SAPIENZA
UNIVERSITÀ DI ROMA

# Control and design of a two-wheeled balancing mobile robot with actuated legs


Faculty of Civil and Industrial Engineering

Corso di Laurea Magistrale in Master of Science in Mechanical Engineering

Candidate

Andrea Patrizi
ID number 1742937

Thesis Advisor

Prof. Leonardo Lanari

Co-Advisor

Phd. Filippo Maria Smaldone



Academic Year 2021/2022

Thesis not yet defended

---

**Control and design of a two-wheeled balancing mobile robot with actuated legs**
Master's thesis. Sapienza – University of Rome



This thesis has been typeset by LaTeX and the Sapthesis class.

Version: October 22, 2021

Author's email: patrizi.1742937@studenti.uniroma1.it

# Abstract

This thesis was born and inteded as a way of exploiting all the competencies I acquired during the years spent at my university and, more importantly, as a way of challenging myself by facing topics on which I previously had limited knowledge and proficiency. It was also a way of exerting my maker side and expand my experience on control and power electronics, as well as on machining tools, 3D printing, etc. . I ended up learning a lot, more than I could have imagined when I started it and more than any conventional thesis can provide a student with.

Due to the prototyping aspect of this thesis (which, in my opinion, may be righteously addressed as a project) I had to invest conspicuous amounts of resources in improving what, at the time, was a small home laboratory; a relevant fraction of the spent resources was also used to purchase all the necessary components. Fortunately, I had some money coming from a tutoring activity at my faculty, so I was happy to invest it in my project.

Now that I am near the end line, I can say that I probably underestimated the amount of work, study, resources and stress that such a thesis would entail. Nonetheless, even if I ended up not achieving all my objectives, I do not regret it at all.

# Acknowledgments

*First, I have to thank my thesis advisor Prof. Leonardo Lanari who had a crucial role in guiding, supporting and directing the work of my thesis.*
*My gratitude also goes to Ph.D. Filippo Smaldone, who provided me with substantial support throughout the whole thesis by giving me many advices and directions on control aspects and also by helping me in the task of setting up Gazebo simulations.*
*Special thanks to Marco Kanneworff, whose work served as a crucial reference throughout this thesis.*
*I also have to thank my family and friends for having supported me in various ways during this thesis.*

*There are also a huge amount of sites, wiki, documentation pages, CAD repositories etc. I had to consult and that helped me during my endless work of documentation and learning. I have reported some of these resources in the bibliography. My gratitude goes to all the people working on these resources whose main merit is to make them available to everybody.*

# Contents

# Chapter 1

# Thesis structure overview

This thesis is structured in the following way:

- Chapter 2 is a brief review of the current state of the art.
- Chapter 3 contains all the used dynamics/kinematics models and a synthetic description of their derivation.
- Chapter 4 expands upons the simulation environments used to test and validate all the explored control strategies.
- Chapter 5 is dedicated to the analysis of models derived in Chapter 3 from a control perspective.
- Chapter 6 discusses the implementation and test results of various control policies to perform trajectory tracking.
- Chapter 7 contains all the details entailing the design, prototyping and implementation aspects of the thesis and the current state of the prototype.
- Chapter 8 sums up the main contents of this thesis and also highlights some of the possible future developments.

# Chapter 2

# State of the art

Wheeled balancing robots have very peculiar characteristics which make them particularly suited for navigation on flat surfaces (for example in urban environments). Compared to classical legged robot, for instance quadrupeds, they generally show a higher degree of agility, maneuverability and also efficiency, if used on flat surfaces. Legged system, on the contrary, are particularly suited when navigating harsh environments.

The system under analysis, i.e. a wheeled balancing robot equipped with an actuated knee, is somewhat a hybrid between the two classes of robots. It retains and extends all the agility and maneuverability of a simple wheeled balancing system (e.g. Segway), while potentially also allowing for more flexibility in navigating difficult surfaces, like stairs.

During last years, several hardware platforms following this concept were implemented ([19,23,44] are examples of such platforms). In particular, [19] represents an important reference for this work, particularly for the design aspect (the prototype is visible in Fig. 2.1). Another example is given by Boston Dynamics' Handle, shown in Fig. 2.2.

The systems under analysis have some peculiar dynamical characteristics which make them particularly challenging from the control point of view: *underactuation*, *instability* and *nonlinearity* of the dynamics. Chapter 5 gives some insights into these properties and also allows for the reader to comprehend why designing controls for biped wheeled systems is quite challenging (yet stimulating). In particular, a very helpful tool for comprehending the properties of such systems is the *second order normal form*, which will be derived in Chapter 5. This form is dependent upon the chosen output for the system and highlights the so called *forced internal dynamics*. Starting from the internal dynamics, the so called *zero dynamics* is obtained by simply setting the output and its derivative identically to zero. Systems with unstable zero dynamics are called *non-minimum phase* and are particularly difficult to handle from the control point of view. This characteristic makes performing trajectory tracking, which is the main focus of Chapter 6, quite challenging.

Several different control frameworks for velocity tracking and balancing have been developed; examples include LQR ([19]) and PID controllers. Pairing those controllers with a suitable planner allows to perform navigation tasks. However, as stresses by [8], many of those approaches fail to properly address the planning

problems and, as a consequence, end up limiting the range of motion of the robot.

Part of Chapter 5 is dedicated to tackling this planning problem to perform trajectory tracking via a modified LQR controller. Mainly due to the underactuation property of the dynamics, the problem of planning becomes crucial. As will be pointed out in Chapter 5, an underactuated system cannot realize arbitrary state trajectories and hence particular care needs to be taken in generating dynamically feasible reference state trajectories and control inputs (depending on the control scheme under use); being the system non-minimum phase, this has to be done while avoiding the internal dynamics divergence.

A so called partial feedback linearization approach was widely adopted on underactuated systems ([41, 42]). An example of this approach, inserted in an IS-MPC framework (see [36]) and applied to wheeled balancing systems is contained in [17]. In particular, using an MPC framework is highly beneficial, since it is also allows to impose constraints on the optimization variables and it naturally adapts to real-time implementations, due to its intrinsic discreteness.

**Figure 2.1.** Ascento (see [19]): two wheeled balancing robot capable of navigation and jumping motion.

**Figure 2.2.** Latest version of Boston Dynamics' Handle. It is a two wheeled balancing robot with an actuated arm for handling packages.

# Chapter 3

# Models description and derivation

This chapter is dedicated to the modeling aspects of the project/thesis.

Several different kinematic and dynamics models were derived throughout the thesis and this chapter is intended as an ordered way of referencing those models. Separating the modeling aspects from the control ones allows for a much clearer comprehension and readability of the thesis. Furthermore, the models are here reported in the chronological order they were derived and used.

## 3.1 Modeling approach, used symbols and conventions

A Lagrangian approach was chosen for deriving the dynamic equations.

Throughout this work, generalized state coordinates of the system under study will be collected and addressed using the state vector $q(t)$, while Lagrangian coordinates will be denoted by $q_p(t)$. The meaning of its elements, if not explicitly specified, can be directly deduced from the context of the model being discussed or used. For this reason, this notation will be purposely overloaded when discussing different models.

For the modeling part, a right handed stationary coordinate system was used. The way this coordinate frame is placed is clearly intelligible from the various pictures inserted in this chapter.

To avoid confusion, note that the operator $\nabla_{var}()$, i.e. the gradient of the input function w.r.t. the vector variable $var$, returns a row vector in the case of scalar functions. As a consequence, in the case of a vectorial function, the operator returns a matrix, with each row representing the gradient of the single elements. Keeping this in mind, matrix operations with this operator can be performed consistently.

Reference quantities, when needed, will be denoted by an superscript $^{ref}$.

For referencing moments of inertia symbol $I_{ij}$ will be used, with the axes specified by the subscripts; this way confusion with Jacobians or cost functions (indicated using the letter $J$) is avoided.

All the dynamic models presented in this chapter were verified and derived with

the aid of Matlab's Symbolic Math Toolbox. These models are made available for reading and consulting in dedicated scripts.

## 3.2 Leg kinematic planar model and dimensional synthesis

The first important step towards the development of a dynamical model of the system is to choose a structure for the legs and model their kinematics.

The choice of using a four bar linkage (as done by [19]) is supported by the following simple considerations:

- More leg d.o.f. allow for more complex dynamical motion. However, more d.o.f are also more challenging to handle from the control point of view and, most importantly, they require more actuators, which in turn are source of additional weight and cost. At the price of limiting more complex dynamics, using a single d.o.f. leg seems the most reasonable and balanced choice.

- Once the decision on the number of d.o.f for the leg is made, the next step is to select a suitable link arrangement to realize the leg kinematics. Using two links leaves too many d.o.f.s free, using three links leaves no d.o.f. at all. The first viable way is to employ a four bar linkage, since it leaves only one d.o.f to the leg. Other options with more links would be possible, at the price of more kinematic complexity (and also weight, since the number of links increases).

Next, once the desired kinematics is chosen, the dimensional synthesis of the linkage needs to addressed properly. This aspect will be discussed in Section 3.2.2.

### 3.2.1 Leg kinematic model

A schematic of the leg mechanism is shown in Fig. 3.1.

In the picture, a local reference frame $x^{'}z^{'}$ is defined and also all the dimensional parameters of the linkage are represented. In particular, for naming the links, the notation $L_i$ will be used, where $i$ is a unique identifier, while the associated dimensions will be addressed as $l_i$. From now on, for simplicity, link names will be omitted from the upcoming pictures and only the associated dimensions will be shown, if necessary. Note that link $L_b$ actually belongs to the robot body. Links $L_a, L_h, L_c$ and $L_d$ are actual links of the robots. For the sake of clarity, a schematization of the wheel is also depicted (red disk).

Looking at the linkage, the following trigonometric relationships can be written (the first two represent a vectorial relationship on the links, while the last ones are the fundamental trigonometric relationships necessary to close the system):

$$\begin{cases} l_a \cos\alpha + l_c \sin\gamma = l_b - l_d \cos\beta \\ l_a \sin\alpha + l_c \cos\gamma = l_d \sin\beta \\ \cos^2\alpha + \sin^2\alpha = 1 \\ \cos^2\gamma + \sin^2\gamma = 1 \end{cases} \tag{3.1}$$

To solve for $\gamma(\beta)$ and $\alpha(\beta)$, let us define the following $\beta$-dependent coefficients:

$$\begin{cases} A(\beta) \coloneqq \dfrac{l_a^2 - l_b^2 - l_c^2 - l_d^2 + 2\,l_b\,l_d\cos\beta}{2\,l_c\,(l_d\cos\beta - l_b)}\,; & B(\beta) \coloneqq \dfrac{l_d\sin\beta}{l_d\cos\beta - l_b} \\ C(\beta) \coloneqq \dfrac{-l_a^2 - l_b^2 + l_c^2 - l_d^2 + 2\,l_b\,l_d\cos\beta}{2\,l_a\,(l_d\cos\beta - l_b)}\,; & D(\beta) \coloneqq B(\beta) \end{cases} \tag{3.2}$$

For simplicity, from now on, the dependence of $A, B, C, D$ on $\beta$ will be omitted.

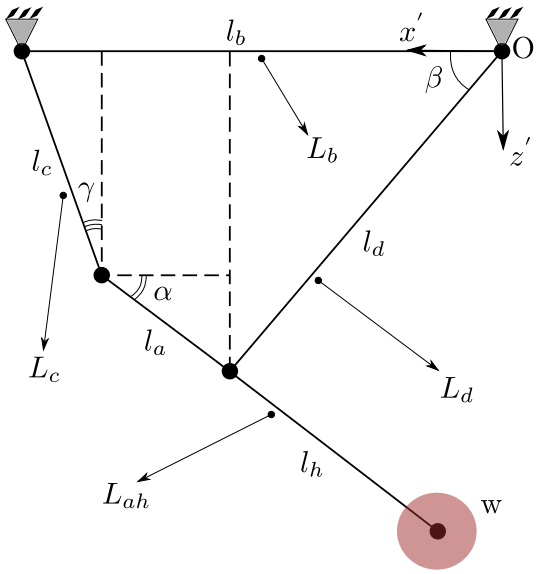



**Figure 3.1.** Linkage schematic

By rearranging the first and second equations of (3.1), the system can be rewritten as

$$\begin{cases} \sin\gamma = A(\beta) + B(\beta)\cos\gamma \quad \wedge \quad \cos^2\gamma + \sin^2\gamma = 1 \\ \cos\alpha = C(\beta) + D(\beta)\sin\alpha \quad \wedge \quad \cos^2\alpha + \sin^2\alpha = 1 \end{cases} \tag{3.3}$$

which leads to the following second order polynomial equations in, respectively, $\cos\gamma$ and $\sin\alpha$:

$$\begin{cases} (1 + B^2)\cos^2\gamma + 2\,A\,B\cos\gamma + (A^2 - 1) = 0 \\ (1 + D^2)\sin^2\alpha + 2\,C\,D\sin\alpha + (C^2 - 1) = 0 \end{cases} \tag{3.4}$$

A possible solution algorithm is to solve the first equation w.r.t. $\cos\gamma$:

$$\cos\gamma = \frac{-A\,B \pm \sqrt{1+B^2-A^2}}{1+B^2} \tag{3.5}$$

From (3.3):

$$\sin\gamma = A + \frac{B}{1+B^2}\left(-A\,B \pm \sqrt{1+B^2-A^2}\right) \tag{3.6}$$

Going back to (3.1) and substituting (3.5) and (3.6):

$$\cos\alpha = \frac{l_b}{l_a} - \frac{l_d}{l_a}\cos\beta - \frac{l_c}{l_a}\left[A + \frac{B}{1+B^2}\left(-A\,B \pm \sqrt{1+B^2-A^2}\right)\right] \tag{3.7}$$

$$\sin\alpha = \frac{l_d}{l_a}\sin\beta - \frac{l_c}{l_a}\left[\frac{-A\,B \pm \sqrt{1+B^2-A^2}}{1+B^2}\right] \tag{3.8}$$

Note that, alternatively, it would have been possible to solve for $\alpha$ first using the second equation of (3.4) and then use the other relationships to obtain $\gamma$. Also note that, $\forall\,\beta$ assigned, there are potentially two admissible pairs $(\alpha,\gamma)$ satisfying (3.1). This has a very straightforward geometric interpretation (see Fig. 3.1).

One possible way of resolving such kinematic redundancy is to search for solutions satisfying the inequality $-\pi/2 \leq \alpha+\gamma \leq \pi/2$, which clearly corresponds to cases in which the knee doesn't inflect towards the internal area of the quadrilateral linkage.

Furthermore, a mathematical singularity occurs when $l_d\cos\beta = l_b$; however, this singularity can be removed by solving again starting from the original system and imposing $l_b - l_d\,cos\,\beta = 0$. This singularity can be neglected by simply choosing $l_d < l_b$. If this is not possible due to dimensional constraints, care must be used to ensure that the mathematical model is robust w.r.t. the singularity occurrence. This singularity corresponds, as it will be shown in the following pages, to a change of knee convexity. This procedure was not carried out since, for the vast majority of useful dimensional synthesis, this singularity never happens.
Another singularity occurs when $1+B^2-A^2$ becomes negative. Imposing $1+B^2-A^2 \geq 0$ leads to a second order polynomial trigonometric expression which allows to find an admissible $\beta$ range of values. Since the expressions are not particularly simple and compact, these are not reported here.

Once the linkage is solved, it is possible to express the coordinates of the wheel center w.r.t. the local frame of the linkage.
These coordinates, expressed in the local reference frame $x'y'$ (shown in Fig. 3.1), are

$$x'_w = l_d\cos\beta - l_h\cos\alpha \tag{3.9}$$

$$z'_w = l_d\sin\beta + l_h\sin\alpha \tag{3.10}$$

Inserting (3.7) and (3.5) in, respectively, (3.9) and (3.10) leads to:

$$x'_w = l_d\cos\beta - l_h\left\{\frac{l_b}{l_a} - \frac{l_d}{l_a}\cos\beta - \frac{l_c}{l_a}\left[A + \frac{B}{1+B^2}\left(-A\,B \pm \sqrt{1+B^2-A^2}\right)\right]\right\} \tag{3.11}$$

$$z_w^{'} = l_d \sin\beta + l_h \left\{ \frac{l_d}{l_a} \sin\beta - \frac{l_c}{l_a} \left[ \frac{-A\,B \pm \sqrt{1+B^2-A^2}}{1+B^2} \right] \right\} \tag{3.12}$$

Let us compute the following expressions:

$$\begin{pmatrix} \cos(\alpha+\gamma) \\ \sin(\alpha+\gamma) \end{pmatrix}_1 = \begin{pmatrix} \frac{(l_b - l_d\cos(\beta))\sqrt{\frac{l_d^2\sin(\beta)^2}{(l_b-l_d\cos(\beta))^2} - \frac{\left(-l_a^2+l_b^2-2\cos(\beta)\,l_b\,l_d+l_c^2+l_d^2\right)^2}{4\,l_c^2\,(l_b-l_d\cos(\beta))^2}+1}}{l_a} \\ -\frac{l_a^2+2\cos(\beta)\,l_b\,l_d+l_c^2-l_b^2-l_d^2}{2\,l_a\,l_c} \end{pmatrix} \tag{3.13}$$

$$\begin{pmatrix} \cos(\alpha+\gamma) \\ \sin(\alpha+\gamma) \end{pmatrix}_2 = \begin{pmatrix} \frac{-(l_b - l_d\cos(\beta))\sqrt{\frac{l_d^2\sin(\beta)^2}{(l_b-l_d\cos(\beta))^2} - \frac{\left(-l_a^2+l_b^2-2\cos(\beta)\,l_b\,l_d+l_c^2+l_d^2\right)^2}{4\,l_c^2\,(l_b-l_d\cos(\beta))^2}+1}}{l_a} \\ -\frac{l_a^2+2\cos(\beta)\,l_b\,l_d+l_c^2-l_b^2-l_d^2}{2\,l_a\,l_c} \end{pmatrix} \tag{3.14}$$

Where the subscripts refer, respectively, to the first (+ sign) and second solution (recall that there exists a kinematic redundancy).

When solving the robot's dynamics, it is necessary to know which solution needs to be used for computing the leg's position. Since redundancy is exactly associated with knee convexity, it is sufficient to determine which expression of the two corresponds to the desired knee convexity. Once this is done, the corresponding kinematic solution can be used to determine the leg position. We know $-\pi/2 \leq \alpha+\gamma \leq \pi/2$ corresponds to a concave internal knee angle; this implies $\cos(\alpha+\gamma) \geq 0$. Looking at the first components of (3.13) and (3.14) it is clear that the first solution is the one preserving the required convexity of the knee. As a consequence, this is the solution to be used.

This is actually true, to be more precise, only if $l_b - l_d\cos(\beta) \geq 0$ which proves to be a legit assumption for most conceivable dimensional synthesis of the linkage and usable $\beta$ ranges.
Looking at (3.13) and (3.14) it is also clear how those solutions collapse onto each other when $l_b - l_d\cos(\beta) = 0$.

All this considerations will be clear in Section 3.2.2, where the optimization used for the dimensional synthesis of the linkage will be discussed.

### 3.2.2 Leg dimensional synthesis

Having carried out the kinematic analysis of the leg, it is now possible to face the problem of determining a suitable dimensional synthesis for all links.

This synthesis can be driven by a simple physical principle: if the local cartesian wheel trajectory in the $x^{'}z^{'}$ reference frame (as a function of the leg angle $\beta$) was linear w.r.t. the robot body and passed through the total center of mass (from here on addressed as CoM) of the robot, a leg motion would produce, in principle, relatively small perturbation of the vertical equilibrium. In other words, the robot's vertical dynamics would be locally decoupled w.r.t. the horizontal one (*only* during

vertical balancing). Note that, under such assumptions, this behaviour would in general be true only for small leg excursions. In fact, as will be seen in Section 3.2.2, even if the wheel trajectory can be considered approximately linear, the leg linkage possesses nonetheless a nonlinear trajectory of its CoM.

However, as shown by [15], the end-point trajectory of a four bar linkage cannot be made exactly linear, since it can intersect a line, at most, a finite number of times. This is somewhat a downside of the chosen design; however, as will be seen in the following discussion, the local cartesian wheel trajectory in the $x^{'}z^{'}$ reference frame can be made linear with very good approximation.

Note also that this design requirements would not necessarily be needed since, in theory, the complete knowledge of the system's kinematics and dynamics would potentially be sufficient to be able to control it. This approach is only a way of efficiently designing the leg to make the job of a controller easier.

As already remarked at the beginning of the chapter, the guiding principle of the dimensional synthesis is to make the trajectory of the wheel w.r.t. to $x^{'}z^{'}$ local frame as linear as possible. This, however, cannot be sufficient: it is clear that a trajectory almost perfectly linear but very short makes little or no sense at all. As a consequence, another guiding principle is that the wheel trajectory needs to be long "enough", where the meaning of trajectory length and "enough" still needs to be defined rigorously.
Let

$$z^{'} = e_t\, x^{'} + f_t \tag{3.15}$$

be the linear approximation of the wheel trajectory, in a least-squares sense, in the defined local reference frame.

The dimensional synthesis has a total of seven parameters, which for simplicity are collected in the following vector

$$\mathcal{D} \coloneqq [l_a, l_b, l_c, l_d, l_h, e_t, f_t] \tag{3.16}$$

This vector contains, respectively, all link dimensions, the angular coefficient and intercept of the least-squares linear approximation of the local trajectory.

The approach used here is to build a *nonlinear* objective function and solve the associated nonlinear optimization program. To solve it, Matlab's nonlinear programming solver *fmincon* was used.

Let us start building the first part of the cost function.
The versor normal to the linear approximation of the trajectory has the following expression:

$$\hat{n} = \frac{1}{\sqrt{(1+e_t^2)}}\,[-e_t,\, 1]$$

The knowledge of this versor allows to write the squared distance from the approximate trajectory line as

$$d^2_{lin}(\mathcal{D}, \beta) = \frac{\left[e_t\, x^{'}_w(l_a, l_b, l_c, l_d, l_h, \beta) + f_t - z^{'}_w(l_a, l_b, l_c, l_d, l_h, \beta)\right]^2}{e_t^2 + 1}$$

The first term of the objective function becomes

$$\int_{\beta_{min}}^{\beta_{max}} d_{lin}^2(\mathcal{D}, \beta)\, d\beta$$

where $\beta_{min}$ and $\beta_{max}$ are the extremes of the selected $\beta$ range of values. These are chosen and fixed before the optimization.

For simplicity, the dependence of $x_w^{'}$ nd $z_w^{'}$ on $(l_a, l_b, l_c, l_d, l_h, \beta)$ will be omitted where it is not needed.

Then, it is necessary to add at least another term for maximizing the trajectory length.
First, let us establish a definition of the trajectory length (squared) in the following way

$$d_{length}^2(\mathcal{D}) = \left[x_w^{'}(\beta_{max}) - x_w^{'}(\beta_{min})\right]^2 + \left(e_t\, x_w^{'}(\beta_{max}) + f_t - e_t\, x_w^{'}(\beta_{min}) - f_t\right)^2 =$$

$$= \left[x_w^{'}(\beta_{max}) - x_w^{'}(\beta_{min})\right]^2 \left(1 + e_t^2\right)$$

The second term is then built as

$$\frac{1}{d_{length}^2(\mathcal{D})}$$

With this formulation, the solution of the optimization can be retrieved with

$$\mathcal{D} = \underset{\mathcal{D}}{\operatorname{argmin}} \left( w_{lin} \int_{\beta_{min}}^{\beta_{max}} d_{lin}^2(\mathcal{D}, \beta)\, d\beta + w_{length}\, \frac{1}{d_{length}^2(\mathcal{D})} \right)$$

The optimization is also subject to lower and upper bounds on the optimization variable $\mathcal{D}$; these limits were chosen according to reasonable design considerations. The problem is quite sensible to the initial conditions given to the optimizer, since the cost function possesses multiple local minima. As a consequence, the operation of choosing the right initialization was crucial and needed quite a lot of tuning.

Note that, since the dimensional synthesis is not known a priori, it was necessary to run two different optimization problems, one using the first solution of leg kinematics and another using the second one. At the end, the best one is manually selected. In particular, since both solution generally have very good linear performance, it is advisable to select the solutions with bigger trajectory length.

Finally, to carry out the optimization and compute the cost function integral, it was necessary to discretize the interval for $\beta$. A low number of discretization points was indeed sufficient to obtain very good results; increasing the discretization too much only produced a computational overhead, without appreciable solution improvements.

In particular, the optimization was carried out with the following parameters:

- $\beta \in [62°, 100°]$.
- Number of $\beta$ interval samples: 10.

- $w_{lin} = 10^2$.
- $w_{length} = 10^{-4}$.
- $\mathcal{D}_0 = [0.1,\ 0.2,\ 0.4,\ 0.4,\ 0.4, -1, 0.2]$.
- Lower bounds: $[0.1,\ 0.1,\ 0.1,\ 0.1,\ 0.1, -\infty, 0]$.
- Upper bounds: $[0.3,\ 0.3,\ 0.4,\ 0.4,\ 0.4, -1,\ \infty]$.

The result is

$$\mathcal{D}_{opt} = [0.1,\ 0.145,\ 0.236,\ 0.256,\ 0.355, -1, 0] \tag{3.17}$$

where all link dimensions were rounded up to the third decimal point for practical reasons. Also, for brevity sake, the optimizer's specific settings are not reported. In particular, solution (3.17) corresponds to the second solution and was selected over the first one because it is overall more compact than the first one. Since the amount of resources were limited, this solution was selected, even if it is not relative to the longest trajectory.

These results were used throughout the rest of the thesis and were never modified.

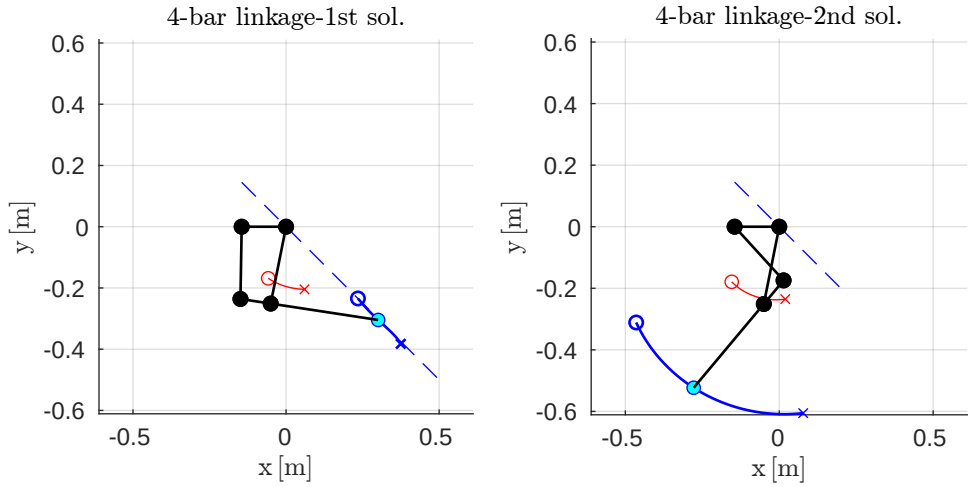



**Figure 3.2.** Optimized four-bar leg trajectory. On the left, the trajectory associated with the kinematic solution with concave knee (which is the one of interest). On the right, the trajectory associated with the convex configuration of the knee.

Obviously, this is not the only viable dimensional synthesis and, if needed, the optimization can simply be rerun to finding a more suitable solution.

Fig. 3.2 shows a plot of the wheel trajectory (blue line), the linkage CoM trajectory (red line), the least-squares linear approximation of the wheel trajectory and the linkage for both kinematic solutions (dashed blue line). The CoM trajectory is computed supposing links with constant density. In particular, aluminum one was chosen.

## 3.3 Leg approximate static planar model

During the project's first stage, even before a dynamics model was derived, there was the need to estimate approximately the leg torques needed to balance vertically the robot. To this purpose, an approximate static model of the knee was derived.

Later on, this approximate model was validated using the complete dynamics of the system. Surprisingly, it proved to be relatively accurate. For this reason and others which will be clear in the following discussion, this model is here reported.

In principle, it is possible to actuate the leg by acting on any of the available hinges. For hypothesis, two possible actuation schemes are considered: the first considers the actuation only on the hinge between links $L_d$ and $L_b$, while the second only on the hinge between links $L_c$ and $L_b$. These two schemes use the following

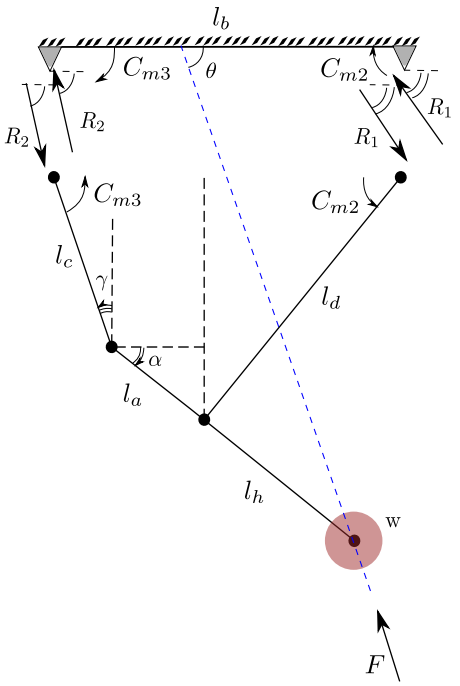



**Figure 3.3.** Approximate static leg model, with both actuations showed.

hypothesis:

- The vertical ground reaction force is parallel w.r.t. the linear approximation of the wheel trajectory in the local frame $x'z'$(shown as a dashed blue line in Fig. 3.3). This would be true if the dashed blue line stayed always vertical during vertical balancing. As will be seen when studying the static equilibrium of the system, this is not true; it is possible, however, to make this approximation relatively good by employing the previously discussed leg design principles. In fact, this is exactly the reason why the approximate models for computing balancing torques are able to perform reasonably well.
- The vertical reaction force on a single wheel is exactly equal to half the total weight of the robot (valid during static vertical balancing, but not during dynamical motion).

- All hinges are ideal.
- All links are indeformable.

These hypothesis are synthetically depicted in Fig. 3.3, where both actuations are shown, as well as all reactions to the body link. $C_{m2}$ is the torque to be applied between links $L_d$ and $L_b$, while $C_{m3}$ is the one to be applied between links $L_c$ and $L_b$.

Note that this leg linkage has 1 d.o.f., hence either $C_{m2}$ or $C_{m3}$ is sufficient to statically balance the system. The single models to be solved are shown, respectively, in Fig. 3.4 for $C_{m2}$ actuation and in Fig. 3.5 for $C_{m3}$ actuation. These are two distinct free body diagrams, in which all joint reactions are shown.

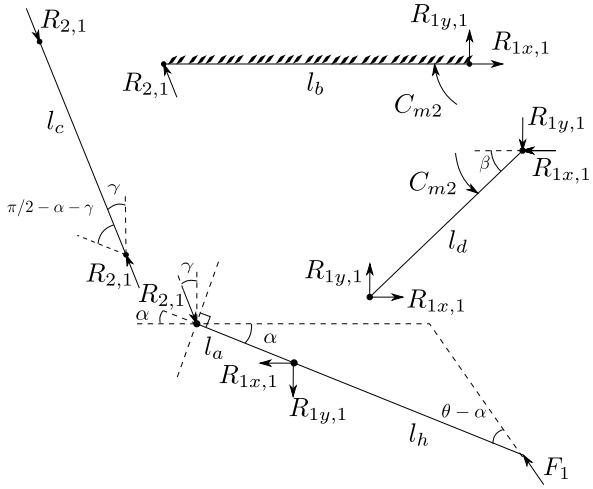



**Figure 3.4.** First approximate static model: actuation of $l_d$ hinge.

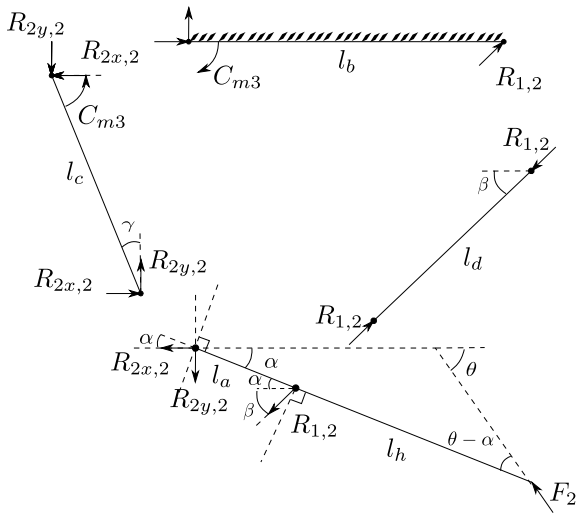



**Figure 3.5.** Second approximate static model: actuation of $l_c$ hinge.

This approximate models, once validated through the real balancing torques provided by the complete dynamics, do not become obsolete, because they can be

used to design leg hinges' mechanical characteristics through the knowledge of all joint reactions. By applying a properly chosen safety coefficient on those values to account for dynamical motion of the robot, leg bearings can be appropriately chosen.

For brevity, balance equations and explicit expressions of hinges reactions, which were double-checked using Matlab's Symbolic Toolbox, are not reported here. These explicit expressions are all contained inside a dedicated Matlab script.

The approximate expressions of $C_{m2}$ and $C_{m3}$ are, respectively,

$$\begin{aligned} C_{m2}(\mathcal{D},\beta) = -F_1\, l_d\, [&l_h\, \cos(\beta)\, \cos(\gamma)\, \sin(\alpha) - l_a\, \cos(\alpha+\gamma)\, \sin(\beta) + \\ &+ l_h\, \sin(\alpha)\, \sin(\beta)\, \sin(\gamma) + e_t\, l_a\, \cos(\alpha+\gamma)\, \cos(\beta) + \\ &+ e_t\, l_h\, \cos(\alpha)\, \cos(\beta)\, \cos(\gamma) + \\ &+ e_t\, l_h\, \cos(\alpha)\, \sin(\beta)\, \sin(\gamma)] \,/\, \left(l_a\, \cos(\alpha+\gamma)\, \sqrt{{e_t}^2+1}\right) \end{aligned} \tag{3.18}$$

$$\begin{aligned} C_{m3}(\mathcal{D},\beta) = -F_2\, l_c\, [&l_h\, \cos(\beta)\, \cos(\gamma)\, \sin(\alpha) - l_a\, \cos(\alpha)\, \cos(\gamma)\, \sin(\beta) + \\ &+ l_a\, \sin(\alpha)\, \sin(\beta)\, \sin(\gamma) + l_h\, \sin(\alpha)\, \sin(\beta)\, \sin(\gamma) + \\ &+ e_t\, l_a\, \cos(\alpha)\, \cos(\beta)\, \cos(\gamma) + e_t\, l_h\, \cos(\alpha)\, \cos(\beta)\, \cos(\gamma) + \\ &- e_t\, l_a\, \cos(\beta)\, \sin(\alpha)\, \sin(\gamma) + \\ &+ e_t\, l_h\, \cos(\alpha)\, \sin(\beta)\, \sin(\gamma)] \left(l_a\, \sin(\alpha+\beta)\, \sqrt{{e_t}^2+1}\right) \end{aligned} \tag{3.19}$$

where $F_1$ and $F_2$ are the share of weight to be sustained by, respectively, $C_{m2}$ and $C_{m3}$. In particular, supposing actuation only on the hinge between $L_d$ and $L_b$ (as done by [19]) , $F_1 = \left[m^{body} + 2\left(m^{ah} + m^c + m^d\right)\right]/2$ and $F_2 = 0$.

Since both $C_{m2}$ and $C_{m3}$ act on the same g.d.l., there is complete freedom in choosing the proportion in which they contribute to the leg actuation. This proportion can be chosen according to design requirements. For example, in the upcoming chapters, $C_{m3}$ will be neglected (hence, set to zero).

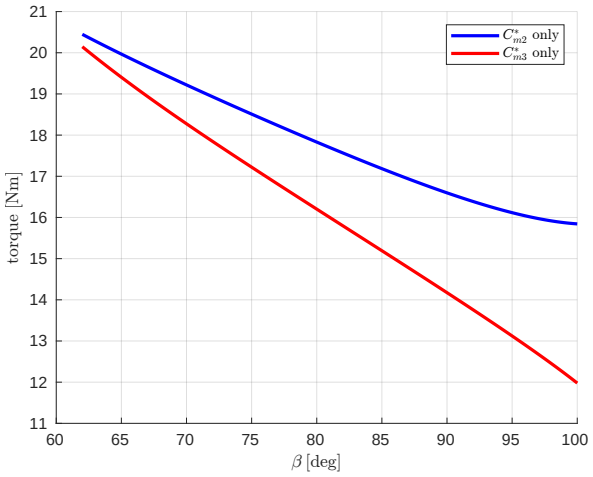



**Figure 3.6.** Approximate balancing torques: $C_{m2}$ (blue) vs $C_{m3}$ (red)

Fig. 3.6 reports a plot of the vertical balancing $C_{m2}$ and $C_{m3}$ computed with the approximate static model, supposing the whole robot weight is to be sustained

by only one between the two available actuations at a time. This plot provides a good design guide to choosing where to put the actuator if only one actuation per leg was available. In this case, for the same weight to the sustained, mounting the actuator on $L_c$ link provides significant advantage for high $\beta$ values (i.e. when the leg is almost fully extended). The inertial properties of the robot used to compute these torques were obtained assuming constant length of links and using aluminum density (this choice makes sense because for the prototype structure aluminum profiles are used). Body's mass is supposed to be 10 Kg ([19] was used as reference).
Furthermore, all link masses are kept constant during this work and can be read in Chapter 5.

## 3.4 Planar model during ground contact

The first derived model of the system is a complete planar model valid during ground contact. Extensions of this model and variations of it are presented in the upcoming sections.

The following figure is a synthetic representation of such a model:

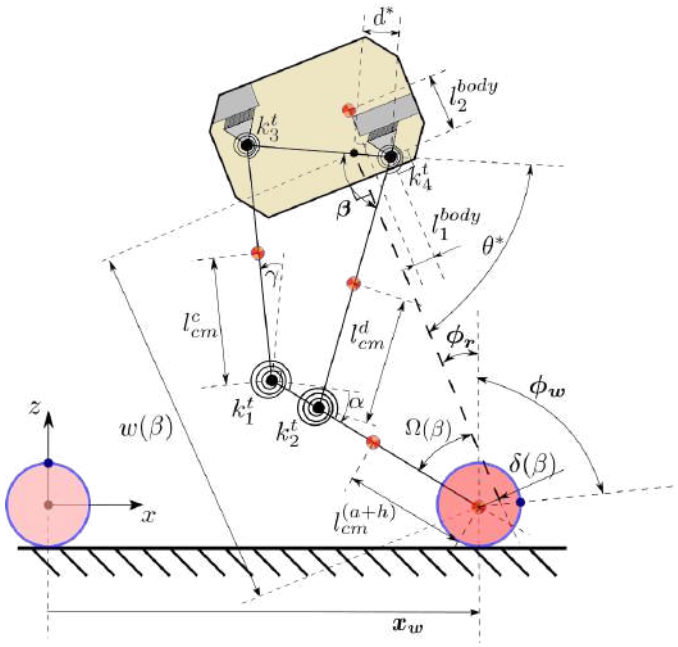



**Figure 3.7.** Complete planar model during ground contact

The reference frame is clearly visible. All the CoMs are shown as orange/yellow circles and their positioning relative to the joints is also indicated. To make the model more general, a total of four torsional springs in parallel with the joints are depicted. During simulations, however, only $k_1^t$ was considered. This mimics somehow the way knee springs are placed in the Ascento prototype [19].

The parameters $d^*$ and $\theta^*$ come from the dimensional synthesis of the knee and represent, respectively, the $x^{'}$ intercept of the wheel trajectory approximation in the local body reference frame and the acute angle of the approximation w.r.t. the same axis. The second parameter is particularly important, since the pitch of the robot is defined based on it. In particular, the figure shows a black dotted line representing the linear approximation of the wheel trajectory in the local $x^{'}y^{'}$ reference frame (see Fig. 3.1), which is by hypothesis orthogonal to the robot body. Parameters $l_1^{body}$ and $l_2^{body}$ define the body CoM position w.r.t. the hinge between $L_d$ and $L_b$.

Furthermore, $\delta(\beta)$ and $w(\beta)$ are, respectively, the distance of the wheel center from the linear approximation of the local wheel trajectory and the actual length of the black dotted line. These parameters are shown only to stress that the wheel center doesn't actually pass through this line (it is however pretty close, if the dimensional synthesis is carried out properly).

In Fig. 3.7 all leg angles are also shown, as well as one additional auxiliary angle $\Omega(\beta)$, which will be necessary in the upcoming discussion.

Finally, the system Lagrangian coordinates are shown in bold. Note that the robot pitch angle $\phi_r$ is positive counter-clockwise (contrary to common conventions) and that the wheel angle $\phi_w$ is measured w.r.t. the vertical. This way, during no-slip ideal contact, this variable is directly related to $x_w$. Clearly, this is not the only possible choice of coordinates.

### 3.4.1 Building the model

The following main assumptions are used:

- The motion stays planar and in particular is parallel w.r.t. $xz$ plane (defined in Fig. 3.7).
- The wheels always stay in contact with the ground.
- There is no ground slip. If using a simple friction model like, for instance, the friction cone (or pyramid since, if the motion is planar, they coincide), this hypothesis requires $T < \mu N$, where $T$ is the horizontal ground reaction component, $N$ is the vertical and $\mu$ is a suitable friction coefficient.
- All dissipative contributions are neglected. This means, in particular, that there is no rolling resistance, no joint dissipation and no viscous damping due to the air presence. This hypothesis are quite popular when developing control policies and they usually do not have a huge impact when controls are tested on real systems.
- The ground surface is perfectly flat and rigid. This is a good approximation for navigation inside urban environments and it is therefore a suitable hypothesis for the system under analysis.
- All links and the robot body are perfectly rigid. The reason why this is a good approximation will be clear in Chapter 7. For now, it is sufficient to state that the robot structure is not structurally critical. Hence, it is not necessary to perform structural analysis on its links. This also implies that the contact with the ground happens on a single point.

Fig. 3.8 shows how actuations are applied to the system. Here, both the potentially available leg actuations $C_{m2}$ and $C_{m3}$ are shown; $C_{m1}$ is the torque provided by the wheel actuator. Note that, since the actuator is mounted on link $L_{ah}$, the same torque acts on both the wheel and link itself.

To derive the equations of motion, the absolute orientation of each link is needed. This can be readily accomplished, since the model is planar, by making use of some elementary trigonometry.

Firstly, it is necessary to retrieve the expression of $\Omega(\beta)$. This is done in Fig. 3.10. With the knowledge of this angle, it is possible to determine the absolute orientations of each link as a function of the Lagrangian variables.

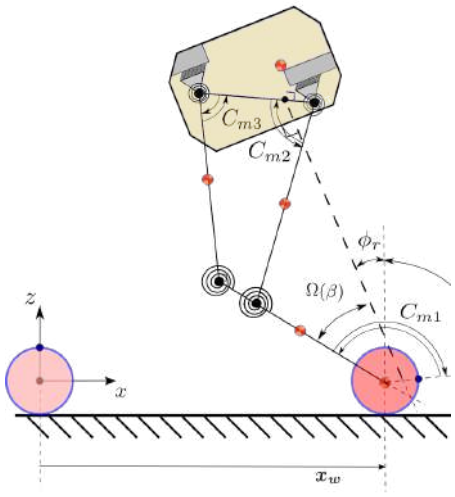



**Figure 3.8.** Complete planar model during ground contact: actions

This is shown, in particular, in Fig. 3.9.

Furthermore, Fig. 3.10 shows an angle $\theta^*_{body}$ which can and will be used as a design parameter for deciding efficiently how to position the body's CoM. Another parameter which will be used with $\theta^*_{body}$ is $l^{body}_{cm}$, which is the distance between body's CoM and $l_d$ hinge. These two parameters are easily retrieved with the knowledge of $l^{body}_1$ and $l^{body}_2$.

### 3.4.2 Writing the dynamic equations

Once all the kinematic parameters are completely defined and the modeling assumptions are clearly outlined, it is possible to start writing the dynamics.

In particular, the Lagrangian coordinates of the system are conveniently grouped in the following generalized position vector:

$$q_p(t) \coloneqq \begin{bmatrix} \phi_w(t) \\ \phi_r(t) \\ \beta(t) \end{bmatrix} \tag{3.20}$$

where the subscript $p$ is to distinguish between these vector and the state vector $q(t)$ which will be defined next. Vector (3.20) describes the positional *configuration* of the system in its configuration space $\mathcal{C}$ (which, in general, coincides with $\mathbb{R}^n$, where $n$ is system's number of d.o.f).

As remarked by [39], any wheeled vehicle subject to kinematic constraints possesses, in general, a reduced local mobility, but the possibility of reaching arbitrary configurations by appropriate manoeuvres remains intact. This would indeed be the

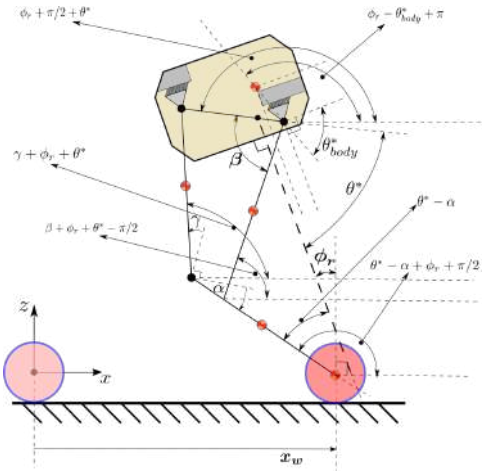



**Figure 3.9.** Complete planar model during ground contact: link absolute angles

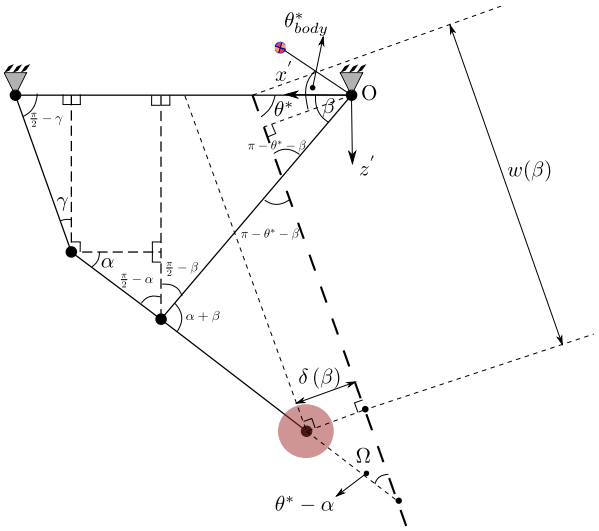



**Figure 3.10.** Retrieving $\Omega(\beta)$

case of a 3D model of the system. In the case of this 2D model (and also the other ones), due to the planar motion, there is not even loss of local mobility.

Note that the variable $x_w(t)$ is missing from the vector since, during contact, the following kinematic contact constraint (which is also in *Pfaffian form*) is active:

$$\dot{x}_w(t) = \dot{\phi}_w(t)\, r_w \tag{3.21}$$

where $r_w$ is the wheel radius.

In the case of planar motion, this simple constraint can be integrated into:

$$x_w(t) = \left(\phi_w(t) - \phi_w^0\right)\, r_w \tag{3.22}$$

In particular $\phi_w^0$ can be set to 0 without loss of generality (in particular refer to Fig. 3.7, where this assumption is made).

This is an example of an integrable kinematic constraint. Since this constraint is integrable, it is also *holonomic*. If it wasn't integrable, then the constraint would have been called *non-holonomic*.

Holonomic contraints like (3.22) are actually very similar to prismatic or revolute joints and, as such, reduce the space of *accessible configurations* (see [39]). This aspect will be clearer, in particular, in Section 3.6, when the floating-base model will be derived.

Before actually starting to write the dynamics, a couple of additional kinematic relationships are needed.
Recall that

$$\begin{aligned} \cos\,\alpha &= f_1(\mathcal{D}, \beta(t)) \\ \sin\,\alpha &= f_2(\mathcal{D}, \beta(t)) \\ \cos\,\gamma &= f_3(\mathcal{D}, \beta(t)) \\ \sin\,\gamma &= f_4(\mathcal{D}, \beta(t)) \\ \cos\,\theta^* &= f_5(\mathcal{D}) \\ \sin\,\theta^* &= f_6(\mathcal{D}) \end{aligned} \tag{3.23}$$

where $\mathcal{D}$ is the vector containing all the dimensional parameters of the leg (see (3.16)).
From now on, explicit dependence on $\mathcal{D}$ will be omitted, since these parameters are fixed, once a dimensional leg synthesis is chosen. Explicit dependence of state variables w.r.t. time will also be omitted, if not necessary.
The time derivates of the $\beta$-dependent leg angles are computed in the following way:

$$\dot{\alpha}(t) = \frac{df_2}{d\beta}(\beta(t))\,\frac{1}{f_1(\beta(t))}\,\dot{\beta}(t) = h_1(\beta(t))\,\dot{\beta}(t) \tag{3.24}$$

$$\dot{\gamma}(t) = -\frac{df_3}{d\beta}(\beta(t))\,\frac{1}{f_4(\beta(t))}\,\dot{\beta}(t) = h_2(\beta(t))\,\dot{\beta}(t) \tag{3.25}$$

where functions $h_1(\beta) = \dfrac{d\alpha(\beta)}{d\beta}$ and $h_2(\beta) = \dfrac{d\gamma(\beta)}{d\beta}$ are known leg Jacobians.
The first step to writing the dynamics is to retrieve all CoM positions:

$$p_{cm}^{w} = \begin{pmatrix} \phi_w\, r_w \\ \\ 0 \end{pmatrix} \tag{3.26a}$$

$$p_{cm}^{(a+h)} = \begin{pmatrix} \phi_w\, r_w - l_{cm}^{(a+h)}\,\sin\left(\phi_r - \alpha + \theta^*\right) \\ \\ l_{cm}^{(a+h)}\,\cos\left(\phi_r - \alpha + \theta^*\right) \end{pmatrix} \tag{3.26b}$$

$$p_{cm}^{d} = \begin{pmatrix} \phi_w\, r_w - l_h\,\sin\left(\phi_r - \alpha + \theta^*\right) + l_{cm}^{d}\,\sin\left(\beta + \phi_r + \theta^*\right) \\ \\ l_h\,\cos\left(\phi_r - \alpha + \theta^*\right) - l_{cm}^{d}\,\cos\left(\beta + \phi_r + \theta^*\right) \end{pmatrix} \tag{3.26c}$$

$$p_{cm}^{c} = \begin{pmatrix} \phi_w\, r_w - \sin\left(\phi_r - \alpha + \theta^*\right)\,\left(l_a + l_h\right) + l_{cm}^{c}\,\cos\left(\gamma + \phi_r + \theta^*\right) \\ \\ \cos\left(\phi_r - \alpha + \theta^*\right)\,\left(l_a + l_h\right) + l_{cm}^{c}\,\sin\left(\gamma + \phi_r + \theta^*\right) \end{pmatrix} \tag{3.26d}$$

$$p_{cm}^{body} = \\ = \begin{pmatrix} \phi_w\, r_w - l_h\, \sin\left(\phi_r - \alpha + \theta^*\right) - l_{cm}^{body}\, \cos\left(\phi_r - \theta_{body}^*\right) + l_d\, \sin\left(\beta + \phi_r + \theta^*\right) \\ \\ l_h\, \cos\left(\phi_r - \alpha + \theta^*\right) - l_{cm}^{body}\, \sin\left(\phi_r - \theta_{body}^*\right) - l_d\, \cos\left(\beta + \phi_r + \theta^*\right) \end{pmatrix} \tag{3.26e}$$

In writing these vectors, holonomic contraint (3.22) was applied. This way the CoM positions are only function of $q_p$. The knowledge of this vector is sufficient to write system's potential energy.

To be able to write the system's kinetic energy, it is also necessary to obtain all CoM velocities, which are simply retrieved by differentiation of expressions (3.26a) - (3.26e) w.r.t. time. The result is the following:

$$v_{cm}^{w} = \begin{pmatrix} j_{11}^{w}\, \dot{\phi}_w + j_{12}^{w}\, \dot{\phi}_r + j_{13}^{w}\, \dot{\beta} \\ \\ j_{21}^{w}\, \dot{\phi}_w + j_{22}^{w}\, \dot{\phi}_r + j_{23}^{w}\, \dot{\beta} \end{pmatrix} = J^{w}\, \dot{q}_p$$

$$v_{cm}^{(a+h)} = \begin{pmatrix} j_{11}^{(a+h)}\, \dot{\phi}_w + j_{12}^{(a+h)}\, \dot{\phi}_r + j_{13}^{(a+h)}\, \dot{\beta} \\ \\ j_{21}^{(a+h)}\, \dot{\phi}_w + j_{22}^{(a+h)}\, \dot{\phi}_r + j_{23}^{(a+h)}\, \dot{\beta} \end{pmatrix} = J^{(a+h)}\left(\phi_r, \beta\right) \dot{q}_p$$

$$v_{cm}^{d} = \begin{pmatrix} j_{11}^{d}\, \dot{\phi}_w + j_{12}^{d}\, \dot{\phi}_r + j_{13}^{d}\, \dot{\beta} \\ \\ j_{21}^{d}\, \dot{\phi}_w + j_{22}^{d}\, \dot{\phi}_r + j_{23}^{d}\, \dot{\beta} \end{pmatrix} = J^{d}\left(\phi_r, \beta\right) \dot{q}_p$$

$$v_{cm}^{c} = \begin{pmatrix} j_{11}^{c}\, \dot{\phi}_w + j_{12}^{c}\, \dot{\phi}_r + j_{13}^{c}\, \dot{\beta} \\ \\ j_{21}^{c}\, \dot{\phi}_w + j_{22}^{c}\, \dot{\phi}_r + j_{23}^{c}\, \dot{\beta} \end{pmatrix} = J^{c}\left(\phi_r, \beta\right) \dot{q}_p$$

$$v_{cm}^{body} = \begin{pmatrix} j_{11}^{body}\, \dot{\phi}_w + j_{12}^{body}\, \dot{\phi}_r + j_{13}^{body}\, \dot{\beta} \\ \\ j_{21}^{body}\, \dot{\phi}_w + j_{22}^{body}\, \dot{\phi}_r + j_{23}^{body}\, \dot{\beta} \end{pmatrix} = J^{body}\left(\phi_r, \beta\right) \dot{q}_p$$

where $J^i$ are velocity Jacobians, at best dependent upon $\phi_r$ and $\beta$. For brevity, Jacobians explicit expressions are here omitted.

