u}) + \mathbf{Q}_f + \mathbf{Q}_d$$

where:

- $\mathbf{q}$ is the configuration vector of the robot (of size $n_q = 6$ in our case)
- $\mathbf{u} = (\tau_l, \tau_r)$ is the vector of motor torques (the left and right wheel torques).
- $\mathbf{M}(\mathbf{q})$ is the mass matrix of the unconstrained system (positive-definite of size $n_q \times n_q$)
- $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the generalized Coriolis and centrifugal force matrix
- $\mathbf{G}(\mathbf{q})$ is the generalized gravity force
- $\mathbf{J}(\mathbf{q})$ is the constraint Jacobian of the robot
- $\boldsymbol{\lambda}$ is a vector of Lagrange mulitpliers.
- $\mathbf{Q}_a(\mathbf{u})$ is the generalized force of actuation, which can be written in the form $\mathbf{E}(\mathbf{q}) \cdot \mathbf{u}$
- $\mathbf{Q}_f$ is the generalised force modelling all friction forces in the system
- $\mathbf{Q}_d$ is a generalised force modelling all disturbance forces considered, which may be added to test disturbance rejection when a feedback controller is used.

Although the Twinbot moves on flat terrain, the chassis can turn about the wheels axis, which changes the height of its center of gravity. For this reason $\mathbf{G}(\mathbf{q})$ will not be 0.

Our task now is to obtain $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, $\mathbf{G}(\mathbf{q})$, $\mathbf{Q}_a(\mathbf{u})$, $\mathbf{Q}_f$, and $\mathbf{Q}_d$.

Mass matrix

Recall that the robot configuration is given by

$$\mathbf{q} = (x, y, \alpha, \varphi_p, \varphi_l, \varphi_r)$$

To find the mass matrix $\mathbf{M}(\mathbf{q})$ we need to write the kinetic energy T of the robot as a function of $\mathbf{q}$ and $\dot{\mathbf{q}}$ and then express it as:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\ \dot{\mathbf{q}}^\top \cdot \mathbf{M}(\mathbf{q}) \cdot \dot{\mathbf{q}}$$

We have to compute the translational and rotational kinetic energies of all bodies in the Twinbot, and add them all to obtain T. In computing such energies we must view the robot as a kinematic tree whose configuration is given by $\mathbf{q}$.

The absolute velocity of the center of gravity G of the chassis is given by:

$$\mathbf{v}(G) = \mathbf{v}(M) + \boldsymbol{\omega}_{\text{chassis}} \times \overrightarrow{MG}$$

In basis B' we have

$$\mathbf{v}(G)=\begin{bmatrix} c_\theta & -s_\theta & 0\\ s_\theta & c_\theta & 0\\ 0 & 0 & 1\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\dot{x}\\ \dot{y}\\ 0\end{bmatrix}+\begin{bmatrix}0\\ \dot{\varphi}_p\\ \dot{\theta}\end{bmatrix}\times\begin{bmatrix} c_{\varphi_p} & 0 & s_{\varphi_p}\\ 0 & 1 & 0\\ -s_{\varphi_p} & 0 & c_{\varphi_p}\end{bmatrix}\begin{bmatrix}x_G\\ y_G\\ z_G\end{bmatrix},$$

where (x_G, y_G, z_G) is the position vector of G in the chassis reference (and written in B''). Thus we have:

```
% Rotation that transforms vectors in B' to vectors in B
Rot_BpB =  [ cos(theta)  -sin(theta)  0;
             sin(theta)   cos(theta)  0;
             0            0           1  ];

% Rotation that transforms vectors in B'' to vectors in B'
Rot_Bpp_Bp = [ cos(varphi_p)   0   sin(varphi_p);
               0               1   0;
              -sin(varphi_p)   0   cos(varphi_p)  ];

% Velocity of the midpoint (in B')
v_M = transpose(Rot_BpB) * [x_dot; y_dot; 0];

% Angular velocity of the chassis (in B')
omega_chas = [0; varphi_dot_p; theta_dot];

% Position vector of G in chassis reference (in B')
MG =  Rot_Bpp_Bp * [x_G; y_G; z_G];

% Absolute velocity of G (in B')
v_G = v_M + cross(omega_chas,MG)
```

```
v_G =
```

$$\begin{bmatrix}\dot{\varphi}_p\,\left(z_G\cos(\varphi_p)-x_G\sin(\varphi_p)\right)-\dot{\theta}\,y_G+\dot{x}\cos(\theta)+\dot{y}\sin(\theta)\\ \dot{\theta}\,\left(x_G\cos(\varphi_p)+z_G\sin(\varphi_p)\right)+\dot{y}\cos(\theta)-\dot{x}\sin(\theta)\\ -\dot{\varphi}_p\,\left(x_G\cos(\varphi_p)+z_G\sin(\varphi_p)\right)\end{bmatrix}$$

```
% Translational kinetic energies (using B')

    % Of the chassis
    T_tra_chas = 1/2 * m_c * transpose(v_G) * v_G;

    % Of the left wheel
    T_tra_l = 1/2 * m_w * transpose(v_Cl) * v_Cl;

    % Of the right wheel
    T_tra_r = 1/2 * m_w * transpose(v_Cr) * v_Cr;

% Rotational kinetic energies (using B')
```

```
    % Chassis
    I_chas = [I_c11  I_c12  I_c13; I_c12  I_c22  I_c23; I_c13  I_c23  I_c33];
    S = Rot_Bpp_Bp;
    I_chas_Bp = S * I_chas * transpose(S);
    T_rot_chas = 1/2 * transpose(omega_chas) * I_chas_Bp * omega_chas;

    % Left wheel
    omega_l = [0; varphi_dot_p + varphi_dot_l; theta_dot];
    I_w = [I_t 0 0; 0 I_a 0; 0 0 I_t];
    T_rot_l = 1/2 * transpose(omega_l) * I_w * omega_l;

    % Right wheel
    omega_r = [0; varphi_dot_p + varphi_dot_r; theta_dot];
    I_w = [I_t 0 0; 0 I_a 0; 0 0 I_t];
    T_rot_r = 1/2 * transpose(omega_r) * I_w * omega_r;


% Total kinetic energy
T = simplify(T_tra_chas + T_tra_l + T_tra_r + T_rot_chas + T_rot_l + T_rot_r);

% Display the translational kinetic energies
display(T_tra_chas); display(T_tra_l); display(T_tra_r);
```

```
T_tra_chas =
```

$$\frac{m_c\ \left(\dot{\theta}\ (x_G \cos(\varphi_p) + z_G \sin(\varphi_p)) + \dot{y} \cos(\theta) - \dot{x} \sin(\theta)\right)^2}{2} + \frac{m_c\ \big(\dot{\varphi}_p\ (z_G \cos(\varphi_p) - x_G \sin(\varphi_p)) - \dot{\theta}\, y_G + \dot{x} \cos}{2}$$

```
T_tra_l =
```

$$\frac{m_w\ \left(v - d\,\dot{\theta}\right)^2}{2}$$

```
T_tra_r =
```

$$\frac{m_w\ \left(v + d\,\dot{\theta}\right)^2}{2}$$

```
% Display the rotational kinetic energies
display(T_rot_chas); display(T_rot_l); display(T_rot_r);
```

```
T_rot_chas =
```

$$\dot{\theta}\ \Bigg(\frac{\dot{\varphi}_p\ (I_{c23} \cos(\varphi_p) - I_{c12} \sin(\varphi_p))}{2} + \frac{\dot{\theta}\ (\cos(\varphi_p)\ (I_{c33} \cos(\varphi_p) - I_{c13} \sin(\varphi_p)) - \sin(\varphi_p)\ (I_{c13} \cos(\varphi_p) - I_{c11}}{2}$$

```
T_rot_l =
```

$$\frac{I_t\,\dot{\theta}^2}{2} + I_a\ \left(\frac{\dot{\varphi}_l}{2} + \frac{\dot{\varphi}_p}{2}\right)\ \left(\dot{\varphi}_l + \dot{\varphi}_p\right)$$

```
T_rot_r =
```

$$\frac{I_t\,\dot{\theta}^2}{2} + I_a\ \left(\frac{\dot{\varphi}_p}{2} + \frac{\dot{\varphi}_r}{2}\right)\ \left(\dot{\varphi}_p + \dot{\varphi}_r\right)$$

Since T is a quadratic form, the Hessian of T gives the desired mass matrix:

```
M = hessian(T,[x_dot y_dot theta_dot varphi_dot_p varphi_dot_l varphi_dot_r])
```

```
M =
```

$$\left[\begin{array}{cc} m_c\cos(\theta)^2+m_c\sin(\theta)^2 & 0 \\ 0 & m_c\cos(\theta)^2+m_c\sin(\theta)^2 \\ -m_c\,y_G\cos(\theta)-m_c\sin(\theta)\ (x_G\cos(\varphi_p)+z_G\sin(\varphi_p)) & m_c\cos(\theta)\ (x_G\cos(\varphi_p)+z_G\sin(\varphi_p))-m_c\,y_G\sin(\theta) \\ m_c\cos(\theta)\ (z_G\cos(\varphi_p)-x_G\sin(\varphi_p)) & m_c\sin(\theta)\ (z_G\cos(\varphi_p)-x_G\sin(\varphi_p)) \\ 0 & 0 \\ 0 & 0 \end{array}\right.$$

```
% See which variables are in M
symvar(M)
```

ans = $\left[I_a\ \ I_{c11}\ \ I_{c12}\ \ I_{c13}\ \ I_{c22}\ \ I_{c23}\ \ I_{c33}\ \ I_t\ \ d\ \ m_c\ \ m_w\ \ \theta\ \ \varphi_p\ \ x_G\ \ y_G\ \ z_G\right]$

Coriolis matrix

Recall that the (i,j) element of $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ is given by the following formula [Murray, Li, Sastry 2017]:

$$\mathbf{C}_{\mathrm{ij}}=\frac{1}{2}\sum_{\mathrm{k}=1}^{\mathrm{n}}\left(\frac{\partial\mathbf{M}_{\mathrm{ij}}}{\partial\mathbf{q}_{\mathrm{k}}}+\frac{\partial\mathbf{M}_{\mathrm{ik}}}{\partial\mathbf{q}_{\mathrm{j}}}-\frac{\partial\mathbf{M}_{\mathrm{kj}}}{\partial\mathbf{q}_{\mathrm{i}}}\right)\dot{\mathbf{q}}_{\mathrm{k}},$$

In Matlab, the computation can be implemented as follows:

```
C = sym(zeros(nq,nq));
for i=1:nq
    for j=1:nq
        for k=1:nq
            C(i,j) = C(i,j) + qdotvec(k) * 0.5 * ( ...
                diff(M(i,j),qvec(k)) + ...
                diff(M(i,k),qvec(j)) - ...
                diff(M(k,j),qvec(i))                   );
        end
    end
end
C = simplify(C);
display(C)
```

```
C =
```

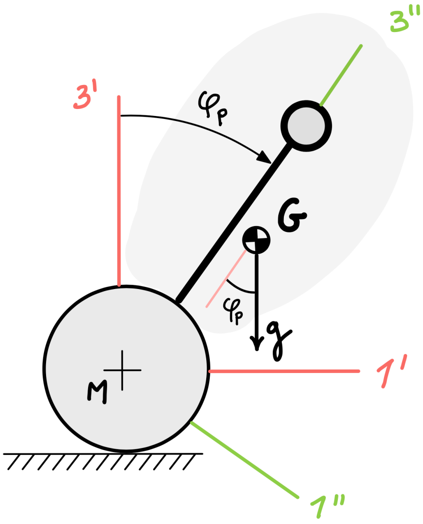
$$\left[\begin{array}{ccc}
0 & 0 & \frac{\dot{\theta}\ (2\,m_c\,y_G\sin(\theta) - 2\,m_c\cos(\theta)\ (x_G\cos(\varphi_p) + z_G\sin(\varphi_p)))}{2} - m_c\,\dot{\varphi}_p\sin(\theta)\ (z_G\cos(\varphi_p) - x_G\,\mathrm{s} \\
0 & 0 & m_c\,\dot{\varphi}_p\cos(\theta)\ (z_G\cos(\varphi_p) - x_G\sin(\varphi_p)) - \frac{\dot{\theta}\ (2\,m_c\,y_G\cos(\theta) + 2\,m_c\sin(\theta)\ (x_G\cos(\varphi_p) + z_G\,\mathrm{si}}{2} \\
0 & 0 & -\frac{\dot{\varphi}_p\ \left(m_c\sin(2\,\varphi_p)\,x_G{}^2 - 2\,m_c\cos(2\,\varphi_p)\,x_G\,z_G - m_c\sin(2\,\varphi_p)\,z_G{}^2 + 2\,I_{c13}\cos(2\,\varphi_p) - I_{c11}\sin(2\,\varphi_p) +\right.}{2} \\
0 & 0 & \frac{\dot{\theta}\ \left(m_c\sin(2\,\varphi_p)\,x_G{}^2 - 2\,m_c\cos(2\,\varphi_p)\,x_G\,z_G - m_c\sin(2\,\varphi_p)\,z_G{}^2 + 2\,I_{c13}\cos(2\,\varphi_p) - I_{c11}\sin(2\,\varphi_p) + I\right.}{2} \\
0 & 0 & 0 \\
0 & 0 & 0
\end{array}\right.$$

```
% See which variables are in C
symvar(C)
```

ans = $\left[I_{c11}\quad I_{c12}\quad I_{c13}\quad I_{c23}\quad I_{c33}\quad m_c\quad \theta\quad \dot{\theta}\quad \dot{\varphi}_p\quad \varphi_p\quad x_G\quad y_G\quad z_G\right]$

