SAPIENZA
UNIVERSITÀ DI ROMA

Elective in Robotics – Underactuated Robots

Academic Year 2020/2021

Modeling and Control of a Hybrid Wheeled Jumping Robot

Master's Degree in Artificial Intelligence and Robotics

Rossella Bonuomo, 1923211

Marco Pennese, 1749223

Veronica Vulcano, 1760405

Contents

1 Introduction

In this work, we apply a Model Predictive Controller to a wheeled robot with a prismatic extension joint, which can be considered a simple **wheeled-legged system**. Such systems are able to move across rough terrains, jump over obstacles and across gaps, while maintaining the fast motion of wheeled systems at the same time. In fact, they can combine the benefits of both mobile robots and legged systems.

After showing the system dynamics, we propose a scenario in which the robot should swing-up and balance, drive upright and finally jump over a gap under the control of the MPC. We also consider a PD controller in order to make a comparison between the two approaches.
In the latest section, we include noise in sensor's reading so to test the robustness of the proposed approaches.

For all the details concerning the code and simulation results this is the link: `https://github.com/rbonuomo/UR_HybridWheeledJumpingRobot.git`.

2 Model

The wheeled jumping robot is essentially a car with four wheels. The front wheels are actuated and the peculiarity of this robot is its prismatic joint that allows to increase and reduce the body length.
In the starting pose the robot is placed horizontally with respect to the floor, with all the wheel touching the ground.

We divided the simulation in two phases:

- Phase 0: the robot has to swing-up, by braking and accumulating kinetic energy. In this phase it has a starting velocity different from zero that will be used to accumulate energy.
- As soon as the robot velocity is equal to 0, we switch our controller to the proposed wheeled-legged system [1] to be controlled with MPC.

We have also implemented a simplified dynamic model that will be used when we do not take into account the flying phase. In particular, we are going to use it to make the comparison between the MPC and the PD controller.

2.1 Phase 0

As anticipated, the structure of the robot is essentially a car with four wheels. During the first phase of the simulation, the goal of the robot is to slow-down and swing-up by applying torque to the front wheels. The robot starts with constant velocity equal to $\dot{x} = 7m/s$.

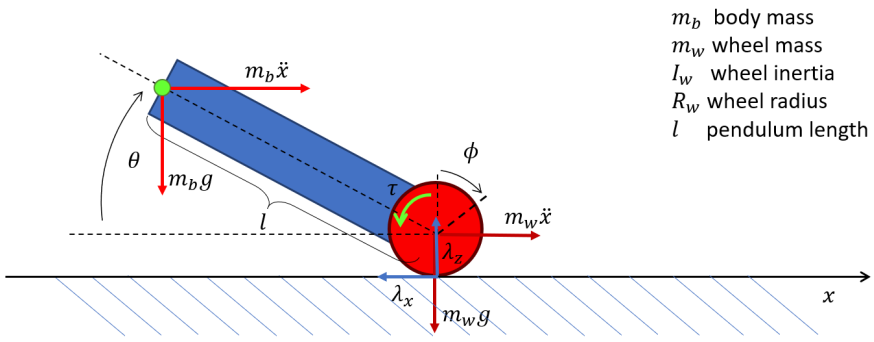

Figure 1: The braking car model

In this phase, we constrain l to be constant; therefore, the model simplifies. For this reason, we can consider as input only the torque acting on the wheel by ignoring the force that will be used to extend

and retract the prismatic joint.

By analysing the forces and the torques applied to the car we can derive its dynamic model:

$$\begin{cases} \ddot{\phi} = \frac{\tau}{(I_w + m_t R_w^2)} \\ \ddot{\theta} = \frac{1}{m_b l^2}(-m_b l \sin\theta R_w \ddot{\phi} + m_b g l \cos\theta - \tau) \end{cases} \tag{1}$$

where, ϕ is the angle of rotation of the wheel, θ is the angle of the car with respect to the ground, l is the length between the front wheel and the center of mass of the body, m_w is the mass of the wheel, m_b is the mass of the body, $m_t = m_w + m_b$ is the total car mass, R_w is the radius of the wheel, I_w is the inertia of the wheel, τ is the control input which consists in the torque applied by the wheel.