The knowledge of link velocities allows writing all the kinetic energies:

$$E_k^w\left(\dot{\phi}_w\right) = \frac{1}{2}\, m^w \left[\dot{q}_p^T \left(J^w\right)^T J^w\, \dot{q}_p\right] + \frac{1}{2}\, I_{yy}^w\, \dot{\phi}_w^2 = \frac{1}{2}\, \dot{q}_p^T\, M^w\, \dot{q}_p \tag{3.28a}$$

$$\begin{aligned} E_k^{(a+h)}\left(\phi_r,\, \beta,\, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2}\, m^{(a+h)} \left[\dot{q}_p^T\, J^{(a+h)^T}\, J^{(a+h)}\, \dot{q}_p\right] + \\ + \frac{1}{2}\, I_{yy}^{(a+h)} \left(\dot{\phi}_r - \dot{\alpha}\right)^2 = \frac{1}{2}\, \dot{q}_p^T\, M^{(a+h)}\, \dot{q}_p \end{aligned} \tag{3.28b}$$

$$\begin{aligned} E_k^d\left(\phi_r,\, \beta,\, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2}\, m^d \left[\dot{q}_p^T \left(J^d\right)^T J^d\, \dot{q}_p\right] + \\ + \frac{1}{2}\, I_{yy}^d \left(\dot{\phi}_r + \dot{\beta}\right)^2 = \frac{1}{2}\, \dot{q}_p^T\, M^d\, \dot{q}_p \end{aligned} \tag{3.28c}$$

$$\begin{aligned} E_k^c\left(\phi_r,\, \beta,\, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2}\, m^c \left[\dot{q}_p^T \left(J^c\right)^T J^c\, \dot{q}_p\right] + \\ + \frac{1}{2}\, I_{yy}^c \left(\dot{\phi}_r + \dot{\gamma}\right)^2 = \frac{1}{2}\, \dot{q}_p^T\, M^c\, \dot{q}_p \end{aligned} \tag{3.28d}$$

$$\begin{aligned} E_k^{body}\left(\phi_r,\, \beta,\, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2}\, m^{body} \left[\dot{q}_p^T \left(J^{body}\right)^T J^{body}\, \dot{q}_p\right] + \\ + \frac{1}{2}\, I_{yy}^{body}\, \dot{\phi}_r^{\,2} = \frac{1}{2}\, \dot{q}_p^T\, M^{body}\, \dot{q}_p \end{aligned} \tag{3.28e}$$

where $m_i$ are link masses, $I_{yy}^i$ are the moments of inertia of link $L_i$ w.r.t. axis $y$ (orthogonal to the plane) measured in the CoM frames and $M_i$ is the mass matrix of link $L_i$.

Taking into account the presence of two legs, the total kinetic energy takes the form

$$E_k^{tot}\left(\phi_r,\, \beta,\, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = 2\left(E_k^w + E_k^{(a+h)} + E_k^d + E_k^c\right) + E_k^{body} = \frac{1}{2}\, \dot{q}_p^T\, M\, \dot{q}_p \tag{3.29}$$

where

$$M(\phi_r, \beta) = 2\left(M^w + M^{(a+h)} + M^d + M^c\right) + M^{body} \tag{3.30}$$

Using the second elements of expressions from (3.26a) to (3.26e), the gravitational potential energies of each link and the body can be written as

$$E_{p,g}^w = 0 \tag{3.31a}$$

$$E_{p,g}^{(a+h)}\left(\phi_r, \beta\right) = m^{(a+h)}\, g\, h_{cm}^{(a+h)} \tag{3.31b}$$

$$E_{p,g}^d\left(\phi_r, \beta\right) = m^d\, g\, h_{cm}^d \tag{3.31c}$$

$$E_{p,g}^c\left(\phi_r, \beta\right) = m^c\, g\, h_{cm}^c \tag{3.31d}$$

$$E_{p,g}^{body}\left(\phi_r, \beta\right) = m^{body}\, g\, h_{cm}^{body} \tag{3.31e}$$

where $g$ is the gravitational acceleration and $h_{cm}^i$ is the CoM height of link $L_i$ measured with respect to the absolute frame $xyz$ (defined in Fig. 3.7). Note that

(3.31a) is 0 because the system's potential energy is here defined w.r.t. a height equal to the wheel radius.

The total gravitational potential energy becomes

$$E_{p,g}^{tot}\left(\phi_r,\beta\right)=2\left(E_{p,g}^{w}+E_{p,g}^{(a+h)}+E_{p,g}^{d}+E_{p,g}^{c}\right)+E_{p,g}^{body} \tag{3.32}$$

For brevity, its explicit expression is not here reported.
Moreover, the elastic potential energy of a single leg takes the following form:

$$\begin{aligned}E_{p,el}(\beta)=&\frac{k_3^t\left[\gamma\left(\beta\right)-\gamma_0\left(\beta_0^{el}\right)\right]^2}{2}+\frac{k_1^t\left[\alpha\left(\beta\right)+\gamma(\beta)-\alpha_0\left(\beta_0^{el}\right)-\gamma_0\left(\beta_0^{el}\right)\right]^2}{2}\\&+\frac{k_2^t\left[\beta+\alpha\left(\beta\right)-\alpha_0\left(\beta_0^{el}\right)-\beta_0\right]^2}{2}+\frac{k_4^t\left[\beta-\beta_0\right]^2}{2}\end{aligned} \tag{3.33}$$

where $k_i^t$ are joint torsional rigidities (defined in Fig. 3.7) and the subscript 0 indicates reference angles for the elastic potential energy.

The total elastic potential energy simply becomes

$$E_{p,el}^{tot}\left(\beta\right)=2\,E_{p,el}\left(\beta\right) \tag{3.34}$$

Finally, the total potential energy is

$$E_{p}^{tot}\left(\phi_r,\beta\right)=E_{p,g}^{tot}\left(\phi_r,\beta\right)+E_{p,el}^{tot}\left(\beta\right) \tag{3.35}$$

Before actually writing the dynamics using Lagrange's method, it is necessary to obtain the Lagrangian components of the inputs.

The easiest way of obtaining those components is to write the total infinitesimal work made on each Lagrangian d.o.f by the actions as

$$\begin{aligned}\delta\mathcal{W}&=2\,C_{m1}\,\delta(\phi_r+\Omega(\beta)+\phi_w)+2\,C_{m2}\,\delta\beta-2\,C_{m3}\,\delta(\pi/2-\gamma(\beta))=\\&=2\,C_{m1}\,\delta(\phi_r+\theta^*-\alpha(\beta)+\phi_w)+2\,C_{m2}\,\delta\beta-2\,C_{m3}\,\delta(\pi/2-\gamma(\beta))=\\&=2\,C_{m1}\,\delta\phi_w+2\,C_{m1}\,\delta\phi_r-\left[2\,C_{m1}\,h_1-2\,C_{m2}-2\,C_{m3}\,h_2\right]\,\delta\beta\end{aligned} \tag{3.36}$$

As can be seen in (3.36), $C_{m1}$ does work on the relative angle between the wheel and link $L_{ah}$. This is because the wheel actuation is mounted on link $L_{ah}$. Due to this, $C_{m1}$ possesses a Lagrangian component on all langrangian coordinates. Also note that torque inputs are thought to act on a single leg; as a consequence, during planar motion, their contribution to the dynamics is multiplied by a factor of 2.
In (3.36), the previously defined leg Jacobians $h_1(\beta)$ and $h_2(\beta)$ are used.

All the three Lagrangian components of the input can be readily extracted from (3.36) by looking at the coefficients multiplying the infinitesimal increment of each langragian coordinate.
In particular, indicating with $f_j$ the Lagrangian component of coordinate $j$,

$$f_{\phi_w}=2\,C_{m1}(t) \tag{3.37a}$$

$$f_{\phi_r} = 2\, C_{m1}(t) \tag{3.37b}$$

$$f_\beta = 2\, [C_{m2}(t) + C_{m3}(t)\, h_2\,(\beta(t))\,] - 2\, C_{m1}(t) \tag{3.37c}$$

Here, to make the model slightly more general, also $C_{m3}$ was included. In numerical simulations, however, only $C_{m2}$ was used for simplicity.

Once all fundamental blocks have been computed, the Lagrangian of the system can be written easily as

$$\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}) = E_k^{tot}(\phi_r, \beta, \dot{\phi}_w,\, \dot{\phi}_r, \dot{\beta}) - E_p^{tot}\,(\phi_r, \beta) \tag{3.38}$$

Defining

$$F_{lgr} \coloneqq \begin{bmatrix} f_{\phi_w} \\ \\ f_{\phi_r} \\ \\ f_\beta \end{bmatrix} \tag{3.39}$$

the dynamics can be written synthetically in the following vectorial form:

$$\frac{d}{dt}\left(\nabla_{\dot{q}_p}^T\left(\mathcal{L}(\phi_r, \beta, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta})\right)\right) - \left(\nabla_{q_p}^T\left(\mathcal{L}(\phi_r, \beta, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta})\right)\right) = F_{lgr} \tag{3.40}$$

Just for clarity, single components of equation (3.40) are here reported:

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{\phi}_w}\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,)\right) - \frac{\partial}{\partial \phi_w}\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,) = f_{\phi_w} \tag{3.41a}$$

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{\phi}_r}\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,)\right) - \frac{\partial}{\partial \phi_r}\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,) = f_{\phi_r} \tag{3.41b}$$

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{\beta}}\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,)\right) - \frac{\partial}{\partial \beta}\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,) = f_\beta \tag{3.41c}$$

Let us now work on expanding each term of the equations of motion as much as possible.
First consider that, since potential energy does not depend on $\dot{q}_p$,

$$\nabla_{\dot{q}_p}^T\left(\mathcal{L}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,)\right) = \nabla_{\dot{q}_p}^T\left(E_c^{tot}\,(\phi_r, \beta, \dot{\phi}_w\,, \dot{\phi}_r\,, \dot{\beta}\,)\right) = M(\phi_r, \beta\,)\,\dot{q}_p \tag{3.42}$$

Using equation (3.42), we obtain

$$\frac{d}{dt}\left(\nabla_{\dot{q}_p}^T\left(E_c^{tot}\right)\right) = M\,\ddot{q}_p + \dot{M}\,\dot{q}_p = M\,\ddot{q}_p + \left(\frac{\partial M}{\partial \phi_w}\dot{\phi_w} + \frac{\partial M}{\partial \phi_r}\dot{\phi_r} + \frac{\partial M}{\partial \beta}\dot{\beta}\right)\dot{q}_p \tag{3.43}$$

Note that $\dfrac{\partial M}{\partial \phi_w} = 0$, but was left to make the scheme more general.
The right side of this equation can also be rewritten in the following form:

$$M\,\ddot{q}_p + \left(\frac{\partial M}{\partial \phi_w}K_{\phi_w} + \frac{\partial M}{\partial \phi_r}K_{\phi_r} + \frac{\partial M}{\partial \beta}K_\beta\right)\dot{q}_p^{quad} \tag{3.44}$$

where

$$K_{\phi_w} \coloneqq \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$K_{\phi_r} \coloneqq \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.45}$$

$$K_{\beta} \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

and

$$\dot{q}_p^{quad} \coloneqq \begin{bmatrix} \dot{\phi}_w^2 \\ \dot{\phi}_r^2 \\ \dot{\beta}^2 \\ \dot{\phi}_w \, \dot{\phi}_r \\ \dot{\phi}_w \, \dot{\beta} \\ \dot{\phi}_r \, \dot{\beta} \end{bmatrix} \tag{3.46}$$

Equation (3.44) is simply a more concise way of writing (3.43). This way of writing this term allows to group centrifugal and Coriolis contributions and is also slightly easier to handle when implementing the dynamics in software.
Passing to the next term,

$$\begin{aligned} \nabla_{q_p}^T (\mathcal{L}) &= \nabla_{q_p}^T E_k^{tot} - \nabla_{q_p}^T E_p^{tot} = \\ &= \nabla_{q_p}^T (e^{tot}) \, \dot{q}_p^{quad} - \nabla_{q_p}^T E_p^{tot} \end{aligned} \tag{3.47}$$

where

$$e^{tot} (\phi_r, \beta) \coloneqq \begin{bmatrix} e_1^{tot} \\ e_2^{tot} \\ e_3^{tot} \\ e_4^{tot} \\ e_5^{tot} \\ e_6^{tot} \end{bmatrix} \tag{3.48}$$

Note that $\nabla_{q_p}^T E_p^{tot}$ can be computed employing the explicit expression of (3.35). Moreover, the elements of (3.48) are so that

$$M\left(\phi_r(t), \beta(t)\right) \coloneqq \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \coloneqq \begin{bmatrix} 2\,e_1^{tot} & e_4^{tot} & e_5^{tot} \\ e_4^{tot} & 2\,e_2^{tot} & e_6^{tot} \\ e_5^{tot} & e_6^{tot} & 2\,e_3^{tot} \end{bmatrix} \tag{3.49}$$

Defining

$$u \coloneqq \begin{bmatrix} C_{m1}(t) \\ C_{m2}(t) \\ C_{m3}(t) \end{bmatrix} \tag{3.50}$$

then

$$F_{lgr} \coloneqq G^{'}(\beta)\, u(t) \tag{3.51}$$

where

$$G^{'} \coloneqq \begin{bmatrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ -2\,h_1 & 2 & 2\,h_2 \end{bmatrix} \tag{3.52}$$

Using all the obtained expressions, the vectorial form of the dynamics can be written as:

$$\begin{aligned} &M\,\ddot{q}_p - \left\{ \left[ \nabla_{q_p}^T \left(e^{tot}\right) - \left( \frac{\partial\, M}{\partial \phi_w}\, K_{\phi_w} + \frac{\partial\, M}{\partial \phi_r}\, K_{\phi_r} + \frac{\partial\, M}{\partial \beta}\, K_\beta \right) \right] \dot{q}_p^{quad} - \nabla_{q_p}^T \left(E_p^{tot}\right) \right\} = \\ &= G^{'}\, u \end{aligned} \tag{3.53}$$

This equation can be synthetically rewritten as

$$M(q_p(t))\, \ddot{q}_p(t) + C(q_p(t), \dot{q}_p(t)) = G^{'}(q_p(t))\, u(t) \tag{3.54}$$

where the vector $C(q_p, \dot{q}_p)$ is equal to

$$\begin{aligned} C(q_p, \dot{q}_p) \coloneqq &- \left[ \nabla_{q_p}^T \left(e^{tot}\right) - \left( \frac{\partial\, M}{\partial \phi_w}\, K_{\phi_w} + \frac{\partial\, M}{\partial \phi_r}\, K_{\phi_r} + \frac{\partial\, M}{\partial \beta}\, K_\beta \right) \right] \dot{q}_p^{quad} + \\ &+ \nabla_{q_p}^T \left(E_p^{tot}\right) \end{aligned} \tag{3.55}$$

and contains all Coriolis and centrifugal contributions (between square brackets in (3.54)) and also gravitational and elastic ones.

This system can be converted to a first order one by defining the augmented state vector $q(t)$ as

$$q(t) \coloneqq \begin{bmatrix} q_1(t) \\ q_2(t) \\ q_3(t) \\ q_4(t) \\ q_5(t) \\ q_6(t) \end{bmatrix} \coloneqq \begin{bmatrix} \phi_w \\ \phi_r \\ \beta \\ \dot{\phi}_w \\ \dot{\phi}_r \\ \dot{\beta} \end{bmatrix} \tag{3.56}$$

This result into the first order dynamics given by

$$\dot{q}(t) = f(q(t)) + G(q(t))\, u(t) \tag{3.57}$$

where

$$f(q) \coloneqq \begin{bmatrix} q_4(t) \\ q_5(t) \\ q_6(t) \\ -M^{-1}(q)\, C(q) \end{bmatrix} \tag{3.58}$$

and

$$G(q) \coloneqq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ M^{-1}(q)\, G'(q) \end{bmatrix} \tag{3.59}$$

Some important properties of system (3.57) are discussed in Section 5.2 and in the following ones. Note that (3.57) is a first order input-affine nonlinear system.

## 3.5 eWSLIP model during contact

The eWSLIP model is an equivalent model to (3.57) and was used in several occasions during the control part of the work for deriving useful equivalent quantities.

eWSLIP stands for " Equivalent Wheeled Spring Loaded Inverted Pendulum" and it is a model which captures the whole dynamics of (3.57), while also allowing the identification of a couple of fundamental parameters. eWSLIP's wheel is meant to represent both wheels of 3.7, while the upper part (i.e. the pendulum) is meant to capture the centroidal dynamics of the legs and the body as a whole. Fig. 3.11 contains a representation of such an equivalent model.
The Lagrangian variables of the model are $\theta_w$, $\theta_r$ and $l_{cm}^p$ and they can all be written

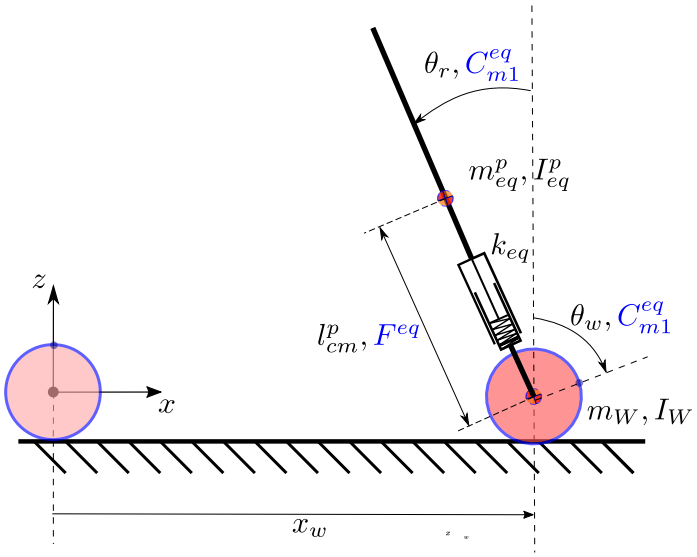



**Figure 3.11.** eWSLIP model. This model is made of two distinct bodies: an equivalent wheel and an equivalent pendulum.

in terms of the ones of model 3.7.
In particular:

- The center of mass of the equivalent pendulum is shown as a yellow and orange circle. It corresponds to the partial CoM of the set {two legs, robot body} of the parent system 3.7.
- As a consequence, $l_{cm}^p$ represents the distance of the equivalent pendulum CoM to the wheel center and it is a function of $\beta$.
- $k_{eq}$ is the equivalent linear stiffness. Due to the presence of the knee, this stiffness will be a nonlinear function of $\beta$ (or, equivalently, $l_{cm}^p$).
- $m_{eq}^p = m^{body} + 2\,(m^{ah} + m^c + m^d)$.
- $I_{eq}^p$ is the equivalent moment of inertia of the pendulum around axis $y$ and will be a function of $\beta$. This fact has a trivial interpretation, since the mass distribution of the upper part of the robot is completely determined by the leg angle or, equivalently, the equivalent pendulum length.
- $m_W = 2\,m^w$.
- $I_W = 2\,I_{yy}^w$.
- $C_{m1}^{eq} = 2\,C_{m1}$.
- $F^{eq}$ is the equivalent linear actuation.
- $\theta_w \equiv \phi_w$.

- $\theta_r = \phi_r - \phi_r^*$, where $\phi_r^*$ is the vertical balance pitch angle for a given leg angle. This quantity will be defined precisely in Section 5.2, where the vertical equilibrium of the system will be addressed. For now, it is sufficient to know that it is a function of $\beta$.

- Once again, ground contact without slipping is assumed, so the $x_w$ variable is not necessary.

The knowledge of these parameters/variables, allows to establish a 1-1 correspondence between model 3.7 and 3.11. The problem of determining all of them, however, still remains. Some of them can be obtained trivially, while for others it is necessary to employ some equivalence relationships.

It can be easily shown that the mass matrix for the eWSLIP during contact is

$$M(\theta_r, l_{cm}^p) = \begin{bmatrix} \left(m_{eq}^p + m_W\right) {r_w}^2 + J_W & -l_{cm}^p \, m_{eq}^p \, r_w \, \cos(\theta_r) & -m_{eq}^p \, r_w \, \sin(\theta_r) \\ -l_{cm}^p \, m_{eq}^p \, r_w \, \cos(\theta_r) & m_{eq}^p \, {l_{cm}^p}^2 + I_{eq}^p(l_{cm}^p) & 0 \\ -m_{eq}^p \, r_w \, \sin(\theta_r) & 0 & m_{eq}^p \end{bmatrix} \tag{3.60}$$

The first thing the eWSLIP model needs to satisfy is the equivalence of kinetic energy between the models. Since $l_{cm}^p(\beta)$ is trivially obtained by computing the relative distance between the wheel and the partial CoM of the robot (details here omitted for simplicity) and $m_{eq}^p$ is already defined, the comparison between kinetic energies of the models allows to identify $I_{eq}^p(\beta)$.

By equating the two kinetic energies and expressing all equivalent quadratic terms in the new Lagrangian variables as a function of the old ones, it is easy to see that

$$I_{eq}^p(\beta) = m_{22} - m_{eq}^p \, {l_{cm}^p}^2 \tag{3.61}$$

where $m_{22}$ is the second diagonal element of (3.49).
The dependence of $I_{eq}^p$ on variable $\beta$ only was confirmed via Matlab's Symbolic Toolbox. $I_{eq}^p$ and $l_{eq}^p$ are the only equivalent parameters which will be actually used in Chapter 5. However, for completeness, also the remaining ones will be derived.

To determine $k_{eq}$ it is sufficient to impose elastic energy equivalence with

$$\frac{1}{2} \, k_{eq} \, (l_{cm}^p - l_{cm,0}^p)^2 = E_{p,el}^{tot} \tag{3.62}$$

where $E_{p,el}^{tot}$ is (3.34) and $l_{cm,0}^p = l_{cm,0}^p(\beta_0^{el})$.
From (3.62),

$$k_{eq}(\beta) = 2 \, \frac{E_{p,el}^{tot}}{(l_{cm}^p - l_{cm,0}^p)^2} \tag{3.63}$$

To find the equivalent actions, it is sufficient to impose the infinitesimal work

equivalence between the models. This is done as

$$\begin{aligned}\delta\mathcal{W}^{eq} &= C_{m1}^{eq}\,\delta(\theta_w+\theta_r) + F_{eq}\,\delta\,l_{cm}^{p} = \\ &= C_{m1}^{eq}\delta\phi_w + C_{m1}^{eq}\delta\phi_r - C_{m1}^{eq}\delta\phi_r^{*} + F_{eq}\,\delta l_{cm}^{p} = \\ &= C_{m1}^{eq}\delta\phi_w + C_{m1}^{eq}\delta\phi_r + \left[-C_{m1}^{eq}\frac{\partial\phi_r^{*}}{\partial\beta} + F_{eq}\frac{\partial l_{cm}^{p}}{\partial\beta}\right]\delta\beta\end{aligned} \tag{3.64}$$

where $\dfrac{\partial\phi_r^{*}}{\partial\beta}$ and $\dfrac{\partial l_{cm}^{p}}{\partial\beta}$ are known functions of $\beta$.
Equating (3.36) to (3.64) and comparing each term multiplying the Lagrange variables's infinitesimal increments, we obtain

$$C_{m1}^{eq} = 2\,C_{m1} \tag{3.65}$$

$$F_{eq}(t,\beta) = \frac{2\left\{C_{m1}\left[\dfrac{\partial\phi_r^{*}}{\partial\beta} - h_1\right] + [C_{m2} + C_{m3}\,h_2]\right\}}{\dfrac{\partial l_{cm}^{p}}{\partial\beta}} \tag{3.66}$$

From (3.66) it is clear how the equivalent linear force $F_{eq}$ depends linearly on the actuations of system (3.7) and nonlinearly on $\beta$, through $h_1$, $h_2$, $\dfrac{\partial\phi_r^{*}}{\partial\beta}$ and $\dfrac{\partial l_{cm}^{p}}{\partial\beta}$.

Up to now, only the transition between the parent model (3.7) and the child one (3.11) was discussed. The inverse procedure is however conceptually very simple; the only problem is the inversion of the relationship $l_{cm}^{p} = l_{cm}^{p}(\beta)$, which is not trivial to solve analytically. The easiest solution is, given a dimensional synthesis of the leg, inertial properties, elastic properties, etc..., to invert the relationship numerically (e.g. via interpolation). Plots of $I_{eq}^{p}$ and $l_{cm}^{p}$ as a function of $\beta$ are reported in Section 6.4.1, in particular in Fig. 6.4.

## 3.6 Floating base planar model

Model 3.7 is valid during ground contact.

To be able to extend this model to cases in which ground contact might be broken, a widely used approach is to derive the so called *floating-base* model of the system. Examples of use of this kind of models can be found in [48], [8] and [47]. In particular, [8] and [47] use those models to develop controllers for jumping motion.

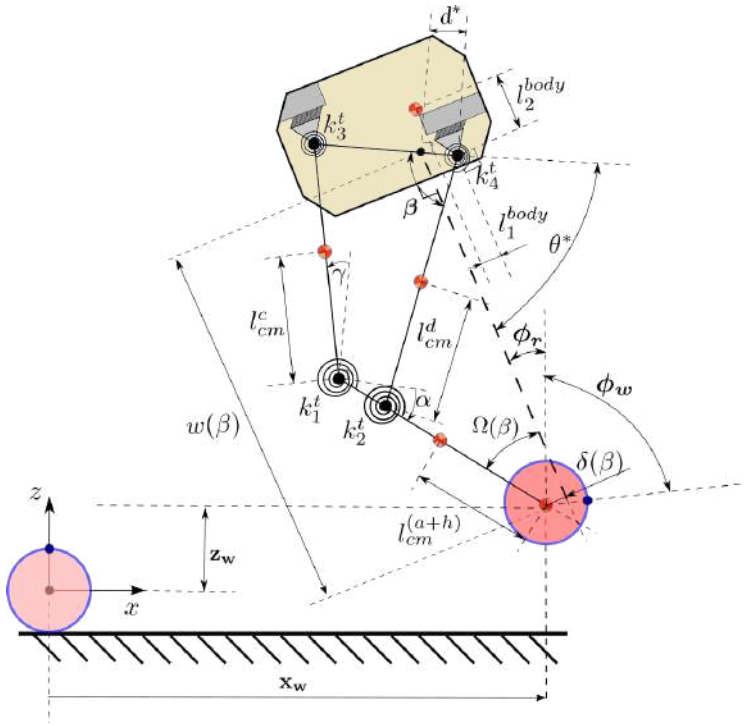



**Figure 3.12.** Complete floating-base planar model

The key is to remove the hypothesis of ground contact by extending the system's state vector and then reintroduce ground reactions as additional inputs to the dynamics. This procedure is usually called *dynamics extension* (see [42]). Ground reactions, as all other inputs, will have an associated Lagrangian component which will act on some of the d.o.f.s of the system. Their contribution will enter the dynamics through a suitable *contact Jacobian*.

Fig. 3.12 is a representation of the floating-base model. It is very similar to 3.7, the only difference being that now the system has two additional Lagrangian coordinates, $x_w$ and $z_w$. Note that $z_w$ is measured from an height equal to wheel's radius.

The new Lagrangian state becomes

$$q_p(t) \coloneqq \begin{bmatrix} x_w(t) \\ z_w(t) \\ \phi_w(t) \\ \phi_r(t) \\ \beta(t) \end{bmatrix} \tag{3.67}$$

The positions of the links' CoM are simply computed as

$$p_{cm}^{w} = \begin{pmatrix} x_w \\ z_w \end{pmatrix} \tag{3.68a}$$

$$p_{cm}^{(a+h)} = \begin{pmatrix} x_w - l_{cm}^{(a+h)} \sin\left(\phi_r - \alpha + \theta^*\right) \\ z_w + l_{cm}^{(a+h)} \cos\left(\phi_r - \alpha + \theta^*\right) \end{pmatrix} \tag{3.68b}$$

$$p_{cm}^{d} = \begin{pmatrix} x_w - l_h \sin\left(\phi_r - \alpha + \theta^*\right) + l_{cm}^{d} \sin\left(\beta + \phi_r + \theta^*\right) \\ z_w + l_h \cos\left(\phi_r - \alpha + \theta^*\right) - l_{cm}^{d} \cos\left(\beta + \phi_r + \theta^*\right) \end{pmatrix} \tag{3.68c}$$

$$p_{cm}^{c} = \begin{pmatrix} x_w - \sin\left(\phi_r - \alpha + \theta^*\right)\left(l_a + l_h\right) + l_{cm}^{c} \cos\left(\gamma + \phi_r + \theta^*\right) \\ z_w + \cos\left(\phi_r - \alpha + \theta^*\right)\left(l_a + l_h\right) + l_{cm}^{c} \sin\left(\gamma + \phi_r + \theta^*\right) \end{pmatrix} \tag{3.68d}$$

$$p_{cm}^{body} = \begin{pmatrix} x_w - l_h \sin\left(\phi_r - \alpha + \theta^*\right) - l_{cm}^{body} \cos\left(\phi_r - \theta_{body}^*\right) + l_d \sin\left(\beta + \phi_r + \theta^*\right) \\ \\ z_w + l_h \cos\left(\phi_r - \alpha + \theta^*\right) - l_{cm}^{body} \sin\left(\phi_r - \theta_{body}^*\right) - l_d \cos\left(\beta + \phi_r + \theta^*\right) \end{pmatrix} \tag{3.68e}$$

The only difference between (3.26a) - (3.26e) and (3.68a) - (3.68e) is the insertion of $x_w$ where before was $\phi_w\, r_w$ (there is no contact constraint) and the addition of $z_w$ to the second element of each position vector.

Like before, the velocities of the CoMs will take the form

$$v_{cm}^{w} = J^{w}\, \dot{q}_p \tag{3.69a}$$

$$v_{cm}^{(a+h)} = J^{(a+h)}\left(\phi_r, \beta\right) \dot{q}_p \tag{3.69b}$$

$$v_{cm}^{d} = J^{d}\left(\phi_r, \beta\right) \dot{q}_p \tag{3.69c}$$

$$v_{cm}^{c} = J^{c}\left(\phi_r, \beta\right) \dot{q}_p \tag{3.69d}$$

$$v_{cm}^{body} = J^{body}\left(\phi_r, \beta\right) \dot{q}_p \tag{3.69e}$$

where now all velocity Jacobians $J^i \in \mathbb{R}^{2x5}$. In the same way of Section 3.4.2, the kinetic energies take the are

$$E_k^w\left(\dot{x}_w, \dot{z}_w, \dot{\phi}_w\right) = \frac{1}{2} m^w \left[\dot{q}_p^T \left(J^w\right)^T J^w \dot{q}_p\right] + \frac{1}{2} I_{yy}^w \dot{\phi}_w^2 = \frac{1}{2} \dot{q}_p^T M^w \dot{q}_p \tag{3.70a}$$

$$\begin{aligned} E_k^{(a+h)}\left(\phi_r,\, \beta,\, \dot{x}_w, \dot{z}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2} m^{(a+h)} \left[\dot{q}_p^T J^{(a+h)^T} J^{(a+h)} \dot{q}_p\right] + \\ + \frac{1}{2} I_{yy}^{(a+h)} \left(\dot{\phi}_r - \dot{\alpha}\right)^2 = \frac{1}{2} \dot{q}_p^T M^{(a+h)} \dot{q}_p \end{aligned} \tag{3.70b}$$

$$\begin{aligned} E_k^d\left(\phi_r,\, \beta,\, \dot{x}_w, \dot{z}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2} m^d \left[\dot{q}_p^T \left(J^d\right)^T J^d \dot{q}_p\right] + \\ + \frac{1}{2} I_{yy}^d \left(\dot{\phi}_r + \dot{\beta}\right)^2 = \frac{1}{2} \dot{q}_p^T M^d \dot{q}_p \end{aligned} \tag{3.70c}$$

$$\begin{aligned} E_k^c\left(\phi_r,\, \beta,\, \dot{x}_w, \dot{z}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2} m^c \left[\dot{q}_p^T \left(J^c\right)^T J^c \dot{q}_p\right] + \\ + \frac{1}{2} I_{yy}^c \left(\dot{\phi}_r + \dot{\gamma}\right)^2 = \frac{1}{2} \dot{q}_p^T M^c \dot{q}_p \end{aligned} \tag{3.70d}$$

$$\begin{aligned} E_k^{body}\left(\phi_r,\, \beta,\, \dot{x}_w, \dot{z}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = \frac{1}{2} m^{body} \left[\dot{q}_p^T \left(J^{body}\right)^T J^{body} \dot{q}_p\right] + \\ + \frac{1}{2} I_{yy}^{body} \dot{\phi_r}^2 = \frac{1}{2} \dot{q}_p^T M^{body} \dot{q}_p \end{aligned} \tag{3.70e}$$

The total kinetic energy becomes

$$E_k^{tot}\left(\phi_r,\, \beta,\, \dot{x}_w, \dot{z}_w,\, \dot{\phi}_w,\, \dot{\phi}_r,\, \dot{\beta}\right) = 2\left(E_k^w + E_k^{(a+h)} + E_k^d + E_k^c\right) + E_k^{body} = \frac{1}{2} \dot{q}_p^T M \dot{q}_p \tag{3.71}$$

with

$$M(\phi_r, \beta) = 2\left(M^w + M^{(a+h)} + M^d + M^c\right) + M^{body} \tag{3.72}$$

The total elastic energy, since it only depends upon the leg configuration, remains exactly the same as (3.34).
Using the second coordinates of expressions from (3.68a) to (3.68e), the gravitational potential energies can be written as

$$E_{p,g}^w(z_w) = m_w\, g\, h_{cm}^w \tag{3.73a}$$

$$E_{p,g}^{(a+h)}\left(z_w, \phi_r, \beta\right) = m^{(a+h)}\, g\, h_{cm}^{(a+h)} \tag{3.73b}$$

$$E_{p,g}^d\left(z_w, \phi_r, \beta\right) = m^d\, g\, h_{cm}^d \tag{3.73c}$$

$$E_{p,g}^{c}\left(z_w, \phi_r, \beta\right) = m^c \, g \, h_{cm}^{c} \tag{3.73d}$$

$$E_{p,g}^{body}\left(z_w, \phi_r, \beta\right) = m^{body} \, g \, h_{cm}^{body} \tag{3.73e}$$

Once again, the total gravitational potential energy becomes

$$E_{p,g}^{tot}\left(z_w, \phi_r, \beta\right) = 2\left(E_{p,g}^{w} + E_{p,g}^{(a+h)} + E_{p,g}^{d} + E_{p,g}^{c}\right) + E_{p,g}^{body} \tag{3.74}$$

Finally, the total potential energy is

$$E_{p}^{tot}\left(z_w, \phi_r, \beta\right) = E_{p,g}^{tot}\left(z_w, \phi_r, \beta\right) + E_{p,el}^{tot}\left(\beta\right) \tag{3.75}$$

In analogy to Section 3.4.2, the following quantities are also defined:

$$e^{tot}\left(\phi_r, \beta\right) \coloneqq \begin{bmatrix} e_1^{tot} \\ e_2^{tot} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ e_{15}^{tot} \end{bmatrix} \tag{3.76}$$

$$\nabla_{q_p}\left(e^{tot}\right)\left(\phi_r(t), \beta(t)\right) \coloneqq \begin{bmatrix} \frac{\partial e_1^{tot}}{\partial x_w} & \frac{\partial e_1^{tot}}{\partial z_w} & \frac{\partial e_1^{tot}}{\partial \phi_w} & \frac{\partial e_1^{tot}}{\partial \phi_r} & \frac{\partial e_1^{tot}}{\partial \beta} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial e_{15}^{tot}}{\partial x_w} & \frac{\partial e_{15}^{tot}}{\partial z_w} & \frac{\partial e_{15}^{tot}}{\partial \phi_w} & \frac{\partial e_{15}^{tot}}{\partial \phi_r} & \frac{\partial e_{15}^{tot}}{\partial \beta} \end{bmatrix} \tag{3.77}$$

The Lagrangian components of the control inputs are exactly the same as (3.37a) - (3.37c), since they still act on the same degree of freedom.

Now, it is necessary to include the contact forces effect into the dynamics. Firstly, contact forces, in the case of planar motion, are made of simply two components: a horizontal and a vertical one.
Since the wheel is considered rigid, assuming the robot is in contact with the ground through the wheel, the contact point will always lay exactly below the wheel center. For this reason, the vertical component of the ground reaction will only do work on the $z_w$ degree of freedom.
Differently, the horizontal component, which will also act on the contact point, will give its contribution to both $x_w$ and $\phi_w$.
Let us define the contact force as

$$\lambda(t) \coloneqq \begin{bmatrix} T(t) \\ N(t) \end{bmatrix} \tag{3.78}$$

It is important to note how the elements of (3.78), are actually the total horizontal and vertical reactions acting on both wheels. This means that, if the motion is planar, the contact force acting on a single wheel is actually half of (3.78).

This been said, the contact Lagrangian inputs to the dynamics are given by

$$F_\lambda(t) = J_\lambda \, \lambda \tag{3.79}$$

where

$$J_\lambda = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & -r_w \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.80}$$

is the previously mentioned contact Jacobian.

The last step to writing the dynamics is to define the following matrices

$$K_{x_w} \coloneqq \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.81}$$

$$K_{z_w} \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.82}$$

$$K_{\phi_w} \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.83}$$

$$K_{\phi_r} \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.84}$$

$$K_\beta \coloneqq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.85}$$

Proceeding in exactly the same way of Section 3.4.2, the dynamics can be written as:

$$M(q_p(t))\,\ddot{q}_p(t) + C(q_p(t), \dot{q}_p(t)) = G^{'}(q_p(t))\,u(t) + J_\lambda\,\lambda(t) \tag{3.86}$$

where now

$$C(q_p, \dot{q}_p) \coloneqq -\left[\nabla_{q_p}^T\left(e^{tot}\right) - \left(\frac{\partial M}{\partial x_w} K_{x_w} + \frac{\partial M}{\partial z_w} K_\beta + \frac{\partial M}{\partial \phi_w} K_{\phi_w} + \right.\right. \\ \left.\left. + \frac{\partial M}{\partial \phi_r} K_{\phi_r} + \frac{\partial M}{\partial \beta} K_\beta\right)\right] \dot{q}_p^{quad} + \nabla_{q_p}^T\left(E_p^{tot}\right) \tag{3.87}$$

and

$$G'(\beta) \coloneqq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \\ -2\,h_1 & 2 & 2\,h_2 \end{bmatrix} \tag{3.88}$$

while $u$ is still (3.50).

Note that $\frac{\partial M}{\partial x_w}$, $\frac{\partial M}{\partial z_w}$ and $\frac{\partial M}{\partial \phi_w}$ are actually null. They were included to make the discussion more general and also applicable to different systems.
Dynamics (3.86) is valid during contact, but also during a possible *flight phase*. If the contact breaks, then simply $\lambda = [0, 0]$; otherwise $\lambda$ acts in compliance with the used contact model.

During contact, using the same hypothesis made in Section 3.4.2, the following kinematic constraint in Pfaffian form must hold:

$$A_\lambda\, \dot{q}_p = 0 \tag{3.89}$$

where

$$A_\lambda = \begin{bmatrix} 1 & 0 & -r_w & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.90}$$

The constraint (3.89) restricts $\dot{q}_p$ to belong to $\ker(A_\lambda)$.
It is easy to show that the kernel of $A_\lambda$ is generated by

$$S = \begin{bmatrix} r_w & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.91}$$

Now, note that $J_\lambda = A_\lambda^T$. This is not by chance.
It is however crucial to highlight that the contact matrix $A_\lambda$ is not unique: for example, multiplication of a row by a constant, does not alter its validity. This means that, in general, care must be taken when inserting contact actions in (3.86) to ensure that the chosen constraint matrix is physically meaningful. If this is not done, then $\lambda$ might not physically represent the contact forces.
A simple way to verify the consistency of $A_\lambda$, as previously done, is to check the Lagrangian components of contact forces.

Employing the knowledge of (3.91), it is possible to retrieve dynamics (3.54) starting from (3.86). Let us see how.

Firstly note that, since $\dot{q}_p \in \mathrm{Ker}(A_c)$

$$\dot{q}_p = S\,\dot{\nu} \tag{3.92}$$

where

$$\nu \coloneqq \begin{bmatrix} \phi_w \\ \phi_r \\ \beta \end{bmatrix} \tag{3.93}$$

Vector $\dot{\nu}$ is usually called *quasi-velocity* vector, while (3.92) is a kernel decomposition of $\dot{q}_p$.
By substituting the kernel decomposition of $\dot{q}_p$ in (3.86) and pre-multiplying the equation by $S^T$,

$$S^T\,M\,S\,\ddot{\nu} + S^T\,C = S^T\,G'\,u + \cancel{S^T J_\lambda \lambda} \tag{3.94}$$

where

- $S^T\,M\,S$ coincides with the mass matrix of (3.54).
- $S^T\,C$ is exactly the same as (3.55).
- $S^T\,G'$ is (3.52).
- $S^T\,J_\lambda$ is null, since $(S^T\,J_\lambda).^T = J_\lambda^T\,S = A_\lambda\,S$ and $S$ generates the kernel of $A_\lambda$

Dynamics (3.94) and (3.54) are completely equivalent. This fact was also verified symbolically through Matlab.
Equation (3.94) proves to be very useful when used in the intermediate form

$$M\,S\,\ddot{\nu} + C = G'\,u + J_\lambda\,\lambda \tag{3.95}$$

Note that dynamics (3.95) is still five-dimensional, in contrast with (3.94), which is instead three-dimensional.

From the first two equations of (3.95) it is possible to obtain an analytical expression for the ground reactions forces during contact. In fact, looking at (3.88), it is clear that the control input $u$ does not enter the first two equations. In particular, extracting the second equation and rearranging, we obtain

$$\begin{aligned} N(t) = {} & n_1(\beta,\phi_r)\,\ddot{\phi}_r + n_2(\beta,\phi_r)\,\ddot{\beta}_r + n_3(\beta,\phi_r)\,\dot{\phi}_r^2 + \\ & + n_4(\beta,\phi_r)\,\dot{\beta}_r^2 + n_5(\beta,\phi_r)\,\dot{\phi}_r\,\dot{\beta} + n_6 \end{aligned} \tag{3.96}$$

where, as expected, $n_6 = g\left[2\left(m^w + m^{ah} + m^c + m^d\right) + m^{body}\right]$, which is the total robot weight.
While, performing the same operations on the first equation, we obtain

$$\begin{aligned} T(t) = {} & t_1(\beta,\phi_r)\,\ddot{\phi}_w + t_2(\beta,\phi_r)\,\ddot{\phi}_r + t_3(\beta,\phi_r)\,\ddot{\beta}_r + \\ & + t_4(\beta,\phi_r)\,\dot{\phi}_r^2 + t_5(\beta,\phi_r)\,\dot{\beta}_r^2 + t_6(\beta,\phi_r)\,\dot{\phi}_r\,\dot{\beta} \end{aligned} \tag{3.97}$$

which, as expected, is null when the robot is not moving.
Equations (3.96) and (3.97) were used for evaluating the ground reactions during simulations performed in Matlab.

## 3.7 Floating base planar model of the eWSLIP model

With exactly the same reasoning of Section 3.6, it is also possible to obtain a floating-base model for the eWSLIP.

Here a new picture of the model is not shown, since the way system (3.11) is extended to a floating one is the same for passing from (3.7) to (3.12). The contact hypothesis are exactly the same w.r.t. Section (3.6). All the considerations are perfectly extensible.
In this case, however, the expanded state is

$$q_p(t) \coloneqq \begin{bmatrix} x_w(t) \\ z_w(t) \\ \theta_w(t) \\ \theta_r(t) \\ l^p_{cm}(t) \end{bmatrix} \tag{3.98}$$

The derivation of the dynamics is perfectly equivalent w.r.t. previous sections.
Once again, one obtains a dynamics in the form

$$M(q_p(t))\,\ddot{q}_p(t) + C(q_p(t), \dot{q}_p(t)) = G^{'}(q_p(t))\,u(t) + J_\lambda\,\lambda(t) \tag{3.99}$$

where $\lambda$ and $J_\lambda$ are still the same as (3.86).
In this case

$$G^{'} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.100}$$

and

$$u = \begin{bmatrix} C^{eq}_{m1} \\ F_{eq} \end{bmatrix} \tag{3.101}$$

where $C^{eq}_{m1}$ and $F_{eq}$ can be expressed through the equivalence relationships as, respectively, (3.65) and (3.66).

Furthermore, the mass matrix takes the form

$$M(\theta_r, l^p_{cm}) = \begin{bmatrix} m^p_{eq} + 2\,m_w & 0 & 0 & -l^p_{cm}\,m^p_{eq}\,\cos(\theta_r) & -m^p_{eq}\,\sin(\theta_r) \\ 0 & m^p_{eq} + 2\,m_w & 0 & -l^p_{cm}\,m^p_{eq}\,\sin(\theta_r) & m^p_{eq}\,\cos(\theta_r) \\ 0 & 0 & 2\,J_w & 0 & 0 \\ -l^p_{cm}\,m^p_{eq}\,\cos(\theta_r) & -l^p_{cm}\,m^p_{eq}\,\sin(\theta_r) & 0 & m^p_{eq}\,{l^p_{cm}}^2 + I^p_{eq}(l^p_{cm}) & 0 \\ -m^p_{eq}\,\sin(\theta_r) & m^p_{eq}\,\cos(\theta_r) & 0 & 0 & m^p_{eq} \end{bmatrix} \tag{3.102}$$

while vector $C$ is not shown for brevity.

A very similar system is used by [12]. Differently from this article, here the model does not neglect the pendulum's rotational inertia (can be seen by comparing mass matrices) and possesses various additional terms due to the dependence of $I^p_{eq}$ and $k_{eq}$ on $l^p_{cm}$ (or, equivalently, if referring to the parent model (3.12), $\beta$).

Moreover, it is possible to proceed in the same way as in Section 3.6. Once again, by substituting the kernel decomposition of $\dot{q}_p$ in (3.99) and pre-multiplying the equation by $S^T$, we obtain the reduced dynamics described by

$$S^T M S \dot{\nu} + S^T C = S^T G' u + \cancel{S^T J_\lambda \lambda} \tag{3.103}$$

# Chapter 4

# Simulation environments

This chapter contains a synthetic description of the simulation environments which were used for testing the control policies.
Simulations were carried out mainly on a custom simulator written in Matlab.
This simulator has a whole series of characteristics which make it particularly suited for testing control policies. In particular, the current version of the simulator uses the Lagrangian floating-base model (3.86). The insertion of more complex ones, due to how the simulator is structured, is only a matter of modifying properly some functions (this is done very easily via Matlab's Symbolic Toolbox) and adapting the plotting utility.

Towards the end of the thesis, with the purpose of testing the controls on high fidelity dynamics simulators, Gazebo, an Open Source dynamics simulator widely used in research (and not only), was employed. To perform simulations in Gazebo, two fundamental elements are necessary: an SDF model of the robot and a control plugin. Writing those two elements took quite a lot of time, but it was definitely worth it.

Due to time constraints, only the controls described in Section 6.5 and 6.6 were tested in Gazebo; however, once a model is available, inserting different controllers is conceptually very simple, especially using the adopted workflow for migrating controls written in Matlab to Gazebo (more details on this in Section 4.2).

Testing controllers in Gazebo is also very useful for evaluating the real time performance of controllers, since the control plugins are written in C++ and, usually, this is the used language for deploying real time software on control hardware. We will see in Section 4.2, that the difference in performance between Matlab's code and the one written in C++ is quite significant (this is to be expected, being Matlab an interpreted language and C++ a compiled one).

## 4.1 Custom Matlab simulator key features and brief description

This simulator integrates the Lagrangian model associated with (3.6). In particular, the system turned out to be stiff and, as a consequence, instead of the classical ode45 integrator, ode15s was used. Trying to use ode45 on this system only results

in extremely long simulation time and decreased accuracy.

The script directly loads optimal dimensions obtained via the dimensional optimization process; these results are stored in a file inside the working directory. This simple feature allows rapid adaptation of the simulation, in case a different dimensional synthesis was chosen.
For all simulation, the gravity vector is assumed to be perfectly vertical (parallel to the $z$ axis) and with module $g = 9.81$.
All other necessary physical parameters are defined in line with (5.29) - (5.35).
Initially, the integrated model was the simpler (3.54), which was ok to test controls during ground contact. However, it lacked any kind of estimate of the ground reaction forces and, as a consequence, did not account for possible wheel separations which may occur during particularly fast manoeuvres.

Later, model (3.6) was integrated into the framework and this allowed an accurate estimate of the ground reactions. Dynamics (3.40) is integrated considering the contact force $\lambda$ as an additional input to the system. As a consequence, during ground contact, this force needs to be computed to be able to integrate the dynamics. To do this, during ground contact, the intermediate equation (3.95) is employed. As soon as the vertical component of the ground reaction becomes $\leq 0$, the model is integrated with $\lambda = 0$. This allows to properly manage the "takeoff" phase.

A boolean parameter is used to distinguish between flight and contact phases. If the model has left the ground at a certain integration step, then at the next one the integration will know the robot is still in "flight" and, consequently, keeps $\lambda = 0$. The handling of the so called "touchdown" phase was quite tricky. At first, an approach with contact rigidity and damping was tried, with little success. Moreover, using such an approach on a handwritten simulator can make the simulation too much dependent upon the selected contact parameters.

The solution was to employ the so called event functions, which can be passed as an additional argument to the integrator's options. Basically, an event function detects when an event happens and, if requested, can stop the integration. This mechanism was used to detect when the robot touches the ground. In particular, the event function detects when the wheel's $z$ coordinate crossed the x axis with a negative velocity and stops the integration. Using also the negative velocity condition avoids to trigger the event function during normal ground contact. The last state before the integrator stops is then used as input to a touchdown function. This function uses the available state to compute a "state after touchdown". Specifically, the function was set so that after the touchdown $z_w = 0$, $\dot{z}_w = -k\,\dot{z}_w^0$ and $\dot{\phi}_w = r_w\,\dot{z}_w^0$, where $k$ is a coefficient of restitution (for the presented simulations, always set to 0) and the superscript $^0$ refers to quantities before the touchdown. This modified state is then used as initial conditions to start a new integration. So, basically, when the robot hits the ground it instantaneously grips on the surface and the vertical motion of the wheel is stopped (this basically implies that the wheel never bounces). The contact surfaces (actually points) of the wheel and the ground are hence considered infinitely rigid.
Clearly, this is not the best possible contact model, but it is good enough for testing control algorithms (and definitely better than using the model during contact).

Another problem arises when the linkage (for example during a failed control)

extends or contracts too much, thus reaching its kinematic limits. In this case the simulation fails. To avoid this and also to model a more realistic behaviour of the legs, the simulation model is set up in such a way that, when linkage limits are almost reached, dissipative and elastic reaction torques appear. This way, the behaviour of a real linkage is simulated. In particular, the parameters characterizing this stopping reactions are the following:

$$k_{diss} = 1000 \tag{4.1a}$$

$$k_{el} = 10000 \tag{4.1b}$$

$$\delta_{diss}^{block} = 0.01^\circ \tag{4.1c}$$

$$\delta_{el}^{block} = 0.01^\circ \tag{4.1d}$$

where the first two parameters characterize how strong the reactions (both elastic and dissipative) should be, while the other two define how close to the leg's limit angles the reactions should start to act.

The simulation can obviously be carried starting from any set of initial conditions. Moreover, the simulator allows to select directly which kind of control approach to use, as well as many other features which are not reported here for simplicity. The only control strategy which was implemented in a separate main script is 6.5. All the other controls are testable from within the same main script.

To introduce further elements of real-world fidelity, there are two important parameters which can be varied: the control loop frequency and the sensing (or sampling) frequency. The first is the frequency at which the actual control loop runs (manipulation of sensors data and computation of control actions). The second is the rate at which a measurement is available. Generally the first is much higher than the second. For example, as will be seen in Chapter 7, the state of the prototype is available at a maximum frequency of 100 Hz, while the control loop, if written in C++, can run at one order of magnitude higher, if not more.

To understand why this two parameters actually play a very important role and to appreciate the reason why implementing them in the simulation is useful, let us look at Fig. 4.1.
When the controller is started (at $t_0$), there is a delay of $\delta_s + \delta_c$ before the first control input is applied. This is basically equivalent to starting from a "delayed" initial condition and can produce significant differences, if the sampling time is big enough.
After that, if the sensing and control computation can be run in parallel, then the control loop will actually run at a rate of $1/\delta_s$. During simulations, $\delta_c$ was kept at a very low value to simulate an instantaneous control loop, while $\delta_s$ is varied according to simulation's the specific needs.

Another important aspect which is clear from Fig. 4.1 is that the state available to the controller at time $t_k$ is actually the state at time $t_{k-1}$. This delay can have detrimental effects on the controller. In the simulator there is an option to use the actual state at time $t_k$ instead of the one relative to $t_{k-1}$. In this case, the simulator

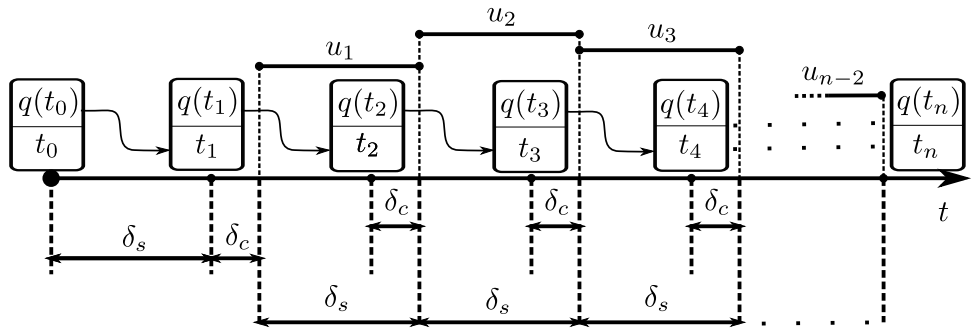



**Figure 4.1.** Control timing diagram

acts like if the measurement at the previous instant of time was *forward propagated* to the current instant of time by the controller. This behaviour can be implemented, if a good enough model of the system is available, in a real system by integrating numerically the known previous state up to the current sampling time. Clearly, this introduces a computational overhead, so its effect on $\delta_c$ would have to be evaluated.

Furthermore, the simulator has an option to evaluate the real time Matlab takes to compute the control input. If this time is bigger than $\delta_s$, then the previous input is used. Since we saw that controllers written in more efficient languages like C++ have considerably higher performance, performing this check in Matlab may have little sense. It is however useful to have an initial idea as to if a hypothetical C++ controller would be fast enough or not without actually implementing it.

The simulator also allows inserting disturbances on each d.o.f. . Clearly, given a general disturbance to the system, it is first necessary to compute its Lagrangian component; only then, it is possible to insert it in the simulator.

Now, let us look briefly at the workflow of the simulator.
After having loaded and initialized all the parameters, variables, reference trajectories (if any), etc.. the simulator starts the integration of the dynamics. The integrator resides inside a loop running at a virtual frequency equal to the sampling frequency. States between adjacent control intervals are passed to the next integrator instance. This is the easiest way to make the input constant inside each control loop.
When the integration is finished, a dedicated function plots a series of graphs.
At the current state of the simulator, the plotted graphs are:

- State evolution over time (generalized positions and their derivatives).
- Disturbances (if present).
- Actuation inputs.
- "Limit switch" actions (present only if linkage limits are approached).
- Ground reaction forces.
- Absolute and relative (to the wheel center) CoM trajectories.

After this, using results from the integration of the dynamics, the simulator produces an animation of the robot over time. In particular, the plotted frames are captured

and then saved in a movie. The movie generation is set up in a way that the video will run in "real time".
Finally, if necessary, simulation results and graphs can be saved to a chosen file.