Generalized gravity force

To find the generalized gravity force, we need the gravitational potential energy of the robot. We assume that a mass has zero potential energy when it is located in the plane through M that is parallel to ground. Thus, the wheels have zero potential energy and we only have to count the potential energy of the chassis. In view of this figure,



this energy can be computed as follows using basis B''

$$U(\mathbf{q}) = m_c \cdot \overrightarrow{GM}^{\mathsf{T}} \cdot \mathbf{g} = m_c \cdot \begin{bmatrix} -x_G & -y_G & -z_G \end{bmatrix} \cdot \begin{bmatrix} g\sin\varphi_p \\ 0 \\ -g\cos\varphi_p \end{bmatrix},$$

where $\mathbf{g}$ is the acceleration of gravity vector, g is its magnitude, and (x_G, y_G, z_G) are the coordinates of G in the chassis frame (green). Therefore:

```
U = m_c * [-x_G -y_G -z_G] * [g*sin(varphi_p); 0; -g*cos(varphi_p)]
```

U = $g\, m_c\, z_G \cos(\varphi_p) - g\, m_c\, x_G \sin(\varphi_p)$

Now, $\mathbf{G}(\mathbf{q}) = \dfrac{\partial U(\mathbf{q})}{\partial \mathbf{q}}$:

```
G = jacobian(U,qvec).'
```

G =

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ -g\, m_c\, x_G \cos(\varphi_p) - g\, m_c\, z_G \sin(\varphi_p) \\ 0 \\ 0 \end{bmatrix}$$

Generalized force of actuation

Our vector $\mathbf{u}$ of motor torques is defined as

$$\mathbf{u} = \begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix}$$

Each of the torques in $\mathbf{u}$ acts directly on a $\dot{\mathbf{q}}_{\mathrm{i}}$ coordinate and, therefore, the generalized force of actuation is

```
Qa = [0 ; 0 ;0 ; 0 ; tau_l ; tau_r];
```

To rewrite this force in the form $\mathbf{Q}_a = \mathbf{E} \cdot \mathbf{u}$ we only have to define

```
E = [zeros(4,2);eye(2)];
```

Generalized force of friction

The friccion forces are given by:

$$\mathbf{Q}_a = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -b\ \ \dot{\varphi}_l \\ -b\ \ \dot{\varphi}_r \end{bmatrix}$$

```
%Qf = [-b_a*x_dot ; -b_a*y_dot ;0 ; 0 ; -b * varphi_dot_l ; -b * varphi_dot_r];
```

```
Qf = [0; 0; 0; 0; -b * varphi_dot_l ; -b * varphi_dot_r]
```

```
Qf =
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -b\,\dot{\varphi}_l \\ -b\,\dot{\varphi}_r \end{bmatrix}$$

Generalized disturbance force

This disturbance will not be used to design the controller. Rather, it will serve to test the controller under unmodelled situations. For example, when there is some wind, or when the ground offers some rolling resistance. For the moment we just model a disturbance force $\mathbf{F}_d = [F_{dx}, F_{dy}, F_{dz}]$ applied at G.

```
Fd = [F_dx;F_dy;F_dz];                          % The force in basis B'
Pvirt = transpose(Fd)*v_G;                      % Its virtual power
Qd = transpose(equationsToMatrix(Pvirt,qdotvec))    % The generalized force
```

```
Qd =
```

$$\begin{bmatrix} F_{\mathrm{dx}}\cos(\theta) - F_{\mathrm{dy}}\sin(\theta) \\ F_{\mathrm{dy}}\cos(\theta) + F_{\mathrm{dx}}\sin(\theta) \\ F_{\mathrm{dy}}\;(x_G\cos(\varphi_p) + z_G\sin(\varphi_p)) - F_{\mathrm{dx}}\,y_G \\ F_{\mathrm{dx}}\;(z_G\cos(\varphi_p) - x_G\sin(\varphi_p)) - F_{\mathrm{dz}}\;(x_G\cos(\varphi_p) + z_G\sin(\varphi_p)) \\ 0 \\ 0 \end{bmatrix}$$

The x model

We now wish to obtain the robot model in the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u})$. This requires writing $\ddot{\mathbf{q}}$ as a function of $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\mathbf{u}$ to then convert the 2nd order ODE into a 1st order one.

Recall that Lagrange's equation is

$$\mathbf{M}\;\ddot{\mathbf{q}} + \mathbf{C}\;\dot{\mathbf{q}} + \mathbf{G} + \mathbf{J}^\top\;\boldsymbol{\lambda} = \mathbf{Q}_a\; + \mathbf{Q}_f$$

(we omit the dependencies on $\mathbf{q}$ and $\dot{\mathbf{q}}$ for simplicity)

Since this equation is a system of 6 equations in 9 unknowns (6 coordinates in $\ddot{\mathbf{q}}$ and 3 in $\boldsymbol{\lambda}$) we need 3 additional equations to determine $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$ for a given $\mathbf{u}$. These can be obtained by taking the time derivative of the kinematric constraint

$$\mathbf{J}\cdot\dot{\mathbf{q}} = \mathbf{0},$$

which gives

$$\mathbf{J}\,\ddot{\mathbf{q}}+\dot{\mathbf{J}}\,\dot{\mathbf{q}}=\mathbf{0},$$

or, equivalently,

$$\mathbf{J}\,\ddot{\mathbf{q}}=-\dot{\mathbf{J}}\,\dot{\mathbf{q}}$$

The latter is called the acceleration constraint of the robot.

Recall that

$$\mathbf{J}=\left[\mathbf{I}_3\quad -\mathbf{N}\right],$$

so

$$\dot{\mathbf{J}}=\left[\mathbf{0}_3\quad -\dot{\mathbf{N}}\right]$$

Let us obtain $\dot{\mathbf{N}}$ and then $\dot{\mathbf{J}}$:

```
% Substitute theta in N for a function of t so we can take the time derivative
syms f1(t)
N_of_t = subs(N,theta,f1(t));

% Actual time derivative
dNdt = diff(N_of_t,t);

% Substitute d/dt of f1 by the original variable using dot notation
Ndot = subs(dNdt,[f1(t) diff(f1,t)],[theta theta_dot]);

% Obtain Jdot
Jdot = [zeros(3) -Ndot]
```

```
Jdot =
```

$$\begin{bmatrix} 0 & 0 & 0 & r\,\dot{\theta}\,\sin(\theta) & \frac{r\,\dot{\theta}\,\sin(\theta)}{2} & \frac{r\,\dot{\theta}\,\sin(\theta)}{2} \\ 0 & 0 & 0 & -r\,\dot{\theta}\,\cos(\theta) & -\frac{r\,\dot{\theta}\,\cos(\theta)}{2} & -\frac{r\,\dot{\theta}\,\cos(\theta)}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The system formed by Lagrange's equation and the acceleration constraint is

$$\begin{cases}\mathbf{M}\,\ddot{\mathbf{q}}+\mathbf{C}\,\dot{\mathbf{q}}+\mathbf{J}^{\top}\,\boldsymbol{\lambda}=\mathbf{E}\,\mathbf{u}\\ \mathbf{J}\,\ddot{\mathbf{q}}=-\dot{\mathbf{J}}\,\dot{\mathbf{q}}\end{cases}$$

These equations can be written as

$$\begin{bmatrix}\mathbf{M} & \mathbf{J}^{\top}\\ \mathbf{J} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{q}}\\ \boldsymbol{\lambda}\end{bmatrix}=\begin{bmatrix}\mathbf{E}\,\mathbf{u}-\mathbf{C}\,\dot{\mathbf{q}}\\ -\dot{\mathbf{J}}\,\dot{\mathbf{q}}\end{bmatrix}.$$

The matrix on the LHS is the **extended mass matrix**. This matrix is invertible, since $\mathbf{M}$ is positive-definite and $\mathbf{J}$ is full row rank. Thus we can write

$$\begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{M} & \mathbf{J}^\top \\ \mathbf{J} & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{E}\,\mathbf{u} - \mathbf{C}\,\dot{\mathbf{q}} \\ -\dot{\mathbf{J}}\,\dot{\mathbf{q}} \end{bmatrix}.$$

Therefore

$$\ddot{\mathbf{q}} = \begin{bmatrix} \mathbf{I}_6 & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{M} & \mathbf{J}^\top \\ \mathbf{J} & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{E}\,\mathbf{u} - \mathbf{C}\,\dot{\mathbf{q}} \\ -\dot{\mathbf{J}}\,\dot{\mathbf{q}} \end{bmatrix}.$$

Finally, by considering the trivial equation $\dot{\mathbf{q}} = \dot{\mathbf{q}}$ in conjunction with the earlier equation we arrive at

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}} \\ \begin{bmatrix} \mathbf{I}_6 & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{M} & \mathbf{J}^\top \\ \mathbf{J} & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{E}\,\mathbf{u} - \mathbf{C}\,\dot{\mathbf{q}} \\ -\dot{\mathbf{J}}\,\dot{\mathbf{q}} \end{bmatrix} \end{bmatrix},$$

which, if we let $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$, gives the robot model in the state space form

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}).$$

We refer to this equation as the $\mathbf{x}$ model of the robot.

A drawback of the $\mathbf{x}$ model is that the $\mathbf{x}$ variables are not independent (they must fulfill $\mathbf{F}(\mathbf{x}) = \mathbf{0}$). Therefore:

- The numerical integration of $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ easily generates errors pulling $\mathbf{x}(t)$ away from the state space $\mathscr{X}$. In practice, this translates into lateral slidings of the robot.
- The equation cannot be used to design an LQR controller to stabilize a given trajectory, as LQR theory assumes the $\mathbf{x}$ variables to be independent.

We next derive an alternative model in independent coordinates, which overcomes these limitations.

The y model

We start again from Lagrange's equation

$$\mathbf{M}\,\ddot{\mathbf{q}} + \mathbf{C}\,\dot{\mathbf{q}} + \mathbf{G} + \mathbf{J}^\top\,\boldsymbol{\lambda} = \mathbf{Q}_a(\mathbf{u}) + \mathbf{Q}_f + \mathbf{Q}_d$$

and consider this parameterization of the feasible velocities $\dot{\mathbf{q}}$

$$\begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\boldsymbol{\varphi}} \end{bmatrix} = \begin{bmatrix} \mathbf{N} \\ \mathbf{I}_3 \end{bmatrix} \cdot \dot{\boldsymbol{\varphi}}$$

which we write as

$$\dot{\mathbf{q}} = \boldsymbol{\Delta} \cdot \dot{\boldsymbol{\varphi}}$$

where $\mathbf{\Delta} = \begin{bmatrix} \mathbf{N} \\ \mathbf{I}_3 \end{bmatrix}$.

Multiplying Lagrange's equation by $\mathbf{\Delta}^\top$ we have

$$\mathbf{\Delta}^\top \mathbf{M}\ \ddot{\mathbf{q}} + \mathbf{\Delta}^\top \mathbf{C}\ \dot{\mathbf{q}} + \mathbf{\Delta}^\top \mathbf{G} + (\mathbf{J}\mathbf{\Delta})^\top \boldsymbol{\lambda} = \mathbf{\Delta}^\top \mathbf{Q}_a(\mathbf{u}) + \mathbf{\Delta}^\top \mathbf{Q}_f + \mathbf{\Delta}^\top \mathbf{Q}_d$$

Since the columns of $\mathbf{\Delta}$ are feasible velocities, and the rows of $\mathbf{J}$ are orthogonal to such velocities (by virtue of $\mathbf{J} \cdot \dot{\mathbf{q}} = \mathbf{0}$), it must be $\mathbf{J}\mathbf{\Delta} = \mathbf{0}$. Therefore, the equation simplifies to

$$\mathbf{\Delta}^\top \mathbf{M}\ \ddot{\mathbf{q}} + \mathbf{\Delta}^\top \mathbf{C}\ \dot{\mathbf{q}} + \mathbf{\Delta}^\top \mathbf{G} = \mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f + \mathbf{T}_d$$

where $\mathbf{T}_a(\mathbf{u}) = \mathbf{\Delta}^\top \mathbf{Q}_a(\mathbf{u})$, $\mathbf{T}_f = \mathbf{\Delta}^\top \mathbf{Q}_f$, and $\mathbf{T}_d = \mathbf{\Delta}^\top \mathbf{Q}_d$. Furthermore, if we apply the substitutions

$$\dot{\mathbf{q}} = \mathbf{\Delta} \cdot \dot{\boldsymbol{\varphi}}$$
$$\ddot{\mathbf{q}} = \mathbf{\Delta} \cdot \ddot{\boldsymbol{\varphi}} + \dot{\mathbf{\Delta}} \cdot \dot{\boldsymbol{\varphi}}$$

we obtain

$$\bar{\mathbf{M}}\ \ddot{\boldsymbol{\varphi}} + \bar{\mathbf{C}}\ \dot{\boldsymbol{\varphi}} + \bar{\mathbf{G}} = \mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f + \mathbf{T}_d$$

where

$$\bar{\mathbf{M}} = \mathbf{\Delta}^\top \mathbf{M} \mathbf{\Delta}$$
$$\bar{\mathbf{C}} = \mathbf{\Delta}^\top (\mathbf{M}\dot{\mathbf{\Delta}} + \mathbf{C}\mathbf{\Delta})$$
$$\bar{\mathbf{G}} = \mathbf{\Delta}^\top \mathbf{G}$$

The latter ODE is the equation of motion in $\boldsymbol{\varphi}$ coordinates. This ODE is remarkable, as all of its terms depend only on $\boldsymbol{\varphi}$ or $\dot{\boldsymbol{\varphi}}$ (see our calculations below). In fact, they only depend on $(\varphi_p, \dot{\varphi}_p, \dot{\varphi}_l, \dot{\varphi}_r)$.

To convert the ODE into first order form, we isolate $\ddot{\boldsymbol{\varphi}}$ and add the trivial equation $\dot{\boldsymbol{\varphi}} = \dot{\boldsymbol{\varphi}}$

$$\begin{cases} \dot{\boldsymbol{\varphi}} = \dot{\boldsymbol{\varphi}} \\ \ddot{\boldsymbol{\varphi}} = \bar{\mathbf{M}}^{-1}(\mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f + \mathbf{T}_d - \bar{\mathbf{C}}\ \dot{\boldsymbol{\varphi}} - \bar{\mathbf{G}}) \end{cases}$$

If we define $\mathbf{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$ and represent the RHS of the previous system by $\mathbf{g}(\mathbf{y}, \mathbf{u})$, we have

$$\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}, \mathbf{u})$$

We refer to $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}, \mathbf{u})$ as the $\mathbf{y}$ model of the robot, and to $\mathbf{y}$ as the "internal state".

The $\mathbf{y}$ model has these advantages:

- It is sufficient to predict the trajectory $\boldsymbol{\varphi}(t)$ for a given input $\mathbf{u}(t)$.
- Since the $\boldsymbol{\varphi}$ coordinates are independent, the result will also be compliant with $\mathbf{F}(\mathbf{x}) = \mathbf{0}$.
- The model allows the design of a feedback controller to stabilize the robot in the upright position, or moving with constant speeds v and $\dot{\theta}$, using propioceptive feedback only.

- Since the terms of the model depend only on $(\varphi_p, \dot{\varphi}_p, \dot{\varphi}_l, \dot{\varphi}_r)$, the linearized model will not depend on (φ_l, φ_r). In other words, the system linearization about a state $\mathbf{y} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$ stays constant as long as φ_p and $\dot{\boldsymbol{\varphi}}$ do not change.

Let us obtain $\bar{\mathbf{M}}$, $\bar{\mathbf{C}}$, $\bar{\mathbf{G}}$, $\mathbf{T}_a$, $\mathbf{T}_f$, and $\mathbf{T}_d$:

```
% Compute Delta and Deltadot
Delta = [N;eye(3)];
Deltadot = [Ndot; zeros(3)];

% Compute Mbar
Mbar = simplify(transpose(Delta) * M * Delta);

% Compute Cbar and substitute thetadot by its expression in Eq. 7
Cbar0 = simplify(transpose(Delta) * (M * Deltadot + C * Delta));
Cbar  = simplify(subs(Cbar0,theta_dot,rhs(eqn7)));

% Compute Gbar, Ta, and Tf
Gbar = simplify(transpose(Delta) * G);
Ta = simplify(transpose(Delta) * Qa);
Tf = simplify(transpose(Delta) * Qf);
Td = simplify(transpose(Delta) * Qd);

% Display the terms
display(Mbar); display(Cbar); display(Gbar); display(Ta); display(Tf); display(Td);
```

```
Mbar =
```

$$\left[\begin{array}{r} m_c\, r^2 - 2\, m_c \sin(\varphi_p)\, r\, x_G + 2\, m_c \cos(\varphi_p)\, r\, z_G + m_c\, {x_G}^2 + m_c\, {z_G}^2 + 2\, I_a + I_{c2} \\ \dfrac{2\, I_a\, d + d\, m_c\, r^2 + I_{c12}\, r \sin(\varphi_p) + m_c\, r^2\, y_G - I_{c23}\, r \cos(\varphi_p) - d\, m_c\, r\, x_G \sin(\varphi_p) + m_c\, r\, y_G\, z_G \cos(\varphi_p) - m_c\, r}{2\, d} \\ \dfrac{2\, I_a\, d + d\, m_c\, r^2 - I_{c12}\, r \sin(\varphi_p) - m_c\, r^2\, y_G + I_{c23}\, r \cos(\varphi_p) - d\, m_c\, r\, x_G \sin(\varphi_p) - m_c\, r\, y_G\, z_G \cos(\varphi_p) + m_c\, r}{2\, d} \end{array}\right.$$

```
Cbar =
```

$$\left[\begin{array}{l} \\ \dfrac{r\, \Big(2\, I_{c23}\, \dot{\varphi}_p \sin(\varphi_p) + 2\, I_{c12}\, \dot{\varphi}_p \cos(\varphi_p) - 2\, d\, m_c\, \dot{\varphi}_p\, z_G \sin(\varphi_p) - 2\, m_c\, \dot{\varphi}_p\, x_G\, y_G \cos(\varphi_p) - 2\, m_c\, \dot{\varphi}_p\, y_G\, z_G\, \mathrm{si}}{} \\ -\dfrac{r\, \Big(2\, I_{c23}\, \dot{\varphi}_p \sin(\varphi_p) + 2\, I_{c12}\, \dot{\varphi}_p \cos(\varphi_p) + 2\, d\, m_c\, \dot{\varphi}_p\, z_G \sin(\varphi_p) - 2\, m_c\, \dot{\varphi}_p\, x_G\, y_G \cos(\varphi_p) - 2\, m_c\, \dot{\varphi}_p\, y_G\, z_G\, \mathrm{s}}{} \end{array}\right.$$

```
Gbar =
```

$$\begin{bmatrix} -g\,m_c\,\left(x_G\cos(\varphi_p)+z_G\sin(\varphi_p)\right) \\ 0 \\ 0 \end{bmatrix}$$

Ta =

$$\begin{bmatrix} 0 \\ \tau_l \\ \tau_r \end{bmatrix}$$

Tf =

$$\begin{bmatrix} 0 \\ -b\,\dot{\varphi}_l \\ -b\,\dot{\varphi}_r \end{bmatrix}$$

Td =

$$\begin{bmatrix} F_{\mathrm{dx}}\,r - F_{\mathrm{dx}}\,x_G\sin(\varphi_p) - F_{\mathrm{dz}}\,z_G\sin(\varphi_p) - F_{\mathrm{dz}}\,x_G\cos(\varphi_p) + F_{\mathrm{dx}}\,z_G\cos(\varphi_p) \\ \dfrac{r\,\left(F_{\mathrm{dx}}\,d + F_{\mathrm{dx}}\,y_G - F_{\mathrm{dy}}\,z_G\sin(\varphi_p) - F_{\mathrm{dy}}\,x_G\cos(\varphi_p)\right)}{2\,d} \\ \dfrac{r\,\left(F_{\mathrm{dx}}\,d - F_{\mathrm{dx}}\,y_G + F_{\mathrm{dy}}\,z_G\sin(\varphi_p) + F_{\mathrm{dy}}\,x_G\cos(\varphi_p)\right)}{2\,d} \end{bmatrix}$$

Note that, except for the dynamic parameters, the previous terms only depend on φ_p, $\dot{\varphi}_p$, $\dot{\varphi}_l$, $\dot{\varphi}_r$

```
symvar(Mbar)
```

ans = $\begin{bmatrix} I_a & I_{\mathrm{c11}} & I_{\mathrm{c12}} & I_{\mathrm{c13}} & I_{\mathrm{c22}} & I_{\mathrm{c23}} & I_{\mathrm{c33}} & I_t & d & m_c & m_w & r & \varphi_p & x_G & y_G & z_G \end{bmatrix}$

```
symvar(Cbar)
```

ans = $\begin{bmatrix} I_{\mathrm{c11}} & I_{\mathrm{c12}} & I_{\mathrm{c13}} & I_{\mathrm{c23}} & I_{\mathrm{c33}} & d & m_c & r & \dot{\varphi}_l & \dot{\varphi}_p & \dot{\varphi}_r & \varphi_p & x_G & y_G & z_G \end{bmatrix}$

```
symvar(Td)
```

ans = $\begin{bmatrix} F_{\mathrm{dx}} & F_{\mathrm{dy}} & F_{\mathrm{dz}} & d & r & \varphi_p & x_G & y_G & z_G \end{bmatrix}$

Odometry

Suppose we have simulated the robot using the $\mathbf{y}$ model $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}, \mathbf{u})$. We then have $\boldsymbol{\varphi}(t)$, but in some situations we would like to recover the full configuration $\mathbf{q}(t)$. The angle $\theta(t)$ is easy to obtain, as from the holonomic constraint we have

$$\theta(t) = \frac{r}{2d}(\varphi_r(t) + \varphi_l(t)) + K_0$$

To recover $\mathbf{r}(t) = (x(t), y(t))$, we can use the first two equations of $\dot{\mathbf{p}} = \mathbf{N}(\theta) \cdot \dot{\boldsymbol{\varphi}}$, which we write as

$\dot{\mathbf{r}} = \mathbf{V}(\theta) \cdot \dot{\boldsymbol{\varphi}}$.

Here, $\mathbf{V}(\theta)$ is the matrix defined by the first two rows of $\mathbf{N}(\theta)$.

Therefore:

$\mathbf{r}(t) = \mathbf{r}(0) + \int_0^t \mathbf{V}(\theta(t)) \cdot \dot{\boldsymbol{\varphi}}(t) \cdot dt$.

This integral can be computed numerically using Matlab's *integral* function. This function admits vector-valued integrands like $\mathbf{V}(\theta(t)) \cdot \boldsymbol{\varphi}(t)$.

If the values $\mathbf{r}_k = \mathbf{r}(t_k)$ have to be computed on-line from a sequence of values

$\theta_k = \theta(t_k)$, $\quad \dot{\boldsymbol{\varphi}}_k = \dot{\boldsymbol{\varphi}}(t_k)$, $\quad$ for $k = 0, 1, 2, \ldots$

we can use the following recurrence, which approximates the integral from t_k to t_{k+1} with the trapezoidal rule:

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \frac{h}{2}\left(\mathbf{V}(\boldsymbol{\theta}_k) \cdot \dot{\boldsymbol{\varphi}}_k + \mathbf{V}(\boldsymbol{\theta}_{k+1}) \cdot \dot{\boldsymbol{\varphi}}_{k+1}\right)$$
$$\mathbf{r}_0 = \mathbf{r}(0)$$

The z model

In other situations, we will need a robot model that contains x and y explicitly. This is necessary, for example, when trying to stabilize the Twinbot along a given trajectory involving reference values for the (x, y) coordinates of M.

For this purpose it is easy to build a model $\mathbf{z} = \mathbf{h}(\mathbf{z}, \mathbf{u})$ involving the state

$\mathbf{z} = (x, y, \varphi_p, \varphi_l, \varphi_r, \dot{\varphi}_p, \dot{\varphi}_l, \dot{\varphi}_r)$

We only have to augment the $\mathbf{y}$ model with $\dot{\mathbf{r}} = \mathbf{V}(\theta) \cdot \dot{\boldsymbol{\varphi}}$:

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{V} \cdot \dot{\boldsymbol{\varphi}} \\ \dot{\boldsymbol{\varphi}} = \dot{\boldsymbol{\varphi}} \\ \ddot{\boldsymbol{\varphi}} = \overline{\mathbf{M}}^{-1}(\mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f - \overline{\mathbf{C}}\ \dot{\boldsymbol{\varphi}} - \overline{\mathbf{G}}) \end{cases}$$

The LHS of this system is $\dot{\mathbf{z}}$, and the RHS defines $\mathbf{h}(\mathbf{z}, \mathbf{u})$.

While $\mathbf{V}$ depends on θ in this model, this angle can be written as a function of φ_l and φ_r using the holonomic constraint $\theta = \frac{r}{2d}(\varphi_r - \varphi_l) + K_0$. Thus, the model is self-included and really depends on $\mathbf{z}$ only.

The model does not contain θ but this is irrelevant for control purposes, as θ cannot be controlled independently from x and y. A reference trajectory $(x(t), y(t))$ already determines a trajectory $\theta(t)$.

Defining a constant velocity trajectory

We now wish to find the input trajectory $\mathbf{u}(t)$ that allows the robot to move with $\varphi_p(t) \equiv 0$, while keeping constant its speeds v and $\dot{\theta}$. Since such speeds are constant, it will be $\ddot{\boldsymbol{\varphi}} = \mathbf{0}$.

Assuming there is no disturbance, the equation of motion is

$$\bar{\mathbf{M}}\,\ddot{\boldsymbol{\varphi}} + \bar{\mathbf{C}}\,\dot{\boldsymbol{\varphi}} + \bar{\mathbf{G}} = \mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f$$

If we make $\ddot{\boldsymbol{\varphi}} = \mathbf{0}$ and isolate $\mathbf{T}_a(\mathbf{u})$ we get

$$\mathbf{T}_a(\mathbf{u}) = \bar{\mathbf{C}}\,\dot{\boldsymbol{\varphi}} + \bar{\mathbf{G}} - \mathbf{T}_f$$

which, recall, only depends on $\varphi_p, \dot{\varphi}_p, \dot{\varphi}_l, \dot{\varphi}_r$.

To find the values of $\mathbf{u}$ that generate v and $\dot{\theta}$ we will apply these substitutions to the previous equation:

$$\varphi_p = 0$$
$$\dot{\varphi}_p = 0$$
$$\dot{\varphi}_l = \frac{v - d\,\dot{\theta}}{r}$$
$$\dot{\varphi}_r = \frac{v + d\,\dot{\theta}}{r}$$

```
% Compute the RHS of the equation
varphidotvec = [varphi_dot_p; varphi_dot_l; varphi_dot_r];
RHS = Cbar * varphidotvec + Gbar - Tf;