It is easy to understand that, if the car does not slide, we can also determine the coordinate x as $x = R_w\phi$.
We set as goal $\dot{\phi} = 0$ so, when the car starts with velocity equal to $7m/s$, torque will be provided to the wheel in order to slow-down. This situation tends to bring up the back of the car so that, when the car completely stops, we have both θ and $\dot{\theta}$ different from zero.

Once we have reached the target, we can switch to the Variable-Length Wheeled Inverted Pendulum that will represent our wheeled-legged mobile robot.

2.2 Variable-Length Wheeled Inverted Pendulum

A Wheeled Inverted Pendulum (WIP) consists of a wheel and a pole, which is modeled as a point mass m_b at a certain distance from the wheel. In this work, the dynamic model is an extension of the WIP. In fact, a prismatic joint is included so as to "mimic" the capability of a leg; therefore, the distance of the point mass from the wheel is not fixed anymore. The overall system is called Variable-Length Wheeled Inverted Pendulum (VL-WIP).

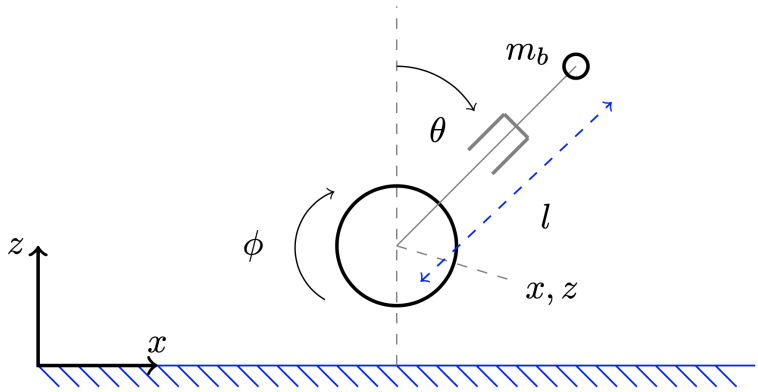

Figure 2: The VL-WIP model

The vector of generalized coordinates is $\mathbf{q} = [x, z, \phi, l, \theta]^T$, where x and z are the coordinates of the center of the wheel, ϕ is the wheel's angle along its axis, l is the distance from the center of the wheel to the body point mass and θ is the rotation of the body from the z-axis. The control inputs are $\mathbf{u} = [\tau, f]^T$, where τ is the actuation torque of the wheel and f is the linear force in the prismatic joint.

The dynamics of the system is derived with the Lagrangian method:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{S}^T\mathbf{u} + \mathbf{J}_C^T\boldsymbol{\lambda}$$

$\mathbf{M}$ is the mass matrix:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_t & 0 & 0 & m_b \sin\theta & m_b l \cos\theta \\ 0 & m_t & 0 & m_b \cos\theta & -m_b l \sin\theta \\ 0 & 0 & I_w & 0 & 0 \\ m_b \sin\theta & m_b \cos\theta & 0 & m_b & 0 \\ m_b l \cos\theta & -m_b l \sin\theta & 0 & 0 & m_b l^2 \end{bmatrix}$$

$\mathbf{C}$ is the Coriolis matrix:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} 0 & 0 & 0 & 2m_b \cos\theta\dot{\theta} & -m_b l \sin\theta\dot{\theta} \\ 0 & 0 & 0 & -2m_b \sin\theta\dot{\theta} & -m_b l \cos\theta\dot{\theta} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -m_b l\dot{\theta} \\ 0 & 0 & 0 & 0 & -m_b l\dot{\theta} \\ 0 & 0 & 0 & 2m_b l\dot{\theta} & 0 \end{bmatrix}$$

$\mathbf{G}$ is the gravity vector:

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} 0 & gm_t & 0 & gm_b \cos\theta & -gm_b l \sin\theta \end{bmatrix}^T$$

Where $m_t = m_b + m_w$ is the total mass of the system, which is given by the sum of the body mass m_b and of the wheel mass m_w, while I_w is the inertia of the wheel.