## 4.2 Gazebo

Gazebo is an Open Source dynamics simulator developed and maintained by the Open Source Robotics Foundation.
As specified by [32], it "offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments".
Gazebo is made up by two different executables: "gzserver", which runs the physics control-loop and sensor data generation, and a client "gzclient" which basically provides the Gazebo's GUI. The client server can be used independently of the graphical interface.
As of now, Gazebo can use a total of four different physics engines: ODE (default, see [35]), DART (see [13]), Bullet (see [7]) and Simbody (see [43]). After many trials with other engines, at the end ODE was used.

All robot are hosted into a so called *world*, which is described by an XML file with extension .world. The world file is used to load *models* into gzserver (robots, ground plane, light sources, cameras...), to set up the simulation scene, to set the physics engine and more.

Models are contained inside dedicated XML files, in particular SDF files. SDF stands for "Simulation Description Format" and it is used by Gazebo for describing objects and environments within the simulator (see [33]). It contains information on all the aspects of the model, including links, joints, collision parameters and so on and so forth. It is an alternative to the also popular URDF format or the newer SRDF, over which has several crucial advantages; the most relevant one being that it supports graph structures in the robot and thus allows for closed loop kinematic chains. Models are made up by rigid links, connected by joints. The collisional aspects of each link is defined by a collision geometry, which can be both chosen from elementary forms or specified via meshes. The appearance of the robot is defined by the visual component of each link, which has to be provided as a Collada (.dae) file. It is advisable to avoid using complicated meshes for the collision models and stick with standard and simple shapes.

The interaction with models inside a world happens through the so called *plugins*. Plugins are line of code, written in C++, which make use of Gazebo C++ API and allow to control almost every aspect of the simulation. This includes, for instance, applying forces and torques to the robot. In particular, there are several types of plugins: World, Model, Sensor, System, Visual and GUI plugins. For the purporse of this thesis, i.e. writing controllers, a Model plugin in more than enough.

Now, let us explore the main characteristics of the used configuration for carrying out Gazebo simulations.
The first element to look at is the world file (it uses the SDF format). Looking at Fig. 4.2, one can distinguish:

- The insertion of a custom ground model. It is simply a copy of the classical one

and has the purpose of allowing modifications of ground's contact parameters (specifically, its static friction coefficient).

- The insertion of the prototype model.
- The insertion of a light source ("sun").
- The insertion of a camera model, with a whole series of settings. Its function is to output image frames from the simulation for video creation.
- A scene tag, used to set shadow properties and modify other aesthetics features.
- A physics tag, used to set physics engines' properties and also the gravity vector.

```
<?xml version="1.0"?>
  <sdf version="1.7">
    <world name="default">
      <!-- A custom ground plane -->
      <include>
        <uri>model://custom_ground_plane</uri>
      </include>
      <!--<include>
        <uri>model://ground_plane</uri>
      </include>-->
      <!-- A global light source -->
      <include>
        <uri>model://sun</uri>
        <pose>0 0 10 0 0 0</pose>
      </include>
      |
      |
      <physics type="ode">
              <ode>
                <solver>
                  <sor>0.8</sor>
                  <iters> 100 </iters>
                  <friction_model> cone_model</friction_model>
                </solver>
                <constraints>
                </constraints>
              </ode>
              <bullet> </bullet>
              <dart>    </dart>
              <simbody> </simbody>
      </physics>
      <gravity> 0 0 -9.81 </gravity>
     </world>
    </sdf>
```

**Figure 4.2.** .world file structure

Secondly, let us look at the used models.
The ground plane is simply a plane, with contact area $100 \times 100\,\mathrm{m}^2$, contact stiffness $k_p$ equal to $10^{12}\,\mathrm{N/m}$, contact dissipation $k_d$ equal to $1\,\mathrm{N\,s/m}$ and static friction coefficients $\mu_1$ and $\mu_2$ equal to 1. This chosen set of parameters allows to model the ground almost as an infinitely rigid body. Desired contact and collision properties are instead assigned to wheels' links.

The sun is a standard light source with pose $[0, 0, 10, 0, 0, 0]$.
The camera is positioned at the desired position and set up to save frames to a specified directory. Its complete settings are not here reported for simplicity.

```
<?xml version="1.0" ?>
 <% #Ruby code begins ("scriplet")
  # all SI units
  require "matrix"
  def alpha_gamma_calc(beta)
    #computes alpha and gamma angles for a given input beta (vectorial output)
    l_a=$link_a_lenght
    l_b=$link_b_lenght
    l_c=$link_c_lenght
    l_d=$link_d_lenght
   |
   |
  %>
 <sdf version="1.7">
    <model name="my_robot">
      <plugin name="proto_controller" filename="libproto_controller.so"/>
      <pose> <%=a_to_s(pose_model ) %> </pose>
      <static>false</static>
      <link name='left_wheel'>
            <pose> <%=a_to_s(pose_link_lft_wheel ) %> </pose>
            <inertial>
              <pose><%=a_to_s(pose_inertial_lft_wheel )%> </pose>
              <mass><%=mass_lft_wheel%></mass>
              <inertia>
                <ixx><%=inertia_lft_wheel[0]%></ixx>
                 |
                 |
    </model>
  </sdf>
```

**Figure 4.3.** Model .sdf.erb file structure

Now, for generating the prototype's SDF file, a different approach was employed w.r.t. the world file. It is particularly useful, expecially with complicated models as the one used, to parametrize the SDF file. In these cases, the SDF can become quite long and, with this approach, modifying the file becomes really easy.
In the particular case of the used model, the SDF file is made of more than 580 lines of code. It is clear that modifying a parameter of the model without using parametrized files becomes a real nightmare.

The parametrization of an SDF file can be made using a particular programming language called RUBY (file extension .erb). Basically, the .erb file is made of an initial part, in which all useful variables (also functions) are defined, and then a second part which is the SDF file of the robot, with all necessary parameters specified through the previously defined variables. Once the .erb file is executed, it generates the SDF file substituting all variables with their values.
This means that each time the model is modified in the .erb file, the SDF file needs to be regenerated. Such a process is made really straightforward by employing a very simple dedicated bash script which was written for the purpose.
In the .erb file all necessary SDF parameters are computed e/o assigned. These include, for example, inertial properties, friction coefficients, kinematic properties, links, joints, visual elements, etc.

Fig. 4.3 shows an extract on the .erb file which highlights its structure. Clearly, all physical parameters were set to be the same used in Matlab's simulations. This way, a comparison between the two simulators is possible.

Since the wheel is made of a rubber-like material and the operational surfaces would reasonably be concrete-like grounds, a static friction coefficient of 0.5 was set on left and right wheel links (this is a quite conservative value). When handling contact between two bodies, Gazebo selects the lower friction coefficient between the two. Since ground's static friction coefficient was purposely set to 1, the simulator will always use the value assigned to the wheel link. Other contact parameters, like for instance $k_p$ and $k_d$, were set to adequate values in order to avoid contact flickering issues and excessive ground penetration.
The resulting model, as shown by Gazebo GUI, is depicted in Fig. 4.4.

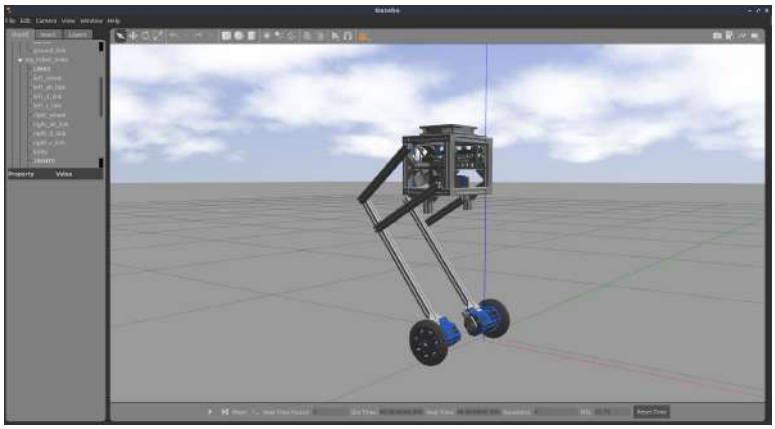


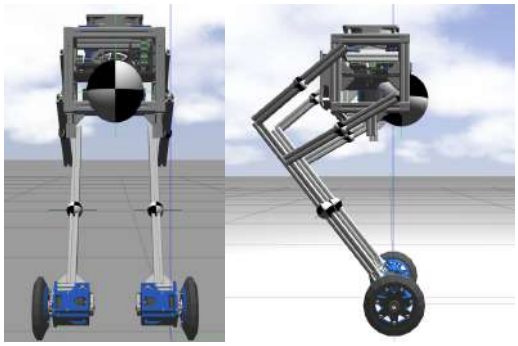
**Figure 4.4.** Spawned model in Gazebo

Fig. 4.5b shows the global $xyz$ reference frame, as well as links and joints local frames. Fig. 4.5a shows the positioning of links' CoMs; clearly, the bigger the sphere, the bigger the mass. All visual elements in these figures were imported by using a .dae file (Collada). To generate a Collada file a possible workflow is the following: export a CAD model to .obj file, open the .obj file in Blender (see [4]), if necessary modify it and export it to a .dae file.

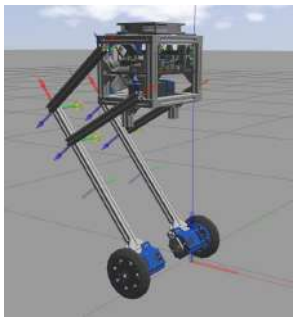
Figures 4.5 - 4.6 contains synthetic CAD representations of the electronics box, the wheel actuator and the linkage. Since the linkage is not yet designed (due to the lack of a suitable leg actuator), here only the aluminum links are shown. To look at a brief description of all used CAD models, refer to Chapter 7.

One crucial point is that link's visual and collision elements are two completely independent entities. The first is only used for visualization, while the second is used to handle contact and collision. As already specified, it is advisable to use as simple as possible collision geometries; this aspect is clear in Fig. 4.6, where the collision elements are shown in orange. We see that, while the visual elements are quite complex and detailed, collision elements are very simple geometric entities (planes, cylinders and boxes).

The last fundamental component necessary to simulate controllers in Gazebo is a Model plugin. A plugin has full access to Gazebo's API and, hence, allows for a



**(a)** Model's CoMs



**(b)** Model's links reference frames and joints

**Figure 4.5.** Model CoMs and frames

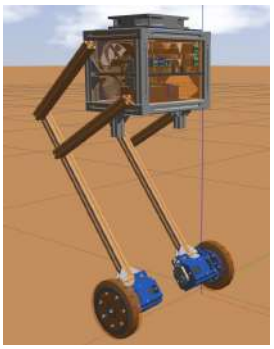


**Figure 4.6.** Collision elements

great degree of control over the simulation.
The used plugin is made of approximately 4120 lines of code and is quite articulated. However, it relies on a relatively simple structure, which is shown in Fig. 4.7. Every plugin must inherit from a Gazebo model class. In this case this is a model plugin, so it will inherit from "ModelPlugin" class.

Aside from the class constructor, a Model plugin must have a public "Load" method, which is called by Gazebo right after loading the plugin. Then, there are other optional public methods, like for instance "OnUpdate", which runs a chunk of code when updating Physics at the start of each simulation iteration. "OnUpdateEnd" and "BeforePhysicsUpdate" were also included in the plugin for performing specific operations and data manipulations. Details are here omitted for brevity.

After the main public methods, the plugins contains a series of "high-level" private method, which provide additional functionalities for running the controllers. Examples of this methods include functions to extract and assign the current state from the model, computation of reference trajectories, control input calculation, etc. Then, a series of "low-level" functions is inserted. They are addressed as low-level, because they serve as basis for the high-level functions. At the end of the code all the private attributes used by the plugin are inserted. The last line of code, which is mandatory, simply registers the control plugin to Gazebo's gz server.

```
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/common/common.hh>
#include <ignition/math/Vector3.hh>
#include <ignition/math/Pose3.hh>
#include <ignition/math/Quaternion.hh>
#include <functional>
#include <cmath>
#include <math.h>
#include <vector>
#include <iostream>
#include <fstream>
#include <Eigen/Core>
#include <chrono>
#include <limits>
#include "include/QPSolverInterface.cpp" //courtesy of F.Smaldone
#include <boost/numeric/odeint.hpp> // numerical integration

using namespace boost::numeric::odeint;
using namespace std;
using namespace Eigen;

namespace gazebo
{
  class ModelController : public ModelPlugin
  {
    public: ModelController(){//class constructor
    public: void Load(physics::ModelPtr _parent, sdf::ElementPtr /*_sdf*/) {
    |
    }
    public:void BeforePhysicsUpdate(){
    |
    }
    public: void OnUpdate(){
    |
    }
    public: void OnUpdateEnd(){
    |
    }
    private: void - - - - - - - - - - {  }// private methods
    |
    private: void - - - - - - - - - - {  }// private  "low-level"
    //private methods
    |
    private:   - - - - - - - - - -  // private attributes
    };
 // Register this plugin with the simulator
 GZ_REGISTER_MODEL_PLUGIN(ModelController)
}
```

**Figure 4.7.** Model plugin structure

The control plugin must run at a control sample interval which is an integer multiple of the integrator step size. The integrator step size is fixed to $0.001s$, which is generally a good value to obtain fast simulations and reliable results. Normally, one would simply modify the integration step size in order to match the desired control frequency; this, however, can produce problems for low control frequency (simulation accuracy decreases significantly) and of slower simulation if the controller must run at very high frequencies. While the second problem cannot be solved, the first one can: it is sufficient to check within the controller wheter or not the current simulation time is also relative to a control instant or not. This way the integration

step can be left to a value which ensures good accuracy and the control can run at an almost arbitrary control frequency. Is is clear that, using an integration step of $0.001s$, the control frequency can be set with a good degree of flexibility. This approach was implemented in the control plugin.

It is also possible to set initial conditions on joints and links; however, at the current state of Gazebo, this procedure has some problems with closed-chain linkages. As a consequence, it is not possible to set initial conditions on the leg freely without causing errors. This problem can be somehow solved by directly modifying the SDF file of the prototype, since it allows to spawn the robot with the desired leg angle. The only limitation is that it is not possible to impose initial conditions of leg angle derivative (this is not really an issue in practice).

The plugin is written in a way that physical parameters to be used by the controller can be either retrieved using the spawned model in Gazebo or, alternatively, they can be manually inserted. This might be useful when testing the controller's robustness w.r.t. parameters uncertainties.

For what concerns control timing, the plugin uses the logic of Fig. 4.1. The only difference is that, for simulations in Gazebo, $\delta_c$ is considered to be 0. As with Matlab's simulator, the plugin is written in a way that it is both possible to simulate the effect of a delayed sampled state or to simulate the propagation of such state to the current instant of time (through a hypothetical numerical integration, for example). Please note that the very first control input will only be applied after the first sampling interval (for the same reasons discussed in Section 4.1).

After the chosen simulation time expires, the plugin stops the simulation and saves a series of data to dedicated txt files. These txt files can then be used, for example by Matlab, to perform analysis on these data and to produce simulation plots (for simple quantities it would also be possible to use Gazebo plotting utility).

Now, to conclude, let us discuss the methodology used for writing the control plugin. These approach can be applied to drastically reduce the coding time and also to avoid coding errors.
Physical models are written using Matlab's Symbolic Toolbox. Using the obtained models, it is possible to employ "matlabFunction" to convert symbolic expression to functions automatically. This is particularly useful when models become quite complex and difficult to handle from a coding point of view. In such cases, writing by hand code might be very time-consuming and can be source of errors.
Once all the necessary functions are available and the dynamics is known, it is possible to start simulating and debugging the system in Matlab. After having verified the correctness of the model, it is possible to develop control policies (if necessary, with the aid of Matlab's Symbolic Toolbox). Then, the controller/s is/are tested extensively in Matlab.
If the controllers work, it is then possible to simulate them on high fidely simulators like Gazebo. Since control plugins are written in C++, it is possible to use Matlab Coder to convert automatically Matlab functions. In the specific case of this thesis, this was done using the option for generating methods inside a class. Once a dummy class is specified, it is sufficient to write a simple script which calls the method to be generated, with the proper inputs. After having carried the process, Matlab generates a lot of files. The only necessary is a .cpp file which contains the method

definition. At that point, it is sufficient to paste the method inside the plugin and, with little modifications, it is usable. This way we have passed almost seamlessly from Matlab's symbolic language to efficient C++ code. Since the functions were previously tested and validated in Matlab, a possible source of error is completely removed. Writing a very long control plugin like the one used by hand would have been otherwise almost impossible (or, at least, very time consuming).

Finally, as a side note, please note that writing a control plugin in Gazebo is actually a good step towards actually deploying that controller on real hardware. In fact, if the hardware supports C++ language (and it is indeed the case of the prototype), then migrating the controller is only a matter of relatively simple adaptations (depending on the software architecture).

# Chapter 5

# Control properties

This chapter explores some important control properties/characteristics of the type of system under study

## 5.1 Control of underactuated and non-minimum phase systems

The system under study belongs to a very important class of robots, i.e. *underactuated* ones.

**Definition 5.1.1** *A system in which the number of independent control inputs is less than the number of independent generalized coordinates describing the system itself is called underactuated*

For example, looking at model (3.6), we can count 4 independent control inputs $C_{m1}$, $C_{m2}$, $T$, $N$ and a total of 5 generalized coordinates (this is clear looking at vector (3.67)). Note that in this case $C_{m3}$ is not an independent control input, since it acts on the same d.o.f. as $C_{m2}$. This system will possess, consequently, one degree of underactuation. The degree of underactuation is directly related to the dimension of a second order constraint (which will here be addressed as *dynamical feasibility constraint*), which naturally arises when studying this kind of systems. This feasibility constraint, not to be confused with the concept of feasibility related to optimization problems, will be addressed in the upcoming sections. For now, it is enough to know that it is a linear constraint w.r.t. the second derivatives of Lagrangian coordinates. Clearly, models 3.4, 3.5 and 3.7 also represent examples of underactuated systems.

Other notable examples of underactued systems include the whole class of walking robots. The book [42] also remarks that, often, systems subject to unilateral or nonholonomic contraints are inherently underactuated.

Usually, solving control problems for underactuated systems is more challenging compared with fully actuated robots, which possess several strong characteristics the first category doesn't. For example, it can be shown (see [42]) that fully-actuated manipulators are globally *feedback linearizable*.

To understand why this property is ideed a strong one, let us report briefly

how a feedback linearizable system is defined, by looking at a single-input nonlinear input-affine system (definition borrowed from [42]).

**Definition 5.1.2** *Given the following dynamics:*

$$\dot{q} = f(q) + g(q)\, u \tag{5.1}$$

*with $f(q)$ and $g(q)$ vector fields on $\mathbb{R}^n$, the system is said to be feedback-linearizable if there exists a diffeomorphism $T : U \to \mathbb{R}^n$ (basically a nonlinear change of coordinates) with $U \subseteq \mathbb{R}^n$ and a nonlinear feedback law*

$$u = a(q) + b(q)\, v \tag{5.2}$$

*with $b(q) \neq 0$ on $U$ such that*

$$y = T(q) \tag{5.3}$$

*satisfies the linear dynamical system*

$$\dot{y} = A\, y + b\, v \tag{5.4}$$

*where*

$$A = \begin{bmatrix} 0 & 1 & 0 & . & . \\ 0 & 0 & 1 & & . \\ . & . & & . & . \\ . & . & & & 1 \\ 0 & 0 & . & . & 0 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 1 \end{bmatrix} \tag{5.5}$$

System (5.4) is said to be in the *Brunovsky canonical form.*

The nonlinear transformation described by (5.3) applied to system (5.1) produces a linear and controllable system. The nonlinear feedback law (5.2), if the system is fully feedback-linearizable, cancels out all the nonlinear dynamics of the original system. Furthermore, the linearization is said to be global if $U \equiv \mathbb{R}^n$.
It can be shown that there are necessary and sufficient conditions for $T(q)$ to exist for equation (5.1). Since these are out of the scope of this thesis, they are here omitted, but can be consulted in [42].
The crucial point of feedback-linearization is that, if the system is globally feedback-linearizable, then an outer control loop can be developed in the new coordinates $y$. The synthesis of such controller then becomes a matter of classical control design, being the new system linear. Please note that this is only possible thanks to the inner-loop control law described by (5.2).

In the case of underactuated systems, however, one can at best aim at performing a *partial feedback-linearization* of the system using a suitable nonlinear feedback law. This means a nonlinear component of the dynamics will persist and will require special care when developing control policies for the system.
The remaining nonlinear component is called *internal dynamics* and the union of the partially feedback-linearized system with the internal dynamics forms the so called *second-order normal form with input-driven internal dynamics.* As was already specified in Chapter 2, the second-order normal form depends upon the chosen output. In particular, second-order normal forms of systems (3.7) and (3.12) will be

obtained in, respectively, Sections 5.5 and 5.6. Obtaining the second order form of system (3.6) is particularly useful because, as will be clear in the upcoming sections, the equations of the forced internal dynamics can actually be as the previously mentioned dynamical feasibility constraint. The use of such a second order constraint becomes crucial, for example, to the good functioning of the Task-Space controller addressed and implemented in Section 6.6.

Once the internal dynamics associated to a specific choice of output is available, the *zero-dynamics manifold* can be retrieved.

**Definition 5.1.3** *Given an output $y = h(q)$ the zero-dynamics manifold is defined as*

$$\Gamma = \left\{ (q, \dot{q}) : h(q) = 0, \dot{h}(q) = \nabla_q h(q)\, \dot{q} = 0 \right\} \tag{5.6}$$

Dynamics belonging to $\Gamma$ are called *zero dynamics*.
By making use of the zero dynamics, it is possible to distinguish the class of the so called *non-minimum phase* systems.

**Definition 5.1.4** *A system with not asymptotically stable zero dynamics is called non-minimum phase*

In the special case of LTI systems, a non-minimum phase system is a system possessing one or more zeros belonging to the RHP (right-half-plane). This condition is easy to understand since, if the system is linear, inverting its transfer function will move the the RHP zeros to the denominator, making them poles. The inverse will consequently be unstable.

Non-minimum phase systems are a class of systems which, as well as underactuated systems, pose significant control challenges. Another property which is usually employed as a demonstration of what are the implications of having a non-minimum phase system is the so called *undershoot* phenomenon: when plotting, for example, a step response of such systems one notices that the output first goes "in the wrong direction", before actually converging to the steady state solution. This intrinsic "delay" effect is one of the main reasons why controlling this kind of systems is not as easy as with classical ones. Notable examples of non-minimum phase systems, in which the undershoot phenomenon is evident, include gimballed rockets, a plane changing altitute, chart-pole systems, a car during parallel parking, etc. As a side note, it is important to remember underactuation and non-minimum phaseness are independent properties.

Unfortunately, the system under study is both underactuated and non-minimum phase. As a consequence, controlling it can become quite difficult and challenging.

## 5.2 Equilibrium points

This section is dedicated to the study of the equilibrium points for dynamics (3.57), i.e. the complete system during contact.

**Definition 5.2.1** *Given a non-linear system*

$$\dot{q} = \mathcal{F}(q, u) \tag{5.7}$$

*an equilibrium point is a constant vector* $[q^*, u^*]$ *satisfying the condition*

$$\mathcal{F}(q^*, u^*) = 0 \tag{5.8}$$

In particular, equilibrium condition (5.8) applied to (3.57) implies

$$f(q^*) + G(q^*)\, u^* = 0 \tag{5.9}$$

The very first implication is that

$$q^* \coloneqq \begin{bmatrix} q_1^* \\ q_2^* \\ q_3^* \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.10}$$

The second implication, by looking at (3.40), is that

$$G^{'}(q^*)\, u^* = \nabla_{q_p}^T E_p^{tot}(q^*) \tag{5.11}$$

Now, recall that

$$G^{'}(q_3) = G^{'}(\beta) = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ -2\, h_1(\beta) & 2 & 2\, h_2(\beta) \end{bmatrix} \tag{5.12}$$

and

$$\nabla_{q_p}^T E_p^{tot}(\phi_r, \beta) = \nabla_{q_p}^T E_{p,g}^{tot}(\phi_r, \beta) + \nabla_{q_p}^T E_{p,el}^{tot}(\beta) \tag{5.13}$$

where

$$\nabla_{q_p}^T E_{p,g}^{tot}(\phi_r, \beta) = \begin{bmatrix} 0 \\ \dfrac{\partial}{\partial\, \phi_r} E_{p,g}^{tot}(\phi_r, \beta) \\ \dfrac{\partial}{\partial\, \beta} E_{p,g}^{tot}(\phi_r, \beta) \end{bmatrix} \tag{5.14}$$

and

$$\nabla_{q_p}^T E_{p,el}^{tot}(\beta) = \begin{bmatrix} 0 \\ 0 \\ 2\, \dfrac{\partial}{\partial\, \beta} E_{p,el}(\beta) \end{bmatrix} \tag{5.15}$$

Equation (5.11) simply becomes

$$\begin{bmatrix} 2\,C^*_{m1} \\ \\ 2\,C^*_{m1} \\ \\ 2\,\left[C^*_{m2} + C^*_{m3}\,h_2(\beta^*)\right] - 2\,C^*_{m1}\,h_1(\beta^*) \end{bmatrix} = \begin{bmatrix} 0 \\ \\ \dfrac{\partial}{\partial\,\phi_r}\,E^{tot}_{p,g}\,(\phi^*_r,\beta^*) \\ \\ \dfrac{\partial}{\partial\,\beta}\,E^{tot}_{p,g}\,(\phi^*_r,\beta^*) + 2\,\dfrac{\partial}{\partial\,\beta}\,E_{p,el}\,(\beta^*) \end{bmatrix} \tag{5.16}$$

where

$$\begin{aligned} &\frac{\partial}{\partial\,\phi_r}\,E^{tot}_{p,g}(\phi^*_r,\beta^*) = \\ &= -2\,g\left[m^c\,(l_a+l_h) + l^{(a+h)}_{cm}\,m^{(a+h)} + l_h\,\left(m^d + \frac{m^{body}}{2}\right)\right]\,\sin\left(\phi^*_r - \alpha(\beta^*) + \theta^*\right) + \\ &\quad + 2\,g\,\left[l_d\,\frac{m^{body}}{2} + l^d_{cm}\,m^d\right]\,\sin\left(\beta^* + \phi^*_r + \theta^*\right) + 2\,g\,l^c_{cm}\,m^c\,\cos\left(\gamma(\beta^*) + \phi^*_r + \theta^*\right) + \\ &\quad - g\,l^{body}_{cm}\,m^{body}\,\cos\left(\phi^*_r - \theta^*_{body}\right) \end{aligned} \tag{5.17a}$$

$$\begin{aligned} &\frac{\partial}{\partial\,\beta}\,E^{tot}_{p,g}(\phi^*_r,\beta^*) = \\ &= 2\,g\left[m^c\,(l_a+l_h) + l^{(a+h)}_{cm}\,m^{(a+h)} + l_h\,\left(m^d + \frac{m^{body}}{2}\right)\right]\,h_1\,\sin\left(\phi^*_r - \alpha(\beta^*) + \theta^*\right) + \\ &\quad + 2\,g\,\left[l_d\,\frac{m^{body}}{2} + l^d_{cm}\,m^d\right]\,\sin\left(\beta^* + \phi^*_r + \theta^*\right) + 2\,g\,l^c_{cm}\,m^c\,h_2\,\cos\left(\gamma(\beta^*) + \phi^*_r + \theta^*\right) \end{aligned} \tag{5.17b}$$

$$\begin{aligned} &\frac{\partial}{\partial\,\beta}\,E_{p,el}(\phi^*_r,\beta^*) = \\ &= \left[k^t_2 + h_1\,(\beta^*)\,\left(k^t_1 + k^t_2\right) + k^t_1\,h_2\,(\beta^*)\right]\,\alpha\,(\beta^*) + \left[k^t_2 + k^t_4 + k^t_2\,h_1\,(\beta^*)\right]\,\beta^* + \\ &\quad + \left[h_2\,(\beta^*)\,\left(k^t_1 + k^t_3\right) + k^t_1\,h_1\,(\beta^*)\right]\,\gamma\,(\beta^*) - h_1\,(\beta^*)\,\left[(\alpha_0+\gamma_0)\,k^t_1 + (\alpha_0+\beta_0)\,k^t_2\right] + \\ &\quad - h_2\,(\beta^*)\,\left[\alpha_0\,k^t_1 + \gamma_0\,\left(k^t_1 + k^t_3\right)\right] - \left(\alpha_0\,k^t_2 + \beta_0\,k^t_2 + \beta_0\,k^t_4\right) \end{aligned} \tag{5.17c}$$

From the first equation of (5.16) it is clear that

$$C^*_{m1} = 0 \tag{5.18}$$

After substituting (5.18) into the second equation equation of (5.16) it is easy to note that it can be rewritten as

$$H_1(\beta^*)\,\cos(\phi^*_r) + H_2(\beta^*)\,\sin(\phi^*_r) = 0 \tag{5.19}$$

where

$$\begin{aligned} H_1(\beta^*) = &\,g\,\left(2\,l_a\,m^c + 2\,l^{(a+h)}_{cm}\,m^{(a+h)} + 2\,l_h\,m^c + l_h\,m^{body} + 2\,l_h\,m^d\right)\,\sin\left(\alpha(\beta^*) - \theta^*\right) + \\ &+ 2\,g\,l^c_{cm}\,m^c\,\cos\left(\gamma(\beta^*) + \theta^*\right) + g\,\left(l_d\,m^{body} + 2\,l^d_{cm}\,m^d\right)\,\sin\left(\beta^* + \theta^*\right) + \\ &- g\,l^{body}_{cm}\,m^{body}\,\cos\left(\theta^*_{body}\right) \end{aligned} \tag{5.20}$$

and

$$H_2(\beta^*) = -g\left(2\,l_a\,m^c + 2\,l_{cm}^{(a+h)}\,m^{(a+h)} + 2\,l_h\,m^c + l_h\,m^{body} + 2\,l_h\,m^d\right)\cos\left(\alpha(\beta^*) - \theta^*\right) + \\ - 2\,g\,l_{cm}^c\,m^c\,\sin\left(\gamma(\beta^*) + \theta^*\right) + g\,\left(l_d\,m^{body} + 2\,l_{cm}^d\,m^d\right)\cos\left(\beta^* + \theta^*\right) + \\ - g\,l_{cm}^{body}\,m^{body}\,\sin\left(\theta_{body}^*\right) \tag{5.21}$$

Equation (5.19) shows that, for each value of $\beta^*$ (i.e. leg position), there are two $\phi_r^*$ compatible with the vertical equilibrium of the system. Clearly, the only meaningful $\phi_r$ will be one in the range $[-\pi/2, \pi/2]$, since the robot cannot penetrate the ground. This value can simply be retrieved with

$$\phi_r^* = -\arctan\left(\frac{H_1(\beta^*)}{H_2(\beta^*)}\right) \tag{5.22}$$

Finally, the third equation of (5.16) provides, for a given $\beta^*$ and $\phi_r^*(\beta^*)$, the necessary leg torques to keep the robot vertically balanced.

Note that, since $C_{m3}$ is not an independent control input, its value its completely arbitrary and can be chosen, together with $C_{m2}$, to meet design requirements. In particular, during simulations, $C_{m3}$ was set to zero for simplicity.
Summing up, $\exists!$ physically consistent equilibrium point for the system, i.e.

$$q^* = \begin{bmatrix} q_1^* \\ q_2^* \\ q_3^* \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \phi_w^* \\ \phi_r^*(\beta^*) \\ \beta^* \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.23}$$

$$u^* \coloneqq \begin{bmatrix} 0 \\ \\ C_{m2}^*(\beta^*) \\ \\ C_{m3}^*(\beta^*) \end{bmatrix} \tag{5.24}$$

Note that actually $\phi_w^*$ can assume an arbitrary value, since it does not enter into the equilibrium conditions (this is highly expected, since the equilibrium on a flat ground cannot depend upon the abscissa of the wheel).
A relevant physical interpretation of the second equation of (5.16) can be found by computing the expression of the $x$ coordinate of the total center of mass. It is readily seen that the following relationship holds:

$$m_{tot}\,g\,x_{cm}^{tot} = \frac{\partial}{\partial\,\phi_r}\left(E_{p,g}^{tot}\right) + r_w\,\phi_w \tag{5.25}$$

When the system is vertically balanced, the partial derivative is zero and this shows that $x_{cm}$ is the same as the wheel center. Hence, if the total or partial CoM (i.e. knees plus robot body) has an abscissa different from the one of the wheel, the system cannot be in its equilibrium state.

There are actually multiple ways to make use of the third equation of (5.16):

- Firstly, if all the physical parameters of the system are known (in particular joint torsional rigidities), as already remarked, from (5.16) it is possible to retrieve, given a leg angle, the necessary balancing input torques. This provides an "exact" relationship of the balancing torques as a function of $\beta^*$, in contrast with the approximate ones (3.18) and (3.19).

- Secondly, that equation can be used as a design relationship: in particular, it can be used to determine joint torsional springs. For example, let us suppose to use only $k_1^t$ (which is indeed the case for all performed simulations). Zeroing both $C_{m2}$, $C_{m3}$ in (5.16) it is possible to retrieve the value of $k_1^t$ such that the torsional spring is able to statically sustain the system for a given $\beta^*$ of interest and a given $\beta_0^{el}$ (i.e. spring's leg reference angle). This procedure provides $k_1^t$ as a function of $\beta^*$. Since during vertical equilibrium a given $\beta^*$ corresponds to a given robot height, the torsional stiffness can be chosen so that the robot is sustained at the desired ground height.

As stated in Chapter 3.2.2, ideally one would want the equilibrium pitch angle, i.e. $\phi_r^*$ to be small (ideally zero) over the range of interest of $\beta$.
The carried leg dimensional synthesis was oriented towards obtaining a linear trajectory of the wheel w.r.t. the local reference frame of the leg. As previously remarked, this condition is in principle not sufficient to make the perturbations due to leg motion to the vertical equilibrium small.
The second condition to be verified is that the CoM has to belong to such trajectory. Since the achievable trajectory will be only approximately linear, it is evident that small perturbations to the equilibrium are inevitable. This, in particular, will be actually "proved" in a few lines.

It is possible to take the second equation of (5.16) and try to force $\phi_r^* = 0$ in it. This results in the following equation

$$K_1 \sin(\,\alpha(\beta^*)\,) + K_2 \cos(\,\gamma(\beta^*\,) + \theta^*) + K_3 \sin(\,\beta^* + \theta^*\,) = g\, l_{cm}^{body}\, m^{body} \cos(\theta_{body}^*) \tag{5.26}$$

where, in particular,

$$\begin{aligned} K_1 &= g\,(2\,l_a\,m^c + 2\,l_{cm}^{(a+h)}\,m^{(a+h)} + 2\,l_h\,m^c + l_h\,m^{body} + 2\,l_h\,m^d) \\ K_2 &= 2\,g\,l_{cm}^c\,m^c \\ K_3 &= g\,(l_d\,m^{body} + 2\,l_{cm}^d\,m^d) \end{aligned} \tag{5.27}$$

By looking at (5.26), the only remaining variables, since the dimensional synthesis is fixed and also link masses are supposed known, are $\beta^*$, $l_{cm}^{body}$ and $\theta_{body}^*$. Note that here the superscript of the last quantity does not indicate an equilibrium (see Fig. 3.7).
If $m^{body}$ is fixed (fairly reasonable), the combination $[l_{cm}^{body}, \theta_{body}^*]$ is fully determined for a given value of $\beta^*$, once one between the two parameters is fixed. As a consequence, this can be used as an additional design relationship.
By looking at the trigonometric nature of (5.26), it is clear how $\phi_r$ can be made

exactly zero for, at most, a finite number of $\beta^*$ values in the range of interest.

Once the equilibrium is fully explored, it is possible to show some interesting plots of some important quantities previously referenced in this section.
This will also allow to fix in a smart way the remaining physical parameters necessary to perform simulations on the system.
All quantities are expressed, if not differently specified, in SI units. For this reason, units of measurement are here omitted for simplicity.
First, let us recall that the optimal dimensional synthesis (3.17) is

$$\mathcal{D}_{opt} = [\, l_a,\, l_b,\, l_c,\, l_d,\, l_h,\, e_t,\, f_t] = [\, 0.1,\, 0.145,\, 0.236,\, 0.256,\, 0.355, -1, 0\,] \tag{5.28}$$

and that it is computed over a $\beta$ range of $[62°, 100°]$.
Supposing links made of aluminum profiles ($\rho_{Al} = 2700\,\mathrm{Kg/m^3}$) with rectangular section and a volume emptiness coefficient of 0.4 (meaning sections' areas are 60% full of material), it is possible to compute

$$\left[m^w, m^{ah}, m^d, m^c, m^{body}\right] = [\, 0.3, 0.2948, 0.1659, 0.1529, 10\,] \tag{5.29}$$

where $m^w$ and $m^{body}$ are actually fixed based on, respectively, the realistic wheel and body mass provided by the design described in Section 7.
With the same reasoning

$$\left[I^w_{yy}, I^{ah}_{yy}, I^d_{yy}, I^c_{yy}, I^{body}_{yy}\right] = \left[\, 10^{-3}, 0.0051, 9.0597 \cdot 10^{-4}, 7.0979 \cdot 10^{-4}, 10^{-1}\,\right] \tag{5.30}$$

Moreover

$$\theta^* = -\arctan(e_t) = \pi/4 \tag{5.31}$$

and, under the hypothesis of constant density,

$$\left[\, l^{ah}_{cm},\, l^d_{cm},\, l^c_{cm}\right] = [0.2275,\, 0.1280,\, 0.1180] \tag{5.32}$$

Fixing by design

$$\theta^*_{body} = 4/5\,\pi \tag{5.33}$$

it is possible to compute $l^{body}_{cm}\cos(\theta^*_{body})$ using (5.26) over the range of $\beta$ and then obtain an average value by simply dividing by the number of points in the range (in this case 1000). Alternatively, it would have been possible to use a value corresponding to a specific $\beta$.
Then, since $\theta^*$ is fixed, it is possible to retrieve a value for $l^{body}_{cm}$. After carrying out this procedure we obtain

$$l^{body}_{cm} = 0.0178 \tag{5.34}$$

The final parameter necessary to fix all the necessary quantities needed to perform simulations is the elastic reference angle for the torsional spring $k^t_1$.
This is chosen to be

$$\beta^{el}_0 = 90° \tag{5.35}$$

This ideally implies that, when mounting the prototype, one would have to measure as precisely as possible the $\beta$ of each leg at which the springs are mounted. Any

error in positioning the spring will inevitably result in an increased model mismatch between simulations and real world prototype.

Having defined almost all the necessary parameters, it is now possible to look at some interesting plots.
Firstly, Fig. 5.1 shows a plot, over the chosen $\beta$ range, of the vertical equilibrium pitch angle $\phi_r^*$.

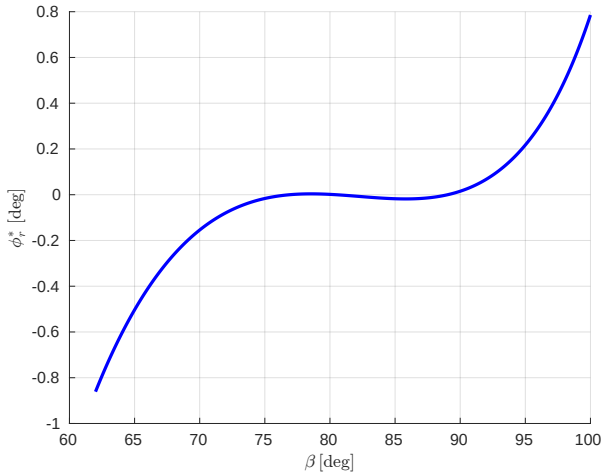



**Figure 5.1.** Vertical balancing pitch angle $\phi_r^*$. Given a leg configuration (i.e. $\beta$), there is only one $\phi_r$ compatible with the vertical equilibrium configuration.

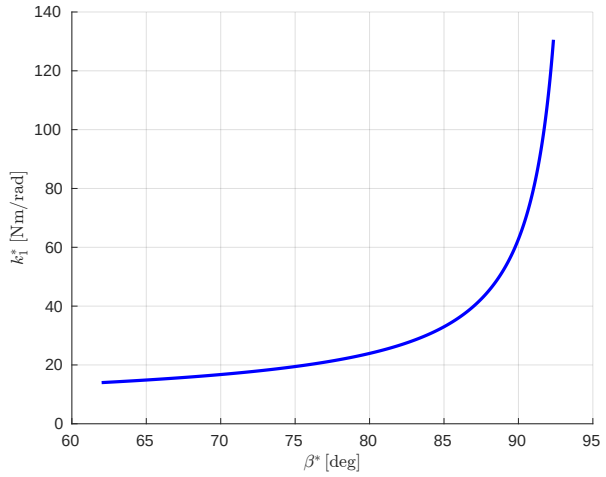



**Figure 5.2.** Vertical balancing knee torsional rigidity, as a function of the leg angle $\beta$. Given a desired vertical balancing leg angle, the torsional spring value capable of sustaining the robot at that configuration is found on the vertical axis.

It is clear how the combined design approaches of making the trajectory of the wheel almost linear and also the use of the design equation (5.26), allow to make the balancing pitch angle quite small. In particular, looking at Fig. 5.1, the balancing pitch angle is always less than 1° in absolute value.

The next graph, Fig. 5.2, shows the plot of $k_1^t(\beta^*)$: this is the torsional spring stiffness value necessary to keep the leg in a certain position, supposing no actuation is present. This plot can be used to select a suitable value for $k_1^t$, given a desired $\beta^*$ (or, equivalently, robot height).
In particular, the chosen value for performing simulations is

$$k_1^t = 30 \tag{5.36}$$

which corresponds, roughly, to a leg angle of 83°.

Since now all the physical parameters are fixed and known, using the last equation of (5.16) and neglecting the elastic potential energy, it is possible to evaluate the exact expression of the balancing torques. Fig. 5.3 show such plot. The blue line is the vertical balancing torque $C_{m2}^*$ to be applied on link $L_d$, supposing $C_{m3} = 0$, and the red line shows the result by using only $C_{m3}$ and no $C_{m2}$.

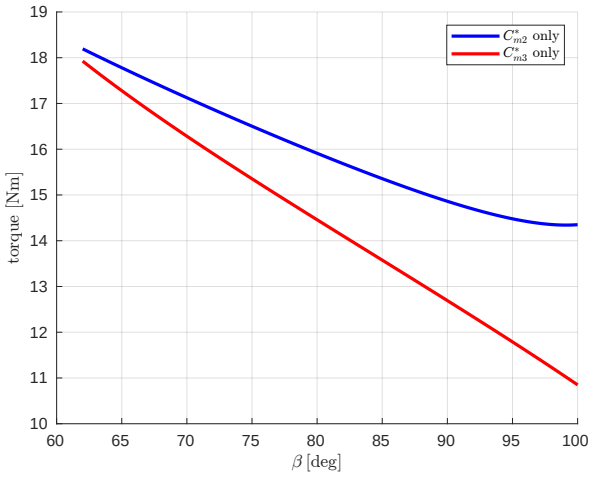



**Figure 5.3.** Vertical balancing leg torques computed using the "exact" expressions obtained imposing the vertical equilibrium on the full dynamics of the system.

Fig. 5.4 shows a comparison between the exact balancing torque expressions retrieved in this section and the approximate ones obtained in Section 3.3. Looking at the graphs, the approximate expressions overestimate by roughly $2Nm$ balancing torques values, which is actually a pretty good result. This shows that the approximate model can potentially be used to get decent (overestimated) values of joint reactions, to be used for designing leg joints; in particular, one can simply stick to the approximate simple relationships obtained in Section 3.3 and use a suitable safety coefficient to account for dynamical effects on joint reactions.

Finally, Fig. 5.5 shows a side view of the robot at four different leg angles, during vertical balancing. Each individual CoM is shown as an orange dot, while the total center of mass is shown in red. There are two dotted lines, a black one representing the linear approximation of wheel trajectory w.r.t. leg's local frame and a blue one, depicting the total CoM trajectory over the range of $\beta$. These two lines are almost overlapped, thanks to the carried leg synthesis and design relationship (5.26).

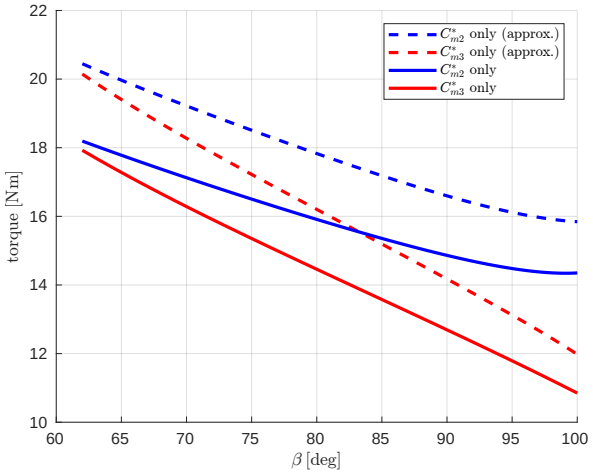



**Figure 5.4.** Comparison between the "exact" vertical balancing leg torques and the ones obtained with the approximate static model.

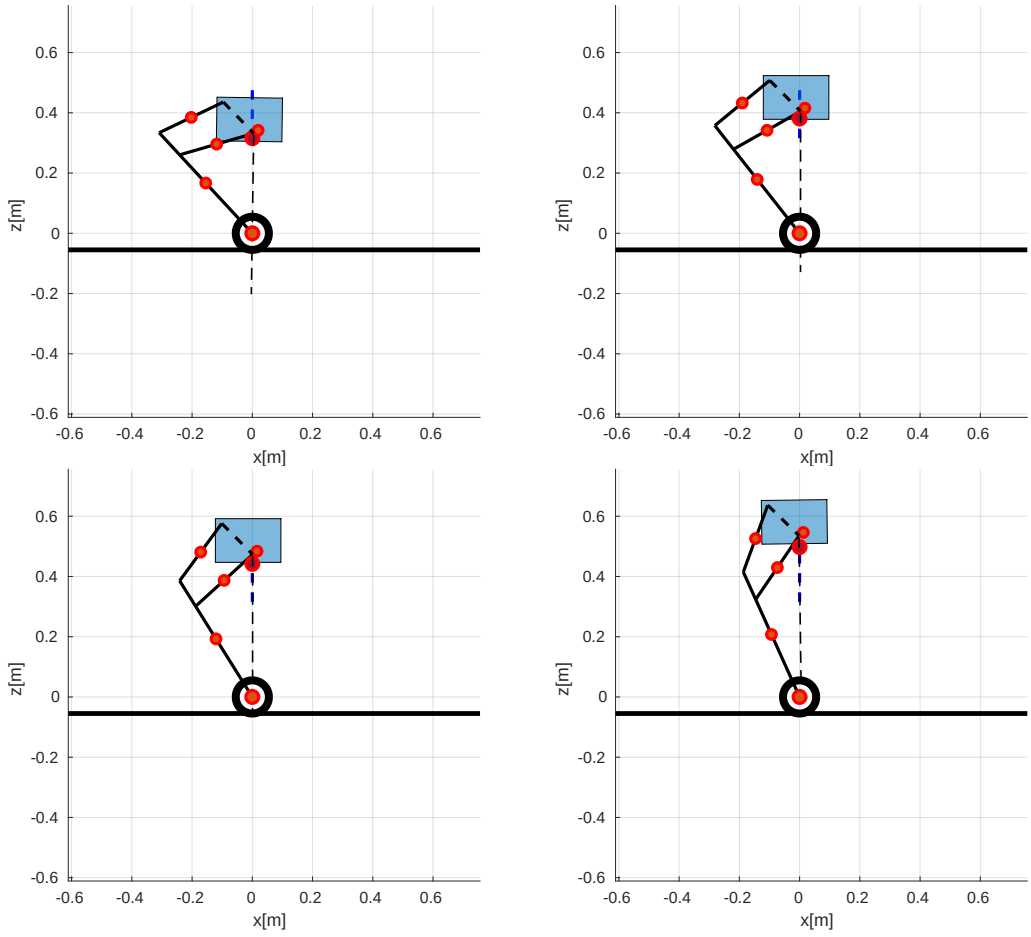



**Figure 5.5.** Side view of the robot during static vertical balancing at four different leg angles

## 5.3 Linearization around a generic reference trajectory

This section is dedicated computing the linearization of a dynamics of the form (3.57) around a suitable reference trajectory (which, for now, is supposed given).

Firstly, let us define

$$u_1 \coloneqq C_{m1}(t) \tag{5.37}$$

$$u_2 \coloneqq C_{m2}(t) \tag{5.38}$$

$$u_3 \coloneqq C_{m3}(t) \tag{5.39}$$

With this definition, it is easy to see that (3.57) can be also written in the form

$$\dot{q} = f(q) + g_1(q)\, u_1(t) + g_2(q)\, u_2(t) + g_3(q)\, u_3(t) = \mathcal{F}(q, u(t)) \tag{5.40}$$

where $g_1$ , $g_2$ and $g_3$ are, respectively, the first, second and third column of (3.59). Secondly, let us suppose that a complete reference trajectory, compatible with the systems dynamics, is available. "Complete" means that this reference trajectory must by available for both the state $q$ and inputs $u_1$, $u_2$ and $u_3$. For now, the problem of how to determine such trajectory is left unaddressed purposely. This problem will be addressed in Section 6.3.
Let this references be

$$q^{ref}(t) \coloneqq \begin{bmatrix} q_1^{ref}(t) \\ q_2^{ref}(t) \\ q_3^{ref}(t) \\ q_4^{ref}(t) \\ q_5^{ref}(t) \\ q_6^{ref}(t) \end{bmatrix} = \begin{bmatrix} \phi_w^{ref}(t) \\ \phi_r^{ref}(t) \\ \beta^{ref}(t) \\ \dot{\phi}_w^{ref}(t) \\ \dot{\phi}_r^{ref}(t) \\ \dot{\beta}^{ref}(t) \end{bmatrix} \tag{5.41}$$

and

$$u^{ref} = \begin{bmatrix} u_1^{ref}(t) \\ u_2^{ref}(t) \\ u_3^{ref}(t) \end{bmatrix} \tag{5.42}$$

Then, the time-varying linearization of (5.40) is

$$\dot{q} = \mathcal{F}(q^{ref}, u^{ref}) + \nabla_q \mathcal{F}(q^{ref}, u^{ref})\, \tilde{q} + \nabla_u \mathcal{F}(q^{ref}, u^{ref})\, \tilde{u} \tag{5.43}$$

which, by employing (5.40), becomes

$$\dot{q} = \nabla_q \mathcal{F}(q^{ref}, u^{ref})\, \tilde{q} + \nabla_u \mathcal{F}(q^{ref}, u^{ref})\, \tilde{u} \tag{5.44}$$

where

$$\tilde{q} = q - q^{ref} \tag{5.45}$$

and

$$\tilde{u} = u - u^{ref} \tag{5.46}$$

Equation (5.44) can be synthetically rewritten as

$$\dot{\tilde{q}} = A\,\tilde{q} + B\,\tilde{u} \tag{5.47}$$

where

$$\begin{aligned} A = \nabla_q \mathcal{F}(q^{ref}, u^{ref}) = \nabla_q\, f(q^{ref}) + \nabla_q\, g_1(q^{ref})\, u_1^{ref} + \nabla_q\, g_2(q^{ref})\, u_2^{ref} + \\ + \nabla_q\, g_3(q^{ref})\, u_3^{ref} \end{aligned} \tag{5.48}$$

and

$$\begin{aligned} B = \left[\frac{\partial}{\partial u_1}\mathcal{F}(q^{ref}, u^{ref}),\ \frac{\partial}{\partial u_2}\mathcal{F}(q^{ref}, u^{ref}),\ \frac{\partial}{\partial u_3}\mathcal{F}(q^{ref}, u^{ref})\right] = \\ = \left[g_1(q^{ref}),\ g_2(q^{ref}),\ g_3(q^{ref})\right] \end{aligned} \tag{5.49}$$

There are several uses of the linearized form (5.47).
For instance, a particular reference trajectory for the system might be its vertical equilibrium state, i.e. (5.23). The linearization can be used to study the control properties of the system around its vertical equilibrium and also to design a stabilizing controller (see (6.1)).
The linearized dynamics (5.47) will also be crucial for performing trajectory tracking in Section 6.4.

Supposing the knowledge of a compatible reference state trajectory (5.41), reference inputs (5.42) cannot be chosen arbitrarily: they have to satisfy dynamics (3.57).
In particular, expanding the second order dynamics (3.54) and neglecting the additional control input $C_{m3}$, we obtain

$$C_{m1}^{ref} = \frac{m_{11}\,\dot{q}_4^{ref} + m_{12}(q_2^{ref}, q_3^{ref})\,\dot{q}_5^{ref} + m_{13}(q_2^{ref}, q_3^{ref})\,\dot{q}_6^{ref} + c_1(q^{ref})}{2} \tag{5.50}$$

$$C_{m1}^{ref} = \frac{m_{21}(q_2^{ref}, q_3^{ref})\,\dot{q}_4^{ref} + m_{22}(q_3^{ref})\,\dot{q}_5^{ref} + m_{23}(q_3^{ref})\,\dot{q}_6^{ref} + c_2(q^{ref})}{2} \tag{5.51}$$

$$\begin{aligned} C_{m2}^{ref} = \frac{m_{13}(q_2^{ref}, q_3^{ref})\,\dot{q}_4^{ref} + m_{23}(q_3^{ref})\,\dot{q}_5^{ref} + m_{33}(q_3^{ref})\,\dot{q}_6^{ref} + c_3(q^{ref})}{2} + \\ + C_{m1}^{ref}\, h_1(q_3^{ref}) \end{aligned} \tag{5.52}$$

If and only if (5.41) is chosen to be compatible with system's dynamics, any between (5.50) and (5.51) can be used to compute $C_{m1}^{ref}$, while (5.52) can be used to compute $C_{m2}^{ref}$.
Note that the difference between (5.50) and (5.51) actually defines the previously mentioned second order dynamical feasibility constraint. This is a typical constraint which arises when studying underactuated systems.

## 5.4 Control properties of the system around the vertical equilibrium

By using the linearization of systems (3.57) it is possible to study its properties around the vertical equilibrium.

From now on, $C_{m3}$ will be set to 0 for simplicity and, as a consequence, the third column of $B$ will be neglected.

Thanks to the structure and dependencies of (3.57) is it possible to know in advance the expected structure of linearization matrices $A$ and $B$ around a generic reference trajectory.
In particular, these matrices will have the following structure:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & a_{42} & a_{43} & 0 & a_{45} & a_{46} \\ 0 & a_{52} & a_{53} & 0 & a_{55} & a_{56} \\ 0 & a_{62} & a_{63} & 0 & a_{65} & a_{66} \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ b_{42} & b_{43} \\ b_{52} & b_{53} \\ b_{62} & b_{63} \end{bmatrix} \tag{5.53}$$

This structure was also confirmed numerically during simulations.
Since the linearization will be performed around the vertical equilibrium, the reference full trajectory (both state and input references) becomes

$$q^{ref}(t) = \begin{bmatrix} 0 \\ \phi_r^*(\beta^{ref}) \\ \beta^{ref} \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u^{ref} = \begin{bmatrix} 0 \\ C_{m2}^*(\beta^{ref}) \\ 0 \end{bmatrix} \tag{5.54}$$

where $C_{m2}^*$ is computed from the third equation of (5.16).
For carrying out this analysis, $\beta^{ref}$ is chosen to be 80°, right in the middle of the range.