% Apply the substitutions
RHS1 = simplify(subs(RHS,...
    [varphi_p,  varphi_dot_p,  varphi_dot_l,        varphi_dot_r],...
    [0,         0,             (v-d*theta_dot)/r,   (v+d*theta_dot)/r]))
```

RHS1 =

$$\begin{bmatrix} I_{c13}\,\dot{\theta}^2 - g\,m_c\,x_G - m_c\,r\,\dot{\theta}^2\,x_G - m_c\,\dot{\theta}^2\,x_G\,z_G \\ \dfrac{b\,\left(v - d\,\dot{\theta}\right)}{r} - \dfrac{m_c\,r^2\,x_G\,\left(v + d\,\dot{\theta}\right)\,\left(\dfrac{v + d\,\dot{\theta}}{r} - \dfrac{v - d\,\dot{\theta}}{r}\right)}{4\,d^2} \\ \dfrac{b\,\left(v + d\,\dot{\theta}\right)}{r} + \dfrac{m_c\,r^2\,x_G\,\left(v - d\,\dot{\theta}\right)\,\left(\dfrac{v + d\,\dot{\theta}}{r} - \dfrac{v - d\,\dot{\theta}}{r}\right)}{4\,d^2} \end{bmatrix}$$

Recall that $\mathbf{T}_a$ must be of the form $\mathbf{T}_a = [0, \tau_l, \tau_r]^\top$, so the first component of the previous vector should be zero. We can achieve so if we assume the chassis has a symmetric design, with $x_G = 0,\ y_G = 0,\ I_{c13} = 0$:

```
RHS2 = simplify(subs(RHS1,[x_G,y_G,I_c13],[0,0,0]))
```

RHS2 =

$$\begin{bmatrix} 0 \\ \dfrac{b\,\left(v - d\,\dot{\theta}\right)}{r} \\ \dfrac{b\,\left(v + d\,\dot{\theta}\right)}{r} \end{bmatrix}$$

The second and third components of the previous vector provide the torques τ_l and τ_r that must be applied by the motors to keep the robot along the desired trajectory:

```
disp(Ta(2) == RHS2(2)); disp(Ta(3) == RHS2(3));
```

$$\tau_l = \frac{b\,\left(v - d\,\dot{\theta}\right)}{r}$$

$$\tau_r = \frac{b\,\left(v + d\,\dot{\theta}\right)}{r}$$

Particularization for a symmetric chassis design

We next particularize the $\mathbf{y}$ model assuming the chassis has a symmetric design in which:

- G is located on axis 3", so $x_G = y_G = 0$.
- The chassis inertia tensor at G is diagonal.

Particularized y model

```
% Parameters of the symmetric design
psymm = [x_G, y_G, I_c12, I_c13, I_c23];
psymmvals = [0, 0, 0, 0, 0];

% Mbar, Cbar, Gbar of the symmetric design
Mbar_symm = simplify(subs(Mbar,psymm,psymmvals));
Cbar_symm = simplify(subs(Cbar,psymm,psymmvals));
Gbar_symm = simplify(subs(Gbar,psymm,psymmvals));
Tf_symm   = simplify(subs(Tf,psymm,psymmvals));
Td_symm   = simplify(subs(Td,psymm,psymmvals));

% Display results
display(Mbar_symm); display(Cbar_symm); display(Gbar_symm); display(Tf_symm); display(Td_symm)
```

```
Mbar_symm =
```

$$\left[\begin{array}{ll} m_c\,r^2 + 2\,m_c\cos(\varphi_p)\,r\,z_G + m_c\,{z_G}^2 + 2\,I_a + I_{c22} & \frac{m_c\,r^2}{2} + \\ \frac{m_c\,r^2}{2} + \frac{m_c\,z_G\cos(\varphi_p)\,r}{2} + I_a & \frac{8\,I_a\,d^2 + I_{c11}\,r^2 + I_{c33}\,r^2 + 4\,I_t\,r^2 + 2\,d^2\,m_c\,r^2 + 4\,d^2\,m_w\,r^2}{} \\ \frac{m_c\,r^2}{2} + \frac{m_c\,z_G\cos(\varphi_p)\,r}{2} + I_a & -\frac{r^2\,\left(I_{c11} + I_{c33} + 4\,I_t - 2\,d^2\,m_c + 4\,d^2\,m_w +\right.}{} \end{array}\right.$$

```
Cbar_symm =
```

$$\left[\begin{array}{c} -m_c\, r\, \dot{\varphi}_p\, z_G \sin(\varphi_p) \\ \frac{r\,\left(\frac{I_{c11}\, r \sin(2\,\varphi_p)\,\left(\dot{\varphi}_l - \dot{\varphi}_r\right)}{2\,d} - 2\,d\, m_c\, \dot{\varphi}_p\, z_G \sin(\varphi_p) - \frac{I_{c33}\, r \sin(2\,\varphi_p)\,\left(\dot{\varphi}_l - \dot{\varphi}_r\right)}{2\,d} + \frac{m_c\, r^2\, z_G \sin(\varphi_p)\, (\dot{\varphi}_l -}{d}\right.}{4\,d} \\ -\frac{r\,\left(2\,d\, m_c\, \dot{\varphi}_p\, z_G \sin(\varphi_p) + \frac{I_{c11}\, r \sin(2\,\varphi_p)\,\left(\dot{\varphi}_l - \dot{\varphi}_r\right)}{2\,d} - \frac{I_{c33}\, r \sin(2\,\varphi_p)\,\left(\dot{\varphi}_l - \dot{\varphi}_r\right)}{2\,d} + \frac{m_c\, r^2\, z_G \sin(\varphi_p)\, (\dot{\varphi}_l -}{d}\right.}{4\,d} \end{array}\right.$$

Gbar_symm =

$$\begin{bmatrix} -g\, m_c\, z_G \sin(\varphi_p) \\ 0 \\ 0 \end{bmatrix}$$

Tf_symm =

$$\begin{bmatrix} 0 \\ -b\, \dot{\varphi}_l \\ -b\, \dot{\varphi}_r \end{bmatrix}$$

Td_symm =

$$\begin{bmatrix} F_{dx}\, r - F_{dz}\, z_G \sin(\varphi_p) + F_{dx}\, z_G \cos(\varphi_p) \\ \frac{r\,\left(F_{dx}\, d - F_{dy}\, z_G \sin(\varphi_p)\right)}{2\,d} \\ \frac{r\,\left(F_{dx}\, d + F_{dy}\, z_G \sin(\varphi_p)\right)}{2\,d} \end{bmatrix}$$

Explicit dynamics function

For the symmetric design (and maybe for the general one) it is feasible to write the dynamics function explicitly, but this is costly.

For this we consider the equation of motion in $\boldsymbol{\varphi}$ coordinates (again without disturbance)

$$\overline{\mathbf{M}}\ \ddot{\boldsymbol{\varphi}} + \overline{\mathbf{C}}\ \dot{\boldsymbol{\varphi}} + \overline{\mathbf{G}} = \mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f$$

and write it as

$$\ddot{\boldsymbol{\varphi}} = \overline{\mathbf{M}}^{-1} \cdot \mathscr{F}(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}, \mathbf{u})$$

where

$$\mathscr{F}(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}, \mathbf{u}) = \mathbf{T}_a(\mathbf{u}) + \mathbf{T}_f - \overline{\mathbf{C}}\ \dot{\boldsymbol{\varphi}} - \overline{\mathbf{G}}$$

In state space form, the equation of motion is then

$$\dot{\mathbf{y}} = \mathbf{g}_{\text{dyn}}(\mathbf{y}, \mathbf{u}),$$

where

$$\mathbf{g}_{\mathrm{dyn}}(\mathbf{y},\mathbf{u}) = \begin{bmatrix} \dot{\boldsymbol{\varphi}} \\ \bar{\mathbf{M}}^{-1} \cdot \mathscr{F}(\boldsymbol{\varphi},\dot{\boldsymbol{\varphi}},\mathbf{u}) \end{bmatrix}$$

Let us compute $\mathbf{g}_{\mathrm{dyn}}$:

```
% Function Fcal_symm (Fcal for the symmetric design)
Fcal_symm = simplify(Ta + Tf_symm - Cbar_symm * varphidotvec - Gbar_symm)
```

```
Fcal_symm =
```

$$\left[\begin{array}{r} g\, m_c\, z_G \sin(\varphi_p) + m_c\, r\, \dot{\varphi}_p{}^2\, z_G \sin(\varphi_p) + \dfrac{r^2\, \dot{\varphi}_l\, \left(\dot{\varphi}_l - \dot{\varphi}_r\right)\, \left(m_c \sin(2\,\varphi_p\right.}{} \\ \\ \dfrac{4\,d^2\,\tau_l - 4\,b\,d^2\,\dot{\varphi}_l - I_{c11}\,r^2\,\dot{\varphi}_l\,\dot{\varphi}_p \sin(2\,\varphi_p) + I_{c11}\,r^2\,\dot{\varphi}_p\,\dot{\varphi}_r \sin(2\,\varphi_p) + I_{c33}\,r^2\,\dot{\varphi}_l\,\dot{\varphi}_p \sin(2\,\varphi_p) - I_{c33}\,r^2\,\dot{\varphi}_p\,\dot{\varphi}_r\, \mathrm{si}}{} \\ \\ \dfrac{4\,d^2\,\tau_r - 4\,b\,d^2\,\dot{\varphi}_r + I_{c11}\,r^2\,\dot{\varphi}_l\,\dot{\varphi}_p \sin(2\,\varphi_p) - I_{c11}\,r^2\,\dot{\varphi}_p\,\dot{\varphi}_r \sin(2\,\varphi_p) - I_{c33}\,r^2\,\dot{\varphi}_l\,\dot{\varphi}_p \sin(2\,\varphi_p) + I_{c33}\,r^2\,\dot{\varphi}_p\,\dot{\varphi}_r\, \mathrm{s}}{} \end{array}\right.$$

The symbolic computation of $\mathbf{g}_{\mathrm{dyn}}$ will take some time (minutes maybe). Uncomment the following if you want to try it. The mlx file becomes rather unmanagable afterwards.

```
% % Function varphiddot giving the acceleration
% varphiddot = simplify(Mbar_symm\Fcal_symm);
% display(Fcal_symm); display(varphiddot);
%
% % Dynamics function
% g_dyn = [varphidot;varphiddot]
```

Jacobians A and B

In many applications it will be necessary to have the Jacobians

$$\mathbf{A} = \frac{\partial \mathbf{g}_{dyn}}{\partial \boldsymbol{\varphi}}, \qquad \mathbf{B} = \frac{\partial \mathbf{f}_{dyn}}{\partial \mathbf{u}},$$

as, for example, when linearizing the system dynamics about a state $\boldsymbol{\varphi}_0$. Again this computation is costly (uncomment if you dare):

```
% % Jacobian of g_dyn wrt the state
% A = simplify(jacobian(g_dyn,yvec));
%
% % Jacobian of g_dyn wrt the action
% B = simplify(jacobian(g_dyn,uvec));
%
% display(A); display(B);
```

Computation of A and B without symbolic inversion of Mbar

An alternative and faster way to compute $\mathbf{A}$ and $\mathbf{B}$ is through the differentiation formulas in Kelly (2017). These avoid the inversion of $\bar{\mathbf{M}}$ and produce much nicer expressions.

Since

$$\mathbf{A} = \frac{\partial \mathbf{g}_{dyn}}{\partial \mathbf{y}}, \qquad \mathbf{B} = \frac{\partial \mathbf{g}_{dyn}}{\partial \mathbf{u}},$$

and

$$\mathbf{g}_{dyn}(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}, \mathbf{u}) = \begin{bmatrix} \dot{\boldsymbol{\varphi}} \\ \mathbf{a}(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}, \mathbf{u}) \end{bmatrix}$$

where $\mathbf{a} = \bar{\mathbf{M}}^{-1}\, \mathscr{F}$, we have

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \dot{\boldsymbol{\varphi}}}{\partial \boldsymbol{\varphi}} & \frac{\partial \dot{\boldsymbol{\varphi}}}{\partial \dot{\boldsymbol{\varphi}}} \\ \frac{\partial \mathbf{a}}{\partial \boldsymbol{\varphi}} & \frac{\partial \mathbf{a}}{\partial \dot{\boldsymbol{\varphi}}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \frac{\partial \mathbf{a}}{\partial \boldsymbol{\varphi}} & \frac{\partial \mathbf{a}}{\partial \dot{\boldsymbol{\varphi}}} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \frac{\partial \dot{\boldsymbol{\varphi}}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{a}}{\partial \mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times2} \\ \frac{\partial \mathbf{a}}{\partial \mathbf{u}} \end{bmatrix}$$

We can now apply the following formulas in Kelly (2017):

$$\frac{\partial \mathbf{a}}{\partial \varphi_i} = \bar{\mathbf{M}}^{-1}\left(-\frac{\partial \bar{\mathbf{M}}}{\partial \varphi_i} \cdot \ddot{\boldsymbol{\varphi}} + \frac{\partial \mathscr{F}}{\partial \varphi_i} \right)$$

$$\frac{\partial \mathbf{a}}{\partial \dot{q}_i} = \bar{\mathbf{M}}^{-1} \cdot \frac{\partial \mathscr{F}}{\partial \dot{\varphi}_i}$$

$$\frac{\partial \mathbf{a}}{\partial \mathbf{u}} = \bar{\mathbf{M}}^{-1} \cdot \frac{\partial \mathscr{F}}{\partial \mathbf{u}}$$

Thus, if we have the symbolic expressions of $\bar{\mathbf{M}}$ and $\mathscr{F}$ and the Jacobians

$\partial \bar{\mathbf{M}} / \partial \boldsymbol{\varphi}$, $\quad \partial \mathscr{F} / \partial \boldsymbol{\varphi}$, $\quad \partial \mathscr{F} / \partial \dot{\boldsymbol{\varphi}}$, $\quad$ and $\quad \partial \mathscr{F} / \partial \mathbf{u}$

it is easy to write a Matlab function that numerically evaluates $\mathbf{A}$ and $\mathbf{B}$ for a given $\mathbf{y}$ and $\mathbf{u}$. Such a function should first obtain $\ddot{\boldsymbol{\varphi}}$ by solving $\bar{\mathbf{M}}\, \ddot{\boldsymbol{\varphi}} = \mathscr{F}$, to then compute the derivatives using the mentioned Jacobians. In this function, it will be helpful to obtain the LU decomposition of $\bar{\mathbf{M}}$, which minimizes the number of operations required to evaluate $\bar{\mathbf{M}}^{-1} \cdot \mathbf{v}$ when this has to be done for many $\mathbf{v}$ vectors.