$\mathbf{S}$ is a selection matrix that translates controls into the generalized coordinates of the system:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & \end{bmatrix}$$

$\mathbf{J}_C^T$ is the contact Jacobian that relates the velocity of the contact point to the generalized velocities of the robot:

$$\dot{\mathbf{r}}_C = \mathbf{J}_c \dot{\mathbf{q}} = \begin{bmatrix} 1 & 0 & -R_w & 0 & 0 \\ 0 & 1 & 0 & 0 & \end{bmatrix} \dot{\mathbf{q}}$$

Where R_w is the radius of the wheel.

Finally, $\boldsymbol{\lambda} = [\lambda_x, \lambda_z]$ are the contact forces acting on the x and z directions.

3 Controls

The main control strategies explored in this work are Model Predictive Control and Proportional-Derivative (PD) controller.

3.1 Model Predictive Control

Model Predictive Control includes a family of control methods that allow to achieve optimal performance while satisfying constraints. It finds the optimal control action by solving an optimal control problem (OCP) over a *finite prediction horizon*. The cost function is minimized taking into account the process dynamics along the horizon.

The goal is to control the system during different phases:

- Swing-up and Balance: the robot has to switch from a driving mode on four wheels to a balancing mode on two wheels.
- Driving upright: in the upright modality, the robot has to balance and to drive forward.
- Jumping over a gap: the robot has to overcome a gap by jumping over it. In the jumping phase, we recognize three different stages. The first is the pre-takeoff in which the robot is ready to leave the ground. Then, there is the flight phase and finally the post-touchdown one in which the robot has to keep balancing once it has come back in contact with the ground.

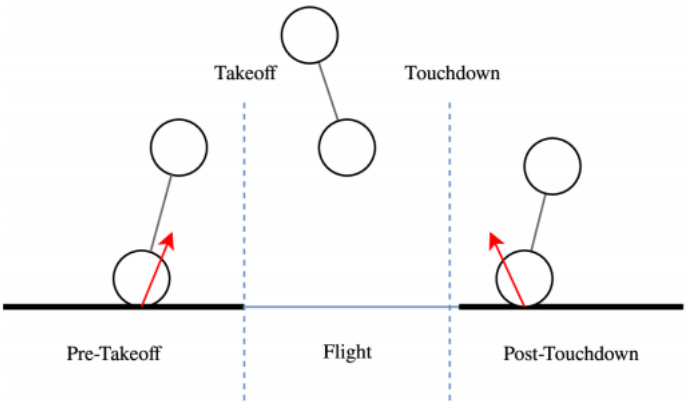

Figure 3: Jumping over a gap

This is the general problem formulation:

$$\begin{aligned}
&\min_{X,U,\Lambda} \sum_i (\mathbf{x}_i^T Q \mathbf{x}_i + \mathbf{u}_i^T U \mathbf{u}_i) + \mathbf{x}_N^T P \mathbf{x}_N \text{ (Cost function)} \\
&\qquad\qquad s.t. \\
&x_0 = x_0^* \text{ (Start state)} \\
&x_N = x_N^* \text{ (Goal state)} \\
&x_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \lambda_t), t \in [0, N] \text{ (Dynamics)} \\
&\\
&\mathbf{x}^- \leq \mathbf{x}_t \leq \mathbf{x}^+ \, t \in [0, N] \text{ (State Bounds)} \\
&\mathbf{u}^- \leq \mathbf{u}_t \leq \mathbf{u}^+ \, t \in [0, N-1] \text{ (Control Bounds)} \\
&\\
&\text{if } \; t \in [T_{tf}, T_{td}] \text{ (Flight Phase)} \\
&\boldsymbol{\lambda}_t = \mathbf{0} \text{ (No Ground Force)} \\
&\\
&\text{if } t \notin [T_{tf}, T_{td}] \text{ (Ground Phases)} \\
&|\boldsymbol{\lambda}_t^x| \leq \mu \boldsymbol{\lambda}_t^z \text{ } (\mu : \text{ friction coefficient}) \text{ (Friction Cone)} \\
&\boldsymbol{\lambda}_t^z \geq 0 \text{ (Unilateral Force)} \\
&\\
&\dot{x}_t = R_w \dot{\phi}_t \text{ (No Slip on Ground)}
\end{aligned} \tag{2}$$