This implies

$$q^{ref}(t) = \begin{bmatrix} 0 \\ -5.0502 \cdot 10^{-6} \\ 1.3963 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u^{ref} = \begin{bmatrix} 0 \\ 2.3253 \\ 0 \end{bmatrix} \tag{5.55}$$

which results into

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 826.4622 & 245.2300 & 0 & 0 & 0 \\ 0 & 127.3409 & 37.7429 & 0 & 0 & 0 \\ 0 & -5.2811 & 13.1909 & 0 & 0 & 0 \end{bmatrix} \tag{5.56}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 307.2738 & -1.4492 \\ 40.4346 & -0.2290 \\ -2.6348 & 2.0307 \end{bmatrix} \tag{5.57}$$

The first important property to be address is the local stability of the system around its vertical equilibrium. This should clearly be an unstable equilibrium since, for a fixed $\beta$, the system is basically a WIP.
The resulting eigenvalues of (5.56) are

$$[0,\ 0,\ 11.2057,\ 3.8684,\ -11.2057,\ -3.8684] \tag{5.58}$$

For completeness sake, computed eigenvalues at, respectively, the start (62°) and end (100°) of $\beta$ range are here reported:

$$[0,\ 0,\ 11.3508,\ 2.5201,\ -11.3508,,\ -2.5201] \tag{5.59a}$$

$$[0,\ 0,\ 10.6868,\ 5.6185,\ -10.6868,\ -5.6185] \tag{5.59b}$$

These values confirm, as it should be, the instability of system's vertical equilibrium.
Next, let us look at the zeros of the system computed choosing as output

$$y = C\,q = [r_w,\ 0,\ 0,\ 0,\ 0,\ 0]\ q \tag{5.60}$$

which corresponds to looking at $x_w$.
Now, let us look at two different transfer functions (computed numerically via Matlab).
First, the transfer function from $C_{m1}$ to $x_w$ is

$$T_1(s) = \frac{17.79\,s^4 + 1.58 \cdot 10^{-14}\,s^3 - 570\,s^2 + 1.46e - 12\,s + 4215}{s^6 - 9.326 \cdot 10^{-15}\,s^5 - 139.5\,s^4 + 1.705 \cdot 10^{-13}\,s^3 + 1566\,s^2} \tag{5.61}$$

with zeros

$$[4.5226,\ 3.4031,\ -4.5226,\ -3.4031] \tag{5.62}$$

Secondly, the transfer function from $C_{m2}$ to $x_w$ is

$$T_2(s) = \frac{-0.1591\,s^4 + 5.652 \cdot 10^{-16}\,s^3 + 74.49\,s^2 + 4.312 \cdot 10^{-21}\,s - 1.352 \cdot 10^{-11}}{s^6 - 9.326 \cdot 10^{-15}\,s^5 - 139.5\,s^4 + 1.705 \cdot 10^{-13}\,s^3 + 1566\,s^2} \tag{5.63}$$

with zeros

$$\left[17.5272,\ -17.5272,\ 1.9438 \cdot\ 10^{-07},\ -1.9438 \cdot\ 10^{-07}\right] \tag{5.64}$$

Looking at (5.63), there are very small coefficients at the numerator which suggest that, due to the output choice, there are actually two zero-pole cancellations happening (in particular relative to the 0 eigenvalues). The small coefficients are in fact numeric zeros.
This means that, while carrying out measurements of (5.61) would be in principle sufficient to reconstruct system's dynamics ($A$ matrix) around its vertical equilibrium, doing the same using (5.63) wouldn't be sufficient, since there is a hidden dynamics.
The presence of a hydden dynamics is to be expected and has a straightforward physical interpretation: the canceled dynamics is associated to the CoM which, by using only the leg torque, cannot be controlled; this is due to $C_{m2}$ being an internal action.
Now, both systems represented by (5.61) and (5.63) have right half plane zeros, making them *non-minimum phase systems* (at least, locally).
This property was also verified when $\phi_r$ and $\beta$ are chosen as output. The details are here omitted for brevity.

Transfer functions (5.61) and (5.63) are also useful for identifying another property, yet to be defined, which is the so called *relative degree* of a system.

**Definition 5.4.1** *A dynamical system of the form* (3.57) *with output* $y = h(q) \in \mathbb{R}^n$ *is said to have vector relative degree* $[r_1,\ r_2, ....., r_n]$ *at a point* $q^\circ$ *if it is necessary to differentiate* $r_k$ *times the element* $y_k$ *of the output in order to have at least one component of the input* $u$ *appear in it.*

Note that property (5.4.1) is a defined locally.
In the particular case of system (3.7) with the scalar output (5.60),

$$\dot{y} = \nabla_{q_p} h(q_p)\,\dot{q}_p = J^y(q_p)\,\dot{q}_p \tag{5.65}$$

where $J^y$ is the output Jacobian. Choosing as output $x_w$, $J^y = [r_w, 0, 0]^T$.
As a consequence, differentiating w.r.t. time once more and employing dynamics (3.54), we obtain

$$\ddot{y} = J^y \ddot{q}_p + \dot{J}^y \dot{q}_p = \left( J^y M^{-1} G^{'} \right) u - J^y M^{-1} C + \dot{J} \dot{q}_p \tag{5.66}$$

In this particular case

$$J^y M^{-1} G^{'} = [17.7934, -0.0865, -0.0930] \tag{5.67}$$

meaning that the system has scalar relative degree two at its vertical equilibrium. This has a very straightforward physical interpretation: looking at dynamics (3.54) it is clear that inputs affect directly the second derivative of the Lagrangian state. It is also possible to verify this property by looking at transfer functions (5.61) and (5.63).
In fact, in the case of single input LTI systems, the relative degree can also be retrieved by computing $n_p - n_z$, where $n_p$ is the number of poles of the transfer function and $n_z$ is the number of its zeros (see [16]). Following the logic of Definition (5.4.1), the system under study will relative degree two, since at least one between (5.61) and (5.63) have relative degree two (actually both).

There is another useful property which can be explored: *linear controllability.*

**Definition 5.4.2** *A system of the form* (3.57) *is said to be linearly controllable at an equilibrium point* $[q^*, u^*]$ *if its linear approximation* (5.47) *around* $[q^*, u^*]$ *is controllable.*
*The controllability of* (5.47) *can simply be checked by verifying that*

$$\text{rank} \left( \left[ B, A\,B, A^2\,B...., A^{n-1}\,B \right] \right) = n \tag{5.68}$$

*where n is equal to the dimension of A.*

Computing the controllability matrix by employing (5.56) and (5.57), the system turns out to be indeed linearly controllable around its vertical equilibrium (this would not true if instead of using the whole $B$, only its second column is chosen). The explicit values of the controllability matrices are here omitted for brevity. The reason why the system is linearly controllable is due to the planarity of the model. In fact, in general, wheeled underactuated systems subject to nonholonomic constraints are usually not linearly controllable according to Definition (5.4.2) (see [42]).
Given a system subject to a pfaffian constraint of the form (3.89), if the constraint matrix has rank $k$, then

$$\dot{q}_p = r_1(q_p)\, a_1 + .... + r_m(q_p)\, a_m \tag{5.69}$$

where $r_i$, with $i = 1, .., m$ is a base of $A$ null space and $a_i$ are suitable coefficients. Systems of the form (5.69) is are called *driftless systems*; this is due to the fact that, if $a_i = 0$ then $\dot{q}_p = 0$. For such systems, a more general definition of controllability is usually employed. This definition is here reported for completeness.

**Definition 5.4.3** *The nonlinear driftless system* (5.69) *is controllable if, for any* $q_p^0$ *and* $q_p^f \in \mathbb{R}$*, there exists a time* $T > 0$ *and a control input* $a(t) = [a_1(t), ..., a_m(t)]^T$ *such that the solution of* (5.69) *satisfies* $q_p(0) = q_p^0$ *and* $q_p(T) = q_p^f$

In particular *Chow's theorem* gives a necessary and sufficient condition for system (5.69) to be controllable in the sense of Definition (5.4.3). This theorem is out of the scope of this thesis, but can be consulted, for example, in [42].

The very last control property which is tested in this section if the *linear observability* of the system around its vertical equilibrium.
In analogy with definition (5.4.2):

**Definition 5.4.4** *A system of the form* (3.57) *with output* (5.60) *is said to be linearly observable at an equilibrium point* $[q^*, u^*]$ *if its linear approximation* (5.47) *around* $[q^*, u^*]$ *is observable.*
*The observability of* (5.47) *can simply be checked by verifying that*

$$\operatorname{rank}\left(\begin{bmatrix} C \\ C\,A \\ C\,A^2 \\ . \\ . \\ C\,A^{n-1} \end{bmatrix}\right) = n \tag{5.70}$$

*where* $n$ *is equal to the dimension of* $A$*.*

In particular, using output (5.60) makes the system observable.
On the contrary, using as outputs $\phi_r$ or $\beta$ (or both), it is possible to verify that the rank of the observability matrix is actually 4 and the system is hence not observable. This has a clear physical interpretation, since one cannot reasonably aim at reconstructing the system's state by using only information on the leg position and the robot's pitch (it is intuitively impossible to know the absolute position of the wheel).

## 5.5 Second order normal form of the system under contact and partial feedback linearization

Let us look again at the expanded form of dynamics (3.54):

$$2\,C_{m1} = m_{11}\,\ddot{q}_1 + m_{12}(q_2, q_3)\,\ddot{q}_2 + m_{13}(q_2, q_3)\,\ddot{q}_3 + c_1(q_2, \dot{q}_2, q_3, \dot{q}_3) \tag{5.71a}$$

$$2\,C_{m1} = m_{21}(q_2, q_3)\,\ddot{q}_1 + m_{22}(q_3)\,\ddot{q}_2 + m_{23}(q_3)\,\ddot{q}_3 + c_2(q_2, \dot{q}_2, q_3, \dot{q}_3) \tag{5.71b}$$

$$2\,C_{m2} - 2\,C_{m1}\,h_1(q_3) = m_{13}(q_2, q_3)\,\ddot{q}_1 + m_{23}(q_3)\,\ddot{q}_2 + m_{33}(q_3)\,\ddot{q}_3 + c_3(q_2, \dot{q}_2, q_3, \dot{q}_3) \tag{5.71c}$$

where recall that $q_1$, $q_2$ and $q_3$ are defined by (3.56).
As was already stressed (5.3), the first two equations actually define a constraint. In particular, by subtracting (5.71b) from (5.71a), one obtains the following equivalent form of the dynamics:

$$[m_{11} - m_{21}(q_2, q_3)]\;\ddot{q}_1 + [m_{12}(q_2, q_3) - m_{22}(q_3)]\;\ddot{q}_2 + [m_{13}(q_2, q_3) - m_{23}(q_3)]\;\ddot{q}_3 + \\ + c_1(q_2, \dot{q}_2, q_3, \dot{q}_3) - c_2(q_2, \dot{q}_2, q_3, \dot{q}_3) = 0 \tag{5.72a}$$

$$2\,C_{m1} = m_{11}\,\ddot{q}_1 + m_{12}(q_2, q_3)\,\ddot{q}_2 + m_{13}(q_2, q_3)\,\ddot{q}_3 + c_1(q_2, \dot{q}_2, q_3, \dot{q}_3) \tag{5.72b}$$

$$2\,C_{m2} - 2\,C_{m1}\,h_1(q_3) = m_{13}(q_2, q_3)\,\ddot{q}_1 + m_{23}(q_3)\,\ddot{q}_2 + m_{33}(q_3)\,\ddot{q}_3 + c_3(q_2, \dot{q}_2, q_3, \dot{q}_3) \tag{5.72c}$$

This form of the system will be here addressed as a "pseudo-lower actuated form" to stress the similarity with the definition used by [42].
In this case, the lower actuated form of the system is not directly obtained by writing the dynamics equation due to the particular choice of the Lagrangian coordinates. It would have been possible, on the contrary, if the Denavit-Hartenberg conventions were used. These conventions are particularly useful for robotic manipulators, but they are not always the most convienient choice for different types of systems.
This been said, any underactuated system in the form (5.5) can be written in an upper or lower actuated way.

Equation (5.72a) is the previously mentioned dynamical feasibilty constraint second order constraint. Since the system has dimension 3 and degree of actuation 2, the dimension of this constraint should be $3 - 2 = 1$, which is indeed the case.
Now, supposing the objective is to perform tracking of $\phi_w$ (or, equivalently, $x_w$) and $\beta$, let us extract $\ddot{\phi}_r$ (i.e. $\ddot{q}_2$) from (5.72a). The result is

$$\ddot{q}_2 = \frac{c_1 - c_2 + \ddot{q}_1\,(m_{11} - m_{21}) + \ddot{q}_3\,(m_{13} - m_{23})}{m_{22} - m_{12}} \tag{5.73}$$

where all the dependencies are omitted for brevity. This substitution is clearly valid only if $m_{22} - m_{12} \neq 0$.

Substituting back into (5.72c) and (5.72b), we obtain

$$\begin{aligned}&\left[m_{11}+\frac{m_{12}\,(m_{11}-m_{12})}{m_{22}-m_{12}}\right]\ddot{q}_1+\left[m_{13}+\frac{m_{12}\,(m_{13}-m_{23})}{m_{22}-m_{12}}\right]\ddot{q}_3+\\&+c_1+\frac{m_{12}\,(c_1-c_2)}{m_{22}-m_{12}}=2\,C_{m1}\end{aligned}\tag{5.74a}$$

$$\begin{aligned}&\left[m_{13}+h_1\,m_{11}+\frac{(m_{23}+h_1\,m_{12})\,(m_{11}-m_{12})}{m_{22}-m_{12}}\right]\ddot{q}_1+\\&+\left[m_{33}+h_1\,m_{13}+\frac{(m_{23}+h_1\,m_{12})\,(m_{13}-m_{23})}{m_{22}-m_{12}}\right]\ddot{q}_3+c_3+h_1\,c_1+\\&+\frac{(m_{23}+h_1\,m_{12})\,(c_1-c_2)}{m_{22}-m_{12}}=2\,C_{m2}\end{aligned}\tag{5.74b}$$

Equations (5.74a) and (5.74b) after a very simple manipulations, can be synthetically rewritten as

$$\bar{M}\,\ddot{\bar{q}}_p+\bar{C}=\bar{u}\tag{5.75}$$

where

$$\bar{q}_p=\begin{bmatrix}q_1\\q_3\end{bmatrix}\tag{5.76}$$

and the other terms have obvious meaning.
Now, it is clear that by using the control law

$$\bar{u}=\bar{M}\,v+\bar{C}\tag{5.77}$$

where $v\in\mathbb{R}^2$ is an additional outer-loop control, the system can be written in the general form

$$\ddot{\bar{q}}_p=v\tag{5.78}$$

$$M^{int}(q_p)\,\ddot{q}_p^{int}+C^{int}(q_p,\dot{q}_p)=G^{int}(q_p)\,v\tag{5.79}$$

where, in this particular case,

$$M^{int}=m_{12}-m_{22}\tag{5.80}$$

$$C^{int}=c_1-c_2\tag{5.81}$$

$$G^{int}=[m_{11}-m_{21},m_{13}-m_{23}]\tag{5.82}$$

System (5.78)-(5.79) is the so called *second-order normal form with input-driven internal dynamics* (refer to [42]).
In particular, equation (5.79) is the *internal dynamics* of the system. This system is completely *feedback equivalent* to the original system.
Furthermore, as opposed to a system of the form (5.4), a system of the form (5.79) - (5.78) is said to have been only *partially* feedback linearized.
An aspect which deserves to be highlighted is that the second order normal form of

the system depends on which variable/s are extracted from the constraint equation (5.72b) and, hence, it is not unique.
Moreover, there is a very important interpretation of the form (5.78)- (5.79):
once reference trajectories for $\bar{q}_p$ over time are given (for example, consider the case of tracking of $x_w$ and $\beta$), the future evolution of $\phi_r$ starting from the current instant of time is completely determined by its initial conditions (i.e. $\phi_r(t_0)$ and $\dot{\phi}_r(t_0)$). The corresponding necessary inputs to the system are uniquely determined by (5.77).
This interpretation provides an insight as to why an underactuated system cannot realize arbitrary state trajectories.

The role of the internal dynamics (5.79) is crucial when developing control policies for underactuated systems. In particular, when performing trajectory tracking, care must be taken in order to ensure internal dynamics remains bounded in time. In the specific case of the system under study this aspect is particularly evident, since the internal dynamics represents pitch angle's forced dynamics and its divergence would make the robot fall to the ground.

Now, with the aid of numerical tools, it is possible to study the properties of the zero dynamics manifold of the system when the output is chosen to be

$$y(q) = \begin{bmatrix} r_w\, q_1 \\ q_3 - q_3^0 \end{bmatrix} = \begin{bmatrix} x_w \\ \beta - \beta^0 \end{bmatrix} \tag{5.83}$$

where $\beta^0$ is a reference leg angle necessary for computing the zero dynamics (chosen to be 80°). Note that, in fact, if the second component of the output was simply chosen to be exactly equal to $\beta$, then computing the zero dynamics manifold according to definition (5.1.3) would lead to numerical problems ($\beta$ cannot be to small, due to the linkage presence). As a consequence, a shift of the output was performed for convenience.
According to Definition (5.1.3) the zero dynamics for this system has a very simple physical interpretation: it is the dynamics of the pitch angle when the wheel is kept still at the origin and leg's knee is fixed at a reference position. As a consequence, one expects the zero dynamics to be completely equivalent to the dynamics of a simple inverted pendulum.
To verify these intuitions, it is possible to look at the phase plot of the zero dynamics. Firstly, let us obtain the equations describing the zero dynamics by simply setting the output identically to 0 in (5.79).
This leads to

$$\left[m_{12}(q_2, \beta^0) - m_{22}(\beta^0)\right]\, \ddot{q}_2 = c_2(q_2, \dot{q}_2, \beta^0, 0) - c_1(q_2, \dot{q}_2, \beta^0, 0) \tag{5.84}$$

Equation (5.84) is the zero dynamics of the system with output (5.83).
Expanding (5.84) to a first order system, one can plot the phase plane plot of the zero dynamics. This is shown in Fig. 5.6.
It is now clear that the zero dynamics of the system is completely equivalent to a simple WIP.
In particular, looking at Fig. 5.6 one can easily distinguish:

- The equilibrium points of the system. The red dot corresponds to the unstable vertical equilibrium, while the two blue dots are the stable one. Note that this

two points are not $[0, 0]$ and $[\pm\pi, 0]$ since, as already stressed, the equilibrium pitch angle for the system is in general different from zero (in this case, however, is very close).

- The red lines represent the unstable zero dynamics trajectories over the zero dynamics manifold, while the blue ones are stable.

- The vector field defined by the first order extension of (5.84).

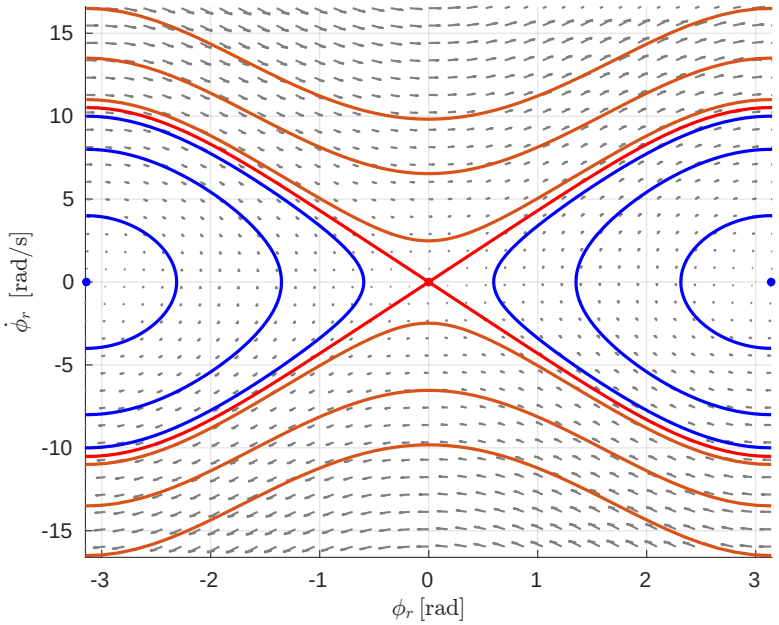



**Figure 5.6.** Phase plane plot of the zero dynamics, which is completely equivalent to the one of a simple inverted pendulum. The stable orbits are shown in blue, while the unstable ones are shown in red.

Clearly, almost all trajectories are periodic orbits (with the exception of the two equilibrium points). As a consequence, the zero dynamics cannot be asymptotically stable.

The analysis carried out in Section 5.4 confirmed that the system is non-minimum phase in a neighbourhood of the vertical equilibrium. The analysis of zero dynamics' phase plot actually clarifies that the system is non-minimum phase over its entire configuration space $\mathcal{C}$.
Note that trajectories of the zero dynamics are actually constant energy levels of the reduced order system (5.84) (see [42]).
The study of the zero dynamics was carried out for a specific value of $\beta^0$; it is however clear that, as long $\beta^0$ is chosen in the range of allowable $\beta$, the analysis of the zero dynamics would lead to the same exact conclusions (if the leg extends or contracts, the zero dynamics behaves always as a WIP).
Furthermore, not that the second derivative of the output is

$$\ddot{y} = \ddot{\bar{q}}_p = \bar{M}^{-1} \left( u - \bar{C} \right) \tag{5.85}$$

Hence, according to definition (5.4.1), this also confirms that the system has vector relative degree $[2, 2]$ w.r.t. the chosen output over $\mathcal{C}$, since all elements of $M^{-1}$ are different from zero.
For completeness, Fig. 5.7 shows a quite general graphical representation of the system, useful for highlighting both the internal dynamics and the remaining part of the dynamics which is then canceled by a suitable partially feedback linearizing input (5.77).

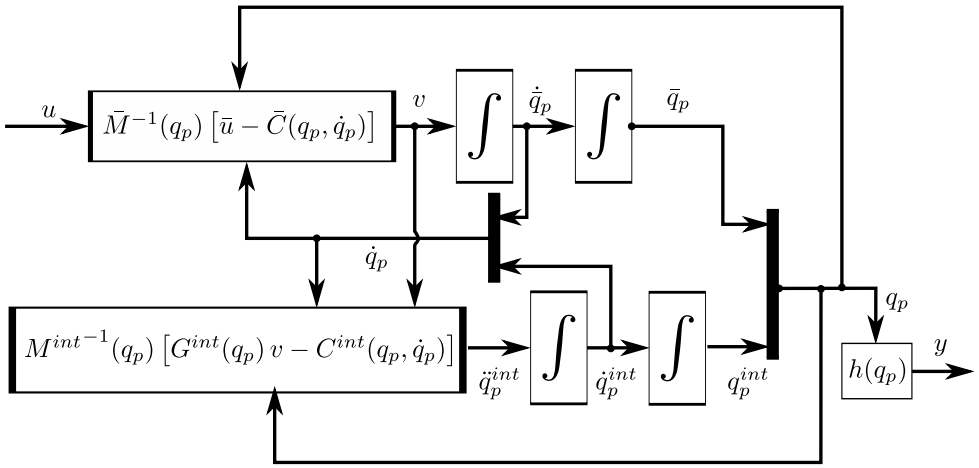



**Figure 5.7.** Representation of the second order normal form of the system. If the control input $u$ is chosen as $\bar{M}\, v + \bar{C}$, then all that is left is a chain of integrators and the internal dynamics.

## 5.6 Second order normal form of the floating-base system

It is also possible to obtain a second order normal form of the floating-base model. The first step is to retrieve the dynamical feasibility second order constraint.
Looking at the explicit form of dynamics (3.86) one recognizes the following structure of the system:

$$\begin{bmatrix} m_{11} & 0 & 0 & m_{14}(q_4,q_5) & m_{15}(q_4,q_5) \\ 0 & m_{22} & 0 & m_{24}(q_4,q_5) & m_{25}(q_4,q_5) \\ 0 & 0 & m_{33} & 0 & 0 \\ m_{41}(q_4,q_5) & m_{42}(q_4,q_5) & 0 & m_{44}(q_5) & m_{45}(q_5) \\ m_{51}(q_4,q_5) & m_{52}(q_4,q_5) & 0 & m_{54}(q_5) & m_{55}(q_5) \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \\ \ddot{q}_4 \\ \ddot{q}_5 \end{bmatrix} + \tag{5.86}$$

$$+ \begin{bmatrix} c_1(q_4,q_5,\dot{q}_4,\dot{q}_5) \\ c_2(q_4,q_5,\dot{q}_4,\dot{q}_5) \\ 0 \\ c_4(q_4,q_5,\dot{q}_4,\dot{q}_5) \\ c_5(q_4,q_5,\dot{q}_4,\dot{q}_5) \end{bmatrix} = \begin{bmatrix} T \\ N \\ 2\,C_{m1} - T\,r_w \\ 2\,C_{m1} \\ 2\,C_{m2} - 2\,h_1(q_5)\,C_{m1} \end{bmatrix}$$

where now $q$ is defined accordingly to (3.67).

Looking at the equations, it is immediate to see that, in this case, the second order constraint can simply be obtained by multiplying the first equation of (5.86) by $r_w$, then adding the third equation and then subtracting the fourth. This results into

$$(r_w\, m_{11} - m_{41})\ \ddot{q}_1 - m_{42}\,\ddot{q}_2 + m_{33}\,\ddot{q}_3 + (r_w\, m_{14} - m_{44})\ \ddot{q}_4 + (r_w\, m_{15} - m_{45})\ \ddot{q}_5 + \\ + r_w\, c_1 - c_4 = 0 \tag{5.87}$$

As expected, the constraint has dimension 1. Its explicit expression is not here reported it is conceptually identical to (5.72a). As already done in the previous section, one can rewrite the system by using four of the five available equations (it is not important which one) and attaching to them the dynamical feasibility constraint. At that point, it is sufficient to extract, for example, $\ddot{\phi}_r$ from the constraint and insert it inside the other four equations. A partially feedback linearizing input of the form (5.75) can then be used to write the system in the form (5.79) - (5.78). Choosing as output the state of the system (paying attention to $\beta$, as done before), the zero dynamics turns out to be exactly the same as before (details omitted for brevity). This is to be expected, since zeroing the output essentially makes the model identical to the one valid during ground contact.
The second order constraint (5.87) is crucial and will be used in Section 6.6 when writing the Task Space controller.

## 5.7 Second order normal form of the eWSLIP with fixed length

It is also useful to obtain the second order normal form of the equivalent model 3.7 and, in particular, its internal dynamics when $l^p_{cm}$ is kept at a fixed value (or, equivalently, $\beta$ is held constant). The knowledge of this dynamics will be employed as a basis for performing trajectory tracking via LQR and Linear Stable Inversion in Chapter 6.
Being $l^p_{cm}$ fixed, the corresponding dynamics can be neglected. As a consequence, the Lagrangian state simply becomes

$$q_p = \begin{bmatrix} \theta_w \\ \theta_r \end{bmatrix} \tag{5.88}$$

which is exactly the same of a simple WIP.
The explicit expression of the dynamics is (omitting here all the passages for brevity)

$$\begin{bmatrix} 2\,J_w + m^{eq}_p\, {r_w}^2 + 2\,m_w\, {r_w}^2 & -l^p_{cm}\, m^{eq}_p\, r_w\, \cos(\theta_r) \\ -l^p_{cm}\, m^{eq}_p\, r_w\, \cos(\theta_r) & m^{eq}_p\, {l^p_{cm}}^2 + I^{eq}_p(l^p_{cm}) \end{bmatrix} \ddot{q}_p + \\ + \begin{bmatrix} -l^p_{cm}\, m^p_{eq}\, r_w\, \dot{\theta}_r^2\, \sin(q_2) \\ g\, l^p_{cm}\, m^p_{eq}\, \sin(q_2) \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} C_{m1} \tag{5.89}$$

Dynamics (5.89) is exactly the dynamics of a WIP. The second order contraint of this system can be simply obtained by subtracting the first and second equation. Then, it is sufficient to extract $\ddot{\theta}_w$ from one of the two equations of (5.89) and then substitute it inside the second order constraint. This allows to retrieve the explicit expression of the forced internal dynamics of the system. Since this procedure is exactly equivalent to the one carried out in the previous sections, details are here omitted for simplicity.
At the end, the internal dynamics will have, once again, the following structure:

$$M^{int}(q_p)\,\ddot{q}_p^{int} + C^{int}(q_p, \dot{q}_p) = G^{int}(q_p)\,v \tag{5.90}$$

where, in this case,

$$q_p^{int} = \begin{bmatrix} \theta_r \\ \dot{\theta}_r \end{bmatrix} \text{ and } v = \ddot{\theta}_w = \ddot{\phi}_w \tag{5.91}$$

Please refer to [17] for further details of the explicit form of the internal dynamics. Once an explicit expression for the internal dynamics is available, it is possible to retrieve its linearization around the vertical equilibrium. This equilibrium is characterized by $v = 0$ and $q_{int} = [0; 0]$.
The linearization becomes

$$\dot{\tilde{q}}^{int} = A^{int}\,\tilde{q}^{int} + B^{int}\,\tilde{v} \tag{5.92}$$

where

$$\begin{aligned} A^{int} &= \begin{bmatrix} 0 & 1 \\ \lambda^2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \dfrac{g\,l_{cm}^p\,m_{eq}^p}{m_{eq}^p\,{l_{cm}^p}^2 + m_{eq}^p\,r_w\,l_{cm}^p + I_{eq}^p(l_{cm}^p)} & 0 \end{bmatrix} \\ B^{int} &= \begin{bmatrix} 0 \\ \xi \end{bmatrix} = \begin{bmatrix} 0 \\ \dfrac{2\,J_w + m_{eq}^p\,{r_w}^2 + 2\,m_w\,{r_w}^2 + l_{cm}^p\,m_{eq}^p\,r_w}{m_{eq}^p\,{l_{cm}^p}^2 + m_{eq}^p\,r_w\,l_{cm}^p + I_{eq}^p(l_{cm}^p)} \end{bmatrix} \end{aligned} \tag{5.93}$$

and $\pm\lambda$ are the two real eigenvalues of matrix $A$. This linearization will be used in Section 6.3.

# Chapter 6

# Controls design and simulation

This Chapter is dedicated to discussing, setting up and testing of suitable control policies for performing trajectory tracking on the system.

Since the addition of Gazebo simulator happened quite close to the end of the thesis, not all the control policies are tested and validated using both the Matlab simulator and Gazebo.
In particular, only the IS-MPC (see Section 6.5) and the Task-Space tracker (see Section 6.6) are implemented and tested on both simulators.

## 6.1 LQR setpoint stabilization

The very first control which can be tested on the system is a simple stabilization around the vertical equilibrium using a state-feedback law of the type $u = -K\,q$. Clearly, to be able to perform a state-feedback control, the whole state needs to be available. In this specific case, the full state is indeed completely available (see Chapter 7).

To compute the gain matrix $K$, different approaches are possible. In particular, the chosen one is to employ a so called Linear Quadratic Regulator (frequently abbreviated with LQR).
The problem is stated in the following way: find the optimal state-feedback gain matrix $K$ such that the quadratic cost function

$$J(\tilde{q}, \tilde{u}) = \int_0^\infty \left( \frac{1}{2}\,\tilde{q}^T\,Q\,\tilde{q} + \frac{1}{2}\tilde{u}^T\,R\,\tilde{u} + \tilde{q}^T\,N\,\tilde{u} \right) dt \tag{6.1}$$

subject to (5.47), is minimized. Note that the general linearized system 5.47 has always an equilibrium point at the origin ($\tilde{q} = 0$ and $\tilde{u} = 0$); as a consequence, this way of formulating the stabilization problem makes sense.
The solution to this problem can be obtained by solving the dual optimization problem (note that, being the problem convex, the strong duality holds). Let us define the optimization Lagrangian as

$$\mathcal{L} = \int_0^\infty \left[ J(\tilde{q}, \tilde{u}) + \lambda^T \left( \dot{\tilde{q}} - A\,\tilde{q} - B\,\tilde{u} \right) \right] dt \tag{6.2}$$

Then, the solution to the constrained optimization problem can be found with

$$\delta\mathcal{L}(\tilde{q}, \dot{\tilde{q}}, \tilde{u}, \lambda) = 0 \tag{6.3}$$

By computing the infinitesimal increment of $\mathcal{L}$, employing integration by parts and by imposing solutions of the form $\lambda = -S\,\tilde{q}$, after rearranging properly the equations, one obtains

$$A^T S + S A - (S B + N) R^{-1} (B^T S + N^T) + Q = 0 \tag{6.4}$$

and

$$K = R^{-1} (B^T S + N^T) \tag{6.5}$$

where (6.4) is the matrix form of the well-known algebraic Riccati equation.
Substituting the control input into system's linearized dynamics, one obtains the closed-loop equation $\dot{\tilde{q}} = (A - B\,K)\ \tilde{q}$, where now the plant has all of its poles in the left complex plane, if the linearization of the system is stabilizable. In particular, we saw in Section 5.4 that the system is indeed linearly controllable around its vertical equilibrium; this also implies that a suitable state feedback, like for instance the one provided by the LQR, is potentially able to vertically stabilize the system.
The solution of (6.4) can be computed using numerical tools (e.g. Matlab's LQR function). To provide more reliable results, simulation involving LQR were carried out using the discrete-time equivalent of LQR function, which is lqrd. Furthermore, the cross-coupling weight matrix $N$ was not used, and $Q$ and $R$ matrices were set to diagonal ones for simplicity.

Specifically, simulations were carried out using the following weights:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix};\ R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.6}$$

Note that, for example, the state weight matrix $Q \in \mathbb{R}^{6\times 6}$. While the complete state of the floating-base model belongs to $\mathbb{R}^{10}$. This is simply because the LQR controller actually employs the contact model 3.4.
Furthermore, the following torque limits were imposed (via controller saturation):

$$u_{lim} = \begin{bmatrix} C_{m1}^{lim} \\ C_{m2}^{lim} \end{bmatrix} = \begin{bmatrix} 2\,\text{Nm} \\ 20\,\text{Nm} \end{bmatrix} \tag{6.7}$$

These limits are quite realistic values (see Chapter 7). Imposing the saturation of the controller is not the best way of employing input limits. Other control schemes, like for instance MPC, allow to impose limits directly inside the controller (see Section 6.5).

Let us look at two groups of simulations.
For all simulations, the target setpoint was chosen to be

$$q^{ref} = \left[1\,\text{m}, 0, 16.6667\,\text{rad}, -5.0502 \cdot 10^{-6}\,\text{rad}, 80 \cdot \frac{\pi}{180}\,\text{rad}, 0, 0, 0, 0, 0\right]^T \tag{6.8}$$

The first control simulation tests the performance of the controller for three different control frequencies (200 Hz, 50 Hz and 20 Hz) and the following initial condition:

$$q_0^{(0)} = \left[0, 0, 0, 0, 75 \cdot \frac{\pi}{180}\,\text{rad}, 0, 0, 0, 0, 0\right]^T \tag{6.9}$$

The results are shown in Fig. 6.1 (state evolution and actuation) and in Fig. 6.3 (ground reactions).

The controller successfully balances the system around the required setpoint for the tested frequencies. Looking at the graphs, the following considerations can be made:

- The horizontal ground reaction converges to a null value, which is to be expected and the vertical one converges to the total robot weight (approximately 116 N).
- Analogously, the horizontal actuation torques converges to a null value, while the vertical one settles around a value of approximately 2.5 Nm.
- The actuation graphs are actually staircase plots (this is particularly evident looking at the ones associated with a 20 Hz control frequencies).
- Some graphs, specifically the ones at 20 Hz, can have some derivative discontinuities. This is simply due to the fact that the sampled states are plotted at the sampling frequency. This is not a problem, since the underlying integration is performed at a much lower integration step.

Fig. 6.2 shows the results using the following set of initial conditions and a sampling frequency of 50 Hz:

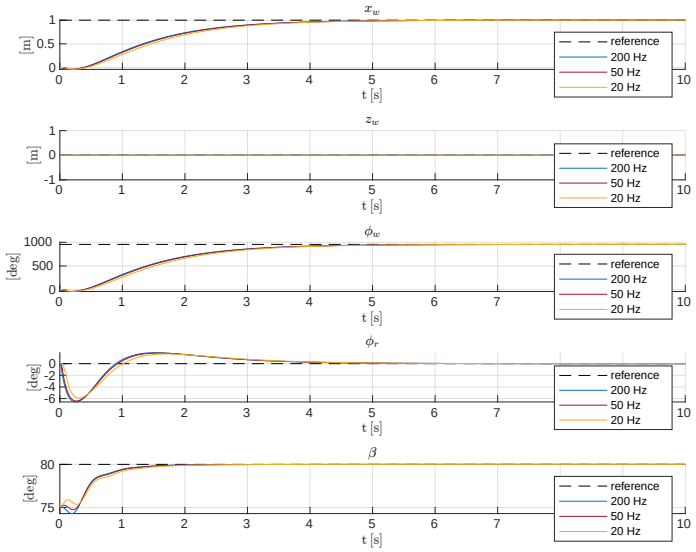
$$q_0^{(1)} = \left[-1\,\text{m}, 0, -16.666\,\text{rad}, 0, 75 \cdot \frac{\pi}{180}\,\text{rad}, 0, 0, 0, 0, 0\right] \tag{6.10a}$$

$$q_0^{(2)} = \left[0, 0, 0, 0\,\text{rad}, 75 \cdot \frac{\pi}{180}\,\text{rad}, 0.5, 0, 8.33, 0, 0\right] \tag{6.10b}$$

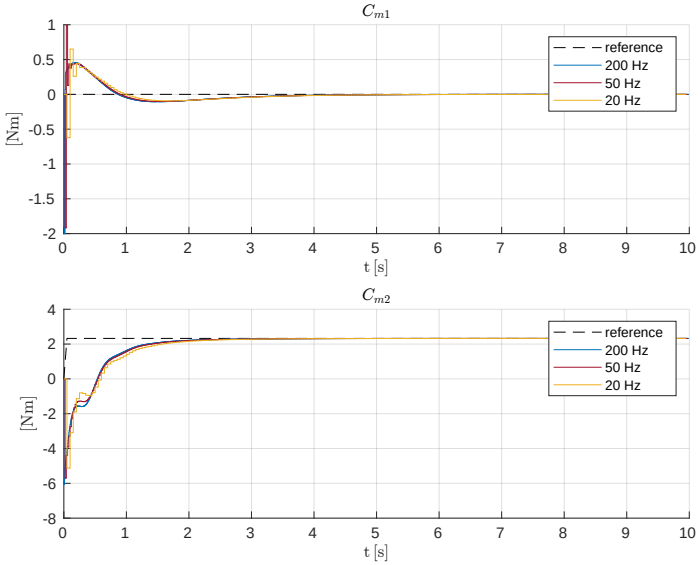
$$q_0^{(3)} = \left[0, 0, 0, 10 \cdot \frac{\pi}{180}\,\text{rad}, 75 \cdot \frac{\pi}{180}\,\text{rad}, 0, 0, 0, 0, 0\right] \tag{6.10c}$$

$$q_0^{(4)} = \left[0, 0, 0, 0, 75 \cdot \frac{\pi}{180}\,\text{rad}, 0, 0, 0, 20 \cdot \frac{\pi}{180}\text{rad/s}, 0\right] \tag{6.10d}$$

Looking at the plots, the controller is able to easily stabilize the system for the given range of initial conditions.




**(a)** $q_p$ evolution




**(b)** Actuation inputs

**Figure 6.1.** Comparing LQR setpoint simulation results for different control frequencies. At the top, a plot of the full Lagrangian state while, at the bottom, a plot of the control inputs. The associated ground reaction forces are shown in Fig. 6.3.

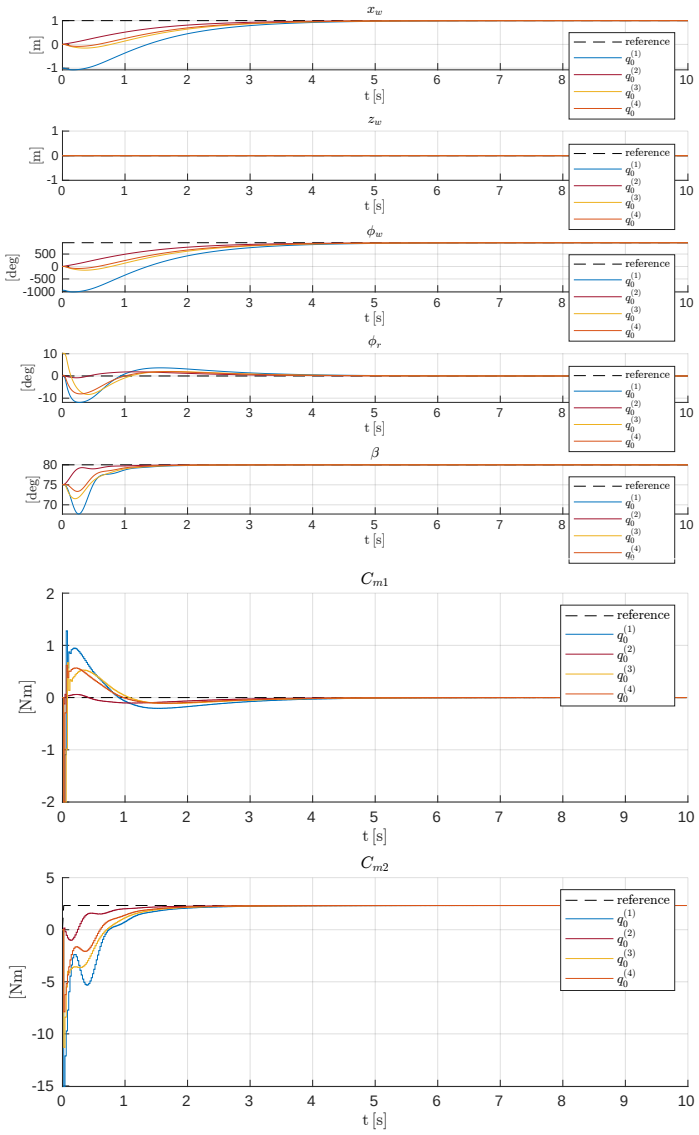



**Figure 6.2.** Comparing LQR setpoint simulation results for different initial conditions, defined by (6.10a)-(6.10d). At the top, a plot of the full Lagrangian state while, at the bottom, a plot of the control inputs.

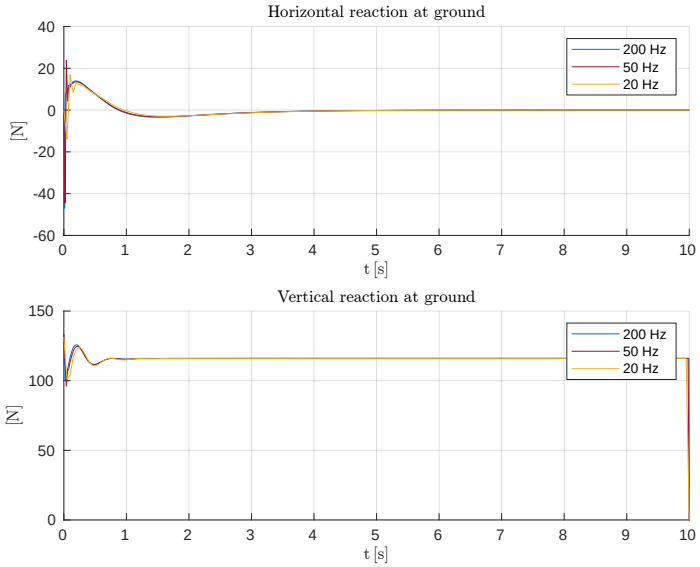



**Figure 6.3.** Comparing LQR setpoint ground reaction forces for different control frequencies. The associated Lagrangian state plot and actuations are shown in Fig. 6.1.

## 6.2 Brief introduction to linear stable inversion

Clearly, simply stabilizing the system around its vertical equilibrium is seldom enough.
Performing trajectory tracking on the system, due to the control properties discussed in Chapter 5, requires special attention.

To see why, let us consider the partially feedback linearized form (5.78)-(5.79). Once trajectories for $\bar{q}_p$ are assigned, (5.79) dynamics only depends on $q_p^{int}$ initial conditions. Specifically, looking at the system under contact, the explicit form of the internal dynamics is given by (5.73). If trajectories for $x_w$ and $\beta$ over time are assigned, then the evolution of the forced internal dynamics, i.e. the pitch angle, is completely determined by its initial conditions. Therefore, the only dynamically feasible pitch trajectory will be the one determined by the reference trajectories of the remaining states and the initial conditions on the pitch dynamics. As already stressed, this is a direct implication of the underactuation of the system.
Since we saw that the zero dynamics is unstable, the evolution of the forced internal dynamics will in general diverge. Being the internal dynamics the dynamics of the pitch angle, it cannot be allowed to do so.
This means that a controller, aside from performing trajectory tracking on the prescribed output, must also avoid the divergence of the forced internal dynamics of the system.

One of the ways of tackling this problem is to employ the so called *Stable Inversion* technique. The objective of Stable Inversion is to compute a non exponentially-divergent (also referred to as a *bounded*) trajectory for the unstable internal dynamics of a dynamical system. Examples of this technique applied to humanoid gait generation are given by [21] and [22].

Once a non-divergent trajectory for the internal dynamics is available, a suitable tracking controller for tracking the whole state can be employed.

Let use explore the case of *Linear* Stable Inversion.
Consider

$$\dot{\tilde{q}}^{int} = A^{int}\,\tilde{q}^{int} + B^{int}\,\tilde{v} \tag{6.11}$$

Let this system be a linearization of the internal dynamics of an underactuated and non-minimum phase system around an operating condition. Being the system non-minimum phase, matrix $A^{int}$ will have at least one pole with positive real part. By employing the knowledge of the right eigenvectors of $A^{int}$, if $A^{int}$ is diagonalizable, it is possible to use the following change of coordinates to decompose the system into stable and unstable subsystems:

$$\tilde{q}^{int} = T^{-1}\,\zeta \tag{6.12}$$

where $T^{-1}$ is obtained by simply concatenating the stable right eigenvectors of the system with the unstable ones. With this choice, the first subsystem will be the stable one, while the remaining will be unstable.
Applying this change of coordinates and left multiplying the system by $T$ leads to

$$\dot{\zeta} = \Lambda\,\zeta + \left(T\,B^{int}\right)\,\tilde{v} \tag{6.13}$$

where $\Lambda$ is diagonal matrix containing the eigenvalues of the system. Note that, if $A^{int}$ was instead defective, then a suitable Jordan decomposition could be employed. In this case $\Lambda$ would be a block diagonal matrix.
Without loss of generality, the transformed state can partitioned into

$$\zeta = \begin{bmatrix} \zeta_s \\ \zeta_u \end{bmatrix} \tag{6.14}$$

where the subscripts $s$ and $u$ indicate, respectively, the state associated with the stable system and the unstable one.
In particular, let us look at the unstable subsystem

$$\dot{\zeta}_u = \Lambda_u\,\zeta_u + B_u\,\tilde{v} \tag{6.15}$$

It is well known that the solution to this system is given by

$$\begin{aligned} \zeta_u(t) &= e^{\Lambda_u\,t}\,\zeta_u(0) + \int_0^t e^{\Lambda_u\,(t-\tau)}\,B_u\,\tilde{v}(\tau)\,d\tau = \\ &= e^{\Lambda_u\,t}\,\left(\zeta_u(0) + \int_0^t e^{-\Lambda_u\,\tau}\,B_u\,\tilde{v}(\tau)\,d\tau\right) \end{aligned} \tag{6.16}$$

which, as expected, depends on the input to the internal dynamics and on the initial condition of the unstable subsystem.
By choosing the particular initial condition

$$\zeta_u(0) = -\int_0^\infty e^{-\Lambda_u\,\tau}\,B_u\,\tilde{v}(\tau)\,d\tau \tag{6.17}$$

The solution becomes

$$\begin{aligned}\zeta_u(t) &= -\int_t^{\infty} e^{\Lambda_u\,(t-\tau)}\,B_u\,\tilde{v}(\tau)\,d\tau = \\ &= -\int_0^{\infty} e^{-\Lambda_u\,\nu}\,B_u\,\tilde{v}(t+\nu)\,d\nu\end{aligned} \tag{6.18}$$

Under mild hypothesis of boundness $v$, the solution given by (6.18) will not diverge, being the input contribution weighted by a decaying exponential.
The remaining part of the internal dynamics will evolve according to

$$\begin{aligned}\zeta_s(t) &= e^{\Lambda_s\,t}\,\zeta_s(0) + \int_0^{t} e^{\Lambda_s\,(t-\tau)}\,B_s\,\tilde{v}(\tau)\,d\tau = \\ &= e^{\Lambda_s\,t}\,\left(\zeta_s(0) + \int_0^{t} e^{-\Lambda_s\,\tau}\,B_s\,\tilde{v}(\tau)\,d\tau\right)\end{aligned} \tag{6.19}$$

In this case, since the system is stable, the choice of the initial condition $\zeta_s(0)$ is arbitrary. One alternative is to choose the initial condition of the stable component so that its steady state solution is reached immediately at $t = 0$.
There are also inversion-based procedures which can be applied directly to non-linear systems, like for instance [11].

## 6.3 Linear stable inversion on a simple WIP for sinusoidal trajectory tracking

Let us see a simplified yet useful example of linear stable inversion applied to a wheeled inverted pendulum. The results obtained with this analysis will be employed in Section 6.4.

Consider the linear approximation around the vertical equilibrium of the internal dynamics of a simple WIP which was derived in Section 5.7. Let us try to derive a bounded trajectory for the linear approximation of the internal dynamics forced by a sinusoidal wheel trajectory

$$x_w^{ref}(t) = \rho\,\sin(\omega\,t) \rightarrow \theta_w^{ref}(t) = \frac{\rho}{r_w}\,\sin(\omega\,t) \tag{6.20}$$

Given this reference trajectory, the internal dynamics will be forced by the input

$$v(t) = \ddot{\phi}_w^{ref}(t) = -\frac{\rho\,\omega^2}{r_w}\,\sin(\omega\,t) \tag{6.21}$$

The linearized internal dynamics can be decomposed into stable and unstable subsystems by employing the change of coordinates described by

$$T^{-1} = \begin{bmatrix} 1 & 1 \\ -\lambda & \lambda \end{bmatrix} \text{ and } T = \begin{bmatrix} 1/2 & -1/(2\,\lambda) \\ 1/2 & 1/(2\,\lambda) \end{bmatrix} \tag{6.22}$$

where $\pm\lambda$ are the eigenvalues of the linearization (see Section 5.7). In this specific case, the integrals (6.18) and (6.19) can be computed analytically (see [17]). As

already pointed out, it is convenient to choose the initial condition of the stable subsystem so as to cancel any transient behaviour (see again [17]).
After that, it is sufficient to go back to the original coordinates by employing (6.12) to retrieve the bounded trajectory for the internal dynamics. The result is

$$q^{int,ref} = \begin{bmatrix} \theta_r^{ref}(t) \\ \\ \dot{\theta}_r^{ref}(t) \end{bmatrix} = \begin{bmatrix} \rho \dfrac{\xi}{r_w} \dfrac{\omega^2}{(\lambda^2+\omega^2)} \sin(\omega\, t) \\ \\ \rho \dfrac{\xi}{r_w} \dfrac{\omega^3}{(\lambda^2+\omega^2)} \cos(\omega\, t) \end{bmatrix} \tag{6.23}$$

This reference trajectory, as it will be seen in the upcoming sections, can be used to perform trajectory tracking not only on a simple WIP (see [17]), but also on the more complex system 3.4.
Note that both $\xi$ and $\lambda$ depend upon the inertia of the equivalent pendulum $I_{eq}^p$ and its equivalent length $l_{cm}^p$ (see (5.93)).

## 6.4 LQR trajectory tracking with Linear Stable Inversion-generated reference trajectories

The idea is to employ the bounded reference trajectory provided by 6.23 to perform output tracking. Before trying this option, a few things have to be highlighted:

- The reference trajectory 6.23 was derived employing the eWSLIP model with fixed pendulum length (basically a WIP). It is therefore necessary to transform the reference trajectory into the original model.
- The model used to perform Stable Inversion is linear; as a consequence, this will introduce an approximation whose validity will decrease as the system moves away from its vertical equilibrium.
- Furthermore, the obtained bounded trajectory is "exact" when considering the linearization of the internal dynamics of a WIP around the vertical equilibrium. However, this is not a trajectory of the actual system, since the internal dynamics is also forced by the reference trajectory given on the leg d.o.f. (see (5.73)). This is the second approximation introduced.

### 6.4.1 Tracking of sinusoidal output trajectories

Let the chosen outputs of the system be the wheel position $x_w$ (or, equivalently, $\phi_w$) and the leg angle $\beta$. This choice is coherent with the normal forms obtained in Chapter 5 and, in particular, with the normal form (5.79)-(5.78).

Suppose given desired outputs of the form

$$\begin{aligned} \phi_w^{ref}(t) &= \frac{x_w^{off}}{r_w} + \frac{\rho_w}{r_w} \sin(2\,\pi\, f_w\, t) \\ \beta^{ref}(t) &= \beta^{off} + \rho_\beta \sin(2\,\pi\, f_\beta\, t) \end{aligned} \tag{6.24}$$

From the analysis made in Section 6.2 we know that, given desired output trajectories (6.24), the evolution of the remaining state $\phi_r$ will be completely determined by its

initial condition. If this initial condition is chosen properly, as already discussed in Section 6.2, the resulting trajectory will remain bounded over time. This is the guiding principle of the so called Stable Inversion. Performing it on the complete system becomes problematic, since finding an analytical solution for the trajectory is not trivial at all, due to the complexities introduced by the legs.

One way to circumvent the problem is to actually employ as a reference for the internal dynamics (i.e. the pitch angle) the bounded solution provided by (6.23). As already pointed out, this introduces an approximation which may or not be acceptable. As we will see in the upcoming simulations, this approach actually proves to be quite effective.
Since (6.23) actually refers to the eWSLIP model, it is first necessary to back to the original system. This is done using the change of coordinates (see Section 3.5)

$$\phi_r(t) = \theta_r(t) + \phi_r^*(\beta(t)) \tag{6.25}$$

As a consequence, the desired complete state trajectory will be

$$q_p^{ref}(t) = \begin{bmatrix} \phi_w^{ref}(t) \\ \phi_r^{ref} \\ \beta^{ref}(t) \end{bmatrix} = \\ = \begin{bmatrix} \dfrac{x_w^{off}}{r_w} + \dfrac{\rho_w}{r_w} \sin(2\,\pi\, f_w\, t) \\ \phi_r^*(\beta^{ref}) + \rho_w \dfrac{\xi(l_{cm}^p(\beta^{ref}), I_{eq}^p(\beta^{ref}))}{r_w} \dfrac{\omega_w^2}{[\lambda^2(l_{cm}^p(\beta^{ref}), I_{eq}^p(\beta^{ref})) + \omega_w^2]} \sin(\omega_w\, t) \\ \beta^{off} + \rho_\beta \sin(2\,\pi\, f_\beta\, t) \end{bmatrix} \tag{6.26}$$

where $\phi_r^{ref}$ is actually also directly function of $l_{cm}^p(\beta)$ and $I_{eq}^p(\beta)$. Plots of these last two quantities are reported in Fig. 6.4 for completeness.
The first and second derivatives of the reference state trajectory can be simply obtained by differentiation of (6.26) w.r.t. time. As already stressed, references up to the second derivative of the Lagrangian state are necessary to be able to compute the reference input torques.