```
% Jacobian of Mbar_symm wrt varphi
Mbar_symm_varphi = sym(zeros(3,3,3));
for i = 1:3
    Mbar_symm_varphi(:,:,i) = diff(Mbar_symm,varphivec(i));
end

% Jacobian of Fcal wrt varphi
Fcal_symm_varphi = jacobian(Fcal_symm,varphivec);
```

```
% Jacobian of Fcal wrt varphidot
Fcal_symm_varphidot = jacobian(Fcal_symm,varphidotvec);

% Jacobian of Fcal wrt u
Fcal_symm_u = jacobian(Fcal_symm,uvec);

% Display the results
display(Mbar_symm_varphi);
```

```
Mbar_symm_varphi(:,:,1) =
```

$$\begin{bmatrix} -2\, m_c\, r\, z_G \sin(\varphi_p) & -\frac{m_c\, r\, z_G \sin(\varphi_p)}{2} & - \\ -\frac{m_c\, r\, z_G \sin(\varphi_p)}{2} & \frac{2\, I_{c11}\, r^2 \sin(2\,\varphi_p) - 2\, I_{c33}\, r^2 \sin(2\,\varphi_p) + 2\, m_c\, r^2\, {z_G}^2 \sin(2\,\varphi_p)}{8\, d^2} & -\frac{r^2\ (2\, m_c \sin(2\,\varphi_p)\, {z_G}^2}{} \\ -\frac{m_c\, r\, z_G \sin(\varphi_p)}{2} & -\frac{r^2\ \left(2\, m_c \sin(2\,\varphi_p)\, {z_G}^2 + 2\, I_{c11} \sin(2\,\varphi_p) - 2\, I_{c33} \sin(2\,\varphi_p)\right)}{8\, d^2} & \frac{2\, I_{c11}\, r^2 \sin(2\,\varphi_p) - 2\, I_{c}}{} \end{bmatrix}$$

```
Mbar_symm_varphi(:,:,2) =
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
Mbar_symm_varphi(:,:,3) =
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
display(Fcal_symm_varphi);
```

```
Fcal_symm_varphi =
```

$$\begin{bmatrix} g\, m_c\, z_G \cos(\varphi_p) + m_c\, r\, {\dot{\varphi}_p}^2\, z_G \cos(\varphi_p) + \frac{r^2\, \dot{\varphi}_l\ (\dot{\varphi}_l - \dot{\varphi}_r)\ (2\, m_c \cos(2\,\varphi_p)\, {z_G}^2 +}{} \\ \frac{2\, I_{c11}\, r^2\, \dot{\varphi}_p\, \dot{\varphi}_r \cos(2\,\varphi_p) - 2\, I_{c11}\, r^2\, \dot{\varphi}_l\, \dot{\varphi}_p \cos(2\,\varphi_p) + 2\, I_{c33}\, r^2\, \dot{\varphi}_l\, \dot{\varphi}_p \cos(2\,\varphi_p) - 2\, I_{c33}\, r^2\, \dot{\varphi}_p\, \dot{\varphi}_r \cos(2\,\varphi_p) + m}{} \\ \frac{2\, I_{c11}\, r^2\, \dot{\varphi}_l\, \dot{\varphi}_p \cos(2\,\varphi_p) - 2\, I_{c11}\, r^2\, \dot{\varphi}_p\, \dot{\varphi}_r \cos(2\,\varphi_p) - 2\, I_{c33}\, r^2\, \dot{\varphi}_l\, \dot{\varphi}_p \cos(2\,\varphi_p) + 2\, I_{c33}\, r^2\, \dot{\varphi}_p\, \dot{\varphi}_r \cos(2\,\varphi_p) + m}{} \end{bmatrix}$$

```
display(Fcal_symm_varphidot);
```

```
Fcal_symm_varphidot =
```

$$\begin{bmatrix} 2\, m_c\, r\, \dot{\varphi}_p\, z_G \sin(\varphi \\ \frac{I_{c11}\, r^2\, \dot{\varphi}_r \sin(2\,\varphi_p) - I_{c11}\, r^2\, \dot{\varphi}_l \sin(2\,\varphi_p) + I_{c33}\, r^2\, \dot{\varphi}_l \sin(2\,\varphi_p) - I_{c33}\, r^2\, \dot{\varphi}_r \sin(2\,\varphi_p) - m_c\, r^2\, \dot{\varphi}_l\, {z_G}^2 \sin(2\,\varphi_p) +}{4\, d^2} \\ \frac{I_{c11}\, r^2\, \dot{\varphi}_l \sin(2\,\varphi_p) - I_{c11}\, r^2\, \dot{\varphi}_r \sin(2\,\varphi_p) - I_{c33}\, r^2\, \dot{\varphi}_l \sin(2\,\varphi_p) + I_{c33}\, r^2\, \dot{\varphi}_r \sin(2\,\varphi_p) + m_c\, r^2\, \dot{\varphi}_l\, {z_G}^2 \sin(2\,\varphi_p) -}{4\, d^2} \end{bmatrix}$$

```
display(Fcal_symm_u);
```

```
Fcal_symm_u =
```

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Export Matlab functions of the generic symmetric model

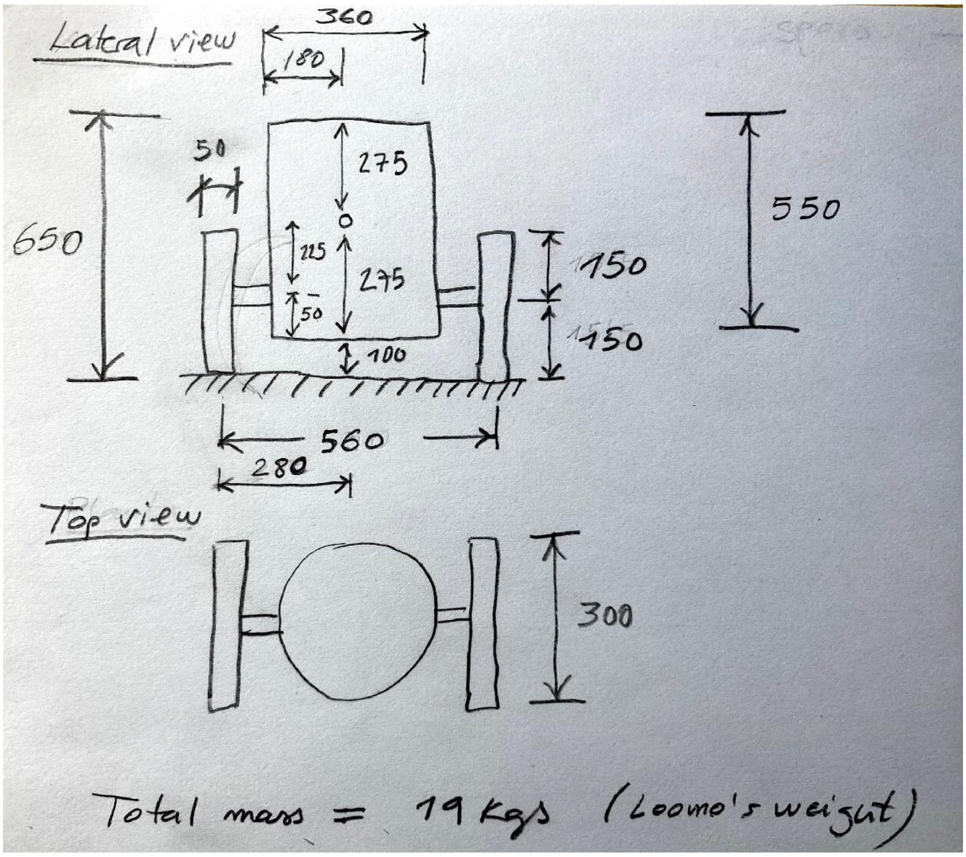
```
% Define the input variables of the exported functions
vars = {yvec,uvec,parsvec};

matlabFunction(Mbar_symm,'File','Compute_Mbar_symm','Vars',vars);
matlabFunction(Cbar_symm,'File','Compute_Cbar_symm','Vars',vars);
matlabFunction(Gbar_symm,'File','Compute_Gbar_symm','Vars',vars);
matlabFunction(Tf_symm,'File','Compute_Tf_symm','Vars',vars);
matlabFunction(Mbar_symm,Fcal_symm,...
    Mbar_symm_varphi,Fcal_symm_varphi,Fcal_symm_varphidot,Fcal_symm_u,...
    'File','Compute_Mbar_Fcal_Jacobians_symm','Vars',vars);
```

Export Matlab functions of the loomo-like symmetric model

Let us find the terms of the $\mathbf{y}$ model for specific parameters. The following values are inspired by the Loomo robot by Segway.

See https://www.segway.com/loomo/loomo-spec/

We assume the wheels and the chassis are cylinders of uniform mass distribution with the following dimensions (in mm):



```
g_val = 9.8;                                    % Acceleration of gravity (m/s^2)

xG_val = 0;                                     % x coord of G (m)
yG_val = 0;                                     % y coord of G (m)
zG_val = 0.225;                                 % z coord of G (m)

d_val = 0.280;                                  % Semiaxis length (m)

rw_val = 0.150;                                 % Wheel radius (m)
hw_val = 0.050;                                 % Wheel width (m)
mw_val = 3;                                     % Wheel mass (kg)
Ia_val = mw_val * rw_val^2;                     % Wheel axial inertia moment (kg·m^2)
It_val = mw_val * (rw_val^2/2 + hw_val^2/12);   % Wheel twisting inertia moment (kg·m^2)

rc_val = 0.180;                                 % Chassis radius (m)
hc_val = 0.550;                                 % Chassis height (m)
mc_val = 13;                                    % Chassis mass (m)

Ic11_val = mc_val * (rc_val^2/2+hc_val^2/12);   % Chassis inertia tensor 11 element (kg·m^2)
Ic12_val = 0;                                   % Chassis inertia tensor 12 element (kg·m^2)
Ic13_val = 0;                                   % Chassis inertia tensor 13 element (kg·m^2)
Ic22_val = mc_val * rc_val^2;                   % Chassis inertia tensor 22 element (kg·m^2)
Ic23_val = 0;                                   % Chassis inertia tensor 23 element (kg·m^2)
Ic33_val = Ic11_val;                            % Chassis inertia tensor 33 element (kg·m^2)
```

```
b_val = 0.1;                                        % Viscous friction coeff. wheels

% Parameters vector
p = [...
            d, r, x_G, y_G, z_G,...
            m_w, I_a, I_t, m_c,...
            I_c11, I_c12, I_c13, I_c22, I_c23, I_c33,...
            b, g
];

% Parameter values vector
pvals = [...
            d_val, rw_val, xG_val, yG_val, zG_val,...
            mw_val, Ia_val, It_val, mc_val, ...
            Ic11_val, Ic12_val, Ic13_val, Ic22_val, Ic23_val, Ic33_val,...
            b_val, g_val
];

% Save parameters vector
pars_loomo = pvals';
save('parameters_loomo.mat','pars_loomo');

% Evaluate Mbar, Cbar, Gbar
Mbar_loomo = vpa(subs(Mbar_symm,p,pvals));
Cbar_loomo = vpa(subs(Cbar_symm,p,pvals));
Gbar_loomo = vpa(subs(Gbar_symm,p,pvals));
Tf_loomo = vpa(subs(Tf_symm,p,pvals));

% Display results
display(Mbar_loomo); display(Cbar_loomo); display(Gbar_loomo); display(Tf_loomo);
```

```
Mbar_loomo =
```

$$\begin{bmatrix} 0.8775\cos(\varphi_p) + 1.506825 & 0.219375\cos(\varphi_p) + 0.21375 \\ 0.219375\cos(\varphi_p) + 0.21375 & 0.24153928172831633049852667729552 - 0.0236093949298469387755 \\ 0.219375\cos(\varphi_p) + 0.21375 & 0.023609394929846938775510204081633\cos(2.0\,\varphi_p) - 0.027789281728 \end{bmatrix}$$

```
Cbar_loomo =
```

$$\begin{bmatrix} -0.43875\,\dot{\varphi}_p\sin(\varphi_p) \\ 0.031479193239795918367346938775510\sin(\varphi_p)\,\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) - 0.219375\,\dot{\varphi}_p\sin(\varphi_p) + 0.02360939492 \\ -0.031479193239795918367346938775510\sin(\varphi_p)\,\left(\dot{\varphi}_l - 1.0\,\dot{\varphi}_r\right) - 0.219375\,\dot{\varphi}_p\sin(\varphi_p) - 0.0236093949 \end{bmatrix}$$

```
Gbar_loomo =
```

$$\begin{bmatrix} -28.665\sin(\varphi_p) \\ 0 \\ 0 \end{bmatrix}$$

```
Tf_loomo =
```

$$\begin{bmatrix} 0 \\ -0.1\,\dot{\varphi}_l \\ -0.1\,\dot{\varphi}_r \end{bmatrix}$$

```
% Export dynamics terms of loomo-like model
matlabFunction(Mbar_loomo,'File','Compute_Mbar_loomo','Vars',vars);
matlabFunction(Cbar_loomo,'File','Compute_Cbar_loomo','Vars',vars);
matlabFunction(Gbar_loomo,'File','Compute_Gbar_loomo','Vars',vars);
matlabFunction(Tf_loomo,'File','Compute_Tf_loomo','Vars',vars);
```

Print ellapsed time