We have formulated a nonlinear optimization problem whose aim is to minimize a cost function while satisfying a set of constraints. We specify the start and goal state and then the state and control bounds that are enforced by physical feasibility of the generated motion.
Then, we specify constraints related to the flight phase and ground phase after having decided the duration of each phase; this is done setting the time of takeoff T_{tf} and the time of touchdown T_{td}. Contact forces must be zero during the flight phase while they must be inside the friction cone and have a positive vertical component during the ground phase. Finally, we set a no-slip constraint for the robot when it is on the ground.
The solution of the problem provides a sequence of optimal control actions: $u^*_{t:t+N-1|t} = [u^*_{t|t}, .., u^*_{t+N-1|t}]$. Anyway, we just apply the first one $u(t) = u^*_{t|t}(x_t)$ and then repeat the process.

Concerning the **code**, at first, we define the model. We have considered two different models, so we are going to define both of them. In the file *CarModelPhase0.py*, first of all we define the model parameters, and the state and control variables. These are:

$$X = \begin{bmatrix} \phi & \theta & \dot{\phi} & \dot{\theta} \end{bmatrix} \tag{3}$$

$$U = \tau \tag{4}$$

Then we define the dynamics in an implicit and explicit way as we have seen in 2.1. To build the discrete version of the model we use Runge-Kutta of 4^{th} order. We do the same also for the Variable-Length Wheeled Inverted Pendulum model. In the file *CarModel.py*, we set the state and controls that are:

$$X = \begin{bmatrix} x & z & \phi & l & \theta & \dot{x} & \dot{z} & \dot{\phi} & \dot{l} & \dot{\theta} \end{bmatrix} \tag{5}$$

$$U = \begin{bmatrix} \tau & f & \lambda_x & \lambda_z \end{bmatrix} \tag{6}$$

and we define all the matrices that are needed to define the dynamic model. The latter is then expressed in an explicit and implicit way. Also in this case, we use Runge-Kutta of 4^{th} order.

In the *main_total.py* file, we define the problems to be solved with the MPC controller. In fact, we have implemented two different optimization problems to be solved with the MPC.
The first uses the model in 2.1 to drive the four-wheel robot from a velocity of $\dot{x} = 7\, m/s$ up to a velocity of $\dot{x} = 0\, m/s$. This phase lasts less than one second. Regarding the state bounds for the MPC constraints, we set the following values:

$$X_{lim} = \begin{bmatrix} -\infty & 0 & -\infty & -\infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

In fact, $z^- = 0$ because otherwise the robot would sink into the ground. For the control inputs we set:

$$u_{lim} = \begin{bmatrix} -10 \\ 10 \end{bmatrix}$$

Once we reach the target velocity, we switch to the model in 2.2 and we solve another optimization problem.
Each phase is characterized by different constraints. In the ground phase before flying we have:

$$X_{lim}^{pre-tf} = \begin{bmatrix} -\infty & R_w & -\infty & 2R_w & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ g^- & R_w & \infty & 1+2R_w & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

During the flight phase we have:

$$X_{lim}^{flight} = \begin{bmatrix} -\infty & R_w & -\infty & 2R_w & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ \infty & \infty & \infty & 1+2R_w & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

and finally, once the robot has come back in contact with the ground after flying, we set:

$$X_{lim}^{post-td} = \begin{bmatrix} g^+ & R_w & -\infty & 2R_w & -\pi/2 & -\infty & -\infty & -\infty & -\infty & -\infty \\ \infty & R_w & \infty & 1+2R_w & \pi/2 & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Where $g^- = 1.5$ and $g^+ = 2$ are the bounds of the gap in the x direction. We can notice that z cannot be less than R_w because of the presence of the ground, while it can be greater than R_w when the robot has to fly. l is bounded to be always between two values that we choose, while θ has to be between $-\pi/2$ and $\pi/2$.