Given the known references $q_p^{ref}$, $\dot{q}_p^{ref}$ and $\ddot{q}_p^{ref}$, it is possible to use (5.50) (or (5.51)) and (5.52) to compute the reference values for $u$, i.e. the control torques.
The control input can then be chosen to be

$$u(t) = u^{ref}(t) + \tilde{u}(t) \tag{6.27}$$

where $u^{ref}$ is a feedforward term given by the reference input and $\tilde{u}$ is a correction term which can be computed using a suitable controller.
The idea is to use a state feedback of the form $\tilde{u} = -K\,\tilde{q}$, where the gain matrix $K$ is computed using an LQR controller. Normally, the LQR is best suited for performing setpoint stabilization; however, if managed properly, it can also be used successfully to perform trajectory tracking.
Since computing the gain matrix via LQR requires a linear approximation of the

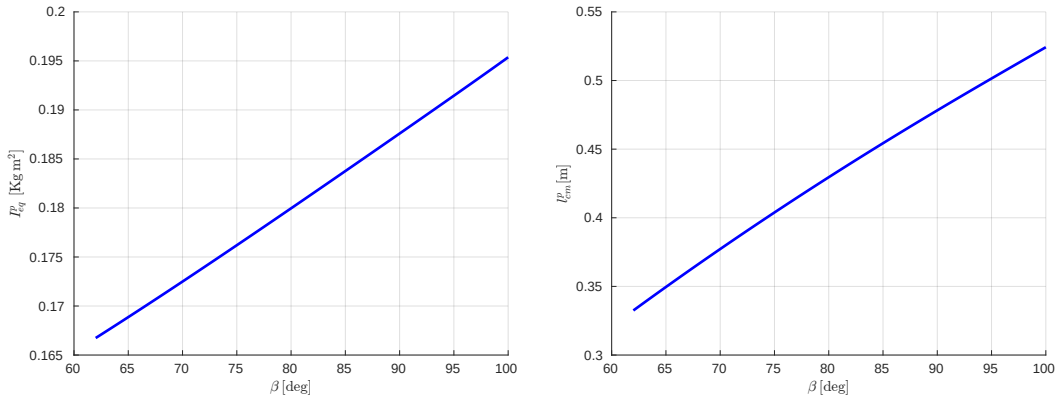



**Figure 6.4.** Plots of the equivalent pendulum length $l^p_{cm}$ (left) and of the equivalent rotational inertial $I^p_{eq}$ (right) as a function of the leg angle.

system, the idea is to use a linearization of the system around the reference trajectory given by (6.26). At each control sample time, the reference trajectory at the current time is read and the linearization around that reference state (5.47) is computed using (5.48) and (5.49).
This linearization is then used to compute the gain matrix $K$ at the current sample time and then obtain, given the current state, the correction input $\tilde{u}$.
Note that, this way, the gain matrix is computed as if the system was frozen around the current reference state from the current time up to $\infty$. This is clearly an approximation which, however, revealed itself to be acceptable.
Furthermore, remember that the obtained bounded trajectory for the pitch angle is actually function of the leg angle $\beta$ through $l^p_{cm}$ and $I^p_{eq}$. This means that, when differentiating the state trajectories w.r.t. time, expressions become quite complicated and difficult to manage. To overcome this issue, the dependence on $\beta$ of $l^p_{cm}$ and $I^p_{eq}$ is completely removed by computing them for a fictitious mean value $\bar{\beta}$ in the range of admissible $\beta$. In the upcoming simulations, this value was chosen to be 75°.

With this approximations, the reference state simply becomes

$$
\begin{aligned}
q_p^{ref}(t) &= \begin{bmatrix} \phi_w^{ref}(t) \\ \phi_r^{ref} \\ \beta^{ref}(t) \end{bmatrix} = \\
&= \begin{bmatrix} \dfrac{x_w^{off}}{r_w} + \dfrac{\rho_w}{r_w}\,\sin(2\,\pi\,f_w\,t) \\ \phi_r^*(\beta^{ref}) + \rho_w\,\dfrac{\xi(\bar{l}^p_{cm},\bar{I}^p_{eq})}{r_w}\,\dfrac{\omega_w^2}{\left[\lambda^2(\bar{l}^p_{cm},\bar{I}^p_{eq}) + \omega_w^2\right]}\,\sin(\omega_w\,t) \\ \beta^{off} + \rho_\beta\,\sin(2\,\pi\,f_\beta\,t) \end{bmatrix}
\end{aligned}
\tag{6.28}
$$

where the bar indicates quantities computed employing $\bar{\beta}$.
Note that, to be able to compute the first and second derivatives of the reference

state trajectory, it is necessary to employ the following relationships

$$\begin{aligned}\dot{\phi}_r^*(\beta(t),\dot{\beta}(t)) &= \frac{\partial \phi_r^*}{\partial \beta}(\beta(t))\,\dot{\beta}(t)\\ \ddot{\phi}_r^*(\beta(t),\dot{\beta}(t),\ddot{\beta}(t)) &= \frac{\partial \phi_r^*}{\partial \beta}(\beta(t))\,\ddot{\beta}(t) + \frac{\partial^2 \phi_r^*}{\partial \beta^2}(\beta(t))\,\dot{\beta}^2(t)\end{aligned} \tag{6.29}$$

As already done for the setpoint stabilization, input limits are handled using a saturation applied directly to the computed control inputs. In particular, the used input limits are (6.7).

Let us look at some representative simulation examples. All the LQR weight were kept constant throughout all the simulations and equal to (6.6).

### Comparing the results for different initial conditions

The first group of simulations are perfomed imposing once again the set of initial conditions (6.10a) - (6.10d) and a fixed sample frequency of 50 Hz. The desired trajectory is given by (6.28), where now

$$\begin{aligned}x_w^{off} = 0\,\text{m};\ \rho_w = 1\,\text{m};\ f_w = 0.2\,\text{Hz}\\ \beta^{off} = 80\,\text{deg};\ \rho_\beta = 10\,\text{deg};\ f_\beta = 1\,\text{Hz}\end{aligned} \tag{6.30}$$

The results are shown in Fig. 6.5. In particular, the tracking of the output trajectory is performed successfully over the whole range of initial conditions.

The controller is able to perform quite a good tracking of the output, while it is possible to appreciate, by looking closely at the evolution of $\phi_r$, that the tracking of the pitch angle has slight oscillations around the reference trajectory. Oscillations around the references are also visible by looking at $C_{m1}$ plot. These are most likely the result of the superposition of all adopted approximations which were made for obtaining the reference trajectory of the pitch angle. One of the most impactful approximations is that, as already stressed, the pitch reference trajectory does not actually obey strictly the second order constraint (5.72a). This, as well as the use of the linearization around the vertical equilibrium to perform Stable Inversion, are undoubtedly potential sources of tracking errors.

Note that, even if the tracked trajectory is quite aggressive, control inputs stay inside their maximum values for most of the simulation time. The only instants in which the input saturates (in particular $C_{m1}$) are the ones right at the beginning of the simulation, where the state error is the biggest.

### Control input saturation effect

Fig. 6.6 shows the results of a group of simulations carried out by using as reference trajectory (6.28) with

$$\begin{aligned}x_w^{off} = 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.3\,\text{Hz}\\ \beta^{off} = 80\,\text{deg};\ \rho_\beta = 15\,\text{deg};\ f_\beta = 0.2\,\text{Hz}\end{aligned} \tag{6.31}$$

and, once again, the set of initial conditions given by (6.10a) - (6.10d). In this case, the reference $\beta$ trajectory was chosen so that saturation of the leg control input is reached. This is clearly visible in $C_{m2}$ plot: when the reference leg angle exceeds the maximum achievable leg extension compatible with control limits, a plateau appears. Saturation effects are also clearly visible in the vertical reaction, as well as in $\beta$ and $\dot{\beta}$ plots, while they have an almost negligible impact on the tracking performance of the other states.

### Constant $\beta$ reference

To further examine the impact of the used approximations on the simulation, it is particularly interesting to perform a test with constant $\beta$ reference and, in particular, equal to the used $\bar{\beta}$ (in this case 75 deg). This way, the reference state makes the system (at steady state) completely equivalent to a WIP. As a consequence, the bounded reference trajectory for $\phi_r$ computed via Linear Stable Inversion becomes "exact". In particular, the simulation is performed using the initial condition (6.9) and the reference trajectory given by (6.28) with, in particular,

$$\begin{aligned} x_w^{off} &= 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.3\,\text{Hz} \\ \beta^{off} &= 75\,\text{deg};\ \rho_\beta = 0\,\text{deg};\ f_\beta = \text{N.D.} \end{aligned} \tag{6.32}$$

In this case, as expected, the tracking of the pitch trajectory is almost perfect (see Fig. 6.7).
The other states are also tracked with very good performance. It is also possible to see that reference inputs are closely tracked without oscillations at steady state (see Fig. 6.5 for comparison), which is an indication that the reference trajectories are substantially compatible with system's dynamics (and hence satisfy the second order constraint (5.72a)).

### Leaving $\beta$ d.o.f. uncontrolled

The last performed simulation is done to test the controller when $\beta$ d.o.f. is left uncontrolled. This, in particular, is achieved setting $C_{m2} = 0$ and using the current (measured) values of $\beta$ and $\dot{\beta}$ to compute both the reference inputs (in this case only $C_{m1}$) and the linearization of the system. Once again, the chosen initial condition is (6.9), while the reference trajectory is

$$x_w^{off} = 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.3\,\text{Hz} \tag{6.33}$$

The results are shown in Fig. 6.8.
The first thing to note is that obviously, since $C_{m2} = 0$, $\beta$ evolves freely and oscillates (there is no damping applied to that d.o.f.). Aside from that, tracking on the remaining states is achieved with good performance, while the "deleterious" effects of a free-evolving $\beta$ are clearly visible by looking at the plots of $C_{m1}$ (recall the approximation relative to the use of $\bar{\beta} = 75\,\text{deg}$). In fact, it is possible to appreciate relatively large oscillations around the reference input.

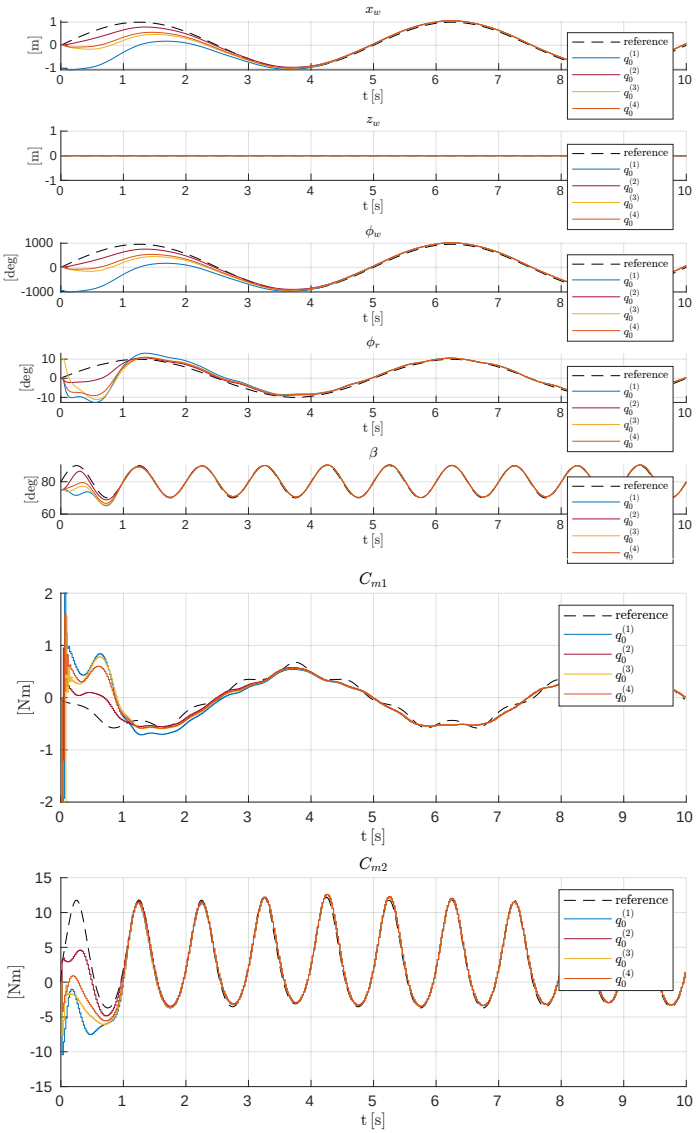



**Figure 6.5.** Comparing the perfomance of the LQR tracker for different initial conditions. In particular, the tracking is performed on a $x_w$ sinusoidal trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 1\,\text{m}$, $f_w = 0.2\,\text{Hz}$ and on a $\beta$ sinusoidal trajectory with $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 1\,\text{Hz}$. The controller runs at $50\,\text{Hz}$.

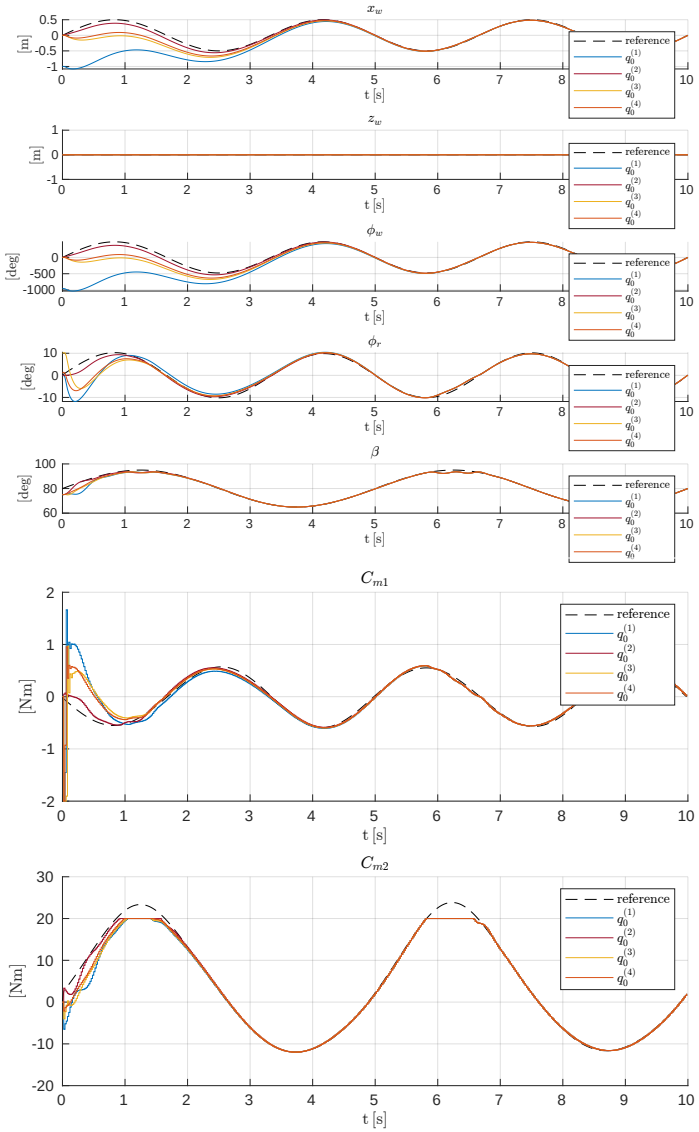



**Figure 6.6.** Comparing the perfomance of the LQR tracker for different initial conditions. In particular, the tracking is performed on a $x_w$ sinusoidal trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and on a $\beta$ sinusoidal trajectory with $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 15\,\text{deg}$, $f_\beta = 0.2\,\text{Hz}$. Due to the amplitude of $\beta$ reference trajectory, the leg control input saturates. The controller runs at 50 Hz.

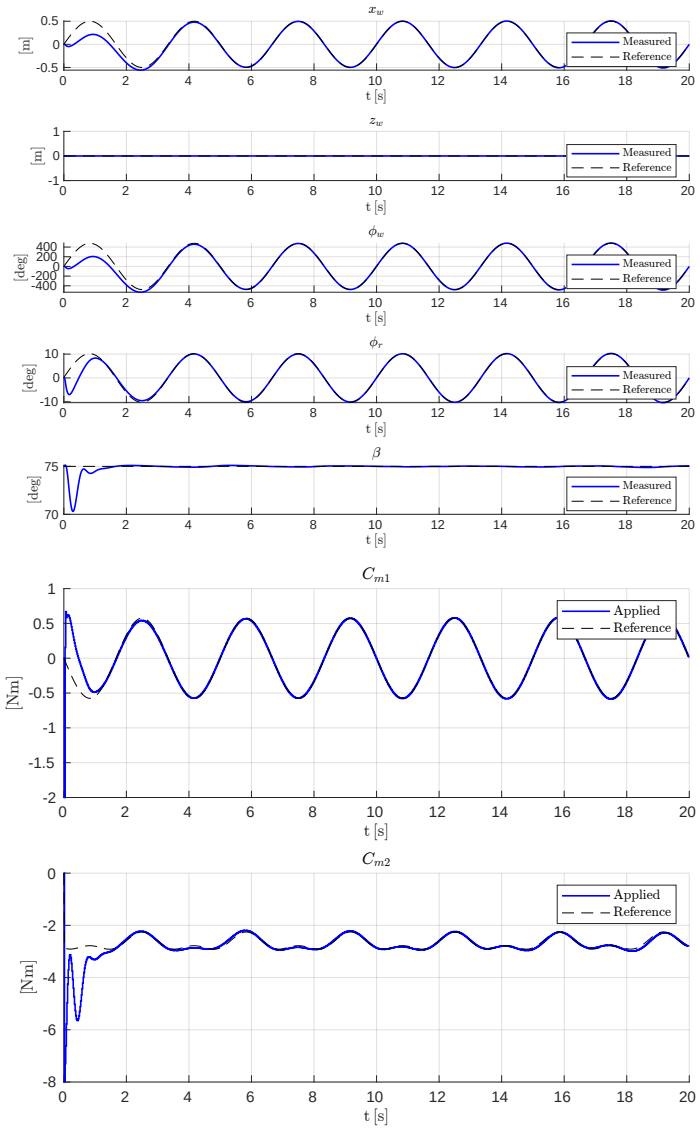



**Figure 6.7.** Results of the LQR tracker when $\beta^{ref}$ is kept to a constant value, equal to $\bar{\beta} = 75\,\text{deg}$. In particular, the tracking is performed on a $x_w$ sinusoidal trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$. The controller runs at 50 Hz.

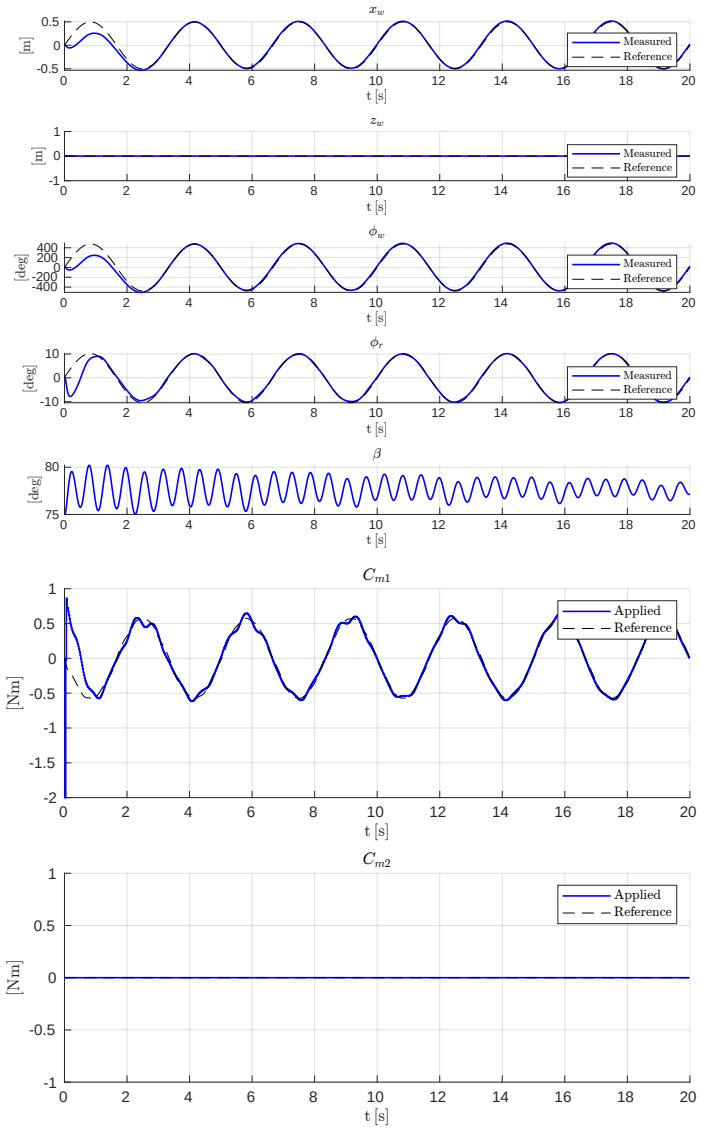



**Figure 6.8.** Results of the LQR tracker when the leg is not actuated. In particular, the tracking is performed on a $x_w$ sinusoidal trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$. The controller runs at $50\,\text{Hz}$.

### 6.4.2 Tracking of more complex output trajectories employing FFT and Linear Stable Inversion

In section 6.4 we have successfully tested and validated the LQR + approximated Linear Stable Inversion control for tracking of harmonic reference trajectories. Equation (6.23) and in particular the second element of (6.28) provide an approximated bounded trajectory for the pitch angle, given sinusoidal reference outputs for $x_w$ (or, equivalently $\phi_w$) and $\beta$.

Since the control works fairly well for simple sinusoidal reference outputs, it is reasonable to expect that it might also work if, instead of simple sinusoids, the output was given as a superposition of an arbitrary number of simple harmonics. Clearly, since here we are dealing with approximated stable-inverted trajectories and a linear approximation of the system, there is no certainty as to if such an approach would actually work.

The test is performed using (6.9) as initial condition and the reference output trajectory

$$\begin{aligned} x_w^{ref}(t) &= x_w^{off} + \sum_{i=1}^{3} \rho_w^{(i)} \, \sin(2\,\pi\, f_w^{(i)}\, t + \psi_w^{(i)}) \\ \beta^{ref}(t) &= \beta^{off} + \rho_\beta \, \sin(2\,\pi\, f_\beta\, t) \end{aligned} \tag{6.34}$$

where

$$\begin{aligned} P_w &= \left[\rho_w^{(1)},\, \rho_w^{(2)},\, \rho_w^{(3)}\right] = [0.1,\, 0.5,\, 0.3] \text{ m} \\ F_w &= \left[f_w^{(1)},\, f_w^{(2)},\, f_w^{(3)}\right] = [1,\, 0.2,\, 0.5] \text{ Hz} \\ \Psi_w &= \left[\psi_w^{(1)},\, \psi_w^{(2)},\, \psi_w^{(3)}\right] = [30,\, 0,\, -60] \text{ deg} \\ x_w^{off} &= 0\,\text{m} \end{aligned} \tag{6.35}$$

and

$$\begin{aligned} \beta^{off} &= 80\,\text{deg} \\ \rho_\beta &= 10\,\text{deg} \\ f_\beta &= 0.2\,\text{Hz} \end{aligned} \tag{6.36}$$

As a consequence, the reference trajectory for the pitch angle $\phi_r$ can simply be obtained as

$$\phi_r^{ref}(t) = \phi_r^*(\beta^{ref}) + \sum_{i=1}^{3} \left( \rho_w^{(i)} \, \frac{\xi(\bar{l}_{cm}^p, \bar{I}_{eq}^p)}{r_w} \, \frac{\left(\omega_w^{(i)}\right)^2}{\left[\lambda^2(\bar{l}_{cm}^p, \bar{I}_{eq}^p) + \left(\omega_w^{(i)}\right)^2\right]} \, \sin(\omega_w^{(i)}\, t + \psi_w^{(i)}) \right) \tag{6.37}$$

where $\xi(\bar{l}_{cm}^p, \bar{I}_{eq}^p)$ and $\lambda^2(\bar{l}_{cm}^p, \bar{I}_{eq}^p)$ are computed using $\bar{\beta} = 75\,\text{deg}$, while $\omega_w^{(i)} = 2\,\pi\, f_w^{(i)}$.

In the performed simulation (and also in the ones of the following subsections), the reference trajectory (6.34) (in particular the first one), is fed into an FFT algorithm to be able to retrieve the amplitudes and phase shifts of the single harmonics. Since these are actually known in advance, it was possible to verify the effectiveness of such an approach. Clearly, performing an FFT of the input trajectory inevitably introduces, if the input trajectory is not perfectly harmonic, additional

tracking errors. Differently w.r.t. previous simulations, this one if performed using a simulation time of $20s$.

The results are shown in Fig. 6.9.

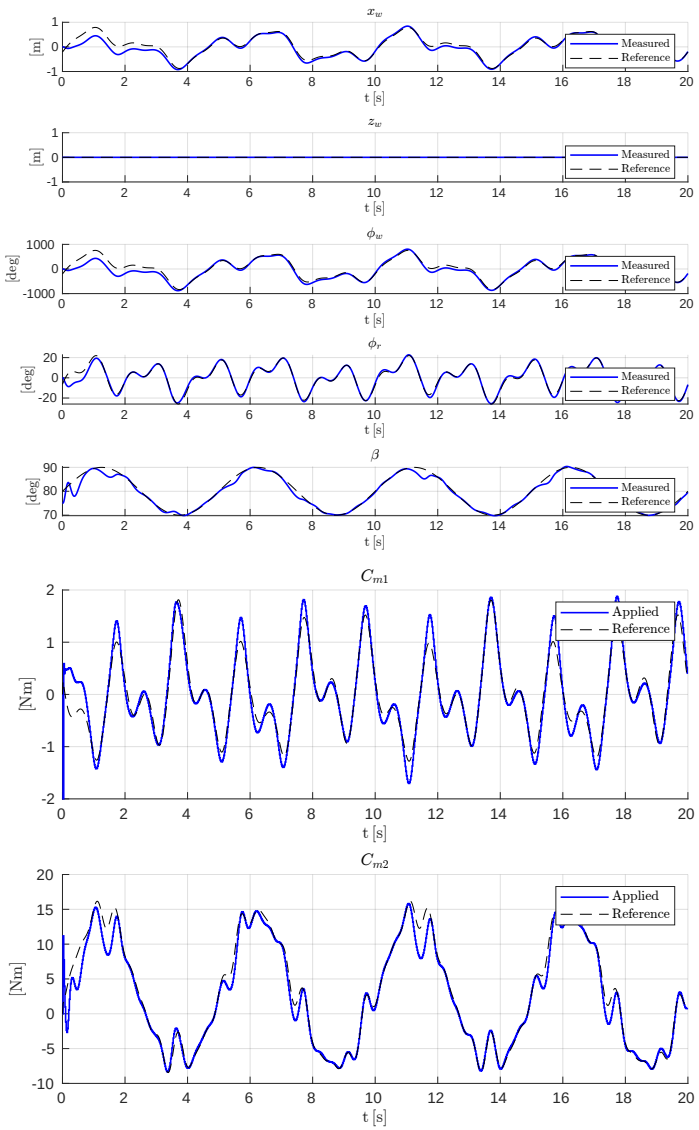



**Figure 6.9.** Results of the LQR tracker when using superposed harmonic wheel trajectories. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $P_w = [0.1, 0.5, 0.3]$ m, $F_w = [1, 0.2, 0.5]$ Hz, $\Psi_w = [30, 0, -60]$ deg and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 0.2\,\text{Hz}$. The controller runs at 50 Hz.

The chosen $x_w$ trajectory is relatively aggressive and, as a consequence, is particularly suited for testing the efficacy of this approach. The tracking of the $x_w$ and the pitch angle $\phi_r$ is quite good, while it is possible to appreciate some periodic tracking errors on $\beta$ d.o.f.; these are most likely the cumulative result of all used approximations.

### 6.4.3 Controlling the height of the center of mass

It might be useful to control the center of mass of the system. In particular, this subsection tests the control of the CoM height using the already developed approach. A more general, flexible and effective way of controlling the CoM of the system will be presented in Section 6.6.

Suppose a given reference trajectory for the CoM height of the form

$$h_{CoM}^{ref}(t) = h^{off} + \rho_h \, \sin(2\,\pi\, f_h\, t) \tag{6.38}$$

The tracking of this trajectory is achieved employing the following steps:

- Use the available $x_w$ reference trajectory (first equation in (6.34)) and perform the approximated Stable Linear Inversion procedure to obtain

$$\theta_r^{ref}(t) = \sum_{i=1}^{3} \left( \rho_w^{(i)} \, \frac{\xi(\bar{l}_{cm}^p, \bar{I}_{eq}^p)}{r_w} \, \frac{\left(\omega_w^{(i)}\right)^2}{\left[\lambda^2(\bar{l}_{cm}^p, \bar{I}_{eq}^p) + \left(\omega_w^{(i)}\right)^2\right]} \, \sin(\omega_w^{(i)}\, t + \psi_w^{(i)}) \right) \tag{6.39}$$

  where $\xi(\bar{l}_{cm}^p, \bar{I}_{eq}^p)$ and $\lambda^2(\bar{l}_{cm}^p, \bar{I}_{eq}^p)$ are computed using a reference value for $\bar{\beta}$ (in the upcoming simulations a value of 75 deg will be used).

- Given (6.38), obtain the required equivalent pendulum length trajectory $l_{cm}^p(t)$ as

$$l_{cm}^p(t) = \frac{h_{CoM}^{ref}(t)}{\cos(\phi_r^{ref} - \phi_r^*(\beta^{ref}))} \, \frac{\left(m_{eq}^p + 2\, m^w\right)}{m_{eq}^p} \tag{6.40}$$

  Differentiating this expression w.r.t. time, one is able to obtain also $\dot{l}_{cm}^p(t)$ and $\ddot{l}_{cm}^p(t)$.

- Interpolate graph 6.4 to invert the relationship $l_{cm}^p = l_{cm}^p(\beta)$ (which is not easily invertible analytically) and retrieve, doing so, a reference trajectory $\beta^{ref}(t)$ for the leg angle.

- Use the known expressions of $\frac{\partial l_{cm}^p}{\partial \beta}(\beta)$, $\frac{\partial^2 l_{cm}^p}{\partial \beta^2}(\beta)$ and the obtained reference leg angle trajectory to compute

$$\begin{aligned} \dot{\beta}^{ref}(t) &= \frac{\dot{l}_{cm}^p(t)}{\frac{\partial l_{cm}^p}{\partial \beta}(\beta^{ref})(t)} \\ \ddot{\beta}^{ref}(t) &= \frac{\ddot{l}_{cm}^p(t) - \frac{\partial^2 l_{cm}^p}{\partial \beta^2}(\beta^{ref}(t)) \, \left(\dot{\beta}^{ref}(t)\right)^2}{\frac{\partial l_{cm}^p}{\partial \beta}(\beta^{ref}(t))} \end{aligned} \tag{6.41}$$

- Finally, compute

$$\phi_r^{ref}(t) = \phi_r^*(\beta^{ref}(t)) + \theta_r^{ref}(t) \tag{6.42}$$

The above described steps allow to obtain a reference trajectory for the Lagrangian state. Let us test this approach on a couple of representative simulations.

First, let us use (6.9) as initial condition, set the reference CoM height at a constant value of 0.4 m and use as a reference trajectory for $x_w$ (6.33). The results are shown in Fig. 6.10 and Fig. 6.13.

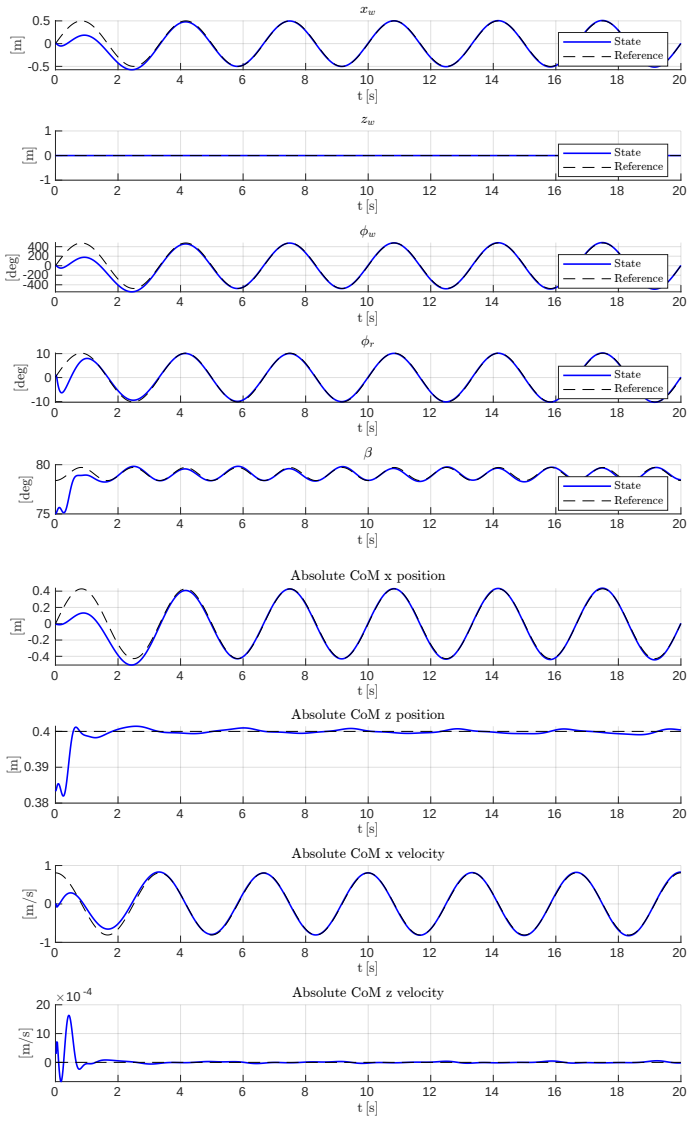



**Figure 6.10.** Results of the LQR tracker for controlling the CoM height, while tracking a reference trajectory of $x_w$. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and a reference CoM height described by 0.4 m. The controller runs at 50 Hz. The associated control inputs are shown in Fig. 6.13.

The controller is able to achieve the required height, while tracking with good results the desired $x_w$ trajectory. There are periodic tracking tracking errors around the reference height most likely due to the employed approximations for computing

the bounded pitch trajectory.

Fig. 6.11 and Fig. 6.14 show the results of a simulation with the same initial conditions as Fig. 6.10 and reference $x_w$ trajectory. This time, the reference CoM height trajectory is set as

$$\begin{aligned} h^{off} &= 0.038\,\text{m} \\ \rho_h &= 0.04\,\text{m} \\ f_h &= 0.3\,\text{Hz} \end{aligned} \tag{6.43}$$

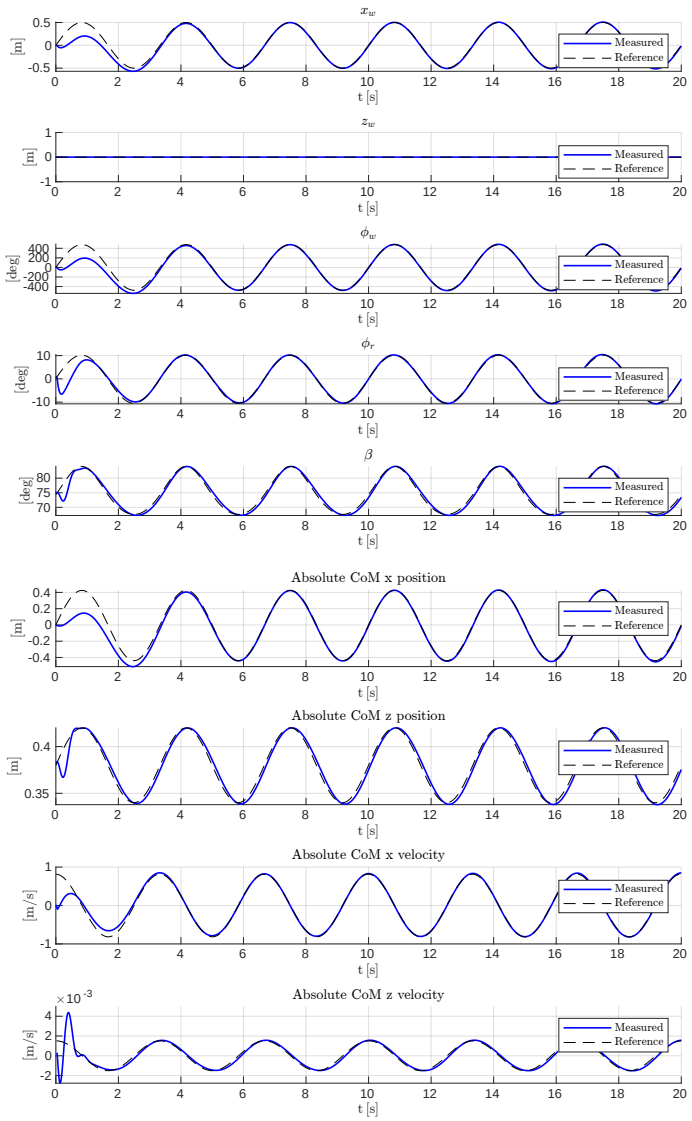



**Figure 6.11.** Results of the LQR tracker for controlling the CoM height, while tracking a reference trajectory of $x_w$. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and the reference CoM height trajectory is $h^{off} = 0.038\,\text{m}$, $\rho_h = 0.04\,\text{m}$, $f_h = 0.3\,\text{Hz}$. The controller runs at 50 Hz. The associated control inputs are shown in Fig. 6.14.

The controller is able to successfully track the wheel trajectory and the height trajectory.

Finally, Fig. 6.12 and 6.15 show a simulation which uses the same initial conditions as Fig. 6.10 and superposed reference harmonic $x_w$ trajectories described by (6.35).
In this case, the tracking of the wheel trajectory is good, while the tracking of the CoM height has some oscillations around the reference value.

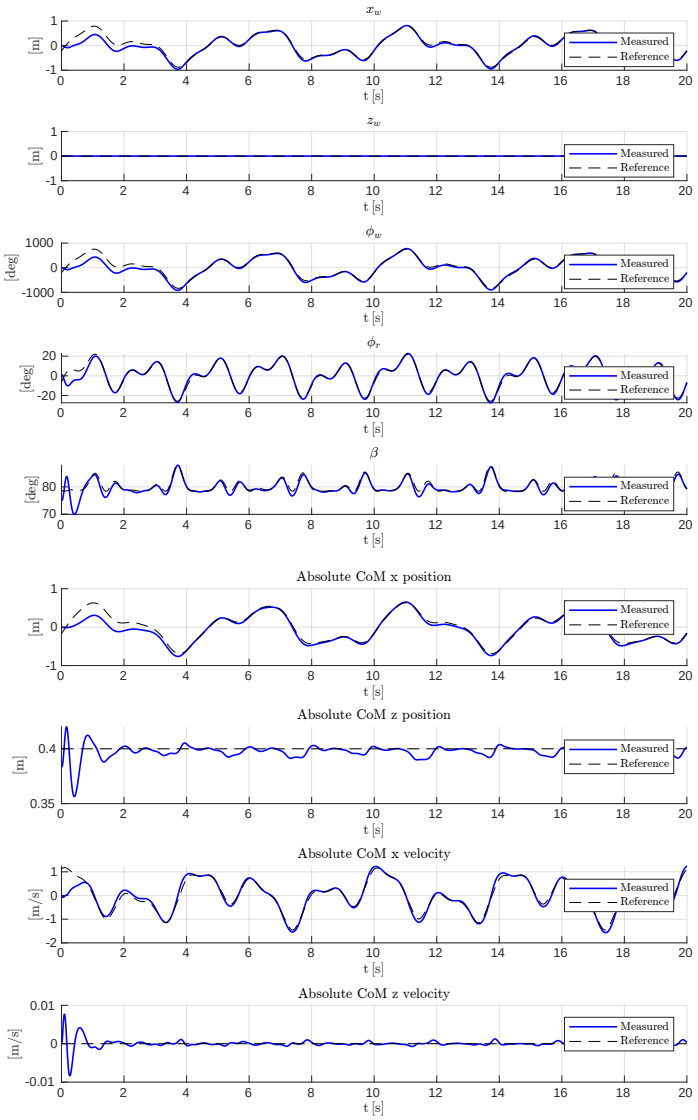



**Figure 6.12.** Results of the LQR tracker for controlling the CoM height, while tracking a reference trajectory of $x_w$. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $P_w = [0.1, 0.5, 0.3]$ m, $F_w = [1, 0.2, 0.5]$ Hz, $\Psi_w = [30, 0, -60]$ deg and the reference CoM height is 0.4 m. The controller runs at 50 Hz. The associated control inputs are shown in Fig. 6.15.

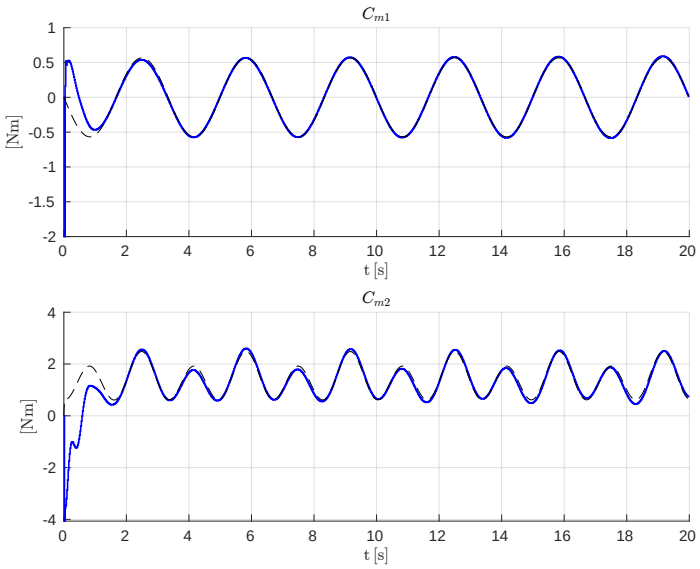



**Figure 6.13.** Results of the LQR tracker for controlling the CoM height, while tracking a reference trajectory of $x_w$. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and the reference CoM height is 0.4 m. The controller runs at 50 Hz. The plot shows the control inputs. The associated Lagrangian state and CoM trajectory are shown in Fig. 6.10.

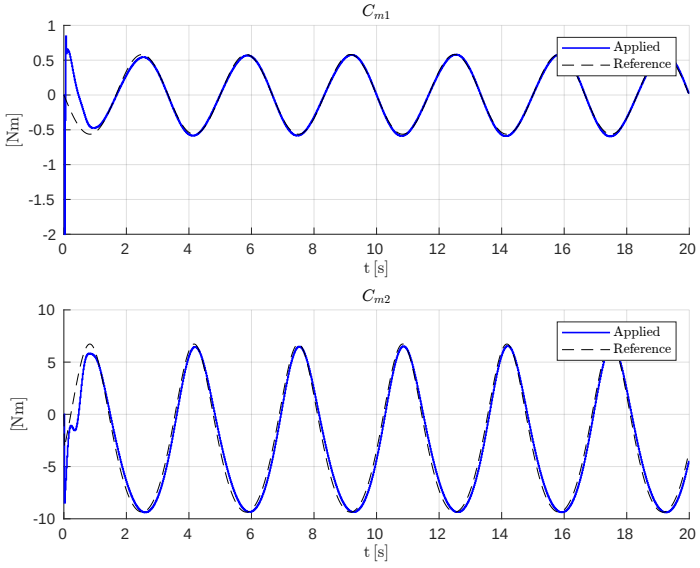



**Figure 6.14.** Results of the LQR tracker for controlling the CoM height, while tracking a reference trajectory of $x_w$. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and the reference CoM height trajectory is $h^{off} = 0.038\,\text{m}$, $\rho_h = 0.04\,\text{m}$, $f_h = 0.3\,\text{Hz}$. The controller runs at 50 Hz. The plot shows the control inputs. The associated Lagrangian state and CoM trajectory are shown in Fig. 6.11.

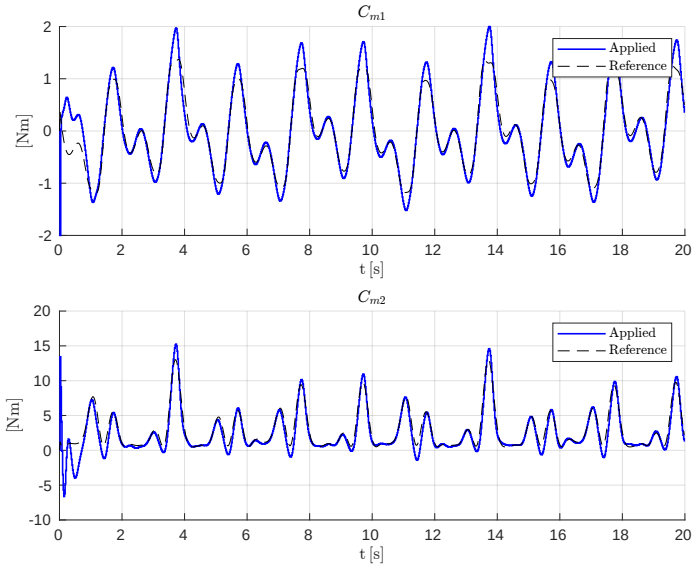



**Figure 6.15.** Results of the LQR tracker for controlling the CoM height, while tracking a reference trajectory of $x_w$ given by a superposition of multiple harmonics. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $P_w = [0.1, 0.5, 0.3]$ m, $F_w = [1, 0.2, 0.5]$ Hz, $\Psi_w = [30, 0, -60]$ deg and the reference CoM height is 0.4 m. The controller runs at 50 Hz. The plot shows the control inputs. The associated Lagrangian state and CoM trajectory are shown in Fig. 6.12.

## 6.5 Instrinsically Stable MPC for trajectory tracking

This section is dedicated to discussing and testing of a particular type of MPC control approach called *Instrinsically Stable* MPC (from here on abbreviated as IS-MPC). This approach was first introduced in 2016 by [36] to control humanoid robots and applied also to wheeled systems by [17].

The idea behind it is to employ a classical MPC scheme and embed the boundness condition (6.17) provided by Linear Stable Inversion as an additional constraint to the optimization problem. This kind of approach is particularly suited for robotics and, in general, industrial applications due to its instrinsic discreteness and its possibility to include constraints on the optimization variables.

### 6.5.1 Brief introduction to MPC

Let us look rapidly at the principle behind MPC control.
MPC stands for "Model Predictive Control" and is also sometimes called "Receding Horizon Control". The reason for such names will be clear in a moment.
The approach behind MPC is synthetically depicted in Fig. 6.16.

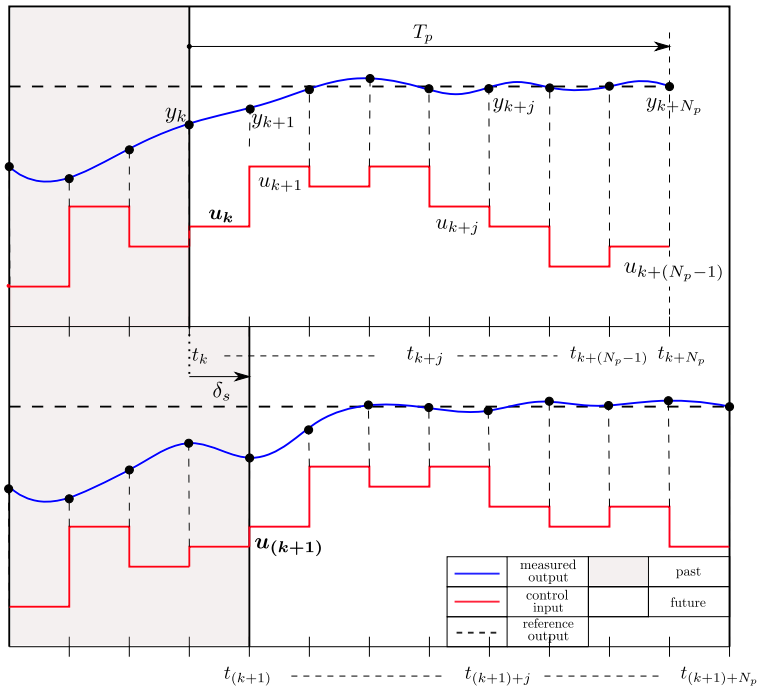



**Figure 6.16.** Receding horizon approach shown for a single input-single output system.

Suppose a known reference trajectory for the output, which is shown as a constant value in Fig. 6.16 for simplicity.

At each sample time $t_k$ the optimization problem defined by

$$U(t_k) = \underset{U(t_k)}{\operatorname{argmin}}\, J(U(t_k)) \tag{6.44}$$

subject, in general, to a number of equality and inequality constraints, is solved by

employing the measured output $y_k$ in conjunction with a mathematical model of the system's dynamics.
In particular,

$$U_k \coloneqq \Big[u_k,\, u_{k+1},\, ....., u_{k+(N_p-1)}\Big] \tag{6.45}$$

and

$$N_p = \frac{T_p}{\delta_s} \tag{6.46}$$

Only the first element of the obtained sequence of control inputs is then applied to the system. At the next sample time, the prediction horizon defined by $T_p$ is shifted by a sample time $\delta_s$, the current state is measured and the whole optimization process is repeated. This approach gives to MPC closed loop properties.

The cost function $J(U_k)$ and the constrains can be, in general, nonlinear functions. However, using nonlinear cost functions and constraints can greatly increase the computational overhead and decrease the reliability of the optimization. It is therefore common practice to employ, if possible, quadratic cost functions and linear constraints.

One of the main advantages of the MPC, differently from the more classical LQR, is that constraints can be embedded into the optimization problem, providing a great degree of flexibility and, furthermore, it is naturally suitable for implementation on discrete-time systems.

### 6.5.2 Building the (quadratic) cost function

Let us build the cost function with the aim of performing, as done in Section 6.4, trajectory tracking on system 3.4 choosing as outputs the position of the wheel $x_w$ (or, equivalently $\phi_w$) and the leg angle $\beta$. With this choice of outputs, consider the partially feedback-linearized form given by (5.78)-(5.79).

As already highlighted, subsystem (5.78) is completely linear and can be partitioned into two independent subsystems each of the form $\dot{\bar{q}}_i(t) = \bar{A}\,\bar{q}_i(t) + \bar{B}\,v_i(t)$, for $i = 1,\, 2$, where

$$\bar{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad \bar{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{6.47}$$

and

$$\bar{q}_1(t) = \begin{bmatrix} \phi_w(t) \\ \dot{\phi}_w(t) \end{bmatrix} \qquad \bar{q}_2(t) = \begin{bmatrix} \beta(t) \\ \dot{\beta}(t) \end{bmatrix} \tag{6.48}$$

Also recall that the feedback linearized inputs $v_1(t)$ and $v_2(t)$ are equal, respectively, to $\ddot{\phi}_w(t)$ and $\ddot{\beta}(t)$. Since the objective is to perform tracking of the output, the first step is to discretize these subsystems for obtaining a prediction model. Thanks to the partial feedback-linearization, the dynamics describing the output is linear. As a consequence, writing the closed form of the solution between $t_k$ and $t_{k+1}$ and considering that the control input $v_i$ is constant during a sample time, leads to

$$\bar{q}_i(t_{k+1}) = e^{\bar{A}\,\delta_s}\,\bar{q}_i(t_k) + \left(\int_{t_k}^{t_{k+1}} e^{\bar{A}\,(t_{k+1}-\tau)}\,B\,d\tau\right) v_i(t_k) \tag{6.49}$$

It is trivial to show that

$$\bar{A}_d = e^{\bar{A}\,\delta_s} = \begin{bmatrix} 1 & \delta_s \\ 0 & 1 \end{bmatrix} \qquad \bar{B}_d = \int_{t_k}^{t_{k+1}} e^{\bar{A}\,(t_{k+1}-\tau)}\, B\, d\tau = \begin{bmatrix} \delta_s^2/2 \\ \delta_s \end{bmatrix} \tag{6.50}$$

The discretization then becomes

$$\bar{q}_i(t_{k+1}) = \bar{A}_d\, \bar{q}_i(t_k) + \bar{B}_d\, v_i(t_k) \tag{6.51}$$

Using (6.51) recursively starting from a generic time $t_{k+N}$ one obtains

$$\bar{q}_i(t_{k+N}) = \bar{A}_d^N\, \bar{q}_i(t_k) + \sum_{j=0}^{N-1} \bar{A}_d^{N-1-j}\, \bar{B}_d\, v_i(t_{k+j}) \tag{6.52}$$

By defining the matrix

$$C = [1,\, 0] \tag{6.53}$$

The output of the system can be written as

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C\,\bar{q}_1 \\ C\,\bar{q}_2 \end{bmatrix} \tag{6.54}$$

Employing (6.52), it is possible to write

$$\bar{y}_i(t_{k+N}) = C\,\bar{A}_d^N\, \bar{q}_i(t_k) + C \sum_{j=0}^{N-1} \bar{A}_d^{N-1-j}\, \bar{B}_d\, v_i(t_{k+j}) \tag{6.55}$$

It is therefore possible to write the predicted output over the entire prediction horizon at the current time $t_k$ as

$$Y_i(t_k) = \begin{bmatrix} y_i^{k+1} \\ y_i^{k+2} \\ \vdots \\ y_i^{k+N_c} \end{bmatrix} = \begin{bmatrix} C\,\bar{A}_d\,\bar{q}_i(t_k) + C\,\bar{B}_d\, v_i(t_k) \\ C\,\bar{A}_d^2\,\bar{q}_i(t_k) + C\,\bar{A}_d\,\bar{B}_d\, v_i(t_k) + C\,\bar{B}_d\, v_i(t_{k+1}) \\ \vdots \\ C\,\bar{A}_d^{N_p}\,\bar{q}_i(t_k) + C\,\sum_{j=0}^{N_p-1} \bar{A}_d^{N_p-1-j}\,\bar{B}_d\, v_i(t_{k+j}) \end{bmatrix} = \tag{6.56}$$

$$= \Omega\,\bar{q}_i(t_k) + \Gamma\, V_i(t_k)$$

where

$$\Omega = \begin{bmatrix} C\,\bar{A}_d \\ C\,\bar{A}_d^2 \\ \vdots \\ \vdots \\ C\,\bar{A}_d^{N_p} \end{bmatrix} \qquad \Gamma = \begin{bmatrix} C\,\bar{B}_d & 0 & 0 & \dots & 0 \\ C\,\bar{A}_d\,\bar{B}_d & C\,\bar{B}_d & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ C\,\bar{A}_d^{N_p-1}\,\bar{B}_d & \dots & \dots & C\,\bar{A}_d\,\bar{B}_d & C\,\bar{B}_d \end{bmatrix} \tag{6.57}$$

and

$$V_i(t_k) = \begin{bmatrix} v_i(t_k) \\ v_i(t_{k+1}) \\ \vdots \\ v_i(t_{k+(N_p-1)}) \end{bmatrix} \tag{6.58}$$

Note that matrices $\Omega_i$ and $\Gamma_i$ are actually constant matrices and are identical for both $Y_1(t_k)$ and $Y_2(t_k)$.
The total optimization variable is simply obtained by stacking up $V_1$ and $V_2$:

$$V(t_k) = \begin{bmatrix} V_1(t_k) \\ V_2(t_k) \end{bmatrix} \tag{6.59}$$

The expression of (6.56) can be adapted to directly include (6.59) in the output predictions.
Specifically

$$Y_1(t_k) = \Omega\, \bar{q}_1(t_k) + \Gamma_1\, V(t_k) \tag{6.60}$$

and

$$Y_2(t_k) = \Omega\, \bar{q}_2(t_k) + \Gamma_2\, V(t_k) \tag{6.61}$$

where

$$\Gamma_1 = [\Gamma,\, 0_{N_p \times N_p}] \qquad \Gamma_2 = [0_{N_p \times N_p},\, \Gamma] \tag{6.62}$$

The contribution of each output to the cost function can be computed as

$$\begin{aligned} J_{i,y} &= \frac{1}{2} \left\| Y_i(t_k) - Y_i^{ref}(t_k) \right\|^2 = \frac{1}{2} \left\| \Omega\, \bar{q}_i(t_k) + \Gamma_i\, V(t_k) - Y_i^{ref}(t_k) \right\|^2 = \\ &= \frac{1}{2} \left[ \Omega\, \bar{q}_i(t_k) + \Gamma_i\, V(t_k) - Y_i^{ref}(t_k) \right]^T \left[ \Omega\, \bar{q}_i(t_k) + \Gamma_i\, V(t_k) - Y_i^{ref}(t_k) \right] = \\ &= \frac{1}{2}\, V(t_k)^T\, \Gamma_i^T \Gamma_i\, V(t_k) + \left[ \Omega\, \bar{q}_i(t_k) - Y_i^{ref}(t_k) \right]^T \Gamma_i\, V(t_k) + \\ &+ \frac{1}{2} \left[ \Omega\, \bar{q}_i(t_k) - Y_i^{ref}(t_k) \right]^T \left[ \Omega\, \bar{q}_i(t_k) - Y_i^{ref}(t_k) \right] \end{aligned} \tag{6.63}$$

where

$$Y_i^{ref}(t_k) = \begin{bmatrix} y_i^{ref}(t_{k+1}) \\ \vdots \\ y_i^{ref}(t_{k+N_p}) \end{bmatrix} \tag{6.64}$$

Note that the term

$$\frac{1}{2} \left[ \Omega\, \bar{q}_i(t_k) - Y_i^{ref}(t_k) \right]^T \left[ \Omega\, \bar{q}_i(t_k) - Y_i^{ref}(t_k) \right] \tag{6.65}$$

is actually constant w.r.t. the optimization variable. Hence, its contribution to the cost function can be neglected.