```
toc
```

```
Elapsed time is 26.239028 seconds.
```

References

Kelly, Matthew. "An introduction to trajectory optimization: How to do your own direct collocation." *SIAM Review* 59.4 (2017): 849-904.

Murray, Richard M., Zexiang Li, and S. Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

A.2 Matlab code

In this section, it figures the application code of the graphical interface, which works as the main function, the external *.m* functions that are being called, and the URDF file that defines the robot's bodies and joints.

A.2.1 Graphical interface application

```
classdef Twinbot_app_neta_graphs_exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                     matlab.ui.Figure
        ExternalForceKnob            matlab.ui.control.Knob
        ExternalForceKnobLabel       matlab.ui.control.Label
        Tau_LGauge                   matlab.ui.control.LinearGauge
        Tau_LGaugeLabel              matlab.ui.control.Label
        Tau_RGauge                   matlab.ui.control.LinearGauge
        Tau_RGaugeLabel              matlab.ui.control.Label
        AngularspeedradsGauge        matlab.ui.control.Gauge
        AngularspeedradsGaugeLabel   matlab.ui.control.Label
        LinearspeedmsGauge           matlab.ui.control.Gauge
        LinearspeedmsGaugeLabel      matlab.ui.control.Label
        ControlPeriodGauge           matlab.ui.control.SemicircularGauge
        ControlPeriodGaugeLabel      matlab.ui.control.Label
        CameraElevationKnob          matlab.ui.control.Knob
        CameraElevationKnobLabel     matlab.ui.control.Label
        QuitButton                   matlab.ui.control.Button
        MetersAroundKnob             matlab.ui.control.Knob
        MetersAroundKnobLabel        matlab.ui.control.Label
        ViewModeButtonGroup          matlab.ui.container.ButtonGroup
        ObliqueViewButton            matlab.ui.control.RadioButton
        ZenithalViewButton           matlab.ui.control.RadioButton
        FirstPersonViewButton        matlab.ui.control.RadioButton
    end


    % User defined app properties: access them as app.axobj, app.quit, ...
    properties (Access = private)
        axobj % Axes object on which we plot the robot
        quit  % Logical. If true, we want to quit
        v     % Desired velocity in m/s
        om    % Desired angular velocity in rad/s
    end
    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            %-------------------------------------------------------------
            % This code is run upon app startup
            %-------------------------------------------------------------

            % Initializations
            fprintf('App started\n');
%             close all;

            % Create separate figure from app main panel
            h = figure;

            % Initialize app properties, in particular the axes in h
            app.axobj = axes(h);
            app.quit = false;
            app.v = 0;
            app.om = 0;

            % Import URDF model of robot ('row' means q will be row vector)
            robot = importrobot('diffbot2.urdf','DataFormat','row');

            % Print robot details in command window (optional)
            showdetails(robot);

            % Show robot in home configuration (delayed until 'drawnow')
            q = homeConfiguration(robot);
            show(robot,q,'Frames','off','collision','off',...
                'visuals','on','PreservePlot',0,'FastUpdate',1,...
                'Parent',app.axobj);

            % Set limits of the axes in the robot window. To avoid plotting
            % in unwanted axes, we specify the axes handle in all axes-
            % related commands.
            mar = app.MetersAroundKnob.Value;
            app.axobj.XLim = [q(1)-mar q(1)+mar];
            app.axobj.YLim = [q(2)-mar q(2)+mar];
            app.axobj.ZLim = [0 1];

            % Use black grid lines with opacity GridAlpha in [0,1]
```

```
app.axobj.LineWidth = 1.5;
app.axobj.GridLineStyle = '–';
app.axobj.GridColor = 'r';
app.axobj.GridAlpha = 0.2;

% Use a boxed plot
box on;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Initialitzations before loop control
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
global FORCE TIME_INI TIME_END
global u_val

% Twinbot parameters
parameters=readmatrix('parameters.xls');
param = load('parameters_twinbot.mat').pars_twinbot;

phi_p_i = 0;                  % Initial pendulum angle (Degree)
FORCE = 0;                    % Disturbance force (N)
TIME_INI = 0;                 % Disturbance initial time (s)
TIME_END = 0;                 % Disturbance final time (s)

pars.g = parameters(1);       % Gravity [m/s^2]
pars.r = parameters(2);       % Wheel radius [m]
pars.d = parameters(3);       % Semiaxis length [m]
pars.zG = parameters(4);      % z coord of G (m)
pars.b = parameters(5);       % Friction coefficient
pars.m = parameters(6);       % Mass of the chassis [kg]

delta_t = 0.02;                % Simulation step time

% Initial twinbot state xi = (xi,yi,thetai,varphi_pi,varphi_li,
    varphi_ri,
% ... xi_dot,yi_dot,thetai_dot,varphi_pi_dot,varphi_li_dot,
    varphi_ri_dot)
% and yi = (varphi_pi,varphi_li,varphi_ri,
% ... varphi_pi_dot,varphi_li_dot,varphi_ri_dot)
xi = [0;0;0;deg2rad(phi_p_i);0;0;0;0;0;0;0;0];
yi= [xi(4:6);xi(10:12)];

%Equilibrium State:
x0 = [0;0;0;deg2rad(0);deg2rad(phi_p_i);deg2rad(phi_p_i)
    ;0;0;0;0;0;0];
y0= [x0(4:6);x0(10:12)];
```

```
app.v= 0;
app.om= 0;
phi_dot = yi(4:6);
x=xi;       % initialitzation of the x vector
y=yi;       % initialitzation of the y vector
xM=xi(1); yM=xi(2); theta=xi(3); xM0=x0(1);

% Torque tau0 needed to maintain equilibrium
% if xG=yG=0 --> l=zG
u0 = pars.m * pars.g * pars.zG * sin(y0(1)) * 0.5 * [1;1];

y0(5) = (app.v - pars.d * app.om) / pars.r;
y0(6) = (app.v+ pars.d * app.om) / pars.r;

% Penalty matrices Q and R
Q = 0.0025*diag([1000,1,1,10,1,1]);      % Penalises state deviations
     from y0
R = 0.0025*diag([1,1]);              % Penalises action deviations
    from tau0
% Ask for a precise simulation (otherwise energy is not conserved)
options = odeset('abstol', 1e-6,'reltol',1e-6);

% Plot the robot at q (limit rate to 20 fps for speedup)
q = x(1:6)';
show(robot,q,'Frames','off','collision','off',...
    'visuals','on','PreservePlot',0,'FastUpdate',1,...
    'Parent',app.axobj);
hold on;

% Set axes limits centered at the robot position
mar = app.MetersAroundKnob.Value;
app.axobj.XLim = [q(1)-mar q(1)+mar];
app.axobj.YLim = [q(2)-mar q(2)+mar];
app.axobj.ZLim = [0 1];

% Set view according to the selected view mode
switch true
    case app.ObliqueViewButton.Value
        % Oblique view
        camproj(app.axobj,'perspective');
        elev = app.CameraElevationKnob.Value;
        view(45,elev);
    case app.FirstPersonViewButton.Value
        % First person view
        camproj(app.axobj,'perspective');
        xM  = q(1); yM  = q(2); th = q(3);
        campos([xM yM 3]-10*[cos(th) sin(th) 0]);
        camtarget([xM yM 0]+3*[cos(th) sin(th) 0]);
    case app.ZenithalViewButton.Value
        % Zenithal view
        camproj(app.axobj,'orthographic');
        view(0,90);
end

drawnow limitrate;

phi_dot0 = y0(4:6);

varphip= [0];
varphip_dot= [0];
theta_dot= [0];
v= [0];
theta_dot0= [0];
v0= [0];
u01= [0];
u1= [0];
u02= [0];
u2= [0];
F= [0];
t= [0];
zero=[0];
xMvec= [0];
xMvec0= [0];

app.quit = false;
pause(3);

tic;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Simulate the closed-loop system
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Start the robot control loop. Note that app.quit can only be
% set to true by the QuitButtonPushed callback
while ~app.quit

    % Infer time increment
    t_act = toc;
    t_future = toc + delta_t;
```

```
% Update the robot configuration q
phi_dot0_ant = phi_dot0;

y0(5) = (app.v - pars.d * app.om) / pars.r;
y0(6) = (app.v + pars.d * app.om) / pars.r;
y0(2) = y0(2) + delta_t * y0(5);
y0(3) = y0(3) + delta_t * y0(6);
u0(1) = pars.b*(app.v-pars.d*app.om)/pars.r;
u0(2) = pars.b*(app.v+pars.d*app.om)/pars.r;
[A,B] = AB_cdiff(y0,u0,param);
[K,~,~] = lqr(A,B,Q,R);

phi_dot0 = y0(4:6);

y_act = y;

% Get handle to function that gives the control torque
u = @(t,y)tau_wheels(t,y,y0,u0,K);

% Get handle that gives ydot = f(y,u)
dydt = @(t,y)ydot_twinbot(t,y,u,pars,param);

% Simulate the closed-loop system for t in [toc,toc + delta_t sec
    ], starting
% from y = y_act
sol = ode45(dydt,[t_act,t_future],y_act,options);

% Evaluate solution for times in t. Size of y is [length(y0) ,1]
y = deval(sol,t_future);

% Calculate x solution. Size of x is [length(x0) , 1]

phi_dot_ant = phi_dot;

phi = y(1:3);
phi_dot = y(4:6);

N0 = Compute_N(theta);

theta = xi(3) + pars.r * (y(3)-y(2)) / (2*pars.d) - ...
    pars.r * (yi(3)-yi(2)) / (2*pars.d);

N1 = Compute_N(theta);
```

```
p_dot = N1 * phi_dot;

Vx0=N0(1,:);
Vy0=N0(2,:);
Vx1=N1(1,:);
Vy1=N1(2,:);

xM = xM + (delta_t/2) * (Vx0*phi_dot_ant + Vx1*phi_dot);
yM = yM + (delta_t/2) * (Vy0*phi_dot_ant + Vy1*phi_dot);

x=[xM;yM;theta;phi;p_dot;phi_dot];

q = x(1:6)';

% Plot the robot at q (limit rate to 20 fps for speedup)
show(robot,q,'Frames','off','collision','off',...
    'visuals','on','PreservePlot',0,'FastUpdate',1,...
    'Parent',app.axobj);
hold on;

% Set axes limits centered at the robot position
mar = app.MetersAroundKnob.Value;
app.axobj.XLim = [q(1)-mar q(1)+mar];
app.axobj.YLim = [q(2)-mar q(2)+mar];
app.axobj.ZLim = [0 1];

% Set view according to the selected view mode
switch true
    case app.ObliqueViewButton.Value
        % Oblique view
        camproj(app.axobj,'perspective');
        elev = app.CameraElevationKnob.Value;
        view(45,elev);
    case app.FirstPersonViewButton.Value
        % First person view
        camproj(app.axobj,'perspective');
        xM  = q(1); yM  = q(2); th = q(3);
        campos([xM yM 3]-10*[cos(th) sin(th) 0]);
        camtarget([xM yM 0]+3*[cos(th) sin(th) 0]);
    case app.ZenithalViewButton.Value
        % Zenithal view
        camproj(app.axobj,'orthographic');
        view(0,90);
end
```

```
        % Draw now
        drawnow limitrate;

        % Store variables for future plots
        xM0 = xM0 + (delta_t/2) * (Vx0*phi_dot0_ant + Vx1*phi_dot0);

        varphip= [varphip,x(4)];
        varphip_dot= [varphip_dot,x(10)];
        theta_dot=[theta_dot,x(9)];
%          v_val=sqrt(x(7)^2+x(8)^2);
          v_val= pars.r*(x(11)+2*x(10)+x(12))/2
        v=[v,v_val];
        theta_dot0= [theta_dot0,app.om];
        v0= [v0,app.v];
        u01= [u01,u0(1)];
        u1= [u1,u_val(1)];
        u02= [u02,u0(2)];
        u2= [u2,u_val(2)];
        F=[F,FORCE];
        t=[t,t_act];
        zero=[zero,0];
        xMvec=[xMvec,xM];
        xMvec0=[xMvec0,xM0];

        % Fix the iteration time to delta_t (the only problem is
        % ifthe iteration time get's higher than delta_t)
        t_pause = t_future - toc;
        pause (t_pause);

        % Update the 'Control period' gauge (miliseconds)
        app.ControlPeriodGauge.Value = (delta_t-t_pause)*1000;

        if app.quit
%              pause(10);
            break
        end

        % Update slider values with app.v and app.om
        app.ExternalForceKnob.Value = FORCE;
        app.Tau_LGauge.Value = u_val(1);
        app.Tau_RGauge.Value = u_val(2);

    end

    app.quit=false;
```

```
% Plots (it is needed to change the commented sections in order
% to get the different graphs)

figure(2);
clf;

% Plot theta_dot(t)
subplot(3,2,1)
plot(t,theta_dot);
hold on;
plot(t,theta_dot0);
hold off;
title('theta dot');
xlabel('Time (s)');
ylabel('\theta dot (ras/s)');
legend('\theta dot (ras/s)','\theta_0 dot (ras/s)')

% Plot v(t)
subplot(3,2,2)
plot(t,v);
hold on;
plot(t,v0);
hold off;
title('v');
xlabel('Time (s)');
ylabel('v (m/s)');
legend('v (m/s)','v0 (m/s)')

% Plot u1(t)
subplot(3,2,3)
plot(t,u1);
hold on;
plot(t,u01);
hold off;
title('u left');
xlabel('Time (s)');
ylabel('u_l (Nm)');
legend('u_l (Nm)','u0_l (Nm)')

% Plot u2(t)
subplot(3,2,4)
plot(t,u2);
hold on;
```

```
            plot(t,u02);
            hold off;
            title('u right');
            xlabel('Time (s)');
            ylabel('u_r (Nm)');
            legend('u_r (Nm)','u0_r (Nm)')