Similarly, for the control inputs we have:

$$u_{lim}^{ground} = \begin{bmatrix} -10 & -200 & -\infty & 0 \\ 10 & 200 & \infty & \infty \end{bmatrix}$$

$$u_{lim}^{flight} = \begin{bmatrix} -10 & -200 & 0 & 0 \\ 10 & 200 & 0 & 0 \end{bmatrix}$$

In fact, when the robot is flying we have no contact forces (i.e., they are constrained to be equal to 0).

At this point, we have all the elements to solve and simulate the problem.

3.2 Proportional Derivative controller

A Proportional Derivative (PD) controller is one of the most important feedback controllers.

We want to control the robot during the balancing phase with a Proportional Derivative controller whose equation is:

$$\begin{aligned} \tau = \tau_{des} - K_P^{\theta}\, e(\theta) - K_D^{\theta}\, e(\dot{\theta}) - K_P^{\phi}\, e(\phi) - K_D^{\phi}\, e(\dot{\phi}) \\ f = f_{des} + K_P^{l}\, e(l) + K_D^{l}\, e(\dot{l}) \end{aligned} \tag{7}$$

where $e(\cdot)$ if the difference between the desired and measured state, while K_D, K_P are the derivative and proportional gains. For the input torque, we take into account the error on the generalized coordinates θ and ϕ and on its derivatives, while for the force we consider the error on l. The force also has a feedforward term equal to the desired value for it; we set it as:

$$f_{des} = m_b\, g\, \cos\theta \tag{8}$$

Let us see which is the interpretation of these two terms:

- Proportional action: it immediately reacts to the variations of the error, so it represents the present action. The error decreases as the proportional constant K_P increases, in fact, the rise time is the main characteristic that we can improve with this action, but, of course, there is a limit for the increase of the proportional constant.
- Derivative action: it is proportional to the derivative of the error, so it represents an anticipation of what will happen to the error. It is used to improve the transient of the response since it can decrease the overshooting.

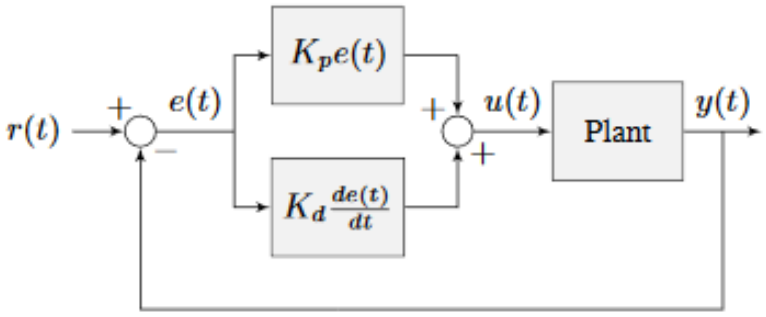

Figure 4: PD controller - block diagram

In order to implement it, we have used the **acados model** in which we define $\mathbf{X} = [\phi, l, \theta, \dot{\phi}, \dot{l}, \dot{\theta}]$ as state variables. In the acados model, we have not set any input, but we have considered τ and f as parameters since we are going to compute them with the PD expression. Finally, we define the explicit and implicit formulation of the model and we are going to integrate it using Runge Kutta of 4^{th} order. We also set constrains for the value of the inputs; in fact, if the torque or the force exceed their maximum values, they will saturate.

The closed-loop PD controller can be implemented straightforwardly: at each time instant, we compute the control input considering the current state, and we give them in input to the system. In this way, we get the updated state to be used in the next iteration for the computation of the control inputs.

This formulation is the one taken into account when we will add noise.

4 Experiments

The simulations have been implemented in Python using the **acados** solver.

Acados is a software package for finding the solution of optimal control problems. It solves a non-linear Optimal Control Problem (OCP) whose aim is to minimize a cost function given the system's dynamics, the constraints and the initial conditions.
Acados provides some libraries written in C which expose very simple interface to Python, and it is compatible with the language of CasADi.

The code is divided into different files:

- utils: it contains a set of functions that are used to create videos and plots.
- Car models: in these files we define the model in an appropriate way, by specifying the system's dynamic model, and, eventually, the constraints.
- mains: in these files we define and solve the problem, so we implement the MPC controller or the PD controller depending on the simulation.