Next, it is always useful to include term for minimizing the control effort.
Let $D_u \coloneqq I_{N_p \times N_p}$. Then, these terms can be written as

$$J_{i,u} = \frac{1}{2}\, V(t_k)^T\, D_{i,u}^T\, D_{i,u}\, V(t_k) \tag{6.66}$$

where

$$D_{1,u} = [D_u,\, O_{N_p \times N_p}] \qquad D_{2,u} = [O_{N_p \times N_p},\, D_u] \tag{6.67}$$

Other important terms, which are useful to smoothen the computed sequence of inputs are the following

$$J_{i,u_{diff}} = \frac{1}{2}\, V(t_k)^T\, D_{i,u_{diff}}^T\, D_{i,u_{diff}}\, V(t_k) \tag{6.68}$$

with

$$D_{1,u_{diff}} = \left[D_{u_{diff}}, O_{(N_p-1)\times N_p}\right] \qquad D_{2,u_{diff}} = \left[O_{(N_p-1)\times N_p}, D_{u_{diff}}\right] \tag{6.69}$$

and

$$D_{u_{diff}} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix} \tag{6.70}$$

where $D_{u_{diff}} \in \mathbb{R}^{(N_p-1)\times N_p}$. This contribution aims at penalizing large differences between two consecutive predicted inputs.
This contribution does not penalize, however, the difference between $v_i(t_k)$ and $v_i(t_{k-1})$. Supposing the previously applied input $v_i(t_{k-1})$ is stored at each sample time, the following contribution can be built:

$$\begin{aligned} J'_{i,u_{diff}} &= \frac{1}{2}\, [v_i(t_k) - v_i(t_{k-1})]^2 = \frac{1}{2}\, V(t_k)^T\, (D'_{i,u_{diff}})^T\, D'_{i,u_{diff}}\, V(t_k) + \\ &+ f_{i,u_{diff}}^T\, V(t_k) \end{aligned} \tag{6.71}$$

with

$$D'_{1,u_{diff}} = \left[D'_{u_{diff}}, O_{N_p \times N_p}\right] \qquad D'_{2,u_{diff}} = \left[O_{N_p \times N_p}, D'_{u_{diff}}\right] \tag{6.72}$$

$$\begin{aligned} D'_{u_{diff}} &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} \qquad f_{1,u_{diff}}(t_{k-1}) = \begin{bmatrix} -v_1(t_{k-1}) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ f_{2,u_{diff}}(t_{k-1}) &= \begin{bmatrix} 0 \\ \vdots \\ -v_2(t_{k-1}) \\ \vdots \\ 0 \end{bmatrix} \end{aligned} \tag{6.73}$$

The final quadratic cost function takes the form

$$\begin{aligned} J(V(t_k)) &= w_{1,y}\, J_{1,y} + w_{1,u}\, J_{1,u} + w_{1,u_{diff}}\, J_{1,u_{diff}} + w'_{1,u_{diff}}\, J'_{1,u_{diff}} + \\ &+ w_{2,y}\, J_{2,y} + w_{2,u}\, J_{2,u} + w_{2,u_{diff}}\, J_{2,u_{diff}} + w'_{2,u_{diff}}\, J'_{2,u_{diff}} = \\ &= \frac{1}{2}\, V(t_k)^T\, H\, V(t_k) + f^T\, V(t_k) \end{aligned} \tag{6.74}$$

where

$$\begin{aligned} H &= w_{1,y}\, \Gamma_1^T \Gamma_1 + w_{1,u}\, D_{1,u}^T\, D_{1,u} + w_{1,u_{diff}}\, D_{1,u_{diff}}^T\, D_{1,u_{diff}} + \\ &+ w'_{1,u_{diff}}\, (D'_{1,u_{diff}})^T\, D'_{1,u_{diff}} + w_{2,y}\, \Gamma_2^T \Gamma_2 + w_{2,u}\, D_{2,u}^T\, D_{2,u} + \\ &+ w_{2,u_{diff}}\, D_{2,u_{diff}}^T\, D_{2,u_{diff}} + + w'_{2,u_{diff}}\, (D'_{2,u_{diff}})^T\, D'_{2,u_{diff}} \end{aligned} \tag{6.75}$$

$$\begin{aligned} f^T &= w_{1,y}\, \left[\Omega\, \bar{q}_1(t_k) - Y_1^{ref}(t_k)\right]^T\, \Gamma_1 + w_{2,y}\, \left[\Omega\, \bar{q}_2(t_k) - Y_2^{ref}(t_k)\right]^T\, \Gamma_2 + \\ &+ w'_{1,u_{diff}}\, f_{1,u_{diff}}^T + w'_{2,u_{diff}}\, f_{2,u_{diff}^T} \end{aligned} \tag{6.76}$$

Note that, by changing the output matrix $C$, it is also possible to include cost terms for penalizing the deviation of the output velocity w.r.t. to the reference. The structure of the prediction model for the velocity stays exactly the same.

### 6.5.3 Input constraints

As already stressed, one of the main advantages of the MPC is that allows to impose constraints over the optimization variable. This characteristic is particularly important because it permits to consider actuators limits inside the optimization process. It is evident why this approach is inherently superior to the a posteriori saturation used in Section 6.4. Also note that, thanks to the used prediction model, is also possible to impose constraints on the outputs (here not used, see [17] for examples).

In this specific case, imposing input limits requires some attention, since the optimization variable is made by the feedback-linearized input and not the actual inputs. The relationship with the actual inputs is given by the inverse of (5.77), which is here reported for clarity

$$v = \bar{M}(q_2, q_3)^{-1}\, \left[\bar{u} - \bar{C}(q_2, q_3, \dot{q}_2, \dot{q}_3)\right] \tag{6.77}$$

Remember that $q_2 = \phi_r$, $q_3 = \beta$ and

$$\bar{u} = 2 \begin{bmatrix} C_{m1} \\ C_{m2} \end{bmatrix} \tag{6.78}$$

Let us define

$$\bar{u}_{lim} = 2 \begin{bmatrix} C_{m1}^{lim} \\ C_{m2}^{lim} \end{bmatrix} \tag{6.79}$$

Then, at the generic instant $t_k$, the input constraint takes the form

$$\begin{aligned} &- \bar{M}(q_2(t_k), q_3(t_k))^{-1} \left[\bar{u}_{lim} + \bar{C}(q_2(t_k), q_3(t_k), \dot{q}_2(t_k), \dot{q}_3(t_k))\right] \leq \\ &\leq v(t_k) \leq \bar{M}(q_2(t_k), q_3(t_k))^{-1} \left[\bar{u}_{lim} - \bar{C}(q_2(t_k), q_3(t_k), \dot{q}_2(t_k), \dot{q}_3(t_k))\right] \end{aligned} \tag{6.80}$$

which can be synthetically written as

$$v_{min}(t_k) \leq v(t_k) \leq v_{max}(t_k) \tag{6.81}$$

Now, there are multiple ways of extending this constraint to the prediction horizon.

The first, and also the simplest, is to simply extend the constraint "as is" to the whole prediction horizon:

$$V_{min}(t_k) \leq V(t_k) \leq V_{max}(t_k) \tag{6.82}$$

where

$$\begin{aligned} V_{min}(t_k) &= [\underbrace{v_{1,min}(t_k), \ldots, v_{1,min}(t_k)}_{1 \times N_p}, \underbrace{v_{2,min}(t_k), \ldots, v_{2,min}(t_k)}_{1 \times N_p}]^T \\ V_{max}(t_k) &= [\underbrace{v_{1,max}(t_k), \ldots, v_{1,max}(t_k)}_{1 \times N_p}, \underbrace{v_{2,max}(t_k), \ldots, v_{2,max}(t_k)}_{1 \times N_p}]^T \end{aligned} \tag{6.83}$$

Constraint (6.82) can then easily be converted to the form

$$A_u^{lim} V(t_k) \leq b_u^{lim} \tag{6.84}$$

which is generally the one used by optimization solvers.
It is clear that with this approach input limits will never be violated at the current sample time. However, extending the constraint over the entire prediction horizon inevitably introduces approximations, since the controller computes future input (with the exception of the first one), without accounting for variations in the constraints due to future values of the state.
This approach could also make the QP unfeasible (this was indeed verified in simulation).

As a consequence, the simplest alternative is to impose (6.80) only at the current time and leave the inputs free over the prediction horizon. This approach ensured that limits are not violated at each $t_k$ and avoids the unfeasibility of the QP. For simplicity, this is the constraint which was implemented in simulations.

Another possible approach is to use the computed sequence of inputs $V(t_{k-1})$ at the previous sample time to integrate both the feedback linearized dynamics (5.78) and the internal dynamics (5.79) forward in time over the prediction horizon (see [17] for more info). This approach should ensure better performance compared with (6.82), at the cost of higher computational overhead. Furthermore, if the prediction at the previous sample time is considerably different from the one at the current time, the accuracy of this approach can potentially be greatly affected.

Lastly, it is also possible to linearize expression (5.77) around an operating condition (for example a suitable vertical equilibrium configuration with given leg angle) and discretize both the linearization of the internal dynamics, and the linearized dynamics of $\bar{q}_2$. An example of this procedure applied to a WIP is contained in [17]. The advantage of this technique is that it basically updates the constraint within the optimization process. Its main weakness is that uses a linearization of the dynamics which may introduce substantial errors in the prediction process.

### 6.5.4 Boundness constraint

The core of the IS-MPC is the integration of the boundness condition (6.17) as an equality constraint into the optimization process.
Condition (6.17) is here explicitly reported for clarity:

$$\zeta_u(0) = -\int_0^\infty e^{-\Lambda_u \tau} B_u \tilde{v}(\tau)\, d\tau \tag{6.85}$$

This equation can be read in two ways:

- It can be interpreted as the particular initial condition $\zeta_u(0)$ which, supposing assigned trajectories for the linearized input $v(t)$, avoids the divergence of the forced internal dynamics of the system. $\zeta_u(0)$ will define through (6.18) a bounded reference trajectory for the internal dynamics which can be then be tracked using a suitable stabilizing controller.
- Alternatively, one can consider the left side of (6.17) assigned by the current measured value of the internal dynamics state. Given the current measured state, it is clear that there are infinite choices of $v(t)$ satisfying the stability/boundness constraint (6.17). As a consequence, this choice can be made by employing a suitable optimization process. In particular, this constraint can be integrated as an equality constraint in the previously described quadratic optimization framework. This way the optimization will automatically choose a trade-off between the tracking of the reference output trajectory and the stability of the system. The main advantage compared to classical Linear Stable Inversion is that this way complex output trajectories can be tracked without the need of explicitly computing a stable reference trajectory for the internal dynamics.

Condition (6.85) needs to be adapted to the MPC formulation.
In particular

$$\zeta_u(t_k) = \underbrace{-\int_{t_k}^{t_{k+N_p}} e^{-\Lambda_u (\tau - t_k)} B_u\, v(\tau)\, d\tau}_{\text{within the prediction horizon}} \underbrace{-\int_{t_{k+N_p}}^{\infty} e^{-\Lambda_u (\tau - t_k)} B_u\, v(\tau)\, d\tau}_{\text{“tail”}} \tag{6.86}$$

The first part of the integral is performed withint the prediction horizon, while the remaining part is usually referred to as the “tail”. To be able to use the boundness condition, the structure of the tail needs to be conjectured somehow. The ways this problem can be addressed are discussed in [36]. In particular, the structure of

the second integral can be conjectured as a *truncated*, *periodic* or *anticipative* tail (see [36]).
Also note that for simplicity, $\tilde{v}$ was dropped in favour of $v$ since the linearization of the internal dynamics around the vertical equilibrium is associated with $v^* = 0$.
In this specific case, the tail is simply neglected (truncated tail). This basically corresponds to conjecturing null output accelerations after the prediction horizon.
The next step is to discretize the constraint over the prediction horizon

$$\begin{aligned}\zeta_u(t_k) &= -\sum_{j=0}^{N_p-1}\left(\int_{t_{k+j}}^{t_{k+(j+1)}} e^{-\Lambda_u\,(\tau-t_k)}\,d\tau\right) B_u\,v(t_{k+j}) = \\ &= \Lambda_u^{-1}\sum_{j=0}^{N_p-1} e^{-\Lambda_u\,j\,\delta_s}\left(e^{-\Lambda_u\,\delta_s}-I\right) B_u\,v(t_{k+j})\end{aligned} \tag{6.87}$$

The resulting expression (6.87) is valid also for multidimensional internal dynamics. In this specific case, since the internal dynamics is monodimensional, (6.87) can be rewritten in the following way

$$\begin{aligned}\zeta_u(t_k) &= \frac{\left(e^{-\lambda_u\,\delta_s}-1\right)}{\lambda_u}\sum_{j=0}^{N_p-1} e^{-\lambda_u\,j\,\delta_s}\,[b_{1,u},\,b_{2,u}]\begin{bmatrix} v_1(t_{k+j}) \\ v_2(t_{k+j})\end{bmatrix} \\ &= \Delta\,B_{uV}\,V(t_k)\end{aligned} \tag{6.88}$$

where

$$\Delta = \frac{\left(e^{-\lambda_u\,\delta_s}-1\right)}{\lambda_u}\left[1,\,e^{-\lambda_u\,\delta_s},\,e^{-\lambda_u\,2\,\delta_s},\ldots,e^{-\lambda_u\,(N_p-1)\,\delta_s}\right] \tag{6.89}$$

and

$$B_{uV} = \begin{bmatrix} b_{1,u} & 0 & 0 & \ldots & 0 & b_{2,u} & 0 & 0 & \ldots & 0 \\ 0 & b_{1,u} & 0 & \ldots & 0 & 0 & b_{2,u} & 0 & \ldots & 0 \\ 0 & 0 & \ddots & \ddots & \vdots & 0 & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & b_{1,u} & 0 & 0 & \ldots & 0 & b_{2,u}\end{bmatrix} \tag{6.90}$$

The final form of the constraint becomes

$$A_{bound}\,V(t_k) = b_{bound} \tag{6.91}$$

where

$$A_{bound} = \Delta\,B_{uV} \qquad b_{bound} = \zeta_u(t_k) \tag{6.92}$$

The use of a boundness constraint has many advantages over more standard control approaches like the continuous LQR or even classical MPCs.

For example, consider the case of a continuous LQR. Note that this kind of controller can be thought as the ancestor of the MPC, since it uses a linearization of the system to solve a continuous optimization problem forward into the future.

We now that, due to its formulation, the classical LQR controller cannot handle constraints. One way of extending its capability would be to use a discrete version of it. The problem could be formulated as a discrete QP and, thus, constraints could be integrated in it. This basically would transform it into an MPC (this is the reason why these control schemes are actually very much related).
In any case, reference state and input trajectories are needed. In the case of fully actuated systems, these can be assigned and computed easily without problems.

However, when dealing with underactuated systems, we already know that state trajectories cannot be assigned freely, due to the second order constraint which naturally arises. In the case of minimum phase systems, output trajectories can be tracked without worrying about the divergence of the internal dynamics, since it is stable. On the contrary, in the case of non-minimum phase systems, it is necessary to avoid the diverge of the internal dynamics by computing a stable reference trajectory for the internal dynamics, for example through Linear Stable Inversion.

When the reference trajectory is far from the current state or in presence of large disturbances, this approach can fail, since the only thing keeping the system stable is the proximity to the stable internal dynamics reference trajectory (which is not guaranteed).
The inclusion of a boundness constraint into an MPC framework provides, on the contrary, much greater robustness, as it will be seen in the upcoming simulations.

### 6.5.5 Testing (on both Matlab and Gazebo)

Note that, to compute the boundness constraint, it is first necessary to obtain a linearization of the internal dynamics around a suitable configuration. Due to how the constraint (6.88) is build, the used linearization will be thought to be fixed for the whole prediction horizon (more complex approaches are however possible). With this premise, the most suitable configuration is the vertical equilibrium which, as we already know, is characterized by a pitch angle $\phi_r^*(\beta^*)$. Being the equilibrium pitch angle a function of the leg angle, it is necessary to choose a reference for $\beta$ to be able to compute the linearization. This angle can be either be fixed by using the current measured state or a generic reference value in the $\beta$ range.

Clearly, the use of a constant linearization within the prediction horizon will inevitably introduce an approximation in the constraint computation, which becomes less valid the more the pitch angle moves away from the vertical equilibrium and the more the predicted $\beta$ angle moves away from its constant reference (recall that $\beta$ evolution can be exactly predicted using the computed feedback linearized inputs). One way of partially relaxing this approximation would be to account for state variations by updating the linearization at each sample time in the prediction horizon. This could be done, for example, by forward integrating the state starting from the current value and using the computed sequence of inputs at $t_{k-1}$.

Let us suppose the acceleration along the leg angle (i.e. $\ddot{\beta}$ or $v_2$) has a negligible effect on the unstable component of the internal dynamics (or, equivalently, the equilibrium of the system in not that much affected by accelerations of the leg angle). If this is valid or $v_2$ is relatively small, then its contribution to (6.88) can be neglected.

This approximation will be used during the testing phase, since it allows for a considerable simplification of constraint (6.88). In fact, using some physical intuition, it is clear that after applying this approximation the boundness constraint becomes substantially equivalent to the one of a simple WIP. As a consequence, instead of computing the explicit linearization of the complete internal dynamics, the already available linearization (5.93) is employed. This allows to use simple analytical expressions for $\lambda_u$ and $b_{1,u}$. These quantities, which are ultimately function of $\beta$ (see Section 5.7), will be updated at each sample time, so that the linearization of the internal dynamics will at least be partially corrected at the beginning of each prediction horizon. Furthermore note that, to be able to compute the $\zeta_u(t_k)$ associated with the above described approximate boundness constraint, it is necessary to use the equivalent variables $\theta_r$ and $\dot{\theta}_r$.

Recall that these variables can be easily computed as

$$\begin{aligned} \theta_r &= \phi_r - \phi_r^*(\beta) \\ \dot{\theta}_r &= \dot{\phi}_r - \frac{d\phi_r^*(\beta)}{d\beta}\,\dot{\beta} \end{aligned} \tag{6.93}$$

With these premises, the approximate boundness constraint takes the form

$$\zeta_u(t_k) = \frac{b_u(t_k)\,\left(e^{-\lambda_u(t_k)\,\delta_s} - 1\right)}{\lambda_u(t_k)} \sum_{j=0}^{N_p-1} e^{-\lambda_u(t_k)\,j\,\delta_s}\, v_1(t_{k+j}) = b_u(t_k)\,\Delta(t_k)\,V(t_k) \tag{6.94}$$

Where

$$\lambda_u(t_k) = \sqrt{\frac{g\,l_{cm}^p(\beta(t_k))\,m_{eq}^p}{m_{eq}^p\,{l_{cm}^p}^2(\beta(t_k)) + m_{eq}^p\,r_w\,l_{cm}^p(\beta(t_k)) + I_{eq}^p(\beta(t_k))}} \tag{6.95}$$

and $\Delta$ is defined by (6.89).
Furthermore, recalling (6.13),

$$b_u(t_k) = \frac{2\,J_w + m_{eq}^p\,{r_w}^2 + 2\,m_w\,{r_w}^2 + l_{cm}^p(\beta(t_k))\,m_{eq}^p\,r_w}{2\,\lambda_u(t_k)\,\left[m_{eq}^p\,{l_{cm}^p}^2(\beta(t_k)) + m_{eq}^p\,r_w\,l_{cm}^p(\beta(t_k)) + I_{eq}^p(\beta(t_k))\right]} \tag{6.96}$$

and

$$\zeta_u(t_k) = \frac{1}{2}\,\left[\theta_r + \dot{\theta}_r/\lambda_u(t_k)\right] \tag{6.97}$$

which is obtained by remembering that

$$\begin{bmatrix} \zeta_s(t_k) \\ \zeta_u(t_k) \end{bmatrix} = T(t_k) \begin{bmatrix} \theta_r(t_k) \\ \dot{\theta}_r(t_k) \end{bmatrix} = \begin{bmatrix} 1/2 & -1/[2\,\lambda_u(t_k)] \\ 1/2 & 1/[2\,\lambda_u(t_k)] \end{bmatrix} \begin{bmatrix} \theta_r(t_k) \\ \dot{\theta}_r(t_k) \end{bmatrix} \tag{6.98}$$

All the presented simulations will be performed employing the above described approximate boundness constraint.
Furthermore, the upcoming simulations will all use the following initial condition:

$$q_0^{(0)} = \left[0, 0, 0, 0, 75 \cdot \frac{\pi}{180}\,\text{rad}, 0, 0, 0, 0, 0\right] \tag{6.99}$$

### Comparing performance for different prediction horizons

The following sinusoidal trajectories are fed into the controller:

$$\begin{aligned}\phi_w(t) &= \frac{x_w^{off}}{r_w} + \frac{\rho_w}{r_w}\,\sin(2\,\pi\,f_w\,t + \psi_w)\\ \beta(t) &= \beta^{off} + \rho_\beta\,\sin(2\,\pi\,f_\beta\,t + \psi_\beta)\end{aligned} \tag{6.100}$$

where

$$\begin{aligned}x_w^{off} &= 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.3\,\text{Hz};\ \psi_w = 0\text{deg}\\ \beta^{off} &= 80\,\text{deg};\ \rho_\beta = 10\,\text{deg};\ f_\beta = 1\,\text{Hz};\ \psi_\beta = 0\text{deg}\end{aligned} \tag{6.101}$$

Furthermore, the following cost function weights are used:

$$\begin{aligned}\left[w_{1,y},\, w_{1,u},\, w_{1,u_{diff}},\, w'_{1,u_{diff}}\right] &= [15,\, 0.01,\, 0.01,\, 0]\\ \left[w_{2,y},\, w_{2,u},\, w_{2,u_{diff}},\, w'_{2,u_{diff}}\right] &= [25,\, 0.0001,\, 0.001,\, 0]\end{aligned} \tag{6.102}$$

The tested prediction horizons and the used sampling interval are, respectively,

$$\begin{aligned}\left[T_p^{(1)},\, T_p^{(2)},\, T_p^{(3)}\right] &= [1.5,\, 0.7,\, 0.6]\ \text{s}\\ \delta_s &= 0.02\,\text{s}\end{aligned} \tag{6.103}$$

The results are shown in Fig. 6.17, where the plots of the ground reaction forces and the Lagrangian state velocities are omitted for simplicity.

The simulations reflect the expected behaviour: a greater prediction horizon allows the optimization to see further into the future and thus the controller's performance is greatly increased. It is possible to note that, while using $T_p^{(2)}$ produces a minor tracking performance decrease, the use $T_p^{(3)}$ diminishes the tracking performance considerably (this is to be expected). Nevertheless, the boundness constraint is able to avoid the divergence of the internal dynamics for all the tested prediction horizons. The strength and efficacy of this constraint is one of the main advantages of this approach, compared to more classical ones (like for instance LQR or even classical MPC). Even if the tracking performance degrades, the stability of the system is ensured over a wide range of motion.

### Comparing performance for different sampling frequencies

It is also particularly useful to simulate the effect of different sampling frequencies on the controller.
The following sinusoidal trajectories are fed into the controller:

$$\begin{aligned}\phi_w(t) &= \frac{x_w^{off}}{r_w} + \frac{\rho_w}{r_w}\,\sin(2\,\pi\,f_w\,t + \psi_w)\\ \beta(t) &= \beta^{off} + \rho_\beta\,\sin(2\,\pi\,f_\beta\,t + \psi_\beta)\end{aligned} \tag{6.104}$$

where

$$\begin{aligned} x_w^{off} &= 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.3\,\text{Hz};\ \psi_w = 0\text{deg} \\ \beta^{off} &= 80\,\text{deg};\ \rho_\beta = 10\,\text{deg};\ f_\beta = 1\,\text{Hz};\ \psi_\beta = 0\text{deg} \end{aligned} \tag{6.105}$$

The following weights were used:

$$\begin{aligned} \left[w_{1,y},\, w_{1,u},\, w_{1,u_{diff}},\, w'_{1,u_{diff}}\right] &= [15,\, 0.01,\, 0.01,\, 0] \\ \left[w_{2,y},\, w_{2,u},\, w_{2,u_{diff}},\, w'_{2,u_{diff}}\right] &= [25,\, 0.001,\, 0.0001,\, 0] \end{aligned} \tag{6.106}$$

The used prediction horizon and the tested sampling intervals are, respectively,

$$\begin{aligned} T_p &= 1.5\,\text{s} \\ \left[\delta_s^{(1)},\, \delta_s^{(2)},\, \delta_s^{(3)}\right] &= [0.02,\, 0.05,\, 0.1]\ \text{s} \end{aligned} \tag{6.107}$$

The results are shown in Fig. 6.18. The controller works perfectly for the first sampling frequency (50 Hz), while the performance decreasing for lower frequencies. This behaviour is to be expected and could also be partially compensated for by adapting the weights. In this case, however, weights were maintained constant during simulations to keep the comparison fair.

Even if the tracking performance decreases considerably, especially on $\beta$ d.o.f., the internal dynamics (i.e. the pitch angle) remains always bounded, even for very low sample frequencies like 10 Hz. This characteristic highlights one of the reasons why this approach is superior to more classical ones. Using this frequency with the LQR and Linear Stable Inversion approach of Section 6.4 would have resulted in the divergence of the internal dynamics.

Looking at the actuation plots, it is also possible to appreciate how the optimization adapts the inputs to the used sample frequency to minimize the offset from the desired output trajectory.

### Tracking of a "fast" sinusoidal trajectory

Another useful test is performed using an aggressive sinusoidal trajectory. On Matlab, the tested trajectory is described by:

$$\begin{aligned} x_w^{off} &= 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.4\,\text{Hz};\ \psi_w = 0\text{deg} \\ \beta^{off} &= 80\,\text{deg};\ \rho_\beta = 10\,\text{deg};\ f_\beta = 0.3\,\text{Hz};\ \psi_\beta = -90\text{deg} \end{aligned} \tag{6.108}$$

while, in Gazebo,

$$\begin{aligned} x_w^{off} &= 0\,\text{m};\ \rho_w = 0.5\,\text{m};\ f_w = 0.3\,\text{Hz};\ \psi_w = 0\text{deg} \\ \beta^{off} &= 80\,\text{deg};\ \rho_\beta = 5\,\text{deg};\ f_\beta = 0.3\,\text{Hz};\ \psi_\beta = -90\text{deg} \end{aligned} \tag{6.109}$$

The cost weights used in Matlab are

$$\begin{aligned} \left[w_{1,y},\, w_{1,u},\, w_{1,u_{diff}},\, w'_{1,u_{diff}}\right] &= [15,\, 0.01,\, 0.01,\, 0] \\ \left[w_{2,y},\, w_{2,u},\, w_{2,u_{diff}},\, w'_{2,u_{diff}}\right] &= [25,\, 0.001,\, 0.001,\, 0] \end{aligned} \tag{6.110}$$

while, in Gazebo,

$$\begin{aligned}\left[w_{1,y},\, w_{1,u},\, w_{1,u_{diff}},\, w'_{1,u_{diff}}\right] &= [15,\, 0.01,\, 0.01,\, 0]\\ \left[w_{2,y},\, w_{2,u},\, w_{2,u_{diff}},\, w'_{2,u_{diff}}\right] &= [1000,\, 0.001,\, 0.001,\, 0]\end{aligned} \tag{6.111}$$

The prediction horizon and the used sampling interval are set to, respectively,

$$\begin{aligned}T_p &= 1.5\,\mathrm{s}\\ \delta_s &= 0.02\,\mathrm{s}\end{aligned} \tag{6.112}$$

The results of the simulations in Matlab and Gazebo are shown, respectively, in Fig. 6.19 and in Fig. 6.20.

In particular, the tracking turns out to be good on both simulators (the performance in Gazebo is inferior due to unmodeled contact dynamics).

### More complex trajectory tracking

A particularly significant test is the tracking of a more general trajectory w.r.t. the ones used until now.
Specifically, the following trajectories are fed to the controller:

$$\begin{aligned}\phi_w(t) &= \frac{\arctan(t - T_s/2)) + (\arctan(T_s/2)}{r_w}\\ \beta(t) &= \beta^{off} + \rho_\beta\, \sin(2\,\pi\, f_\beta\, t + \psi_\beta)\end{aligned} \tag{6.113}$$

where, for Matlab,

$$\beta^{off} = 80\,\mathrm{deg};\ \rho_\beta = 10\,\mathrm{deg};\ f_\beta = 0.3\,\mathrm{Hz};\ \psi_\beta = -90\,\mathrm{deg} \tag{6.114}$$

while, for Gazebo,

$$\beta^{off} = 80\,\mathrm{deg};\ \rho_\beta = 5\,\mathrm{deg};\ f_\beta = 0.3\,\mathrm{Hz};\ \psi_\beta = -90\mathrm{deg} \tag{6.115}$$

and

$$T_s = 20\,\mathrm{s} \tag{6.116}$$

Furthermore, the following weights were used in Matlab:

$$\begin{aligned}\left[w_{1,y},\, w_{1,u},\, w_{1,u_{diff}},\, w'_{1,u_{diff}}\right] &= [15,\, 0.01,\, 0.01,\, 0]\\ \left[w_{2,y},\, w_{2,u},\, w_{2,u_{diff}},\, w'_{2,u_{diff}}\right] &= [25,\, 0.001,\, 0.001,\, 0]\end{aligned} \tag{6.117}$$

while, in Gazebo,

$$\begin{aligned}\left[w_{1,y},\, w_{1,u},\, w_{1,u_{diff}},\, w'_{1,u_{diff}}\right] &= [15,\, 0.01,\, 0.01,\, 0]\\ \left[w_{2,y},\, w_{2,u},\, w_{2,u_{diff}},\, w'_{2,u_{diff}}\right] &= [1000,\, 0.001,\, 0.001,\, 0]\end{aligned} \tag{6.118}$$

The prediction horizon and the used sampling interval are, respectively,

$$\begin{aligned}T_p &= 1.5\,\mathrm{s}\\ \delta_s &= 0.02\,\mathrm{s}\end{aligned} \tag{6.119}$$

The results of the simulations in Matlab and Gazebo are shown, respectively, in Fig. 6.21 and in Fig. 6.22. The tracking performance is nearly perfect in Matlab and acceptable in Gazebo.

This simulation shows another major advantage of this approach. In fact, trying to perform the same trajectory tracking using, for example, Linear Stable Inversion would become quite difficult due to the complexity of the given output trajectories. On the contrary, the IS-MPC does not present such problem since the boundness constraint is imposed internally in the optimization process. Furthermore, as a side product of the control algorithm, note that the resulting integrated pitch trajectory roughly corresponds to the result one would have obtained by performing Stable Inversion. In this sense, the IS-MPC approach can also be used as a way of performing a numerical Stable Inversion on the system and thus obtaining also a bounded reference trajectory for the internal dynamics.

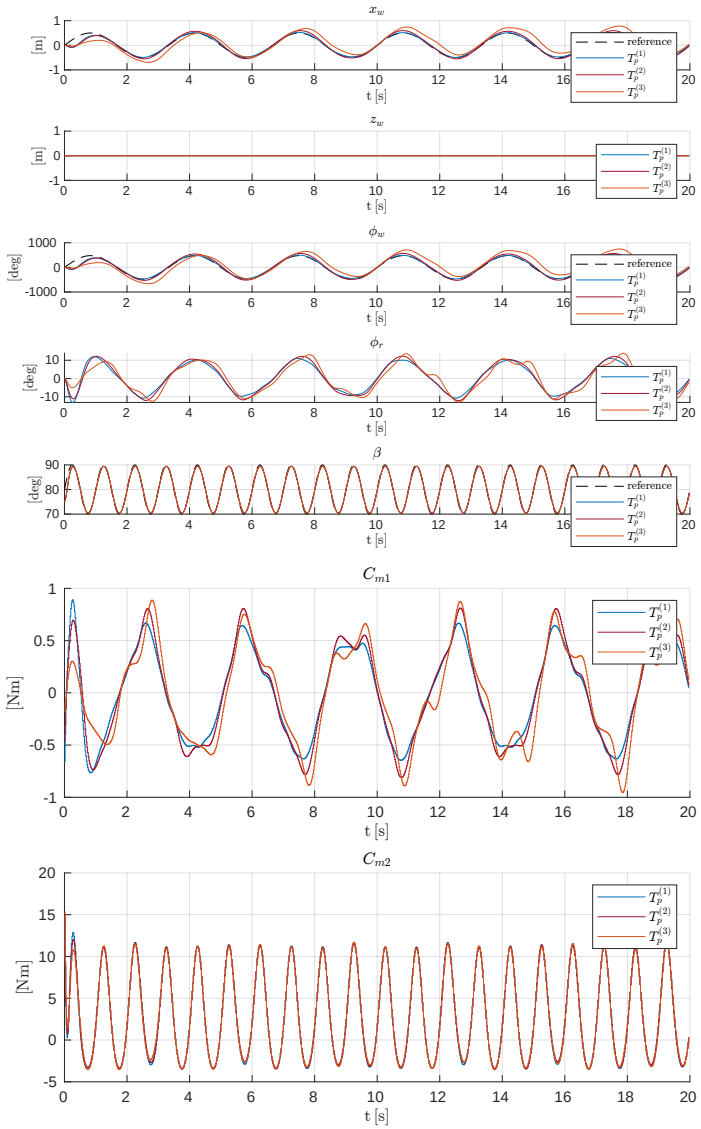



**Figure 6.17.** Results of the IS-MPC tracker, simulated in Matlab, for different prediction horizons. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 1\,\text{Hz}$; furthermore $T_p^{(1)} = 1.5\,\text{s}$, $T_p^{(2)} = 0.7\,\text{s}$ and $T_p^{(3)} = 0.6\,\text{s}$, while $\delta_s = 0.02\,\text{s}$ .

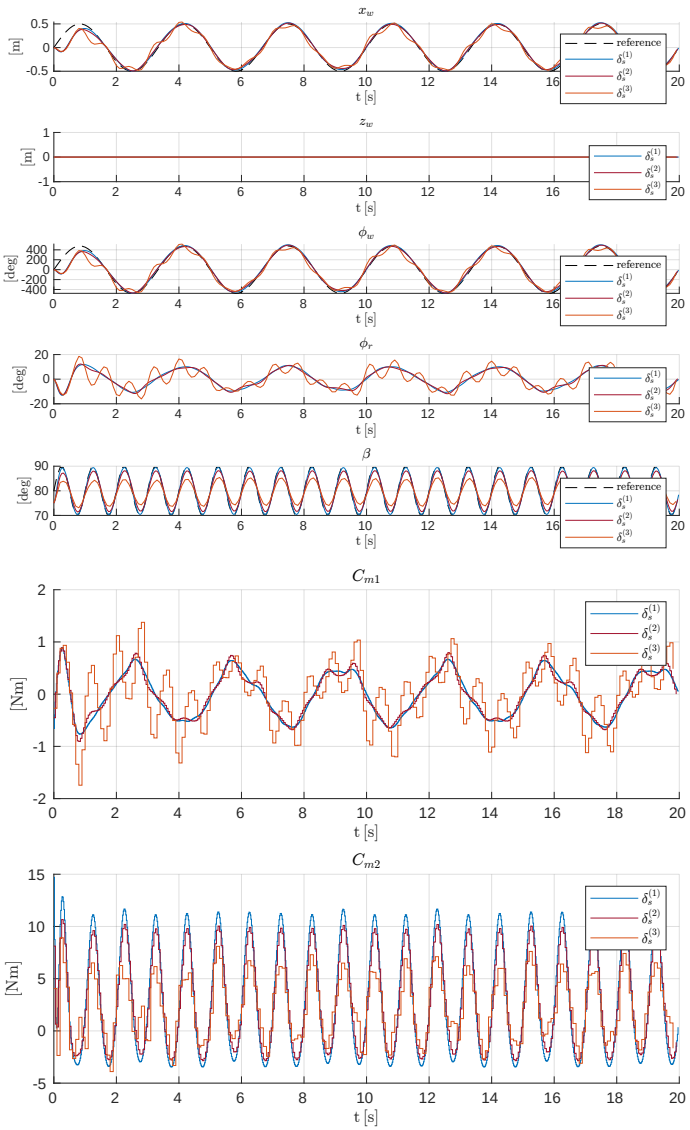



**Figure 6.18.** Results of the IS-MPC tracker, simulated in Matlab, for different sample frequencies. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 0.2\,\text{Hz}$; furthermore $\delta_s^{(1)} = 0.02\,\text{s}$, $\delta_s^{(2)} = 0.05\,\text{s}$ and $\delta_s^{(3)} = 0.1\,\text{s}$, while $T_p = 1.5\,\text{s}$.

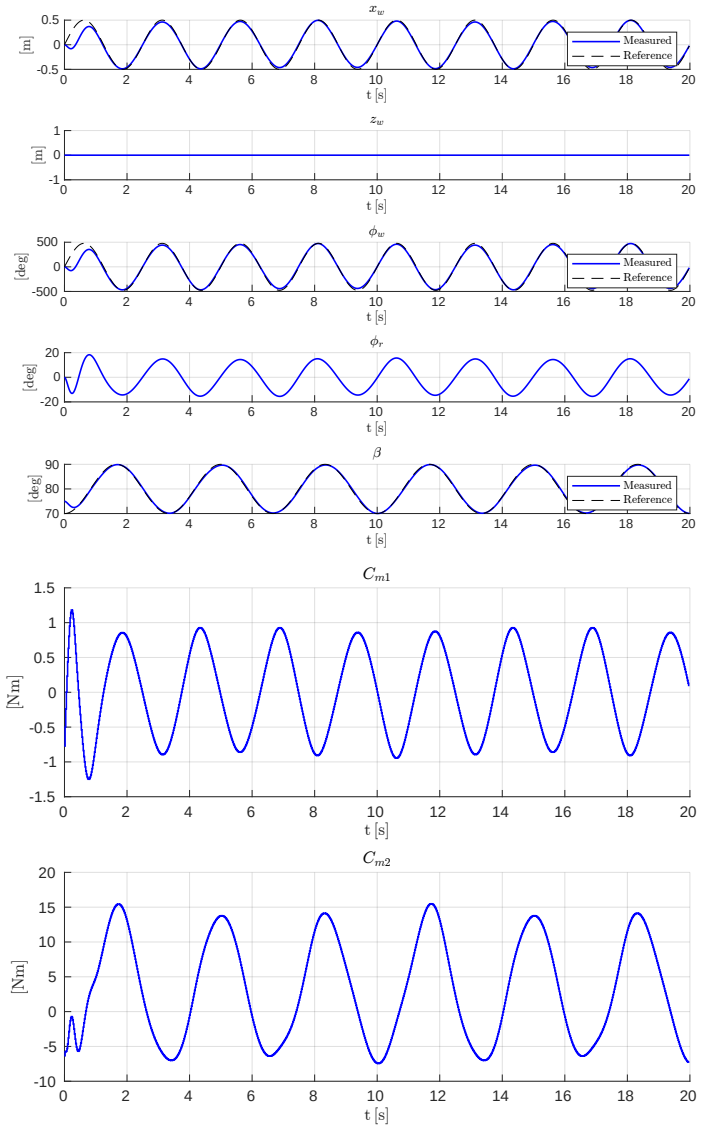



**Figure 6.19.** Results of the IS-MPC tracker, simulated in Matlab, for a quite aggressive $x_w$ trajectory. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.4\,\text{Hz}$ and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 0.2\,\text{Hz}$; furthermore $\delta_s = 0.02\,\text{s}$, and $T_p = 1.5\,\text{s}$. Note that $\beta$ trajectory is also subject to a phase lag of 90 deg.

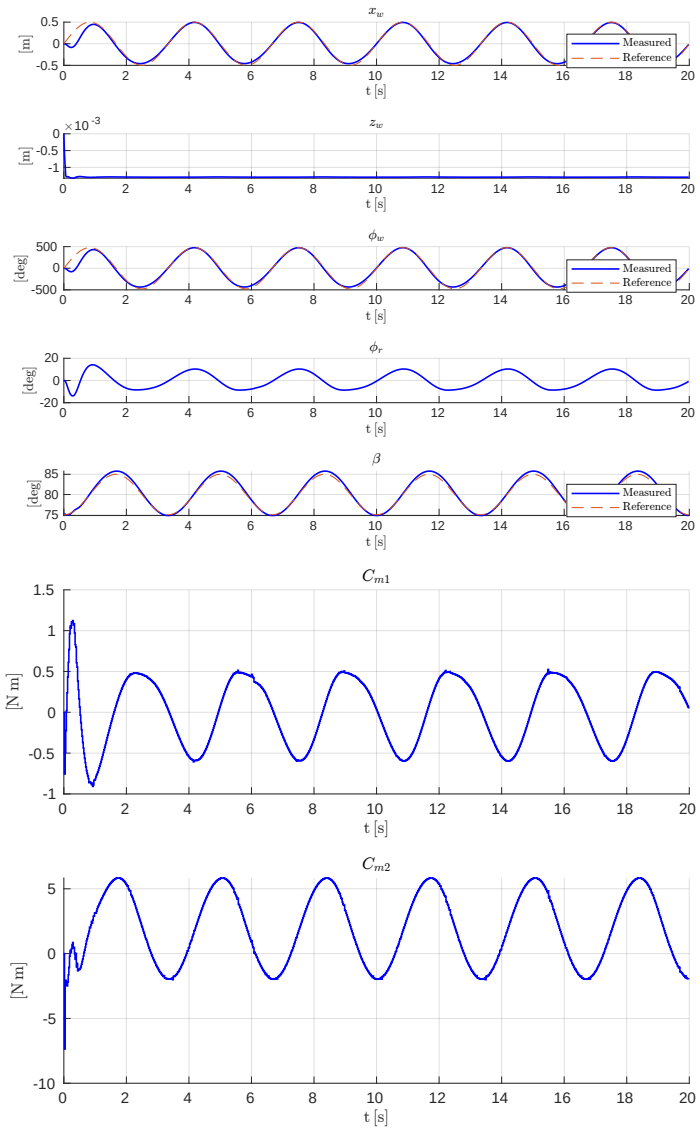



**Figure 6.20.** Results of the IS-MPC tracker, simulated in Gazebo, for a quite aggressive $x_w$ trajectory. In particular, the tracking is performed on a $x_w$ trajectory with $x_w^{off} = 0\,\text{m}$, $\rho_w = 0.5\,\text{m}$, $f_w = 0.3\,\text{Hz}$ and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 5\,\text{deg}$, $f_\beta = 0.3\,\text{Hz}$; furthermore $\delta_s = 0.02\,\text{s}$, and $T_p = 1.5\,\text{s}$. Note that $\beta$ trajectory is also subject to a phase lag of 90 deg.

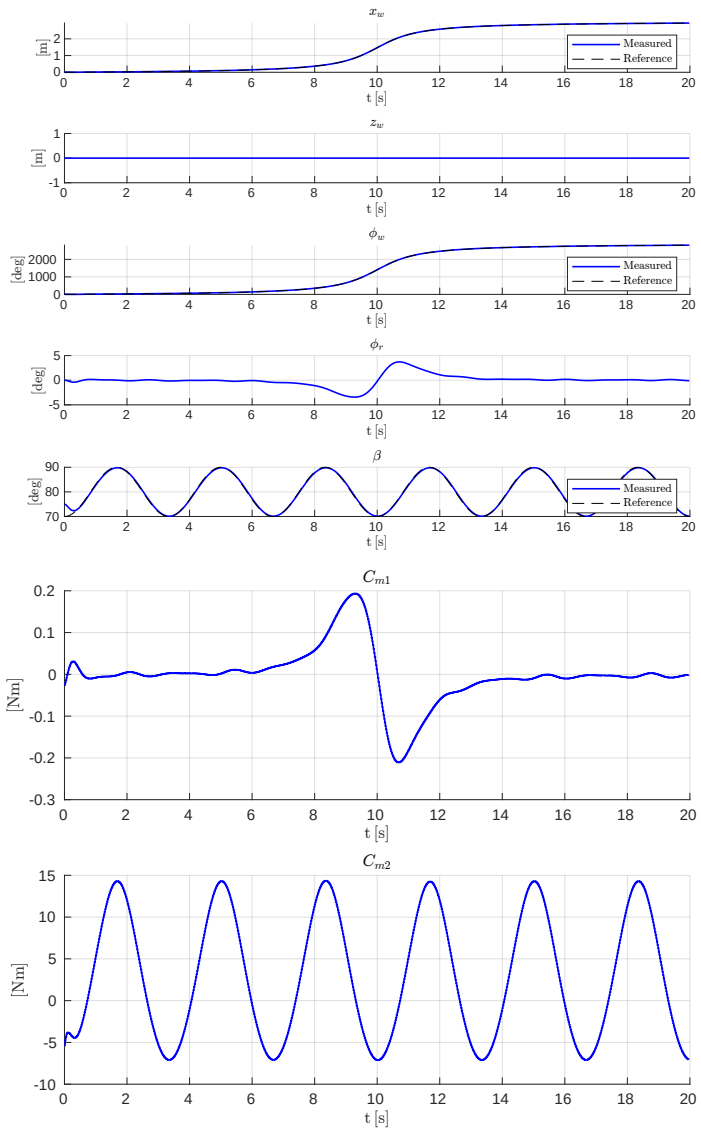



**Figure 6.21.** Results of the IS-MPC tracker, simulated in Matlab, with a more complex $x_w$ trajectory. In particular, the tracking is performed on a arctan-shaped $x_w$ trajectory (refer to (6.113)) and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 0.3\,\text{Hz}$; furthermore $\delta_s = 0.02\,\text{s}$, and $T_p = 1.5\,\text{s}$. The $\beta$ trajectory is also subject to a phase lag of 90 deg.

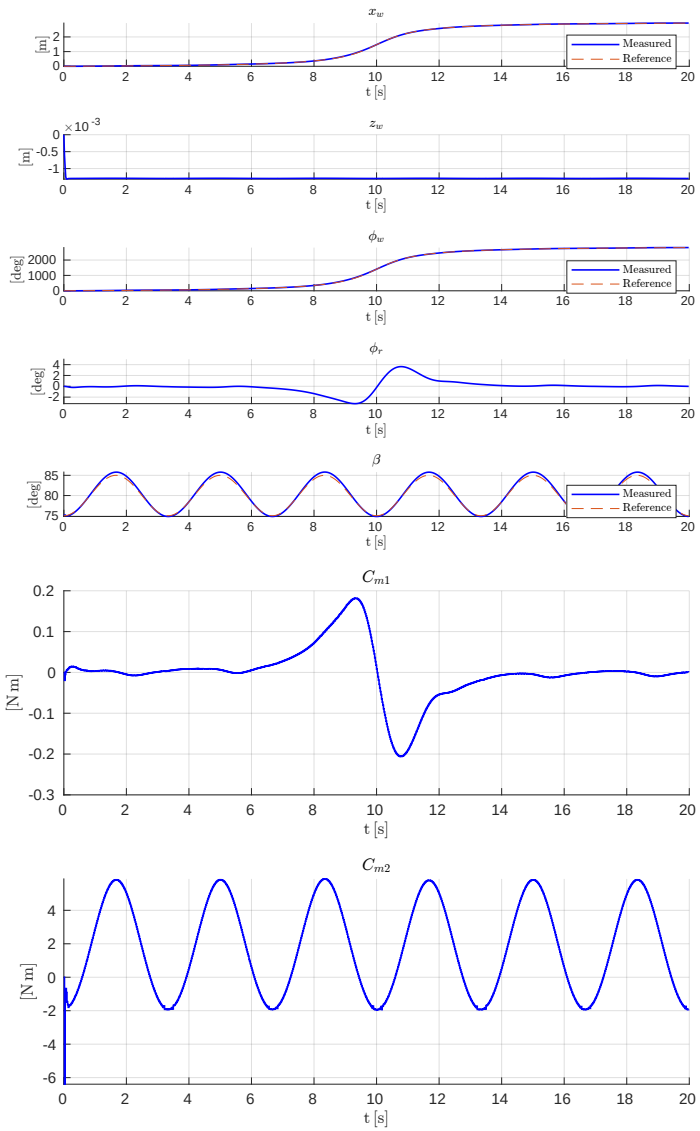



**Figure 6.22.** Results of the IS-MPC tracker, simulated in Gazebo, with a more complex $x_w$ trajectory. In particular, the tracking is performed on a arctan-shaped $x_w$ trajectory (refer to (6.113)) and $\beta^{off} = 80\,\text{deg}$, $\rho_\beta = 10\,\text{deg}$, $f_\beta = 0.2\,\text{Hz}$; furthermore $\delta_s = 0.02\,\text{s}$, and $T_p = 1.5\,\text{s}$. The $\beta$ trajectory is also subject to a phase lag of 90 deg.

## 6.6 Task-space tracker for CoM tracking

This section is dedicated to discussing a powerful approach to trajectory tracking commonly addressed as "Task-Space controller/traker".

Contrary to the previously explored control algorithms, in its classical formulation, does not possess any predictive characteristic.

Suppose a given task, for example tracking a desired CoM trajectory (which is actually the task this approach will be tested on). Examples of this approach applied to similar systems to the one under study are provided by [47] and [6].

For instance, if considering a robotic arm, the task might be to track a desired end-effector trajectory. The set of reachable end-effector positions and orientations will therefore define the associated Task-Space.

The idea behind this approach is below described using as a representative example the tracking of a given relative CoM trajectory (relative to the wheel center). To this purpose, the floating-base system described in Section 3.6 will be used.

### 6.6.1 Building the cost function

Consider the floating base dynamics (3.86). It is immediate to see that it can be partitioned as

$$M_\lambda(q_p)\,\ddot{q}_p + C_\lambda(q_p, \dot{q}_p) = \lambda_c \tag{6.120a}$$

$$M_u(q_p)\,\ddot{q}_p + C_u(q_p, \dot{q}_p) = G_u^{'}(q_p)\,u + \, J_{\lambda,u}\,\lambda_c \tag{6.120b}$$

where the first two-dimensional subsystem is relative to the floating base (hence does not contain any term related to the control inputs) and the second is the remaining dynamics.

Let us define the desired task trajectory using the notation

$$\chi^{ref}(t) = \begin{bmatrix} x_{CoM}^{ref}(t) \\ y_{CoM}^{ref}(t) \end{bmatrix} \tag{6.121}$$

where $x_{CoM}^{ref}(t)$ and $y_{CoM}^{ref}(t)$ are, respectively, the *relative* horizontal and vertical CoM trajectories. In general, the task can be defined arbitrarily to suit specific needs.

Computing the position of the relative CoM and differentiating it w.r.t. time, it is possible to obtain

$$\dot{\chi}(q_p, \dot{q}_p) = J_{CoM}(q_p)\,\dot{q}_p \tag{6.122}$$

and, as a consequence,

$$\ddot{\chi}(q_p, \dot{q}_p, \ddot{q}_p) = J_{CoM}(q_p)\,\ddot{q}_p + \dot{J}_{CoM}(q_p, \dot{q}_p)\,\dot{q}_p \tag{6.123}$$

where $J_{CoM}(q_p)$ is the velocity Jacobian which maps the Lagrangian velocity to the task velocity.

Furthermore, let us define the optimization variable as

$$\Theta = \begin{bmatrix} \ddot{q}_p \\ \lambda \end{bmatrix} \tag{6.124}$$

The reason why the ground reaction forces were included as an optimization variable will be become clear in Section 6.6.2. Note that, with this choice, $\Theta \in \mathbb{R}^7$.

At each sample time $t_k$, the cost to be minimized is

$$J(\Theta,\, t_k) = \frac{1}{2}\,\Theta^T\, H(t_k)\,\Theta + f^T(t_k)\,\Theta \tag{6.125}$$

where $J(\Theta,\, t_k)$ is obtained by computing

$$\begin{aligned} J^{'}(\Theta,\, t_k) &= \frac{1}{2}\,\left\|\ddot{\chi}_c(t_k) - \ddot{\chi}(t_k)\right\|^2 = \\ &= \frac{1}{2}\,\left\|\ddot{\chi}_c(t_k) - \left[J_{CoM}(t_k)\,\ddot{q}_p + \dot{J}_{CoM}(t_k)\,\dot{q}_p(t_k)\right]\right\|^2 = \\ &= \frac{1}{2}\,\ddot{q}_p^T\, J_{CoM}^T(t_k)\, J_{CoM}(t_k)\,\ddot{q}_p + \left[-\ddot{\chi}_c^T(t_k)\, J_{CoM}(t_k) + \dot{q}_p^T(t_k)\,\dot{J}_{CoM}^T(t_k)\, J_{CoM}(t_k)\right]\,\ddot{q}_p \\ &+ \frac{1}{2}\,\dot{q}_p^T(t_k)\,\dot{J}_{CoM}^T(t_k)\,\dot{J}_{CoM}(t_k)\,\dot{q}_p(t_k) + \frac{1}{2}\,\ddot{\chi}_c^T(t_k)\,\ddot{\chi}_c(t_k) - \ddot{\chi}_c^T(t_k)\,\dot{J}_{CoM}(t_k)\,\dot{q}_p(t_k) \end{aligned} \tag{6.126}$$

and neglecting the terms which do not depend upon the optimization variable. In particular,

$$H(t_k) = \dot{J}_{CoM}^T(t_k)\,\dot{J}_{CoM}(t_k) \tag{6.127}$$

$$f^T(t_k) = \left[-\ddot{\chi}_c^T(t_k) + \dot{q}_p^T(t_k)\,\dot{J}_{CoM}^T(t_k)\right]\, J_{CoM}(t_k) \tag{6.128}$$

and

$$\ddot{\chi}_c(t_k) = \ddot{\chi}_{ref}(t_k) + K_d\,\left[\dot{\chi}_{ref}(t_k) - \dot{\chi}(t_k)\right] + K_p\,\left[\chi_{ref}(t_k) - \chi(t_k)\right] \tag{6.129}$$

where $K_p$ and $K_d$ are suitable gain matrices. In this specific case they $\in \mathbb{R}^{2\times 2}$.