            % Plot varphi_p(t)
            varphip=varphip*180/pi;
            subplot(3,2,5)
            plot(t,varphip);
            hold on;
            plot(t,zero);
            hold off;
            title('phi_p');
            xlabel('Time (s)');
            ylabel('phi_p (deg)');

            % Plot varphi_p dot(t)
            subplot(3,2,6)
            plot(t,varphip_dot);
            hold on;
            plot(t,zero);
            hold off;
            title('phi_p dot');
            xlabel('Time (s)');
            ylabel('phi_p dot(rad/s)');


%             % Plot F(t)
%             subplot(3,2,3)
%             plot(t,F);
%             hold off;
%             title('F');
%             xlabel('Time (s)');
%             ylabel('F (N)');

%             % Plot xM(t)
%             subplot(3,2,1)
%             plot(t,xMvec);
%             hold on;
%             plot(t,xMvec0);
%             hold off;
%             title('x_M');
%             xlabel('Time (s)');
%             ylabel('x_M (m)');
%             legend('x_M (m)','x_{M_0} (m)')

%             % Plot u1 i u2(t)
%             subplot(3,2,4)
%             plot(t,u1);
%             hold on;
%             plot(t,u01);
%             hold off;
%             title('u (left and right)');
%             xlabel('Time (s)');
%             ylabel('u(Nm)');
%             legend('u (Nm)','u0(Nm)')

            % Shut down the entire app
            while ~app.quit
                pause(0.1);
            end
            delete(app);
            close all;
            clear;
            fprintf('App closed\n');

        end

        % Button pushed function: QuitButton
        function QuitButtonPushed(app, event)
            app.quit = true;
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            delete(app);
        end

        % Key press function: UIFigure
        function UIFigureKeyPress(app, event)

            global QUIT
            global FORCE
            global TIME_INI
            global TIME_END

            key = event.Key;
            fprintf('key event is: %s\n',key);
```

```
switch key

    case 'uparrow'
        app.v = app.v + 0.5;
    case 'downarrow'
        app.v = app.v – 0.5;
    case 'rightarrow'
        app.om = app.om – deg2rad(15);
    case 'leftarrow'
        app.om = app.om + deg2rad(15);
    case 'space'
        app.v = 0;
        app.om = 0;

    case 'escape'
        QUIT = true;
        app.quit=true;

    case 'f'
        FORCE = 100;            % Disturbance force (N)
        TIME_INI = toc;         % Disturbance initial time (s)
        TIME_END = toc + 0.5;   % Disturbance final time (s)
    case 'g'
        FORCE = –100;           % Disturbance force (N)
        TIME_INI = toc;         % Disturbance initial time (s)
        TIME_END = toc + 0.5;   % Disturbance final time (s)
        end

% Catch slider limits
vmin = app.LinearspeedmsGauge.Limits(1); % m/s
vmax = app.LinearspeedmsGauge.Limits(2); % m/s
omin = deg2rad(app.AngularspeedradsGauge.Limits(1)); % rad/s
omax = deg2rad(app.AngularspeedradsGauge.Limits(2)); % rad/s

% Saturate v to slider limits
app.v = max(app.v,vmin);
app.v = min(app.v,vmax);

% Saturate omega to slider limits
app.om = max(app.om,omin);
app.om = min(app.om,omax);

% Update slider values with app.v and app.om
app.LinearspeedmsGauge.Value = app.v;
```

```
            app.AngularspeedradsGauge.Value = rad2deg(app.om);




        end

        % Callback function
        function VSliderValueChanging(app, event)
            changingValue = event.Value;
            app.v = changingValue;
        end

        % Callback function
        function OmegaSliderValueChanging(app, event)
            changingValue = event.Value;
            app.om = deg2rad(changingValue);
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 504 784];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @
                UIFigureCloseRequest, true);
            app.UIFigure.KeyPressFcn = createCallbackFcn(app, @UIFigureKeyPress,
                true);

            % Create ViewModeButtonGroup
            app.ViewModeButtonGroup = uibuttongroup(app.UIFigure);
            app.ViewModeButtonGroup.Title = 'View Mode';
            app.ViewModeButtonGroup.BackgroundColor = [1 1 1];
            app.ViewModeButtonGroup.Position = [57 490 152 102];

            % Create FirstPersonViewButton
            app.FirstPersonViewButton = uiradiobutton(app.ViewModeButtonGroup);
            app.FirstPersonViewButton.Text = 'First Person View';
            app.FirstPersonViewButton.Position = [11 56 115 22];
```

```
% Create ZenithalViewButton
app.ZenithalViewButton = uiradiobutton(app.ViewModeButtonGroup);
app.ZenithalViewButton.Text = 'Zenithal View';
app.ZenithalViewButton.Position = [11 34 93 22];

% Create ObliqueViewButton
app.ObliqueViewButton = uiradiobutton(app.ViewModeButtonGroup);
app.ObliqueViewButton.Text = 'Oblique View';
app.ObliqueViewButton.Position = [11 12 92 22];
app.ObliqueViewButton.Value = true;

% Create MetersAroundKnobLabel
app.MetersAroundKnobLabel = uilabel(app.UIFigure);
app.MetersAroundKnobLabel.Tag = 'Meters viewed around the robot';
app.MetersAroundKnobLabel.HorizontalAlignment = 'center';
app.MetersAroundKnobLabel.Position = [95 129 85 22];
app.MetersAroundKnobLabel.Text = 'Meters Around';

% Create MetersAroundKnob
app.MetersAroundKnob = uiknob(app.UIFigure, 'continuous');
app.MetersAroundKnob.Limits = [1 50];
app.MetersAroundKnob.Position = [105 181 65 65];
app.MetersAroundKnob.Value = 1.2;

% Create QuitButton
app.QuitButton = uibutton(app.UIFigure, 'push');
app.QuitButton.ButtonPushedFcn = createCallbackFcn(app, @
    QuitButtonPushed, true);
app.QuitButton.Position = [91 39 95 46];
app.QuitButton.Text = 'Quit';

% Create CameraElevationKnobLabel
app.CameraElevationKnobLabel = uilabel(app.UIFigure);
app.CameraElevationKnobLabel.HorizontalAlignment = 'center';
app.CameraElevationKnobLabel.Position = [88 307 100 22];
app.CameraElevationKnobLabel.Text = 'Camera Elevation';

% Create CameraElevationKnob
app.CameraElevationKnob = uiknob(app.UIFigure, 'continuous');
app.CameraElevationKnob.Limits = [0 90];
app.CameraElevationKnob.Position = [105 357 66 66];
app.CameraElevationKnob.Value = 35;

% Create ControlPeriodGaugeLabel
```

```
app.ControlPeriodGaugeLabel = uilabel(app.UIFigure);
app.ControlPeriodGaugeLabel.HorizontalAlignment = 'center';
app.ControlPeriodGaugeLabel.Position = [91 627 83 22];
app.ControlPeriodGaugeLabel.Text = 'Control Period';

% Create ControlPeriodGauge
app.ControlPeriodGauge = uigauge(app.UIFigure, 'semicircular');
app.ControlPeriodGauge.Limits = [0 40];
app.ControlPeriodGauge.MajorTicks = [0 5 10 15 20 25 30 35 40];
app.ControlPeriodGauge.Position = [47 648 171 92];

% Create LinearspeedmsGaugeLabel
app.LinearspeedmsGaugeLabel = uilabel(app.UIFigure);
app.LinearspeedmsGaugeLabel.HorizontalAlignment = 'center';
app.LinearspeedmsGaugeLabel.FontWeight = 'bold';
app.LinearspeedmsGaugeLabel.Position = [308 569 115 22];
app.LinearspeedmsGaugeLabel.Text = ' Linear speed (m/s)';

% Create LinearspeedmsGauge
app.LinearspeedmsGauge = uigauge(app.UIFigure, 'circular');
app.LinearspeedmsGauge.Limits = [−3 3];
app.LinearspeedmsGauge.FontWeight = 'bold';
app.LinearspeedmsGauge.Position = [297 602 137 137];

% Create AngularspeedradsGaugeLabel
app.AngularspeedradsGaugeLabel = uilabel(app.UIFigure);
app.AngularspeedradsGaugeLabel.HorizontalAlignment = 'center';
app.AngularspeedradsGaugeLabel.FontWeight = 'bold';
app.AngularspeedradsGaugeLabel.Position = [303 383 129 22];
app.AngularspeedradsGaugeLabel.Text = 'Angular speed (rad/s)';

% Create AngularspeedradsGauge
app.AngularspeedradsGauge = uigauge(app.UIFigure, 'circular');
app.AngularspeedradsGauge.Limits = [−60 60];
app.AngularspeedradsGauge.FontWeight = 'bold';
app.AngularspeedradsGauge.Position = [299 416 137 137];

% Create Tau_RGaugeLabel
app.Tau_RGaugeLabel = uilabel(app.UIFigure);
app.Tau_RGaugeLabel.HorizontalAlignment = 'center';
app.Tau_RGaugeLabel.Position = [396 188 40 22];
app.Tau_RGaugeLabel.Text = 'Tau_R';

% Create Tau_RGauge
app.Tau_RGauge = uigauge(app.UIFigure, 'linear');
```

```
            app.Tau_RGauge.Limits = [–20 20];
            app.Tau_RGauge.MajorTicks = [–20 –10 0 10 20];
            app.Tau_RGauge.Orientation = 'vertical';
            app.Tau_RGauge.Position = [396 217 40 132];

            % Create Tau_LGaugeLabel
            app.Tau_LGaugeLabel = uilabel(app.UIFigure);
            app.Tau_LGaugeLabel.HorizontalAlignment = 'center';
            app.Tau_LGaugeLabel.Position = [300 188 38 22];
            app.Tau_LGaugeLabel.Text = 'Tau_L';

            % Create Tau_LGauge
            app.Tau_LGauge = uigauge(app.UIFigure, 'linear');
            app.Tau_LGauge.Limits = [–20 20];
            app.Tau_LGauge.MajorTicks = [–20 –10 0 10 20];
            app.Tau_LGauge.Orientation = 'vertical';
            app.Tau_LGauge.Position = [299 217 40 132];

            % Create ExternalForceKnobLabel
            app.ExternalForceKnobLabel = uilabel(app.UIFigure);
            app.ExternalForceKnobLabel.HorizontalAlignment = 'center';
            app.ExternalForceKnobLabel.Position = [325 29 83 22];
            app.ExternalForceKnobLabel.Text = 'External Force';

            % Create ExternalForceKnob
            app.ExternalForceKnob = uiknob(app.UIFigure, 'continuous');
            app.ExternalForceKnob.Limits = [–200 200];
            app.ExternalForceKnob.MajorTicks = [–100 0 100];
            app.ExternalForceKnob.MinorTicks = [–100 0 100];
            app.ExternalForceKnob.Position = [335 85 60 60];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = Twinbot_app_neta_graphs_exported

            % Create UIFigure and components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

A.2.2 Dynamics function

```
function y_dot = ydot_twinbot(t,y,u,pars,param)

% Evaluates the dynamics function ydot = g(y,tau)
% at state y and torque u.
%
% Input:
%
%   t = time instant (scalar)
%   y = (1 x 6) vector of position and velocities
%   u = function handle
%   pars = model paramaters of the twinbot (a tuple)
%
% Output:
%
%   ydot = g(y,tau)
%
global u_val

% Compute actuation torque
u_val = u(t,y);
u_val = min(u_val,10);
u_val = max(u_val,-10);
```

```
% Compute value of the generalised disturbance force
Tau_dist = disturbance(t,y,pars);

% Compute ydot
nq=length(y)/2;
phi=y(1:nq);
phi_dot=y(nq+1:2*nq);

    M = Compute_Mbar_symm(y,u,param);
    C = Compute_Cbar_symm(y,u,param);
    G = Compute_Gbar_symm(y,u,param);
    Tf = Compute_Tf_symm(y,u,param);

Ta = [0 ; u_val ];
Td = Tau_dist;

phi_ddot=M\(Ta+Tf+Td-C*phi_dot-G);

y_dot=[phi_dot;phi_ddot];

end
```

A.2.3 Torque calculation function

```
function tau = tau_wheels(t,y,y0,u0,K)
%
% Evaluates the input torque tau at (t,y). The hardcoded 'kind' variable
% allows to select whether the control law is 'ON', or 'OFF', or
% 'INTERMITTENT' (first ON, then OFF, then ON again).
%

phi = y(1);
phidot = y(2);
phi0 = y0(1);
phidot0 = y0(2);

aerr_p = y(1)-y0(1);          % Angular error of phi_p
aerr_l = y(2)-y0(2);          % Angular error of phi_l
aerr_r = y(3)-y0(3);          % Angular error of phi_r

verr_p = y(4)-y0(4);          % Velocity error of phidot_p
verr_l = y(5)-y0(5);          % Velocity error of phidot_l
verr_r = y(6)-y0(6);          % Velocity error of phidot_r

ctl_action = u0 - K * [aerr_p; aerr_l; aerr_r; verr_p ; verr_l ; verr_r];
```

```
kind = 'ON';

switch kind

    case 'ON'
        % Control law always on
        tau = ctl_action;

    case 'OFF'
        % Control law always off
        tau = 0;

    case 'INTERMITTENT'
        % Control is ON for t<3 and t>4 only
        tau = (t<3|t>4)*ctl_action;

    otherwise
        % Default mode is no control
        tau = 0;

end

end
```

A.2.4 Disturbance force

```
function Tau_dist = disturbance(t,y,pars)

global FORCE
global TIME_INI
global TIME_END

% Current angle of pendulum
varphi_p = y(1);

% Pars needed
l = pars.zG;
r = pars.r;


% Unfold parameters of disturbance force
phi = varphi_p;     % Angle of disturbance force [rad] (hardcoded here)
f = FORCE;          % Magnitude of disturbance force [N]
tdi = TIME_INI;     % Initial time of disturbance force
```

```
tdf = TIME_END;     % Final time of disturbance force

% Compute value of generalised disturbance force
Tau_dist = (t>tdi & t<tdf) * f * [r+l*cos(phi);r/2;r/2] ;
%FORCE = (t>tdi & t<tdf) * FORCE;

end
```