Moreover, we recall that the integrations have been performed using Runge-Kutta of 4^{th} order and the sampling time of the controller is $0.05\,s$.

4.1 Jumping over a gap – MPC

In this experiment, the robot has first to drive horizontally on four wheels, then it should get up and balance on two wheels, drive forward and finally jump over a gap. A two-stage controller is used because we have to switch from the model in 2.1 to the one in 2.2.

After switching to the WIP model, for each of the phases that the robot has to face (e.g. swing-up and balance, driving upright and jumping over a gap) we have defined some simulation times:

- In $5\,s$ the robot has to swing-up and balance starting from the final condition in phase 0.
- In $2\,s$ the robot has to approach the initial edge of the gap.
- In $0.4\,s$ the robot has to jump over the gap.
- In $0.6\,s$ the robot has to settle down and regain its balance after landing.
- In $2\,s$ the robot has to drive upright again.

In the simulations, we set the following values for the robot's parameters:

- $m_b = 4$ Kg
- $m_w = 2$ Kg
- $R_w = 0.17$ m
- $I_w = m_w R_w^2 = 0.0578$ Kg·m^2

The behaviour of the state variables is the following:

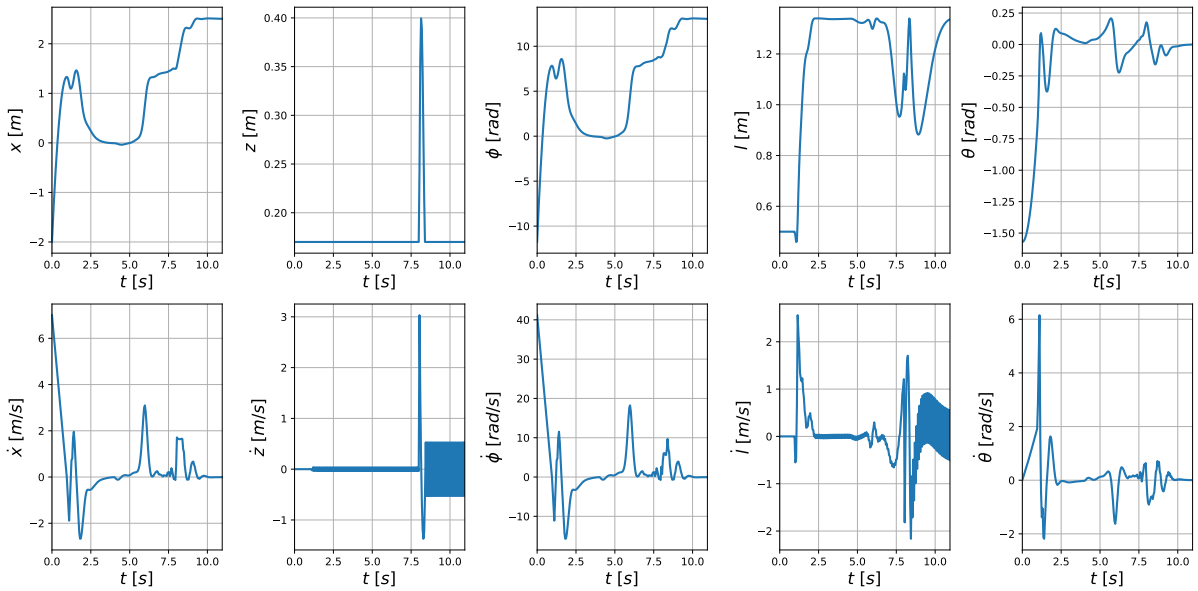

Figure 5: State variables

The variable x and ϕ have the same shape unless a scaling factor; in fact, $x = R_w\,\phi$. At first, the robot will accelerate to reach a target velocity, then it will immediately stop to pass from four wheels to two. It will then move with the aim of balancing, and will jump over the gap situated in $x \in [1.5, 2]\,m$. From the shape of z, we are able to understand the phase in which the robot has to jump over a gap; in fact, the z variable is almost always zero except for few seconds during the fly. Concerning l, it cannot extend more than $1 + 2R_w = 1.34\,m$, so we can state that it will keep its maximum extension for most of the time during the simulation. Finally, θ oscillates around its target value that is $0\,rad$.
At the bottom part, we have the shape of their derivatives.