This concludes the formulation of the cost function.
Note that, once a solution $\Theta(t_k)$ is available, the necessary control inputs can be simply obtained by employing (6.120b) (actually, since there are only two independent control inputs and the second subsystem has dimension 3, two of the equations are actually sufficient).

### 6.6.2 Constraints

Note that the optimization program is function of the current measured state ($q_p$ and $\dot{q}_p$) and $\ddot{q}_p$, which is part of the optimization variable.

This characteristic can be exploited to enforce useful linear equality and inequality constraints w.r.t. the optimization variable, similarly to what was done in Section 6.5.

At the end, the final constraints will be written as

$$A_{eq}\,\Theta = b_{eq} \tag{6.130a}$$

$$A_{ineq}\,\Theta \leq b_{ineq} \tag{6.130b}$$

where $A_{eq}$ and $A_{ineq}$ are simply obtained by staking up, respectively, the single equality and inequality constraints. Let us look at a list of possible constraints (they were all implemented in the upcoming simulations).

### Floating base dynamics constraint

The optimization variables need to be computed accounting for the dynamics of the system. This can be easily done exploiting the linearity of the system w.r.t. to the optimization variables. One fundamental constraint is given by

$$M_\lambda(q_p(t_k))\,\ddot{q}_p - \lambda_c = -C_\lambda(q_p(t_k), \dot{q}_p(t_k)) \tag{6.131}$$

which enforces the dynamics of the floating base.

This constraint is easily converted to the form

$$A_{dyn}(t_k)\;\Theta = b_{dyn}(t_k) \tag{6.132}$$

where

$$A_{dyn}(t_k) = [M_\lambda(t_k),\, -I_{2\times 2}]\;\Theta \tag{6.133}$$

and

$$b_{dyn}(t_k) = -C_\lambda(q_p(t_k), \dot{q}_p(t_k)) \tag{6.134}$$

Note that it is not necessary to employ the remaining part of the dynamics (6.120b), since it will be included indirectly at the time of computing the actual control inputs, after the optimization process is concluded. As already specified, the input are obtained employing two of the three equations of (6.120b), meaning that, apparently, there would be an additional equation which the optimization does not consider.

This remaining relationship is accounted for by the already described second order dynamical feasibility constraint, which will be built in the next subsection.

### Dynamical feasibility constraint

We have already discussed multiple times that, due to the underactuation of the system, the second order constraint (5.87) arises.

This constraint has to be enforced into the optimization process, to be sure that the computed input will be completely coherent with the dynamics of the system. Not doing this would inevitably results in very poor performance.

This constraint can be synthetically written as

$$M_{feas}(q_p(t_k))\,\ddot{q}_p = -C_{feas}(q_p(t_k), \dot{q}_p(t_k)) \tag{6.135}$$

which can easily be rewritten in the form

$$A_{feas}(t_k)\;\Theta = b_{feas}(t_k) \tag{6.136}$$

where

$$A_{feas}(t_k) = [M_{feas}(t_k),\, O_{2\times 2}]\;\Theta \tag{6.137}$$

and

$$b_{feas}(t_k) = -C_{feas}(q_p(t_k), \dot{q}_p(t_k)) \tag{6.138}$$

### Non-holonomic constraint

During contact, it is crucial ensure the inputs are chosen in a way that pure rolling is enforced. This is done by using the constraint (3.89), which is here reported for clarity

$$A_\lambda \, \dot{q}_p = 0 \tag{6.139}$$

where

$$A_\lambda = \begin{bmatrix} 1 & 0 & -r_w & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{6.140}$$

This constraint can be integrated into the optimization process, after differentiating it one time w.r.t. time, using

$$[A_\lambda, O_{2\times 2}] \, \Theta = 0 \tag{6.141}$$

Clearly, this constraint has to be relaxed during flight phases.

### Contact constraint

During contact phases, in order to actually ensure rolling without slipping, it is useful to insert the following inequality constraints

$$\lambda(2) \geq \delta_\lambda > 0; \; \lambda(1) \leq \mu \, \lambda(2) \, \text{and} \; \lambda(1) \geq -\mu \, \lambda(2) \tag{6.142}$$

where $\delta_\lambda$ and $\mu$ are, respectively, an arbitrarily small positive number necessary to ensure that the normal reaction stays always positive and a static friction coefficient (a pyramid friction model is used to keep the constraint linear).

These constraints can be synthetically written as

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -\mu \\ 0 & 0 & 0 & 0 & 0 & -1 & -\mu \end{bmatrix} \Theta \leq \begin{bmatrix} -\delta_\lambda \\ 0 \\ 0 \end{bmatrix} \tag{6.143}$$

### Input constraint

As done in Section 6.5, it is useful to impose limits on the control inputs. Let us write synthetically the last two equations of (6.120b) as

$$\check{M}_u(q_p(t_k)) \, \ddot{q}_p + \check{C}_u(q_p(t_k), \dot{q}_p(t_k)) = \check{G}'_u(q_p(t_k)) \, u \tag{6.144}$$

As a consequence, the input constraint can be written as

$$-u_{lim} \leq \check{G}'^{-1}_u(t_k) \, \left[ \check{M}_u(t_k) \, \ddot{q}_p + \check{C}_u(t_k), \dot{q}_p(t_k)) \right] \leq u_{lim} \tag{6.145}$$

which can be rewritten as

$$-\left[ \check{G}'_u(t_k) \, u_{lim} + \check{C}_u(t_k) \right] \leq \check{M}_u(t_k) \, \ddot{q}_p \leq \left[ \check{G}'_u(t_k) \, u_{lim} - \check{C}_u(t_k) \right] \tag{6.146}$$

This expression can be easily converted into

$$A_{u,lim} \, \Theta = b_{u,lim} \, (t_k) \tag{6.147}$$

where

$$A_{u,lim} = \begin{bmatrix} \check{M}_u(t_k), O_{2\times 2} \\ -\check{M}_u(t_k), O_{2\times 2} \end{bmatrix} \tag{6.148}$$

$$b_{u,lim} = \begin{bmatrix} \check{G}_u'(t_k)\, u_{lim} - \check{C}_u(t_k) \\ \check{G}_u'(t_k)\, u_{lim} + \check{C}_u(t_k) \end{bmatrix} \tag{6.149}$$

**State constraint**

It is also possible to include state constraint.

An approximate expression of the state at the next sample time can be obtained with

$$q_p(t_{k+1}) \simeq q_p(t_k) + \delta_s\, \dot{q}_p(t_k) + \frac{\delta^2}{2}\, \ddot{q}_p \gtreqless q_p^{lim}(t_k) \tag{6.150}$$

As a consequence, supposing given bounds on the pitch angle $\phi_r$ and on the leg angle $\beta$, it is possible to write

$$\begin{aligned} q_p(4) &\leq \phi_r^{ub} \\ q_p(5) &\leq \beta^{ub} \\ -q_p(4) &\leq -\phi_r^{lb} \\ -q_p(5) &\leq -\beta^{lb} \end{aligned} \tag{6.151}$$

where the superscripts "ub" and "lb" refer to, respectively, the upper bound and lower bounds (taken with sign). This constraints can be synthetically written as

$$A_{q_p,lim}^{'}\, q_p \leq b_{q_p,lim}^{'}(t_k) \tag{6.152}$$

where

$$A_{q_p,lim}^{'} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{6.153}$$

and

$$b_{q_p,lim}^{'} = \begin{bmatrix} \phi_r^{ub} \\ \beta^{ub} \\ -\phi_r^{lb} \\ -\beta^{lb} \end{bmatrix} \tag{6.154}$$

Inserting (6.150) into (6.152), leads to

$$A_{q_p,lim}^{'} \left[ q_p(t_k) + \delta_s\, \dot{q}_p(t_k) + \frac{\delta^2}{2}\, \ddot{q}_p \right] \leq b_{q_p,lim}^{'}(t_k) \tag{6.155}$$

which, after rearranging, becomes

$$\frac{\delta^2}{2}\, A_{q_p,lim}^{'}\, \ddot{q}_p \leq b_{q_p,lim}^{'}(t_k) - A_{q_p,lim}^{'}\, [q_p(t_k) + \delta_s\, \dot{q}_p(t_k)] \tag{6.156}$$

The final form of the constraint takes the form

$$A_{q_p,lim}\, \Theta \leq b_{q_p,lim} \tag{6.157}$$

where

$$A_{q_p,lim} = \left[ \frac{\delta^2}{2} A^{'}_{q_p,lim}, O_{4\times 4} \right] \tag{6.158}$$

and

$$b^{'}_{q_p,lim}(t_k) = b^{'}_{q_p,lim}(t_k) - A^{'}_{q_p,lim} \left[ q_p(t_k) + \delta_s \, \dot{q}_p(t_k) \right] \tag{6.159}$$

### 6.6.3 Testing (on both Matlab and Gazebo)

For the testing phase, on both Matlab and Gazebo, the following gain matrices were used:

$$K_p = \begin{bmatrix} 150 & 0 \\ 0 & 300 \end{bmatrix} \quad K_d = \begin{bmatrix} 60 & 0 \\ 0 & 30 \end{bmatrix} \tag{6.160}$$

An optimization algorithm (in particular a genetic one) was used as a tool for obtaining suitable values for these gain matrices. An exhaustive description on this procedure is here omitted for brevity. Basically, the tracking response of the system is simulated using a sample cartesian reference trajectory for the CoM, varying the gains in a suitable manner. After a certain number of iterations (defined by the user), the optimization process is stopped and the combination providing the minimum average value of the cost function over the simulation time is selected.

The contact constraint parameters were selected as

$$\mu = 0.5 \quad \delta_\lambda = 1\,\mathrm{N} \tag{6.161}$$

The torque limits were set to

$$u_{lim} = \begin{bmatrix} 2 \\ 20 \end{bmatrix} \mathrm{N\,m} \tag{6.162}$$

while the state limits were chosen to be

$$\begin{bmatrix} \phi_r^{ub} \\ \phi_r^{lb} \\ \beta^{ub} \\ \beta^{lb} \end{bmatrix} = \begin{bmatrix} 45 \\ -45 \\ 95 \\ 62 \end{bmatrix} \mathrm{deg} \tag{6.163}$$

Furthermore, for all the performed simulations, the chosen initial condition and sample interval are, respectively,

$$q_0^{(0)} = \left[ 0, 0, 0, 0, 75 \cdot \frac{\pi}{180}\,\mathrm{rad}, 0, 0, 0, 0, 0 \right] \tag{6.164}$$

$$\delta_s = 0.02\,\mathrm{s} \tag{6.165}$$

The controller requires the solution of a QP. As a consequence, it was necessary to find a suitable C++ solver to be used by the control plugin in Gazebo. Currently, the plugin makes use of a very powerful library called HPIPM ("High-performance Interior-Point Method", see [14]). This library provides an efficient solver for dense, optimal control and tree-structured convex quadratic programs.

In particular, the structure of the reference trajectory for the *relative* CoM position is given by

$$\begin{aligned} x_{CoM}^{ref}(t) &= x_{CoM}^{off} + \rho_x^{CoM} \sin(2\,\pi\, f_x^{CoM}\, t + \Psi_x^{CoM}) \\ y_{CoM}^{ref}(t) &= y_{CoM}^{off} + \rho_y^{CoM} \sin(2\,\pi\, f_y^{CoM}\, t + \Psi_y^{CoM}) \end{aligned} \tag{6.166}$$

**First comparative simulation**

The first simulation is performed using

$$\begin{aligned} x_{CoM}^{off} &= 0\,\text{m};\ \rho_x^{CoM} = 0.05\,\text{m};\ f_x^{CoM} = 1\,\text{Hz};\ \Psi_x^{CoM} = 0\,\text{deg} \\ y_{CoM}^{off} &= 0.4\,\text{m};\ \rho_y^{CoM} = 0\,\text{m};\ f_y^{CoM} = N.D.;\ \Psi_y^{CoM} = N.D. \end{aligned} \tag{6.167}$$

The results of the first simulation are shown in Fig. 6.23.

In particular, it is possible to appreciate a quite good tracking of the relative CoM trajectory. Compared to the approach employed is Section 6.4.3, it is evident how this one is actually much more powerful in controlling the trajectory of the center of mass.
Furthermore, the actuation inputs, as expected, are kept within the given limits.

The tracking is practically perfect in Matlab, while in Gazebo we see how the y-component has some deviation from the desired value. This is because the simulation results in Gazebo were discovered to be highly dependent on the chosen contact parameters. Nonetheless, being the simulated errors within the mm range, the results are still more than acceptable.

Let us look at the actuation plots. In particular, $C_{m1}$ is practically identical between Matlab and Gazebo, while $C_{m2}$ is on average smaller in Gazebo. Once again, this is due to the complexity of the simulator and it was found to be dependent on the chosen contact and joint parameters.
A thorough discussion of these parameters is here omitted for simplicity.

**Second comparative simulation**

For the second simulation the trajectory is chosen to be

$$\begin{aligned} x_{CoM}^{off} &= 0\,\text{m};\ \rho_x^{CoM} = 0.05\,\text{m};\ f_x^{CoM} = 1\,\text{Hz};\ \Psi_x^{CoM} = 0\,\text{deg} \\ y_{CoM}^{off} &= 0.4\,\text{m};\ \rho_y^{CoM} = 0.04\,\text{m};\ f_y^{CoM} = 1\,\text{Hz};\ \Psi_y^{CoM} = 90\text{deg} \end{aligned} \tag{6.168}$$

The results of the first simulation are shown in Fig. 6.24. Once again, the tracking is quite good and there is little difference between the simulation in Gazebo and in Matlab. As already specified, due to contact modeling, $C_{m2}$ is slightly different between the two simulators.

**Estimated ground reactions**

Note that the use of the floating base model for setting up the optimization process also allows to obtain an estimate of the ground reaction forces. Recall that the QP is set in away that the computed control inputs are chosen to keep the ground reaction forces within the given constraints and consistent with the floating base dynamics.

For completeness, the the obtained ground reactions are shown in Fig. 6.25 for both simulations. Looking at the plots, it is possible to see that all contact constraints are indeed satisfied.

The computed reactions, in particular the vertical components, are slightly different in Gazebo and Matlab. Once again, this is most likely due to the fact that Gazebo involves a much more complex modeling of the contact.

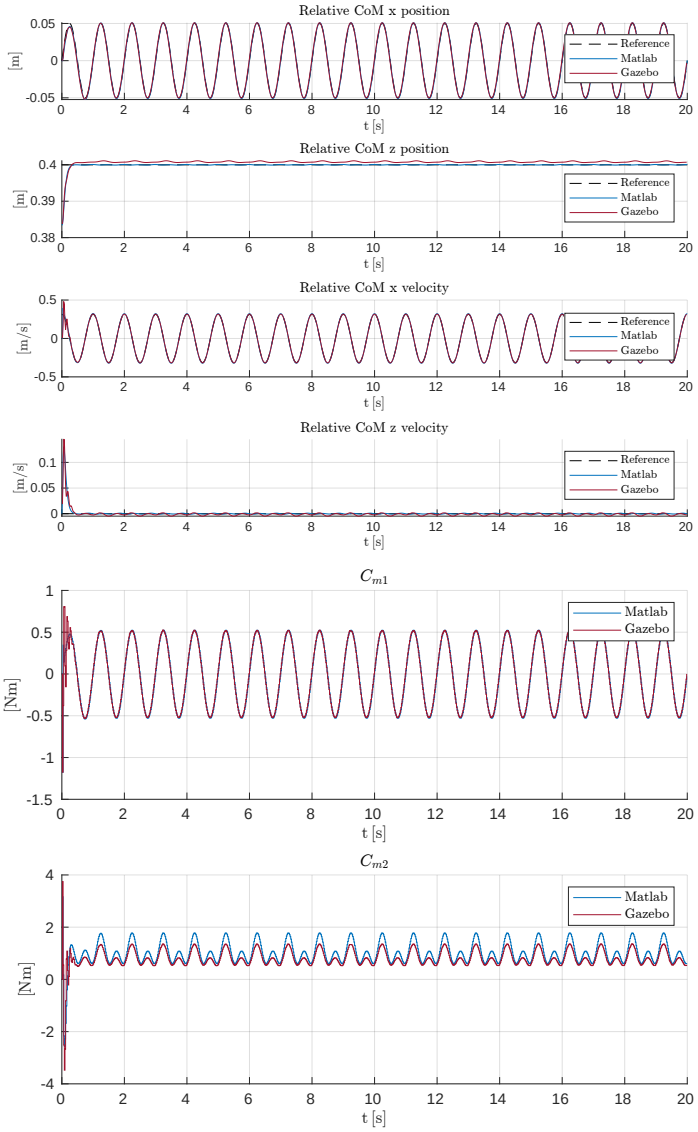



**Figure 6.23.** Comparative results of the Task-Space tracker simulated in Matlab and Gazebo (first simulation). In particular, the tracking is performed on a relative CoM trajectory described by $x_{CoM}^{off} = 0\,\text{m}$; $\rho_x^{CoM} = 0.05\,\text{m}$; $f_x^{CoM} = 1\,\text{Hz}$; $\Psi_x^{CoM} = 0\,\text{deg}$ and $y_{CoM}^{off} = 0.4\,\text{m}$; $\rho_y^{CoM} = 0\,\text{m}$; $f_y^{CoM} = N.D.$; $\Psi_y^{CoM} = N.D.$ .

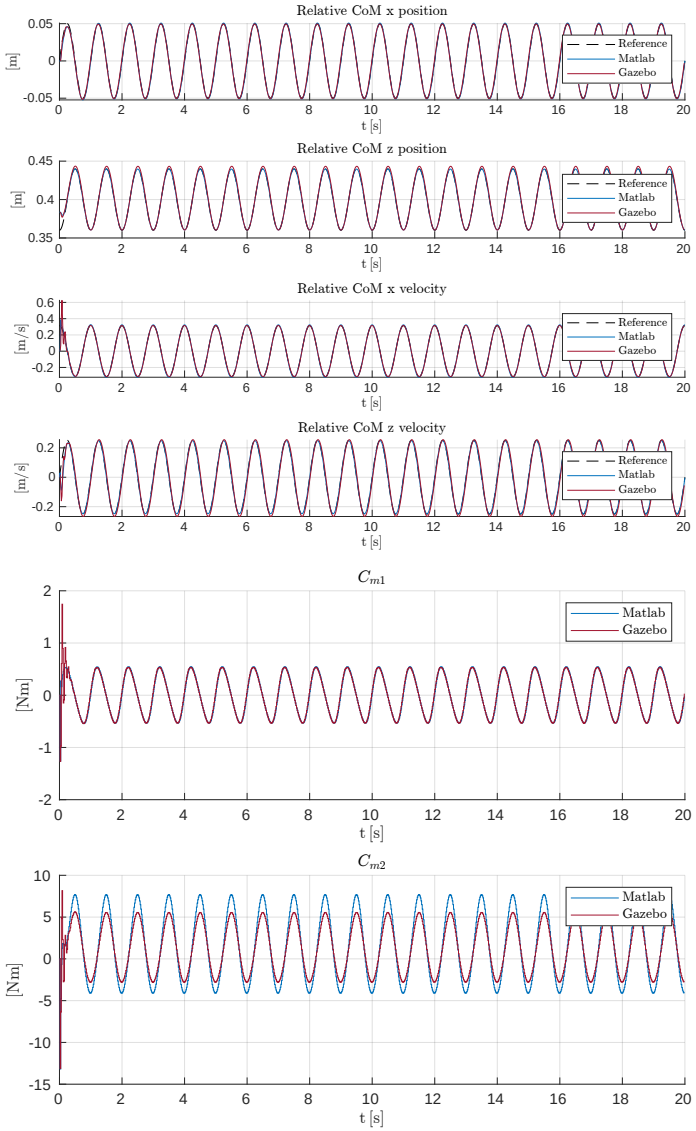
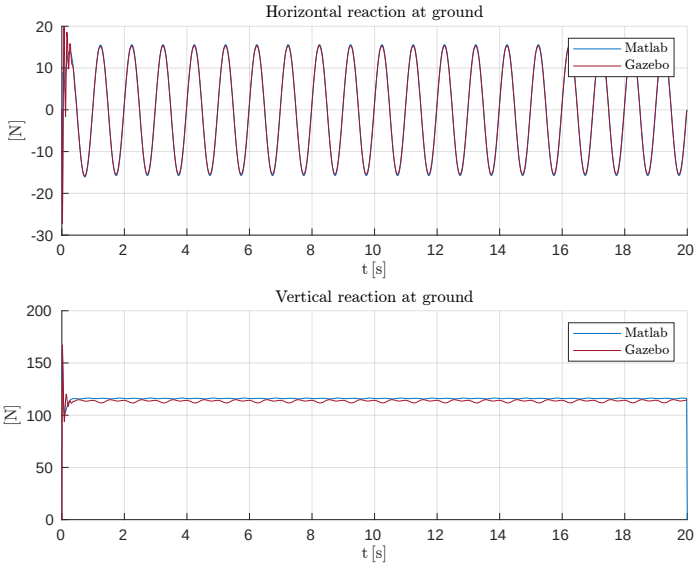
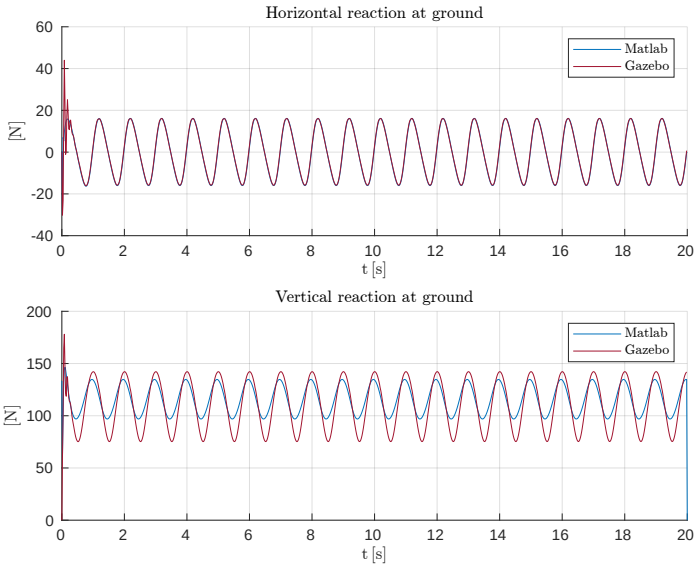



**Figure 6.24.** Comparative results of the Task-Space tracker simulated in Matlab and Gazebo (second simulation). In particular, the tracking is performed on a relative CoM trajectory described by $x_{CoM}^{off} = 0\,\text{m}$; $\rho_x^{CoM} = 0.05\,\text{m}$; $f_x^{CoM} = 1\,\text{Hz}$; $\Psi_x^{CoM} = 0\,\text{deg}$ and $y_{CoM}^{off} = 0.4\,\text{m}$; $\rho_y^{CoM} = 0.04\,\text{m}$; $f_y^{CoM} = 1\,\text{Hz}$; $\Psi_y^{CoM} = 90\,\text{deg}$.




**(a)** First simulation




**(b)** Second simulation

**Figure 6.25.** Ground reaction forces estimation. Results of the first simulation (above) and of the second (bottom). In particular, it is possible to appreciate how the vertical component of the GRF is slightly different between simulation performed in Gazebo and Matlab. This is due to how Gazebo handles ground contact.

# Chapter 7

# Prototype design and implementation

This chapter focuses on the design and implementation of a prototype of the system under study.

The prototype is meant to be a flexible research platform over which testing of control algorithms is possible.
The work on the prototype started together with the control part of the thesis and proceeded in parallel with it.
Unfortunately, at the time of writing, the prototype is not finished yet, both for lack of resources and time.
For brevity sake, the description of all iterations which led to the current design are here omitted and only the present configuration is reported.

The current prototype has a fixed leg and, as a consequence, no knee. This makes it basically an inverted pendulum on wheels. The reason for this choice is very simple: implementing a complete prototype with legs would require time and, more importantly, resources which are not currently available. Unfortunately, while wheel actuators are relatively cheap, leg actuators can become quite expensive due to torque requirements. As a consequence, the choice was to start developing a functional prototype with fixed leg knowing that the integration of a full leg becomes a relatively trivial matter of sourcing the right actuators and components. This aspect will be addressed in Section 7.11, where the integration of a full leg will be briefly described.

Currently, the "no-knee" prototype is made up of the following main components:

- The supporting structure is made of V-slotted 20 mm × 20 mm aluminum profiles.
- Two wheels, obtained from an old push scooter (see Section 7.7).
- Two outrunner brushless motors for wheel actuation (see [27] and Section 7.1).
- Low-cost high performance motor controller/s (ODrive, see [28]).
- Two incremental encoders for detecting the position of the wheel axis (see [10] and Section 7.1).

- Custom-designed and 3D-printed motor enclosures (see Section 7.7).
- Custom-designed and 3D-printed wheel rim couplers (see Section 7.7).
- Aluminum shaft couplers for connecting wheels to motors.
- Two 120 mm × 120 mm high quality cooling fans for the electronics (see Section 7.1).
- Custom-designed and 3D-printed fan mounting plates (see Section 7.7).
- Custom-designed and 3D-printed electronics plate (see Section 7.7).
- 6s 4000 mA h LiPo battery for powering the whole system (see Section 7.5).
- Hand wired and shielded encoder cables for reducing sensing noise.
- Hand wired cables for all electronics.
- Various electronics connectors.
- Prototyping boards.
- DC-DC step-down buck converter for powering cooling, the control and sensing electronics (see Section 7.5).
- Arduino Nano 33 IoT (or, alternatively, BLE 33 Sense) for interfacing with sensors (see [3] and Section 7.2).
- Raspberry Pi model 4B (see [34]) with 8GB of RAM as the Main Control Unit (from here on abbreviated as MCU). Currently, Ubuntu Server 20.04 is the installed operating system (see Section 7.10 for a brief explanation as to why this OS was chosen). Furthermore, to facilitate real-time performance, a *fully preemptable* kernel is installed (`Linux ubuntu 5.10.73-rt54-v8+`). Henceforth, the acronym RPI will be used to reference this component.
- Adafruit BNO055 sensor fusion IMU (see [2], [5] and Section 7.2).
- CAN-SPI click 3.3 V IC for interfacing Arduino with the CAN bus (see [26] and Section 7.4).
- Seedstudio CAN-SPI hat for connecting the RPI (abbreviation of Raspberry Pi) to the CAN bus and also powering it (see [37] and Fig. 7.4).
- A large amount of screws, mechanical connectors, etc.

A hypothetical future complete prototype would also include:

- Two (or more) leg actuators (to be defined).
- More aluminum profiles for assembling the legs' linkage.
- Bearings for the joints of the leg.

- Custom-designed and 3D-printed links couplers.
- Leg torsional springs (also linear springs can be used, but the elastic potential energy of the system has to be modified accordingly).
- Additional motor controllers (depending on how the actuators are chosen).

## 7.1 Actuators and power electronics

The choice of the wheel actuators was initially made using [19] as a reference, combined with very simple dynamical simulations on a simple WIP. Special thanks to Victor Klemm, coauthor of [19], whose help was very useful in the early stage of the thesis.
This was enough to fix torque requirements on the wheel motors (these requirements were then further verified using the latest simulations).

Then, the motor type had to be selected. After some research, PMSM (permanent magnet synchronous motors, also called brushless) motors were selected. Other alternatives would have included stepper motors and DC motors. Brushless motors are nowdays widely used in robotics and industrial automation in general due to their high power density, efficiency, silent operation, easy maintenance, precision control, etc. Brushless motors are available at relatively low price, but they have a major downside: they need dedicated controllers. Many cheap controllers are available at an hobbyist level and they generally allow decent velocity control, but they are not suited for robotics applications. Since performing high level control requires the capability of controlling torques with acceptable precision, having a good controller is of the utmost importance.

Unfortunately, non-industrial alternatives are really rare. At the end, after researching various alternatives, ODrive high performance motor controller was chosen (see Fig. 7.1).

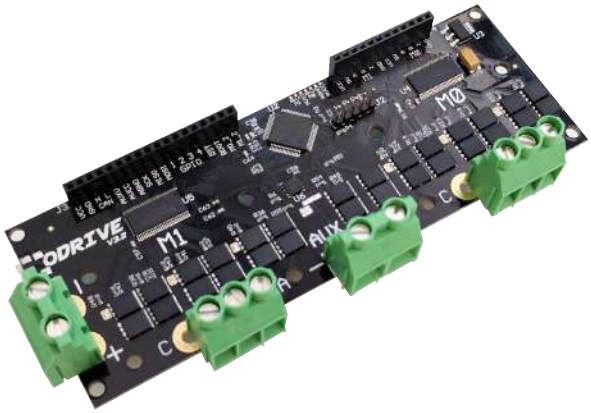



**Figure 7.1.** ODrive

Let us look at the main specifications of this powerful control board (for more details, refer to [28]):

- Two axis control (clearly intelligible by looking at the number of three-phase connector in Fig. 7.1). This means that one ODrive can control two separate motors independently.
- 24 V or 48 V versions available (48 V is used to have potential room for more powerful motors).
- 120 A peak current per motor.

- Support for incremental or absolute encoders.
- Two braking modes: braking resistor/regenerative braking. Since we are using a LiPo battery, the use of a brake resistor is not necessary.
- Open source hardware and software.
- Position control, velocity control, torque control and trajectory control.
- USB interface for connection or software updates, UART ("Universal Asynchronous Receiver-Transmitter"), Servo PWM/PPM, CAN (via a very basic custom protocol called CANSimple)
- General purpose analog pins.

A very important feature of ODrive is that there is a vibrant community gravitating around it. After some confrontation with the community and with ODrive's creators, it was confirmed that it implements *Field Oriented Control* (usually abbreviated as FOC), which is commonly used for controlling permanent magnet synchronous motors and induction motors. This technique allows very precise control of the motor. Details are here omitted since are out of the scope of this thesis.

For what concerns the prototype with no knee a single ODrive is sufficient, since it can control two motors. A complete documentation is available at ODrive GitHub page, specifically at [29], inside "docs" folder.

ODrive website provides also a set of tested motors, with their characteristics and test results. Using this guide, the dual axis D5056 was chosen. This motor is shown in Fig. 7.2


**Figure 7.2.** D5056 dual axis

Let us look briefly at its characteristics:

- 8 mm primary and secondary shaft, compatible with CUI AMT-102 encoder (see [10]).
- 7 pole pairs and 12 stator slots.
- Thermistor thermally connected to motor windings to monitor temperature.
- 270 KV (rpm/V).
- Motor phases with 4 mm male bullet connectors.
- 65Å maximum current, 32 V maximum voltage, phase resistance 39 mΩ

- 420 g mass.
- Maximum torque 1.99 Nm.

The motor is connected to ODrive through its three-phase cables (they can be connected with arbitrary order).

The last necessary component is a suitable rotary encoder for position feedback. Note that it is necessary also when performing torque control, due to how FOC works.

The chosen encoder was the one recommended on ODrive webpage, the CUI ATM102 V capacitative incremental encoder (see Fig. 7.3a). Since it is widely tested with ODrive, this was the easiest choice (it is also quite cheap). Its main characteristics are the following (for full spec see the datasheet):

- Programmable resolution.
- Index pulse.
- 8192 CPR (counts per revolution) maximum resolution.
- Max speed of 7500 rpm.
- Different mounting bases and a selection of shaft collars to fit various shaft diameters.

These encoders connect to ODrive through its two SPI ("Serial Peripheral Interface") interfaces. Since encoders are particularly sensible to noise, it is advisable to shield the SPI cable. Encoders' cables were manually wired and shielded, instead of buying them. These are visible in Fig. 7.22.

The very last power components are two cooling fans for avoiding to overheat both the ODrive/s and the MCU. This is particularly important for the ODrive, since a good cooling also ensures better control performance. In particular, the chosen model was a 12 V 120 mm × 120 mm Noctua NF-P12, visible in Fig. 7.3b.

**(a)** AMT102 incremental encoder

**(b)** Noctua NF-P12 cooling fan

**Figure 7.3**

## 7.2 Sensing electronics

Given the states required to describe the system, a suitable set of sensor has to be selected.

Due to thesis budget, high-cost absolute position cameras, etc. are out of reach. During ground contact, however, there exist relatively low-cost viable alternatives which allow, in principle, to reconstruct the state of the robot. Furthermore, suitable filtering can be employed to improve measurements accuracy and remove noise. This aspect is purposely not addressed here for simplicity (it is however a very important problem).

The encoders used by ODrive for controlling the motors are the first sensing electronics. The disadvantage of using incremental encoders is that ODrives needs to search for the index pulse at each system boot. Under this point of view, absolute encoder would be a more suitable choice. This is, however, a relatively small drawback.
The encoders provide, through the ODrive, wheels' angles w.r.t. the links on which they are mounted and an estimate of the rotational angular velocity. If considering a hypothetical complete system with actuated legs, the encoders mounted on those actuators would provide information on legs state.

Then, there is the need for some sensor capable of measuring information on orientation and, possibly, angular rates.

After researching, the Adafruit BNO055 fusion IMU sensor was chosen. It is a relatively cheap IC which has the capability of performing sensor fusion using data coming from a magnetometer, gyroscope and an accelerometer. There is also a newer version, called BNO085, which offers much better performance.
Its main characteristics are the following (refer to BNO055 datasheet for full specs):

- 9-DOF sensor.
- MEMS accelerometer, magnetometer and gyroscope.
- Onboard high speed ARM Cortex-M0 based processor.
- Absolute Orientation at 100 Hz (yaw-pitch-roll/quaternion).
- Three axial angular velocity at 100 Hz.
- Gravity vector.
- Acceleration vector (with or without gravity) at 100 Hz.
- Magnetic field strength vector at 20 Hz.
- Ambient temperature at 1 Hz.
- 3.3-5.0 V power supply input.
- I2C interface for connection with microcontrollers.
- UART interface (selectable).

One of the most important features to look at is the available sample frequency of the IMU. Looking at the specs, this is limited to 100 Hz. This means that control policies need to be tested at a frequency equal or lower than 100 Hz (better to stay reasonably lower to have a good margin).


**Figure 7.4.** Adafruit BNO055

Note that this sensor's interface allows an easy connection to microcontrollers like, for instance, Arduino-based boards. Furthermore, a dedicated Arduino library is available. It is also possible to interface it with a Raspberry Pi through UART, but this requires some tweaking which could potentially produce problems on other interfaces.

Now, since the application is space-critical, the best option is to go for boards from the Nano family, since they have small footprints. The selected boards are the Nano 33 IoT and the 33 BLE sense (see Fig. 7.5). They are particularly suited for this application and, furthermore, they were both already available for use. These board have the same pinout, so they can be switched with no hardware nor software modification.

Let us compare briefly their relevant technical specifications (Nano 33 IoT left, Nano 33 BLE right):

- SAMD21 Cortex®-M0+ 32bit low power ARM MCU.
- 3.3V operating voltage.
- 5-21V input voltage.
- 7 mA DC current per I/O pin.
- 48 MHz clock speed.
- 256 KB CPU flash memory.
- 32 KB SRAM.
- Native USB.
- LSM6DS3 IMU.
- u-blox NINA-W102 radio module.
- nRF52840.
- 3.3 V operating voltage.
- 5-21 V input voltage.
- 15 mA DC current per I/O pin.
- 64 MHz clock speed.
- 1 MB CPU flash memory.
- 256 KB SRAM.
- Native USB.
- LSM9DS1 IMU, MP34DT05 microphone, APDS9960 gesture, light and proximity sensor, LPS22HB barometric pressure, HTS221 temperature and humidity.

By comparing the specifications, it is clear how the BLE sense is in general a more powerful board, with higher clock frequency and, more importantly, much higher

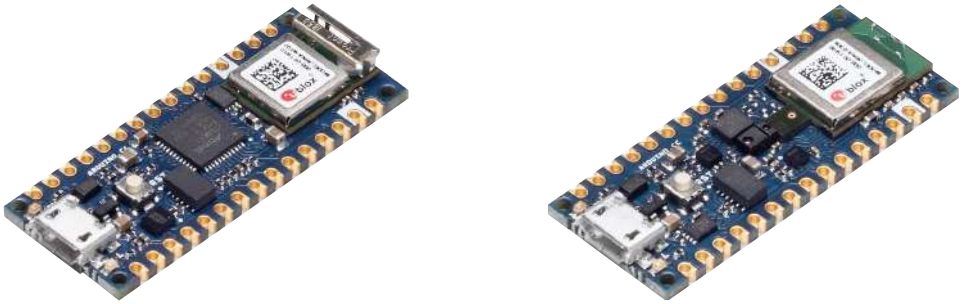


**Figure 7.5.** Arduino Nano 33 IoT (left) and 33 BLE sense

flash memory and RAM. This means that it is capable of holding heavier scripts. The choice between the two is guided by the particular computational requirements; in the particular case of the prototipe, the Nano IoT is more than enough. Note that both boards (specially the BLE sense) potentially provide additional sensing capabilities.

This configuration is able to provide the absolute orientation of the body and also allows the computation of its angular velocity starting from the knowledge of gyroscope's angular rates. These, combined with the knowledge of motor axes position and angular rates, allows to reconstruct the full state during ground contact. Clearly, if the robot loses contact, either an absolute position sensor is available or some techniques employing integration of linear accelerations and other sensor data are necessary.

In the very specific case of planar motion, these sensors are more than enough to reconstruct the state (if ground contact is maintained).

## 7.3 Control electronics

The control electronics is one of the main components of the prototype. The chosen MCU is a Raspberry Pi model 4B with 8GB of RAM (shown in Fig. 7.6).

**Figure 7.6.** Raspberry Pi model 4B

Let us understand why it is indeed a good alternative and what are its limitations.

First, compared with standard microcontrollers available on the maket, the RPI has much bigger computational power, since it is de facto a full fledged computer and can host complete operating systems. This includes the possibility of using high level programming languages like C++ or Python and their optimization libraries. This characteristics are crucial when developing complex control policies.

Second, the RPI has full WiFi connectivity. This allows remote interaction with the robot, including integration in a ROS framework (see Section 7.10).

However, the are at least two issues with using this alternative. The first is that, compared with microcontrollers or FPGA ("Field Programmable Gate Array"), real time capabilities are not ensured. In fact, while running a full operating system multiple processes are executed at the same time and this can somehow disturb an hypothetical running control script and produce high latencies. There are however workarounds which can make the RPI closer to a RT (real time) system: it is possible to apply suitable patches to the kernel. In the specific case of the prototype, this operation was indeed carried out. See Section 7.10 for more details.

The second downside is that, unlike microcontrollers like Arduino, the RPI is not build for interfacing with many sensors. This can make integration into a system slightly complicated.

Let us look rapidly at Raspberry Pi 4B main specifications:

- Broadcom BCM2711, quad-core Cortex-A72 (ARM v8).
- 64-bit SoC @ 1.5 GHz.
- 8 GB LPDDR4.
- 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless.
- LAN, Bluetooth 5.0, BLE.
- Gigabit Ethernet.
- 2 USB 3.0 ports.
- 2 USB 2.0 ports.
- Standard 40-pin GPIO header.
- micro HDMI ports.
- Micro SD card slot for loading operating system.
- 5 V DC via USB-C connector input voltage.
- 5 V DC via GPIO header input voltage.
- Operating ambient temperature 0–50 ºC.

We see that the RPI has all the characteristic of a full-fledged computer. Note that it can also be powered through the general purpose pins but, differently w.r.t. the Arduino boards, it does not have any voltage regulator; as a consequence, the

input voltage needs to match the nominal voltage as closely as possible. In this specific case, the solution to such a problem was provided by the used CAN-SPI shield. In fact, it allows to power the RPI using an external power source in the range $12 - 14\,\mathrm{V}$ (see Section 7.4).

## 7.4 Communication hardware

Now that all the subunits are defined, it is necessary to address the problem of how those units should communicate and interact. Let us look at each main component and at the viable alternatives for interfacing it with the other subunits.

### 7.4.1 ODrive main interfaces

Odrive has many interfaces for communication; one of the main ones is its USB protocol/utility, which is written in Python and can be used for commanding and setting up each axis. Using the USB utility for controlling the motors is not the best option since, as highlighted while discussing with the ODrive community, USB connection is a serial and non-deterministic communication interface and, moreover, the fact that it is written in Python does not help. This means there is no certainty on the timing of sent commands and received messages.

Other viable options would include, for example, using UART interface. This alternative would be best suited if the MCU was a microcontroller-like component, like for instance an Arduino. We saw however that, due to control requirements and budget requirements, using a RPI represents one of the best (if not the best) alternative.

The remaining alternative is using ODrive's CAN interface and simplified CAN protocol (aptly called "CANSimple"). This was the suggested communication protocol by the ODrive community right from the start, since it is a deterministic protocol. ODrive's CANSimple protocol can be read at [29]`/docs`.

### 7.4.2 Brief introduction to CAN

Let us look very briefly at what a CAN bus and protocol is.

CAN stands for "Controller Area Network" and is (borrowing from Wikipedia) "a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other's applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but it can also be used in many other contexts. For each device, the data in a frame is transmitted sequentially but in such a way that if more than one device transmits at the same time, the highest priority device can continue while the others back off. Frames are received by all devices, including by the transmitting device".

As stated by ODrive documentation (see [29]/docs), put in simple terms, it is "a way of communicating between many devices over a single twisted pair of wires" (synthetically sketched in Fig. 7.7).

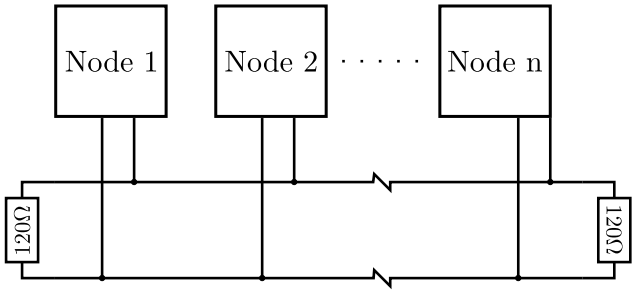



**Figure 7.7.** CAN bus simplified sketch

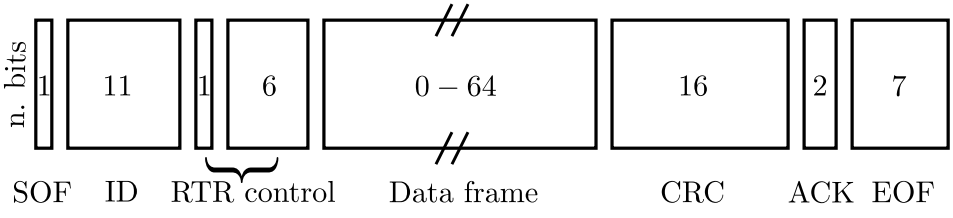
There are many advantages of using CAN communication. These include

- Simple and robust *Physical Layer*. Only a pair of twisted wires with terminating 120 Ω resistors at each end are needed (for preventing signal reflection).
- The signal is transmitted as a potential difference between the two wires, making the bus very robust w.r.t. noise.
- Differently w.r.t. other protocols like I2C or SPI, CAN messages have a unique ID that is also used for priority.
- It has low jitter and low latency, since no host computer is needed.
- It is relatively fast (up to 1 Mbps with CAN 2.0b, up to 8 Mbps with the more recent CAN FD).
- Transceivers and controllers for the physical layer are cheap and widely available.

Fig. 7.8 shows the structure of the message frame used by CAN 2.0A protocol. Different protocols may have slightly different message structure (for example, CANSimple has a simplified structure, which can be consulted at [29] `/docs`). For brevity, details on what each slot precisely means are here omitted. Other standardized protocols include CAN 2.0b and the more recent CAN FD.

ODrive connection to the CAN bus does not require any additional physical interface, since all coding and decoding operations are carried on board. It also has selectable 120 Ω resistors.

More details on how to actually use ODrive's CANSimple protocol will be given in Section 7.10.




| | |
|---|---|
| SOF | "Start of Frame" |
| ID | Message ID |
| RTR | "Remote Transmission Request" |
| CRC | "Cyclic Redundancy Check" |
| ACK | "Acknowledged" |
| EOF | "End of Frame" |

**Figure 7.8.** CAN 2.0a message frame

### 7.4.3 Connecting the Raspberry Pi to a CAN network

Let us look at the available interfaces for the RPI 4B.

As already mentioned, there are 4 USB ports, 2×I2C, 2×UART and 2×SPI (which will be used by the CAN interface).

Since the RPI has to act as the MCU and the ODrive is best commanded via CAN interface, it follows that it is also necessary to connect somehow this computer to the bus. There are a number of shields/hat which allow this. After researching, the choice was to use SeedStudio CAN-SPI hat (see 7.9), which supports the most recent CAN FD protocol and, moreover, has two available CAN interfaces (it employs both SPI interfaces present on the RPI). The hat is shown in Fig. 7.9. Note that, however, this is not a problem since CAN FD is backward-compatible with CAN2.0a/b. Furthermore, CANSimple is compatible with CAN2.0a/b. As a consequence, the network should be able to communicate without problems.

Let us look at the characteristics of Seedstudio CAN-SPI shield:

- 2 channel CAN BUS I/O (up to 8 Mbps).
- On-board 120 Ω terminating resistors with switches.
- MCP2518FD controller.
- MCP2557FD transceiver.
- 12 - 24 V input power screw terminal (also allows to power the RPI).
- Connection to CAN bus through high speed SPI interfaces.
- 2×I2C connector.
- On-board real time clock (RTC-PCF85063). This can be useful, since the RPI does not have it and by default uses the network clock.

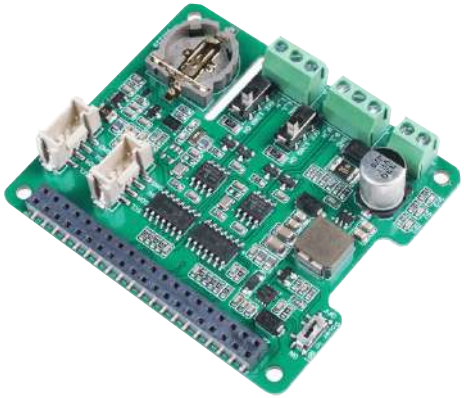


**Figure 7.9.** Seedstudio CAN-SPI hat for RPI

Having two available CAN buses allows further flexibility. In particular, one bus will be used for connecting the RPI with ODrive/s, while the other will be used for interfacing with the sensing electronics. This way message conflicts are avoided and the sensing operation can be carried in parallel with the control one (see Fig. 4.1).

### 7.4.4 Connecting the sensing unit (Arduino) to the CAN bus

Lastly, let us look at what options are available for interfacing with the sensing electronics. Since the choice was already made to use an Arduino Nano as buffer between sensors and the MCU, the problem reduces to exploring ways of connecting the Arduino to the MCU. Since the IMU uses the only I2C available interface, the remaining ones are the SPI, UART and USB serial connection.

USB connection is not the best choice for the already mentioned problems of not


**Figure 7.10.** CAN-SPI click 3.3 V breakout board

being a deterministic interface. If necessary, this interface can be used very easily by connecting to RPI's USB ports. For example, it can be useful for programming the Arduino or for interfacing to ROS via rosserial_Arduino (see Section 7.10). Using UART connection is not practically viable, since access to RPI pins is made quite difficult by the CAN-SPI hat mounted on it. So, only SPI interface remains.

There are many available CAN-SPI hats for Arduino boards. However, almost all of them work at 5V. After a thorough research the best (if not only) option was the CAN-SPI Click 3.3v breakout board from MikroElektronika (see [26]). This

board is shown in Fig. 7.10.
Its main characteristics are the following ones:

- 3.3 V operating voltage (compatible with most recent microcontrollers of the Nano family).
- SN65HVD230 CAN transceiver.
- MCP2515 CAN controller.
- 10 MHz onboard oscillator (important spec).
- Selectable terminating 120 Ω resistor.

Currently, Arduino's SPI CS (Chip Select pin) is wired to pin 10. This is an information which is needed when setting up the used CAN library. Other SPI pins are not here reported, since they cannot be chosen arbitrarily.

One of the most important characteristics of this breakout board is that it mounts an 8 MHz oscillator; these oscillators are needed by the CAN controller for setting up the communication and their frequency can be chosen between a discrete set of values. Even though 8 MHz is actually one of the suggested values (as reported in the MCP2515 datasheet), it is rarely used. In particular, every available Arduino library which was found on the web could only work with different oscillator frequencies. This is because, depending on oscillator and on bus settings, certain configuration values have to be written to three different controller registers. This values also depend on the used bitrate. As a consequence, CAN libraries have a preconfigured matrix of settings for operating a given CAN controller (in this case MCP2515) at different data trasmission rates. This matrix is usually referred to as *bit timing matrix*. After quite a lot of time searching for solutions for computing the necessary configuration values, an online bit timing calculator was used (see [20]). Inserting the oscillator frequency, desired bit rate and controller type, a list of possible configuration values is shown. Fortunately, even if only the MCP2510 was available for selection on that web page, settings proved to work also with the newer MCP2515. Once those configuration values are available, one needs to dig into the chosen Arduino library and manually add this settings. This process was quite tedious, but at the end everything worked fine.

### 7.4.5 Summing up

All subunits can be interfaced choosing from a flexible set of communication hardware. The preferred one should be the CAN bus, but also the use of serial USB communication, as well as I2C and UART are potentially possible. Furthermore, thanks to the presence of the RPI, the robot also gains WiFi connectivity and hence the possibility of interacting with it remotely inside a network. Note that this communication configuration is also valid in the case of a complete prototype with actuated leg.

## 7.5 Power sources

Now that all the main subsystem are outlined, it is possible to address the question of how to power the whole system.

After cross-comparing many alternatives, a 6s LiPo battery was chosen (see Fig. 7.11). The name 6s means that it has 6 standard LiPo cells arranged in series. LiPo batteries generally have relatively high energy density and can withstand high power peaks, making them particularly suited for robotics application. Unlike more classical batteries, they need dedicated chargers, due to the necessity of balancing each single battery cell. If cells become unbalanced, the battery deteriorates rapidly and can become dangerous. Furthermore, to preserve battery life, when they are not used for a long period of time, it is advisable to discharge the battery at a safe voltage level (this is usually done automatically by chargers).


**Figure 7.11.** 4000 mAh 6s LiPo battery

Since a single cell has a nominal voltage of 4.2 V when fully charged, the operating voltage will be approximately 25 V, which is far lower than ODrive's input limit. This choice was motivated by a compromise between several requirements: budget, flexibility, weight, capacity, etc.. The fact that the voltage is also lower than motor's nominal voltage simply implies that the maximum rotational speed will not be reachable. This is not an issue, since what is really important for the prototype is to have as high a torque as possible. After some torque tests (see Section 7.9) on one of the motors and also after having asked for ODrive's creators opinion, the possibility of reaching peak torque with this input voltage was confirmed.

This covers the powering of actuators and their controller.
Considering that cooling fans have a nominal operating voltage of 12 V, noting that, thanks to the CAN hat, the RPI can be powered using input voltages in the range 12-24 V and taking into account that the Arduino Nano can be powered with a source in the range 5-21 V, the best option is to make available a second power line with operating voltage slightly higher than 12 V. This is done to ensure the input voltage to Seedstudio CAN hat is higher than the minimum, i.e. 12 V; note that this does not produce problems for powering cooling fans, since they can tolerate higher voltages to some extent.

To derive this second 12 V power line the best choice is to use a DC-DC buck converter, which is shown in Fig. 7.12. In particular, this buck converter can tolerate input voltages in the range 3.2-40 V and output an arbitrary voltage in the range 1.25-35 V. The output voltage is simply set by rotating a split screw and by checking its value by using a voltmeter.

**Figure 7.12.** DC-DC Buck converter

## 7.6 Wiring design

This section is dedicated to showing a simplified and synthetic diagram of the actual wiring of the prototype's electronics.

Here only two motors, two encoders and one ODrive is shown, since it reflects the wiring of the no-knee prototype. Clearly, when adding more actuators and controllers, nothing really changes conceptually. If more power or capacity is required, it is possible to use two LiPo batteries and connect them in series or in parallel (when connecting them in parallel it is of the utmost importance that cells are properly balanced).

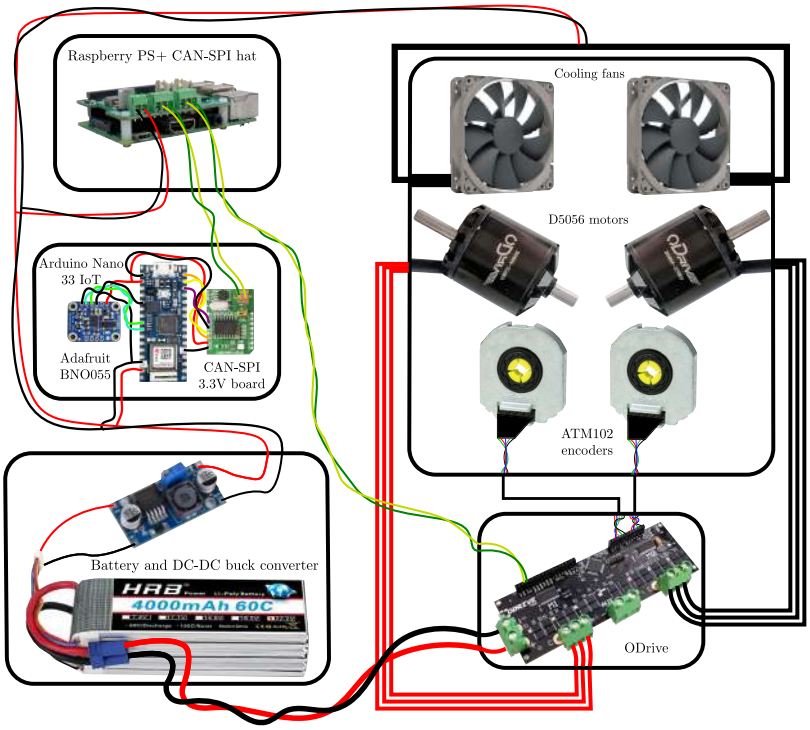



**Figure 7.13.** Wiring synthetic schematics. Note the CAN bus (twisted green-yellow wires), the high-voltage line (thick red and black wires coming out of the battery), the low-voltage line (thin red and black wires) and the three-phase cables powering the motors.

In Fig. 7.13 it is possible to distinguish the high-voltage line, which powers the ODrive, and the low-voltage one, which powers the Raspberry Pi, the Arduino Nano and the cooling fans.