A.2.5 Torque calculation function

```
function [A,B] = AB_cdiff(y0,u0,param)

    epsilon = 1e-6;

    % --- A ---
    A = zeros(6,6);
    incry = zeros(6,1);
    for i=1:6
        incry(i,1) = epsilon;
        Fincr_forw = ydot_twinbot_linealitzacio(y0+incry, u0, param);
        Fincr_back = ydot_twinbot_linealitzacio(y0-incry, u0, param);
        A(:,i) = (Fincr_forw-Fincr_back) / (2*epsilon);
        incry(i,1) = 0;
    end

    % --- B ---
    B = zeros(6,2);
    incru = zeros(2,1);
    for i=1:2
        incru(i,1) = epsilon;
        Fincr_forw = ydot_twinbot_linealitzacio(y0, u0+incru, param);
        Fincr_back = ydot_twinbot_linealitzacio(y0, u0-incru, param);
        B(:,i) = (Fincr_forw-Fincr_back) / (2*epsilon);
        incru(i,1) = 0;
    end

end
```

A.2.6 Dynamics linealization function

```
function y_dot = ydot_twinbot_linealitzacio(y,u,param)

% Compute ydot
nphi=length(y)/2;
phi=y(1:nphi);
phi_dot=y(nphi+1:2*nphi);
```

```
    M = Compute_Mbar_symm(y,u,param);
    C = Compute_Cbar_symm(y,u,param);
    G = Compute_Gbar_symm(y,u,param);
    Tf  = Compute_Tf_symm(y,u,param);

Ta=[0 ; u];

% Compute Fcal
Fcal = Ta + Tf — C * phi_dot — G;

% Compute the acceleration qddot of the system
phi_ddot = M\Fcal;


y_dot=[phi_dot;phi_ddot];

end
```

A.2.7 URDF file

```
<robot name = "diffbot">

        <!-- ============== -->>
        <!-- links section  -->>
        <!-- ============== -->>

        <link name = "link ground">

                <!--
                <visual>
                        <origin xyz = "0 0 -0.05" />
                        <geometry>
                                <box size = "4 4 0.1" />
                        </geometry>
                        <material name = "white">
                                <color rgba = "1  1  1  1" />
                        </material>
                </visual>
                -->>

        </link>

        <link name = "link x">
        </link>

        <link name = "link y">
        </link>

        <link name = "link axis">

                <visual>
                        <origin xyz = "0 0 0" rpy= "1.57079 0 0" />
                        <geometry>
                                <cylinder radius = "0.02" length = "0.65"  />
                        </geometry>
                        <material name = "red">
                                <color rgba = "1  1  1  1" />
                        </material>
                </visual>


        </link>

        <link name = "link chassis">

                <visual name = "chassis base">
                        <origin xyz = "0 0 0" />
                        <geometry>
                                <box size = "0.40 0.51 0.1" />
                        </geometry>
                        <material name = "grey">
                                <color rgba = "0.5 0.5 0.5 1" />
                        </material>
                </visual>

                <visual name = "chassis cylinder">
                        <origin xyz = "0 0 0.25" rpy= "0 0 0" />
        <geometry>
            <cylinder radius = "0.15" length = "0.50"  />
        </geometry>
        <material name = "light grey">
            <color rgba = ".8 .8 .8 1" />
        </material>
    </visual>


</link>

<link name = "link right wheel">
    <visual name = "right wheel tire">
        <origin xyz = "0 0 0" rpy= "1.57079 0 0" />
        <geometry>
            <cylinder radius = "0.150" length = "0.050"  />
        </geometry>
        <material name = "red">
            <color rgba = "1 0 0 1" />
        </material>
    </visual>


    <visual name ="right wheel radius exterior">
        <origin xyz = "0 -0.022 0.110" rpy="0 0 0" />
        <geometry>
            <sphere radius = "0.020"/>
        </geometry>
        <material name = "white">
            <color rgba = "1 1 1 1" />
        </material>
    </visual>

</link>

<link name = "link left wheel">

    <visual name = "left wheel tire">
        <origin xyz = "0 0 0" rpy="1.57079 0 0" />
        <geometry>
            <cylinder radius = "0.150" length = "0.050"  />
        </geometry>
        <material name = "red">
            <color rgba = "1 0 0 1" />
        </material>
    </visual>

    <visual name ="left wheel radius exterior">
        <origin xyz = "0 0.022 0.110" rpy="0 0 0" />
        <geometry>
            <sphere radius = "0.020"/>
        </geometry>
        <material name = "white">
            <color rgba = "1 1 1 1" />
        </material>
    </visual>
```

```
        </link>

        <!-- ============== -->>
        <!-- joints section -->>
        <!-- ============== -->>

        <joint name = "joint x" type = "prismatic">
                <parent link = "link ground" />
                <child link = "link x" />
                <origin xyz = "0 0 0.150" rpy="0 0 0"/>
                <axis xyz = "1 0 0" />
                <limit effort="3000" velocity="100" lower="-10000" upper="10000" />
        </joint>

        <joint name = "joint y" type = "prismatic">
                <parent link = "link x" />
                <child link = "link y" />
                <origin xyz = "0 0 0" rpy="0 0 0"/>
                <axis xyz = "0 1 0" />
                <limit effort="3000" velocity="100" lower="-10000" upper="10000" />
        </joint>

        <joint name = "joint theta" type = "continuous">
                <parent link = "link y" />
                <child link = "link axis" />
                <origin xyz = "0 0 0" rpy="0 0 0"/>
                <axis xyz = "0 0 1" />
        </joint>

        <joint name = "joint phi_p" type = "continuous">
                <parent link = "link axis" />
                <child link = "link chassis" />
                <origin xyz = "0 0 0" rpy="0 0 0"/>
                <axis xyz = "0 1 0" />
        </joint>

        <joint name = "joint phi_l" type = "continuous">
                <parent link = "link chassis" />
                <child link = "link left wheel" />
                <origin xyz = "0 0.280 0" rpy="0 0 0"/>
                <axis xyz = "0 1 0" />
        </joint>

        <joint name = "joint phi_r" type = "continuous">
                <parent link = "link chassis" />
                <child link = "link right wheel" />
                <origin xyz = "0 -0.280 0" rpy="0 0 0"/>
                <axis xyz = "0 1 0" />
        </joint>


</robot>
```

B
Project budget

The budget presented below evaluates the cost of developing and documenting the software implemented in this project, both in human and material resources.

B.1 Assumptions

In this section we specify the hourly costs we assume in this budget. We consider two types of costs:

- The hourly cost of the staff devoted to develop the project.
- The hourly cost of using the equipment.

B.1.1 The hourly cost of the staff

We assume the following salaries:

Analyst	⟶	20	EUR/hour
Programmer	⟶	15	EUR/hour
Operator	⟶	12	EUR/hour

B.1.2 Hourly cost of using the equipment

For the assessment of the hourly cost of using the equipment it is necessary to take into account: its amortization, the cost of the personnel of the Institut de Robòtica i Informàtica Industrial devoted to take care of the equipment, and maintenance contracts, if any.

The only equipment used in this project is a computer with the following associated costs:

1. **Equipment cost:** The used computer is an HP ProDesk 400 G4 PC tower, with 7200 rpm SATA hard drive, 1TB 3.5", Intel HD Graphics 630 board, Realtek RTL8111 HSH GbE LOM network card, 2 USB 3.1 Gen1 ports, 4 USB ports 2.0, 3.4GHz-3.8Ghz Intel Core i5-7500 CPU, 8Gb DDR4-2400 (1×8Gb) SDRAM, HP USB Business Compact Keyboard, HP USB Optical Mouse, 19.5"HP 20kd Monitor (1440x900 at 60Hz), and Windows 10 operating system. The cost of this computer is approximately 750 EUR. This amount also includes the

installation costs of the machine. If we consider that we wish to amortize the equipment in five years, with an interest rate of 15%, the yearly amortization cost is as follows:

$$C(\text{equipment}) = 750 \times \frac{0.15 \times 1.15^5}{1.15^5 - 1} = 223,74 \simeq 224 \text{ EUR/year}$$

2. **Maintenance:** The annual maintenance cost of the complete equipment is assumed to be 10% of its purchase price:

$$C(\text{maintenance}) = 750 \times 0.1 = 75 \text{ EUR/year}$$

3. **Power consumption:** Its yearly cost is assumed to be

$$C(\text{consumption}) = 100 \text{ EUR/year}$$

4. **Cost of the IRI staff:** We consider the costs of an operator working 0.5 hours per week for 40 weeks each year:

$$C(\text{IRI staff}) = 240 \text{ EUR/year}$$

Based on the above partial costs, the total annual cost will be:

C(equipment)	224 EUR/year
C(maintenance)	75 EUR/year
C(consumption)	100 EUR/year
C(IRI staff)	240 EUR/year
C(annual)	639 EUR/year

The hourly cost of CPU usage, considering that the equipment is working 8 hours/day, 5 days/week, 40 weeks/year, and that the CPU is working at 60% of its load on average, is:

$$C(\text{CPU per hour}) = \frac{639}{8 \times 5 \times 40 \times 0.6} = 0.67 \text{ EUR/hour}$$

B.2 Development cost

Taking into account the prices stipulated in the previous section, we next calculate the cost of developing the project, which includes the cost of the project staff, the equipment, and documentation and printing expenses. It should be noted that the costs related to the fixed assets of IRI are not taken into account. We only count those for the development of the project in terms of human resources and computer equipment.

B.2.1 Project staff cost

We estimate that the time invested in the project has been 21 weeks with a full dedication of 15 hours/week. The project involved a person earning the salary of a computer analyst:

$$C(\text{project staff}) = 315 \text{ hours} \times 20 \text{ EUR/hour} = 6,300 \text{ EUR}$$

B.2.2 Cost of using the equipment

We estimate that 60% of the total project time has been devoted to implementing the programs. During that time, however, the CPU has been inactive during 35% of the time approximately. We also consider that, when the CPU is active, we actually use 80% of it (despite the PC allows multiple users to be connected, the usage regime has been practically that of a single user). Therefore, we calculate the following times:

$$\begin{array}{lrcr} \text{Computer time} & 315 \times 0.60 & = & 189 \text{ hours,} \\ \text{Useful computer time} & 189 \times 0.65 & \simeq & 123 \text{ hours,} \\ \text{CPU time} & 123 \times 0.80 & \simeq & 98 \text{ hours,} \end{array}$$

so the CPU cost is

$$C(\text{CPU}) = 98 \times 0.67 = 66 \text{ EUR}.$$

B.2.3 Documentation costs

We here consider the costs of bibliographic sources acquired during the project, which include a book and two journal articles, and those of printing:

C(bibliography)	90.00 EUR
C(printing)	15.00 EUR
C(documentation)	105.00 EUR

B.2.4 Licence costs

We are going to consider the costs of the software used during the project.

C(MATLAB)	840 EUR
C(Creo Elements)	2,118 EUR
C(Licenses)	2,958 EUR

B.2.5 Total cost of development

The total cost of developing the project is finally the sum of the aforementioned costs:

C(project staff)	6,300 EUR
C(CPU)	66 EUR
C(documentation)	105 EUR
C(Licences)	2,958 EUR
C(TOTAL)	9,429 EUR

C
Environmental impact

All research projects should always take into account the environmental impact that have generated. In our case, our project did not have a practical approach, no prototype was developed and no waste was caused , therefore, the carbon footprint generated is minimal, and is only caused by the electrical power consumed by the computer and the room's lightning. The environmental impact of this project will be quantified with the $CO_2\,kg$ emitted by the electricity that has been consumed.

C.1 $CO_2\,kg$ generated

In the annex B, we estimated that this project signified 315 working hours, in which we can consider we had a full time working computer. Moreover, we could estimate that half of this time we needed external lightning in order to work. For the electricity calculations, we will define there were 4 LED light bulbs in the room in order to cover the illumination.

The people form *Sibelga* [18] have studied the $CO_2\,kg$ consumed by different kind of computers, in our case we alternated between a desktop and a laptop, but programs used were quite powerful, therefore, we are going to consider we were using a desktop all the time (for calculation purposes), which has a higher consumption. They say that a computer working for eight hours a day uses almost $600kWh$, and emits $175kg$ of CO_2 every year.

$$\text{Environmental impact of the computer} = \frac{315 \times 175}{8 \times 365} = 18.87\,kg\ of\ CO_2$$

In order to estimate the environmental impact of the illumination, we will take a look at the analysis made by *Towers IT Professional Services Ltd* [19]. From there, we know that a standard LED, with coal based power generation, consumes $22.23\,CO_2\,kg$ every year if is working 8 hours a day. And we know that every coal based kWh of electricity, generates $870\,CO_2\,kg$, whereas if the source is natural gas, which was our case, it generates $487\,CO_2\,kg$. With this data on the table, powering 4 LED would imply:

$$\text{Environmental impact of the illumination} = \frac{315 \times 22.23 \times 487 \times 4}{2 \times 8 \times 365 \times 870} = 0,27\,kg\ of\ CO_2$$

The carbon foot-print generated by the project is therefore:

Environmental impact of the computer	$18.87\,kg\ of\ CO_2$
Environmental impact of the illumination	$0,27\,kg\ of\ CO_2$
Total environmental impact	$19.13 \simeq 19\,kg\ of\ CO_2$

In this study we have seen that the environmental impact of the project is very low, which should make us proud, but our future goal must be to reduce it into 0. For this reason we still have to improve as a society in order to make all the electricity come from renewable sources if we want to significantly minimize the carbon foot-print generated.