As already stressed, there are potentially two available CAN buses, clearly distinguishable looking at the twisted green and yellow wires. The motors are connected to ODrive through three phase tick wires. Encoders interface with ODrive's SPI ports and each of them is connected through five distinct cables (Vcc, GND, A, B, X) contained in a shielded enclosure made of intertwined copper.
The CAN SPI click board interfaces with Nano's SPI pins, while Adafruit BNO055 IMU is connected through I2C. For further details it is possible to look at each component's pinout.

Beyond CAN bus, other possible interfaces include USB ports available on the RPI, the Nano and the ODrive controller.

## 7.7 Structural design and 3D-printed parts

The structure of the prototype is made mainly of aluminum V-slotted extrusions and additional 3D-printed parts for interfacing with the main aluminum structure. For more details on the used 3D printing hardware refer to Section 7.8.

After some research, this alternative was chosen since it provides sufficient rigidity to the structure, while maintaining low weight; furthermore, it is a relatively cheap option and provides a modular structure. This last characteristic is very important, since it allows easy modifications and integration of additional components.

Alternatively, one could have 3D-printed the whole structure. For example, [19] has 3D-printed legs. This is definitely a viable alternative since doing so allows for fast prototyping iterations and also for highly customizable parts. However, the viability of this option also depends on the quality of the available 3D printer and of the printable materials. In most cases, budget 3D printers can at best print modestly rigid plastic materials. For this reason, the choice was to use as many as possible commercially available and standard parts and then print the remaining components. In particular, [45] and [25] were used to obtain 3D models of standard parts. Furthermore, the used 3D modeling software is Solid Edge (see [40]).

Let us explore briefly the main sub-units which make up the prototype. Unessential details are here omitted for brevity.

### 7.7.1 "Foot" design

The very first unit to be designed (and also machined) was the "foot" of the robot. This component uses as template a 3D printable case available at [28]. This simpler design was then adapted and improved to fit the specific needs of the prototype.

The current state of the design is shown in Fig. 7.14, which contains a front and back view of the unit. Clearly, for simplicity, the three-phase cables going out from the motor are not shown.

In particular, a somewhat relevant aspect is that the aluminum extrusion serves at least two purposes: firstly, it makes the strucure of the unit more rigid and,

secondly, it allows for an easy connection with the rest of the robot. The encoder is placed on the back of the enclosure. Note that it connection is made so that encoder's alignment can be adjusted easily; this is crucial to preserve its quality.

**Figure 7.14.** Renderings of prototype's "foot"

### 7.7.2 Wheel design

The second units to be designed and manufactured were the wheels. The actual wheels were obtained from an old push-scooter and suitable wheel couplers were designed. Fig. 7.15 shows at the top two views of the above mentioned couplers (shown in blue). These couplers are then connected to the rims through seven bolts, while the couplers are connected to motors' shafts through aluminum shaft couplers (bottom two figures in Fig. 7.15).

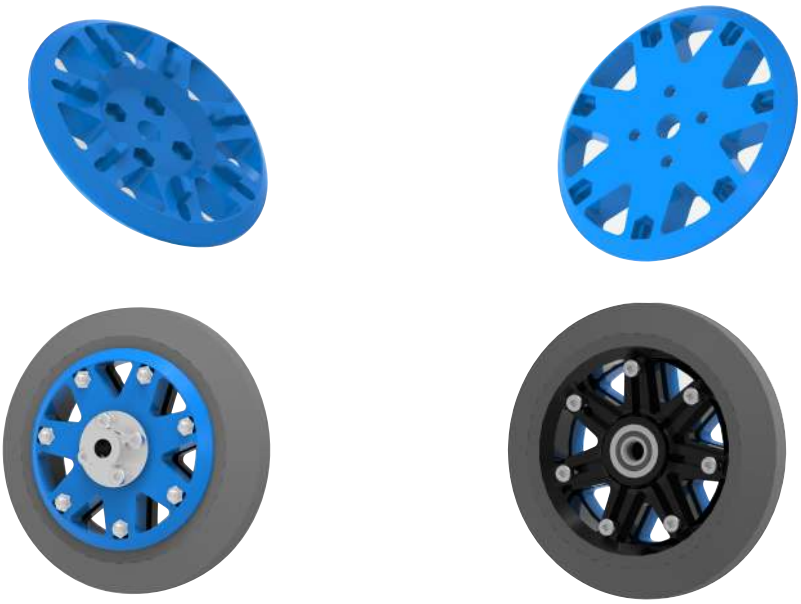


**Figure 7.15.** Renderings of the wheel (3D-printed wheel coupler shown in blue)

### 7.7.3 Electronics enclosure

Then, it was necessary to find a suitable way of enclosing all the electronics. Exploiting the modularity of aluminum extrusions, the choice was to make a simple parallelepiped "cage". Using aluminum profiles allows for easy integration of components inside this cage, electronics included.

The electronics is positioned on a 3D-printed plastic plate, which allows allows for electrical insulation between the electronics and the rest of the robot. The exact positioning was devised so as to make the electronics as easily accessible as possible for outside, without having to unmount anything. In particular, all power inputs to the ODrive/s are easily reachable, as well as its USB port. Basically all Raspberry Pi ports are accessible; these include USB ports, power inputs (both external and via USB-C), HDMI ports, etc .
The electronics is mounted on the plate using plastic screws and hex spacers.

The result is shown in the two top renderings of Fig. 7.16. In these renderings there are actually two ODrives mounted: this is because the electronics box would reasonably be the same also when adding legs to the prototype.

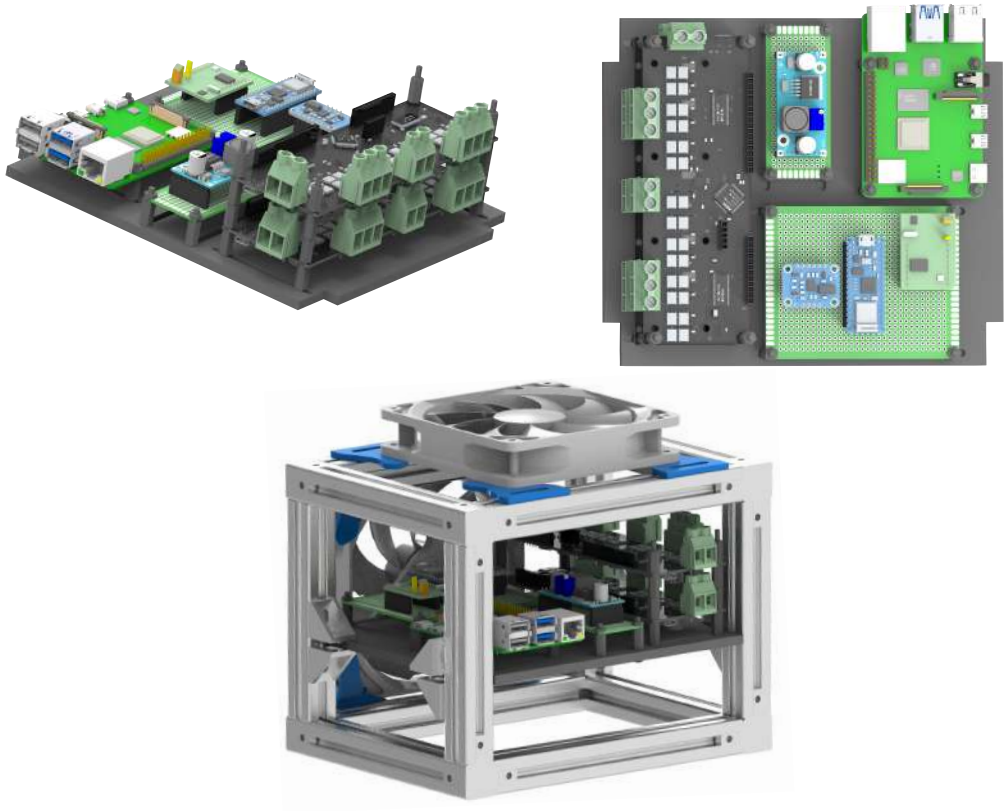


**Figure 7.16.** Synthetic renderings of prototype's electronics plate

Fig. 7.16 also shows the electronics box as it should look after mounting. For simplicity, connection cables and screws are not shown in these renderings. The cooling fans are mounted using simple 3D-printed couplers, visible in Fig. 7.16. On top of these fans, fan grids are mounted to protect the fragile blades from hitting

objects and people (these are not shown in CADs, but are clearly visible in Fig. 7.24).

### 7.7.4 Rendering of the assembled "no-knee" prototype

Finally, Fig. 7.17 shows a rendering of the no-knee prototype. Once again, wiring is not shown here for simplicity, but it is visible in Fig. 7.22, where a real photo of the prototype is shown.

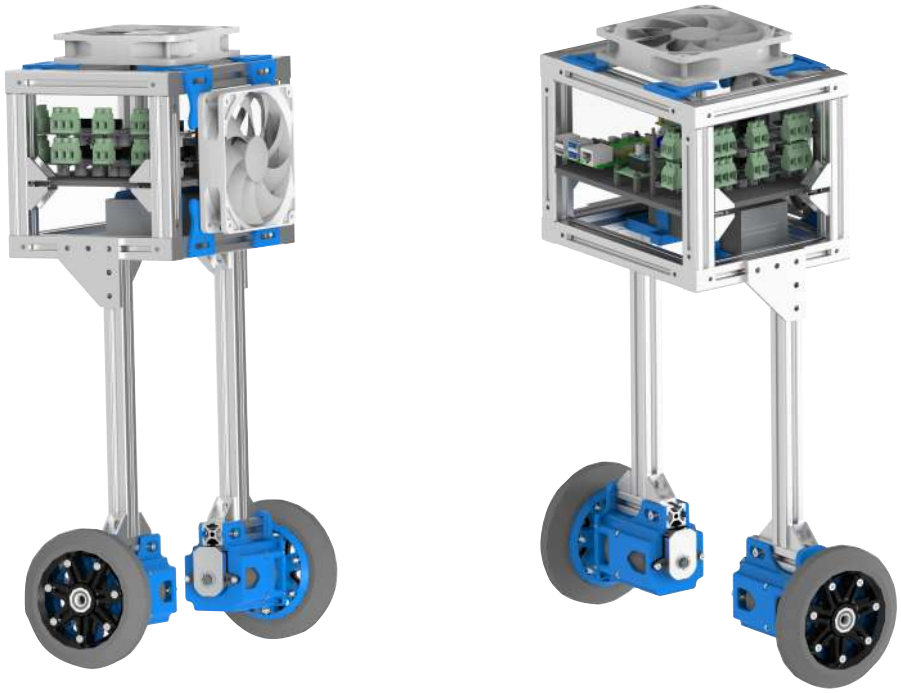


**Figure 7.17.** Synthetic renderings of the "no-knee" prototype

Note that these CAD models can be exploited to obtain an estimate of prototype's physical parameters like, for instance, the center of mass of each component. Clearly, due to a series of unavoidable measurement inaccuracies, those values will not be exact, but will be reasonably close.

## 7.8 Machining, assembling and wiring

This section is dedicated to a synthetic description of the processes of machining, assembling and wiring which were carried out in order to have a functional no-knee prototype.

First, all 3D-printed parts were manufactured using a modified version of an Ender 5 Pro (see Fig. 7.18), which is a standard cartesian FDM 3D printer. Technical specifications of the stock printer and a description of all applied modifications are not reported here, since they are out of the scope of this thesis.

The used material for printing is PETG, which is a relatively high-resistance

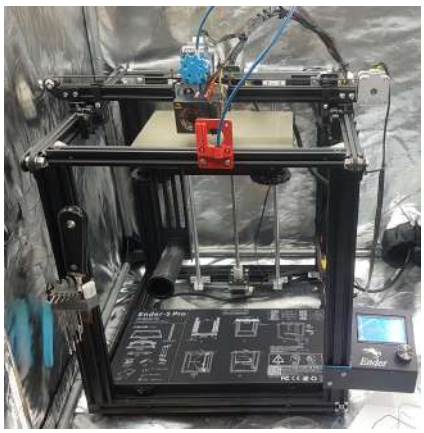


**Figure 7.18.** Used 3D printer (shown inside thermal enclosure)

alternative to the more common PLA. There could also be other alternatives, like for instance Nylon; however, this material in considerably more expensive and also difficult to print (it has a high printing temperature and requires a closed printing chamber). In particular, two colors were used: light blue and black. 3D-printed parts are clearly distinguishable looking at the renderings of Section 7.7.

Next, all electronics is connected using a variety of cables connectors which, in almost all cases, were manually wired.

Let us look at a list of the main wiring components and used tools:

- Thick AWG12 cables (see [46])for powering the ODrive and the actuators.
- Thin AWG24 cables for all other electronics. In particular, encoders cables were packaged inside a shielded copper tube to prevent electromagnetic disturbances.
- Insulating tape and shrink tubes.
- Two prototyping boards, one for supporting sensing electronics and the other for the DC-DC buck converter.
- Bullet connectors (for motors' phases), screw connectors, JST connectors, etc.
- Soldering iron, JST connectors crimper, multimeter for checking electrical connections, etc.

Another necessary step was to cut the aluminum profiles. For doing this, a miter saw was used(shown in Fig. 7.19).

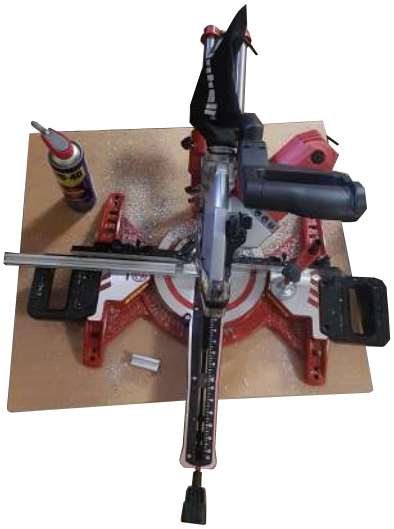


**Figure 7.19.** Used miter saw

The final assembling process required a large amount of screws, nuts and extrusion connectors. Furthermore, a supporting structure for holding the robot was also designed and build (see Fig. 7.24).

Let us look at some details of the prototype.
Fig. 7.20 is a picture of the assembled "ankle". On the left it is possible to look at encoder's shielded cable while, on the right, the three-phase cables coming out of the motor are clearly visible.

**Figure 7.20.** Prototype's assembled "ankle"

Fig. 7.21 shows two views of the complete motor enclosure; the motor, as well as the encoder, are clearly visible.

After having completed the entire electronics wiring process, the result is the one shown in Fig.7.22. Encoders' cables are clearly distinguishable in the foreground,

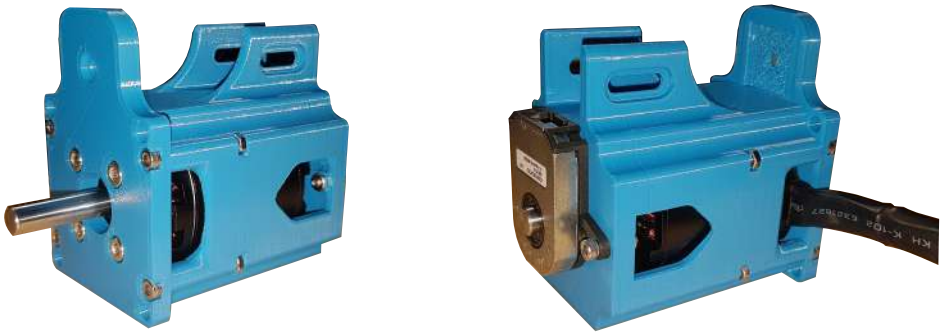


**Figure 7.21.** Front and back view of the assembled motor enclosure (with encoder)

as well as the green and yellow twisted cables of the two CAN buses, the three-phase cables coming from the motors and the input power cables (red and black ones). The Arduino Nano IoT is connected also to a USB cable, which can be used to program the microcontroller from the RPI itself or an external pc (same for the ODrive, but the USB cable is not visible in the photo).

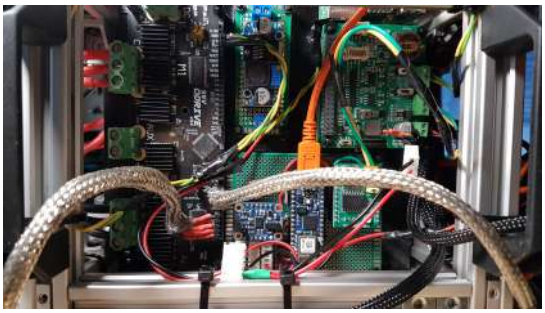


**Figure 7.22.** Wired electronics top view

Fig. 7.23 shows, on the left, a bottom view of the battery and its mounting 3D-printed plates and, on the right, a side view of the electronics box. In particular, cooling fan cables are easily distinguishable.

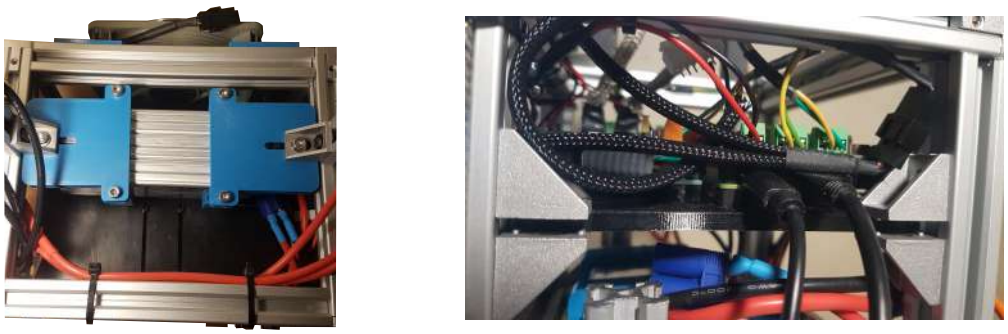


**Figure 7.23.** Bottom view of the battery (left) and side view of the electronics box

Fig. 7.24 shows some pictures of the fully assembled prototype, while it is sustained by a crane.

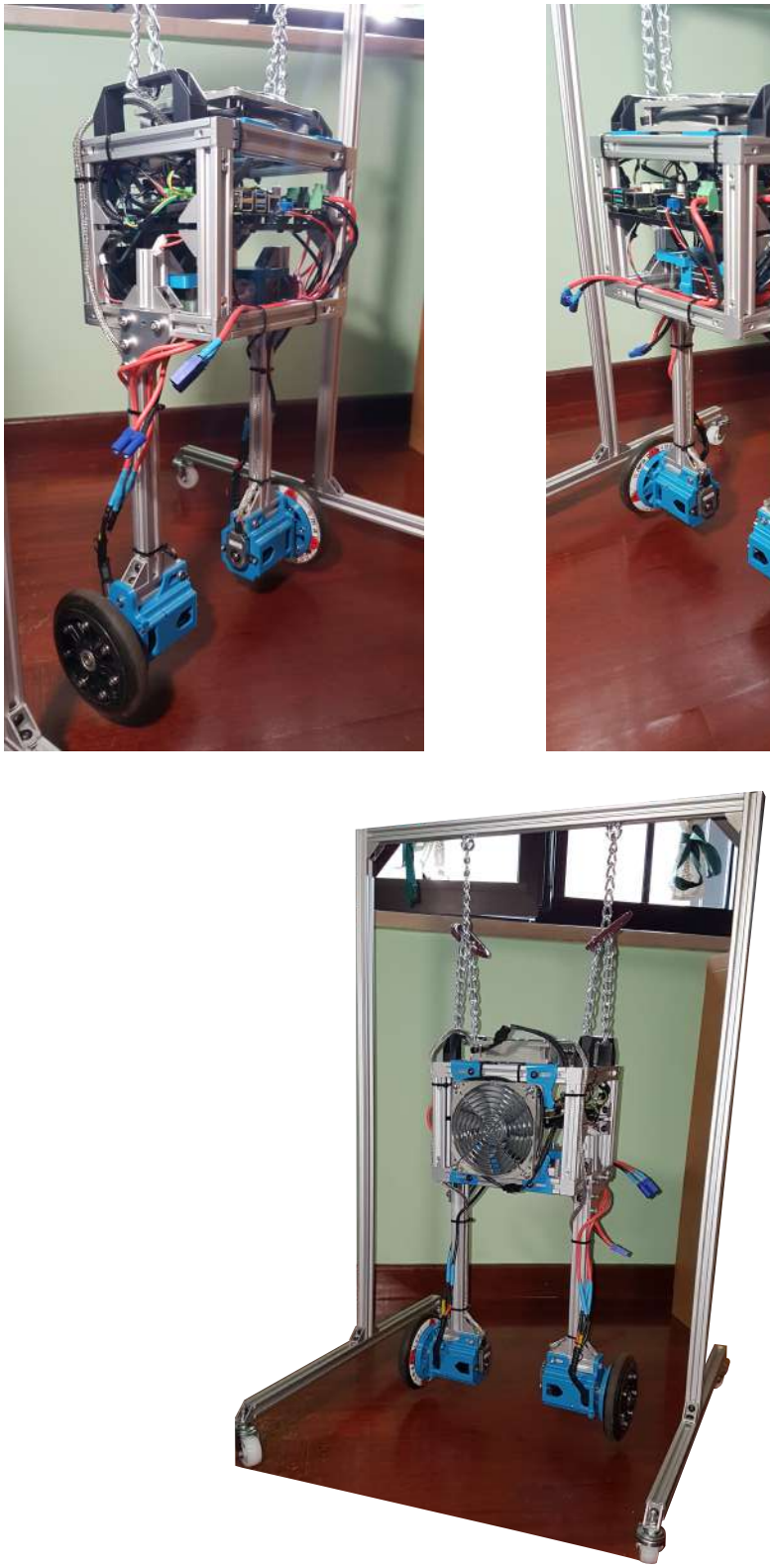


**Figure 7.24.** Completed "no-knee" prototype. At the top, two side-views of the assembled prototype sustained by a supporting structure. On the bottom a back-view of the same prototype, with supporting structure clearly visible.

## 7.9 Subsystems testing

After having discussed the main aspects of the prototype's physical implementation, it is possible to show some representative tests of its subsystems.

### 7.9.1 Testing the IMU

First, let us look at the used IMU, i.e. the Adafruit BNO055. To use these sensors, Adafruit offers a dedicated Arduino library (available at [1]).

As already stated, this IMU needs to be calibrated at each power up. The calibration status is indicated by an index for each hardware component from 0 to 3, where 0 stands for completely uncalibrated. At each component of the IMU, i.e. accelerometer, magnetometer and gyroscope is associated a particular calibration movement. For instance, the magnetometer calibration requires to perform 8-shaped movements, until the calibration status jumps to 3. Clearly, if the sensor is mounted inside a robot or in general an heavy object, this is not practical at all. One way to somehow avoid this, is to perform a full calibration with the sensor unmounted, read the calibration offsets (the library provides a method for doing exactly that), saving them somewhere and then restoring them on each bootup. Since the calibration is quite sensible to magnetism in the area, this procedure should be carried out every time the location is changed.

The orientation outputted by the sensor can be configured to be given either as a quaternion of the classical roll, pitch, yaw triad. Furthermore, the axis of the sensor can be configured by the user. If using Euler angles output, then they are provided in the order yaw-roll-pitch (see datasheet).
The units of measure are modifiable, but the default ones are $m/s^2$ for acceleration, $\mu$T for the magnetic field, rps for the angular rate, degrees for Euler angles and Celsius for the temperature. Moreover, each value is stored as a half precision floating point number (2 bytes), also called float16 by the IEEE standards. This means that, if using the CAN2.0a or b protocol, it is possible to send up to 4 values in a single message frame. Supposing a conservative bit rate over the CAN bus of 250Kbps, this means that sending one message would take $2.56 \cdot 10^{-4}$ seconds (around 0.3 ms). If using the maximum available bit rate, i.e. 1000Kbps, sending the message would take only 64$\mu$s.

Furthermore, the IMU has several possible operation modes. The one needed in this case is called NDOF, which provides acceleration, magnetometer, gyroscope and absolute orientation at a maximum rate of 100Hz. On top of these, temperature is available at 1Hz and there's also the possibility of measuring the gravitational acceleration.

Fig. 7.25 shows the full sensor readings at startup, when no calibration has been carried out. As a consequence, the global calibration index "Sys" is still at 0. Fig. 7.26 shows the results of a script which returns on the Serial Monitor the resulting calibration offsets after successfully completing the procedure.

These values can then be saved and reloaded at startup. Unfortunately, the Arduino Nano IoT (and also the BLE sense) does not have any EEPROM to store the calibration data between shutdowns. A simple solution to this problem is to copy the offsets manually into the main Arduino script.

```
Calibration: Sys=0 Gyro=0 Accel=0 Mag=2
--
Orient: x= 5.94 |       y= 0.25 |       z= -0.75
Rot:    x= -0.06 |      y= -0.06 |      z= 0.06
Linear: x= -0.02 |      y= 0.03 |       z= -0.12
Mag:    x= 17.19 |      y= 19.25 |      z= -33.38
Accl:   x= 0.02 |       y= 0.16 |       z= 9.67
Accl:   x= 0.05 |       y= 0.13 |       z= 9.80

temperature: 29

Calibration: Sys=0 Gyro=0 Accel=0 Mag=2
--
Orient: x= 5.94 |       y= 0.25 |       z= -0.75
Rot:    x= 0.19 |       y= -0.19 |      z= -0.06
Linear: x= 0.01 |       y= 0.06 |       z= -0.11
Mag:    x= 16.06 |      y= 13.00 |      z= -39.38
Accl:   x= 0.06 |       y= 0.19 |       z= 9.69
Accl:   x= 0.05 |       y= 0.13 |       z= 9.80

temperature: 29
```

**Figure 7.25.** Full sensor readings with calibration status

```
X: 134.25       Y: -29.44       Z: 178.00
        Sys:2 G:3 A:1 M:3
X: 133.94       Y: -30.56       Z: 178.00
        Sys:2 G:3 A:1 M:3
X: 133.31       Y: -31.50       Z: 177.94
        Sys:2 G:3 A:1 M:3
X: 133.19       Y: -32.19       Z: 177.94
        Sys:2 G:3 A:1 M:3
X: 133.12       Y: -32.69       Z: 177.94
        Sys:2 G:3 A:1 M:3
X: 133.19       Y: -32.88       Z: 177.94
        Sys:2 G:3 A:1 M:3
Accelerometer: 20 31 -9
Gyro: -1 -3 -1
Mag: -274 587 1008
Accel Radius: 1000
Mag Radius: 715
```

**Figure 7.26.** Obtained calibration offsets after successful calibration

Another option would be to use an external micro-SD, to be connected to the Arduino via a suitable shield.
Fig. 7.27 shows the results right after booting up the sensor with the restored offsets. All sensors are clearly fully calibrated.

For controlling the RPI, since the used operating system (Ubuntu Server 20.04) does not have by default any Desktop environment, the easiest way is to employ a SSH connection to the board. This is easily done, once both the RPI and the PC are connected to the same network and RPI's IP address is known.

```
Calibration: Sys=0 Gyro=3 Accel=3 Mag=3
--
Orient: x= 293.56 |     y= -1.44 |      z= -178.50
Rot:    x= 0.00 |       y= 0.13 |       z= -0.06
Linear: x= 0.00 |       y= 0.00 |       z= 0.05
Mag:    x= 54.50 |      y= -0.88 |      z= 33.19
Accl:   x= -0.25 |      y= 0.24 |       z= -9.74
Accl:   x= -0.25 |      y= 0.25 |       z= -9.80

temperature: 29

Calibration: Sys=0 Gyro=3 Accel=3 Mag=3
--
Orient: x= 293.56 |     y= -1.44 |      z= -178.50
Rot:    x= -0.13 |      y= 0.00 |       z= 0.06
Linear: x= 0.00 |       y= 0.00 |       z= 0.02
Mag:    x= 50.75 |      y= -5.00 |      z= 31.56
Accl:   x= -0.22 |      y= 0.21 |       z= -9.75
Accl:   x= -0.25 |      y= 0.25 |       z= -9.80

temperature: 29

Calibration: Sys=0 Gyro=3 Accel=3 Mag=3
--
```

**Figure 7.27.** Sensor readings right after boot up with calibration offsets restoration

### 7.9.2 Initialiting the RPI SPI-CAN hat

Another important operation, is the initialization of Seedstudio's RPI SPI-CAN hat.

This is done easily by opening the system file `/boot/firmware/usercfg.txt` and adding the line "dtoverlay=seeed-can-fd-hat-v2" to it. The successful initialization can then be verified with "dmesg | grep spi", as shown in the top image of Fig. 7.28. The bus can then be configured and powered on by running `sudo ip link set can0 up type can bitrate chosen_bitrate` in a shell terminal. The resulting bus configuration is shown in the bottom image of Fig. 7.28.

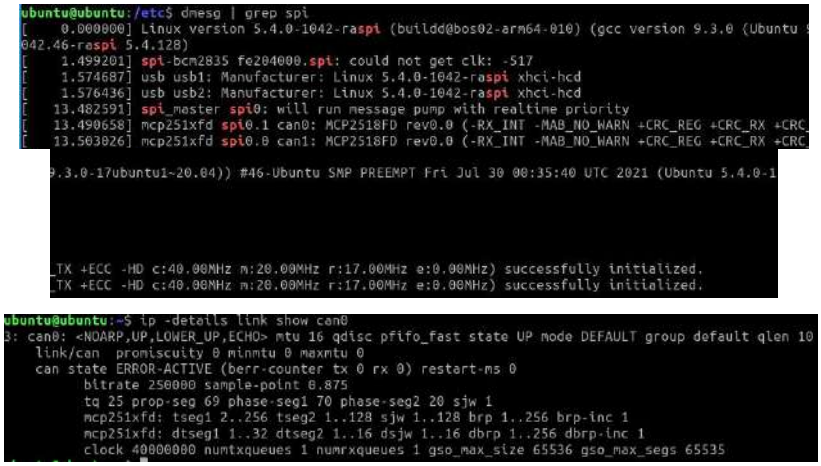



**Figure 7.28.** can0 bus initialization and example configuration

### 7.9.3 Setting up the ODrive

Next, it is necessary to setup both ODrive's axes for operation.

This can be done easily by powering up the board (through an external power source of a battery) and connecting it to a pc through USB. The configuration process can be done entirely using the so called "odrivetool" utility. This utility allows also for the control and calibration of the axes via terminal interface and can be particularly useful to test the whole actuator setup.
The used ODrive and axis0 settings are reported in Fig, 7.29 (axis1 settings are the same). The explicit meaning of each setting is not reported here for brevity.

```
odrv0.config.brake_resistance=0
odrv0.config.dc_max_negative_current=-1
odrv0.config.max_regen_current=100
odrv0.axis0.config.can.node_id = 0
odrv0.axis1.config.can.node_id = 1
odrv0.can.config.baud_rate = 250000

odrv0.axis0.motor.config.current_lim=10
odrv0.axis0.controller.config.vel_limit= 5
odrv0.axis0.motor.config.pole_pairs=7
odrv0.axis0.motor.config.torque_constant=8.27/270
odrv0.axis0.encoder.config.cpr=8192
odrv0.axis0.encoder.config.mode = ENCODER_MODE_INCREMENTAL
odrv0.axis0.encoder.config.calib_range = 0.05
odrv0.axis0.motor.config.calibration_current = 10.0
odrv0.axis0.motor.config.resistance_calib_max_voltage = 2.0
odrv0.axis0.encoder.config.pre_calibrated=True
odrv0.axis0.motor.config.pre_calibrated=True
```

**Figure 7.29.** Used ODrive settings

### 7.9.4 Testing CAN communications and additional tests

Having successfully set both the ODrive and the RPI, it is possible to start using the CAN bus (in particular can0 was used) for communicating between these two devices. It is fundamental to remember that all devices on a CAN bus must use the same bit rate (in particular, 250 Kbps was used).

By default, ODrive sends an "Heartbeat" messages from both axes with info on errors and states. It is hence possible to listen to this message from the RPI and verify the CAN bus is working properly. The result is shown in Fig. 7.30 (ODrive connected to can1 bus).
In particular:

- 0x001 is axis 0 heartbeat (10 Hz by default).
- 0x021 is axis 1 heartbeat (10 Hz by default).
- 0x009 is axis 0 encoder position (100 Hz by default).
- 0x029 is axis 1 encoder position (100 Hz by default).

Another test which was conducted on the ODrive is its control employing the so called ".dbc" files. This is a very convenient way of representing CAN databases offered by a Python library called "cantools". It allows for easy decoding/encoding

of messages and also generation of C code. The used .dbc file is based on ODrive's CANsimple protocol (see [29]`/docs`). Further details are omitted here for brevity.

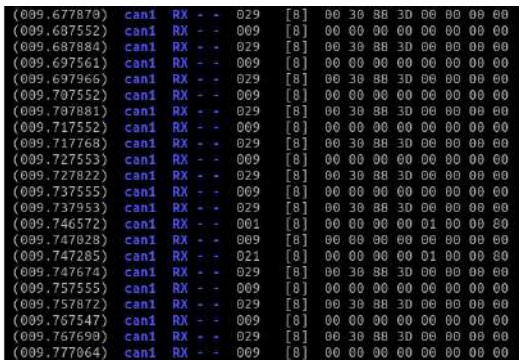



**Figure 7.30.** ODrive heartbeat as seen on the CAN bus (can1)

In particular, two tests were carried out, both using the position control mode.

The first one consists of performing a full calibration sequence, putting the axis in IDLE state and then tracking a sequence of steps (left image, Fig. 7.31).
The second one, tries to track a sinusoidal input signal (right image, Fig. 7.31).

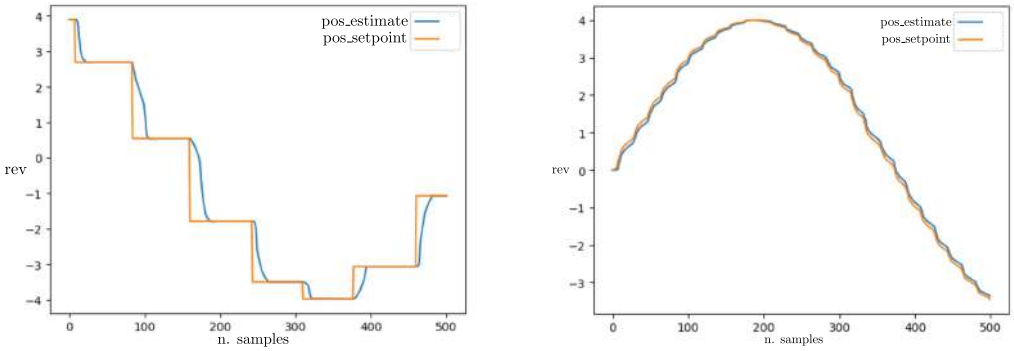



**Figure 7.31.** ODrive CAN tests employing database files

The tracking is not the best one achievable, since the velocity and position loops of the ODrive should have to be tuned. However, since the necessary control mode is torque control, tuning those loops is not necessary.

Additional test which were performed at an early stage using the odrivetool utility consisted of measuring the output torque at different torque input setpoint. For achieving this, a very simple setup was used. This is shown in Fig. 7.32. A 3D-printed arm of known length and a precision weight scale were used to test the resolution and capacity of the actuators. Given the quality of the experimental setup, one could not expect the most accurate results. However, after performing tests at various torque inputs, the results show a pretty good resolution (of the order of mN m) and a quite high torque capability (coherent with the motors' specifications).

Lastly, the only remaining test is to verify the connection of the sensing unit

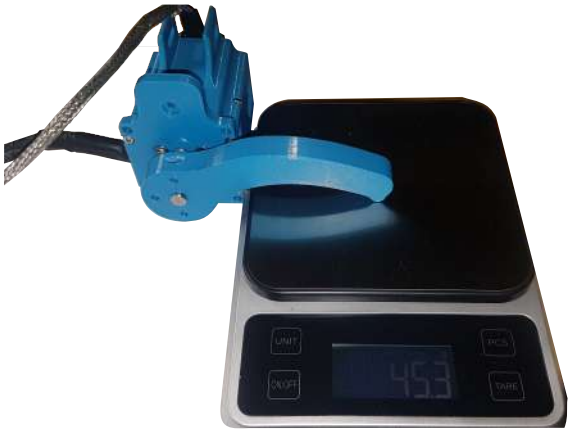



**Figure 7.32.** Torque test setup

on the CAN bus. As already specified, due to the unusual oscillator which the Mikroe CAN-SPI board mounts, it was necessary to compute the bit rate registers configuration and then add it manually to the used Arduino library (see [38]). Further details on these procedure are omitted here for simplicity.

Fig. 7.33 shows the received messages on the CAN bus, as seen by the RPI. In particular the used Arduino script simply sends a message with ID 0×000 and a data packet which is incremented by one bit at each loop iteration. The message is

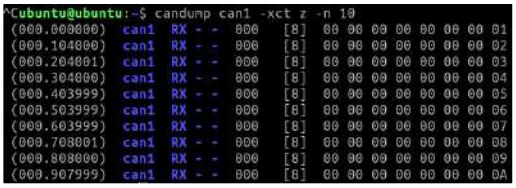



**Figure 7.33.** Arduino CAN bus connection test

sent at 10Hz and the used bit rate is 250 Kbps.

## 7.10 Software architecture

We saw in the previous section that all the subunits of the prototype are perfectly functional.

To be able to actually write and test control algorithms on the prototype, it is first necessary to develop a suitable software architecture. The software architecture should allow to:

1. Interact with the actuators (ODrive) for sending control commands, reading encoders and battery state.
2. Interact with the IMU, through the Arduino, for reading the orientation and angular velocity of the prototype.
3. Process the raw sensor data, if necessary.
4. Use high-level optimization tools for setting up advanced control strategies.
5. Ensure a reliable and stable control sample frequency.
6. Store on-board data for debugging.
7. Handle properly the CAN communication between all the subunits.
8. Interact remotely with the prototype, possibly exploiting the integrated WiFi connectivity of the RPI for sending, if necessary, high-level control directives to the robot and for offline post-processing of sensor data.

Let us see how the above enumerated requirements can be satisfied. At the time of writing, the software is not written yet.
Point 5 can be easily satisfied. In fact, the used RPI has 64 GB of available memory (part of it is used by the operating system) located in a micro-SD; this memory can be used to store the states of the robot, for example in one or more .txt files. This data can then be transferred to an external computer by manually copying them from the SD or, alternatively, by moving them remotely, for example, through an SSH connection.

Point 5 can become crucial when developing control strategies, especially when they have to be deployed on non-minimum phase systems like the one under study. Currently, the operating system which is installed on the RPI is Ubuntu 20.04 Server, which is a much lighter distribution compared with its full-Desktop version, and has no graphical interface, aside from a terminal. This choice goes towards the minimization of latencies and unnecessary overheads on processor usage and RAM consumption. However, latencies on full fledged operating systems can be even of the order of seconds, due to how the scheduler of the operating system handles threads. The operation of interrupting temporarily specific lower-priority processes is called *preemption*. This capability is crucial when working on real-time systems. Unfortunately, the standard Linux kernel is not *preemptable*. The solution is to apply a suitable patch to it. The best alternative is to use the the PREEMPT_RT patch, which can be installed relatively easily on the used operating system. With such an operation, it should be possible to bring latencies down to more than

acceptable values (100 µs or less). To install the *preemptable* version of the kernel, the easiest choice is to employ [18], which provides pre-compiled kernels through .deb packages. This makes the installation of the kernel really easy; details are here omitted for brevity. In particular, at the time of writing, the installed fully preemptable kernel is `Linux ubuntu 5.10.73-rt54-v8+`. To appreciate the impact of this patch, latencies were measured using a test available at [9]; the resulting minimum latency was found to be 6 µs, while the maximum turned out to be 56 µs, which are even better than the values reported at [24]. Clearly, these are still not enough to make the RPI a true real-time system, but it shoud be enough for the purposes of this thesis.
This only solves , however, part of the issue. The other important aspect is that a sufficiently precise clock/counter should be available for timing properly the control loop. The RPI, by default, does not have any RTC (real time counter). It is however possible to add it using the available RTC mounted on the shield used for the CAN communication. Alternatively, it is also possible to use the clock of the network (if the RPI is connected to it). A further alternative is to employ the used microcontroller, i.e. the Nano IoT, to provide the control timing to the RPI. In fact, this microcontroller has a 32 bit RTC, up to five 16 bit timer/counters and up to four 24 bit timer/counters. Furthermore, if using this last option, suitable libraries for executing parallel loops and handling efficiently timer interrupts are available and can be exploited to ensure a precise timing of the control loop.

The use of a full-fledged operating system like Ubuntu 20.04 Server also addresses point 3 and 4. In fact, all the employed high level C++ libraries which were used to write the controllers on Gazebo will also be available on the RPI.

Points 1, 2 and 7 are actually closely related. The commands to ODrive can be sent through the associated CAN bus employing the .dbc database file generated using the CANSimple protocol. Similarly, a very similar approach can be used to interact with the IMU. On the RPI side, this is simply done by establishing a suitable message packaging for the IMU data. On the microcontroller side, message coding and decoding has to be written almost completely manually.

Finally, point 8 can be addressed in multiple ways. Since it will be necessary to mix both Python (for using .dbc files) and C++ code (for using efficient optimization libraries), while also allowing a remote connection/interaction with the robot, one viable alternative is to integrate the whole software inside the ROS framework. ROS stands for “Robotics Operating System” and it is “an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers”. For detailed information on the concepts behind ROS, refer to [30]. Details on the theory behind ROS are here omitted for brevity. Actually, since ROS was not built accounting for real-time performance requirements, it might be better to explore the possibility of using ROS2, the latest version of ROS; apparently, this new version was developed completely from scratch, with real-time performance in mind and also offers many improvements over the older version. The documentation of ROS2 is available at [31].

## 7.11 Current prototype status and ToDo

At the time of writing, the "no-knee" prototype is fully assembled and all its subunits have been tested.
Things which still need to be addressed are:

- Designing the actuated legs and sourcing suitable knee actuators. In particular, the design of the joints can be carried out using the approximate expressions of Section 3.3.
  Let us substitute $\left[2\left(m^w + m_{ah} + m^d + m^c\right) + m^{body}\right] g/2$ for $F_1$ and $F_2$ and plot the approximate joint reactions over the range of $\beta$. The result is shown is Fig. 7.34.

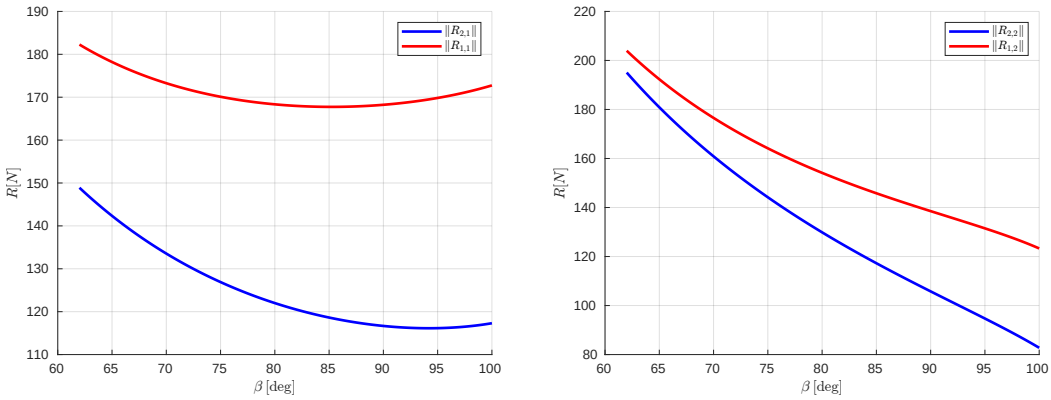



**Figure 7.34.** Approximate joint reactions (in modulus) during vertical balancing. On the left, the case of vertical balancing using only $C_{m2}$ while, on the right, the case of balancing with only $C_{m3}$.

The left figure is relative to the case of balancing when using only $C_{m2}$, while the right one depicts the case of balancing using only $C_{m3}$. Note that only two joint reactions are plotted for each figure simply because these are only two independent reactions (see Figures 3.4 and 3.5 for interpreting these graphs).

- Writing the software following the guidelines outlined in Section 7.10. This includes the integration into the ROS/ROS2 framework.

# Chapter 8

# Conclusions and future work

This thesis focused on the study and control of a particular class of systems, i.e. wheeled inverted pendulum ones.

The first step, after a brief review of the state of the art, was to obtain suitable planar dynamical models of the system (Chapter 3). This was done employing a Lagrangian formulation of the equations of motion. Multiple models were used. Initially, a complete 2D model of the system valid during ground contact and a 2D equivalent model (addressed with "eWSLIP") were derived. Later on, more general floating-base models of both systems were derived.

After having derived all the models, a brief analysis of the used simulation environments was presented (specifically, in Chapter 4). In particular, two simulators were employed: a self-written Matlab simulator and Gazebo. The latter was introduced towards the end of the thesis as a way of simulating control strategies on a high fidelity simulator. This open source robotics simulator is far from perfect: the contact performance is quite difficult to tune and the simulator, at the time of writing, has some problems with robots equipped with closed kinematic chains, like the one under study. This problems include, for example, the inability to set initial conditions on the linkage directly through Gazebo API and some strange issues appearing when reading joint derivatives. Nonetheless, it is still a very good alternative for simulation, being it free and open source. Gazebo11 is the latest and the last version of Gazebo. The future, apparently, will be Ignition Gazebo, which is already available, but not mature yet.

Next, a relatively detailed analysis of some relevant control properties of the system was carried out (see Chapter 5.4). The feedback-linearization technique was introduced, as well as the second order normal form of a system. Furthermore, concepts like the internal and zero dynamics were discussed. The equilibrium points of the system were also studied and its properties around those points (linear controllability, linear observability, local relative degree, etc..) were analyzed. A linearization of the full dynamics around a general trajectory was also obtained. Finally, the second order normal forms, the internal and zero dynamics of the contact and floating-base model were derived.

All the studied properties and concepts served as a base for writing and testing suitable control strategies for trajectory tracking, which was the main focus of

Chapter 6. In particular, this chapter contains the development and testing of three separate control approaches, two of which were tested on both Gazebo ad Matlab. The first and also most crude approach employs a classical LQR, combined with some results from the Linear Stable Inversion on a simple WIP (borrowed from [17])) and allows to perform state trajectory tracking of relatively complex trajectories (with some limitations) and also to control the height of the CoM. The whole scheme relies heavily on a linearization around the reference trajectory and employs approximate results from Linear Stable Inversion; nonetheless, it showed good tracking performance over a wide range of trajectories.
The second approach employs a more advanced IS-MPC (Intrinsically Stable-MPC) controller which contains, at its core, a constrained quadratic optimization (discrete) program. The peculiarity of this approach is the use of a boundness constraint within the optimization scheme which avoids, to some extent, the divergence of the internal dynamics of the system; this allows to completely remove the problem of generating bounded reference trajectories for the internal dynamics. Furthermore the MPC naturally adapts, due to its intrinsic discreteness, to deployment on hardware. One disadvantage of this approach, if implemented as described in this thesis, is that it heavily relies on a model of the system. System uncertainties, unmodeled dynamics and disturbances may decrease its performance considerably. In the performed simulations, this approach shows good tracking results.
The third approach uses a so-called Task-Space tracker. Differently to the other schemes, this one has no predictive characteristic and uses a constrained QP and a tracking controller to compute, at each sample time, suitable control inputs, with the aim of tracking a given Task. In the performed simulations, this task was chosen to be a relative cartesian CoM trajectory. This approach shows very good performance and has the advantage of also providing, if set up properly, estimates of the ground reaction forces.

Finally, in Chapter 7 a synthetic description of the current "no-knee" prototype is inserted. Note that, even if the implementation only occupies one chapter, the related work was actually one of the most time-consuming and also profitable in terms of acquired skills. As already stated, the implementation of a full-fledged prototype with knee was not possible due to time and resource limits.
This chapter contains a relatively detailed description of the chosen hardware components, including actuators, sensing, control and communication electronics, a description of the wiring design, the machining and the testing of all the main subunits.

There are many possible future developments. These include:

- The derivation of a complete 3D floating-base model of the system for performing more complex navigation tasks.
- Testing control policies for performing jumping motion. One possible approach was already tested on the 2D model but non included in this thesis since there was no time of exploring it adequately.
- Completing the software architecture as described in Section 7.11. These include setting up properly the software on the prototype and designing the

leg.

- Testing of control policies on the "no-knee" prototype and evaluation of the appropriateness of the chosen hardware.
- Testing of control policies for performing navigation and jumping motions on the complete prototype.

# Bibliography

[1] Adafruit. Adafruit Unified BNO055 Driver. Available from: https://github.com/adafruit/Adafruit_BNO055.

[2] Adafruit. Adafruit BNO055. Available from: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor.

[3] Arduino. Arduino open source electronics platform. Available from: https://www.arduino.cc/.

[4] Blender Foundation. Blender. Available from: https://www.blender.org/foundation/.

[5] Bosch Sensortec. BNO055. Available from: https://www.bosch-sensortec.com/products/smart-sensors/bno055/.

[6] Bouyarmane, K., Chappellet, K., Vaillant, J., and Kheddar, A. Quadratic programming for multirobot and task-space force control. *IEEE Transactions on Robotics*, **35** (2019), 64. doi:10.1109/TRO.2018.2876782.

[7] Bullet Physics development team. Bullet. Available from: https://github.com/bulletphysics.

[8] Chen, H., Wang, B., Hong, Z., Shen, C., Wensing, P. M., and Zhang, W. Underactuated motion planning and control for jumping with wheeled-bipedal robots. *IEEE Robotics and Automation Letters*, **6** (2021), 747. doi:10.1109/LRA.2020.3047787.

[9] Clark Williams. RT test utils. Available from: https://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git.

[10] CUI devices. AMT102-V. Available from: https://www.cuidevices.com/product/motion/rotary-encoders/incremental/modular/amt10-series.

[11] Devasia, S., Chen, D., and Paden, B. Nonlinear inversion-based output tracking. *IEEE Transactions on Automatic Control*, **41** (1996), 930. doi:10.1109/9.508898.

[12] Dinev, T., Xin, S., Merkt, W., Ivan, V., and Vijayakumar, S. Modeling and control of a hybrid wheeled jumping robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2563–2570 (2020). doi:10.1109/IROS45743.2020.9341339.

[13] Georgia Tech and Carnegie Mellon University. Dart. Available from: `http://dartsim.github.io/`.

[14] Gianluca Frison. hpipm. Available from: `https://github.com/giaf`.

[15] Hartenberg, R. S. and Denavit, J. *Kinematic Synthesis of Linkages*. McGraw-Hill, New York (1964).

[16] Isidori, A. *Nonlinear Control Systems*. Springer-Verlag, Berlin, Heidelberg, 3rd edn. (1995). ISBN 3540199160.

[17] Kanneworff, M. *Stable trajectory control for wheeled inverted pendulum vehicles*. Master thesis, La Sapienza Università di Roma (2016).

[18] Kevin Doren. PREEMPT RT kernel for Raspberry Pi. Available from: `https://github.com/kdoren/linux/releases/`.

[19] Klemm, V., et al. Ascento: A two-wheeled jumping robot. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 7515–7521 (2019). `doi:10.1109/ICRA.2019.8793792`.

[20] Kvaser. Can bus bit timing calculator. Available from: `https://www.kvaser.com/support/calculators/bit-timing-calculator/`.

[21] Lanari, L. and Hutchinson, S. Inversion-based gait generation for humanoid robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1592–1598 (2015). `doi:10.1109/IROS.2015.7353580`.

[22] Lanari, L., Hutchinson, S., and Marchionni, L. Boundedness issues in planning of locomotion trajectories for biped robots. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 951–958 (2014). `doi:10.1109/HUMANOIDS.2014.7041478`.

[23] Li, X., Zhou, H., Feng, H., Zhang, S., and Fu, Y. Design and experiments of a novel hydraulic wheel-legged robot (wlr). In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3292–3297 (2018). `doi:10.1109/IROS.2018.8594484`.

[24] Mauro Riva. Raspberry Pi 4B: Real-Time system using Preempt-RT. Available from: `https://lemariva.com/blog/2019/09/raspberry-pi-4b-preempt-rt-kernel-419y-performance-test`.

[25] McMASTER-CARR. Mcmaster-carr. Available from: `https://www.mcmaster.com/`.

[26] MikroElektronika. CAN SPI click 3.3V. Available from: `https://www.mikroe.com/can-spi-33v-click`.

[27] ODrive Robotics. D6374 150 KV. Available from: `https://odriverobotics.com/shop/odrive-custom-motor-d5065`.

[28] ODrive Robotics. ODrive. Available from: `https://odriverobotics.com/`.

[29] ODrive Robotics. ODrive (GitHub). Available from: `https://github.com/odriverobotics/ODrive`.

[30] Open Robotics. Robot Operating System. Available from: `http://wiki.ros.org/`.

[31] Open Robotics. Robot Operating System. Available from: `https://docs.ros.org/en/foxy/index.html`.

[32] Open Source Robotics Foundation. Gazebo. Available from: `http://gazebosim.org/`.

[33] Open Source Robotics Foundation. SDFormat. Available from: `http://sdformat.org/`.

[34] Raspberry Pi foundation. RPI 4B. Available from: `https://www.raspberrypi.org/products/raspberry-pi-4-model-b/`.

[35] Russ Smith. Ode. Available from: `https://www.ode.org//`.

[36] Scianca, N., Cognetti, M., De Simone, D., Lanari, L., and Oriolo, G. Intrinsically stable mpc for humanoid gait generation. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 601–606 (2016). `doi:10.1109/HUMANOIDS.2016.7803336`.

[37] Seeed Technology Co. 2 Channel CAN BUS FD Shield for Raspberry Pi. Available from: `https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor`.

[38] Seeed Technology Co. Seeed Arduino CAN. Available from: `https://github.com/Seeed-Studio/Seeed_Arduino_CAN`.

[39] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. *Robotics: Modelling, Planning and Control*. Springer (2010). ISBN 1849966346.

[40] Siemens. Solid edge. Available from: `https://solidedge.siemens.com/it/`.

[41] Spong, M. Partial feedback linearization of underactuated mechanical systems. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, vol. 1, pp. 314–321 vol.1 (1994). `doi:10.1109/IROS.1994.407375`.

[42] Spong, M., Hutchinson, S., and Vidyasagar, M. Robot modeling and control. *IEEE Control Systems*, **26** (2006), 113. `doi:10.1109/MCS.2006.252815`.

[43] Stanford University Bioengineering. Simbody. Available from: `https://github.com/simbody/simbody`.

[44] Stilman, M., Wang, J., Teeyapan, K., and Marceau, R. Optimized control strategies for wheeled humanoids and mobile manipulators. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pp. 568–573 (2009). `doi:10.1109/ICHR.2009.5379514`.

[45] Stratasys. Grabcad. Available from: `https://grabcad.com/`.

[46] Wikipedia. American wire gauge. Available from: `https://it.wikipedia.org/wiki/American_wire_gauge`.

[47] Xin, S. and Vijayakumar, S. Online dynamic motion planning and control for wheeled biped robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3892–3899 (2020). `doi:10.1109/IROS45743.2020.9340967`.

[48] Zambella, G., et al. Dynamic whole-body control of unstable wheeled humanoid robots. *IEEE Robotics and Automation Letters*, **4** (2019), 3489. `doi:10.1109/LRA.2019.2927961`.