For the control input we obtain:

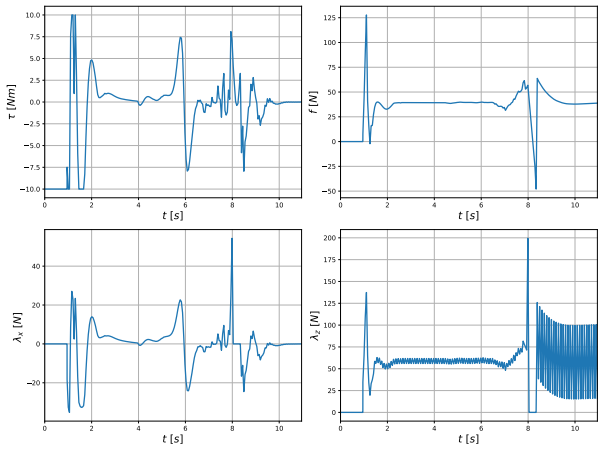

Figure 6: Control variables

The torque τ at the beginning saturates at its maximum and minimum values that are $\tau = \pm 10$ and then it will oscillates around the 0 value depending on the phase's target. The force f is 0 when the robot stays on four wheel since it was not included in the model (the link could not extend in the four-wheel drive), but then suddenly reaches a high value; this is due to the fact that the robot has to swing up. Anyway, it will never reach its limit value that is $\pm 200N$. The negative peak will be reached during the jumping phase; in that seconds, in fact, the contact force λ_z is 0. The shape of λ_x is quite similar to τ.

In fact, this is the force that is needed to avoid slipping.

If we focus on the jumping phase, the robot will perform this way:

Figure 7: Jumping over a gap phase

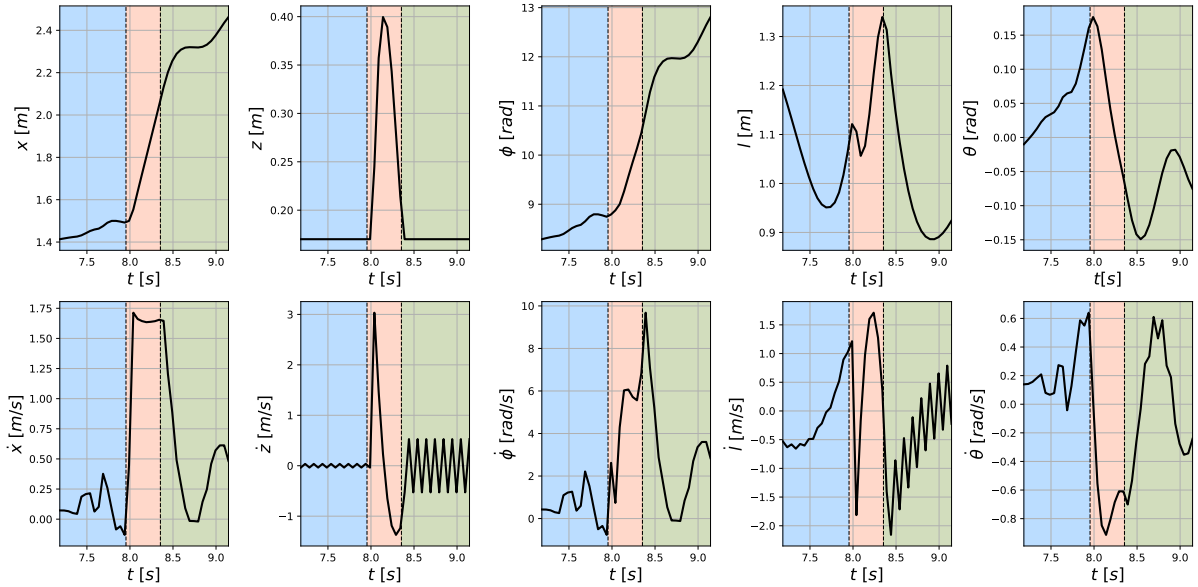

Figure 8: Evolution of the state variables during the jumping over a gap phase

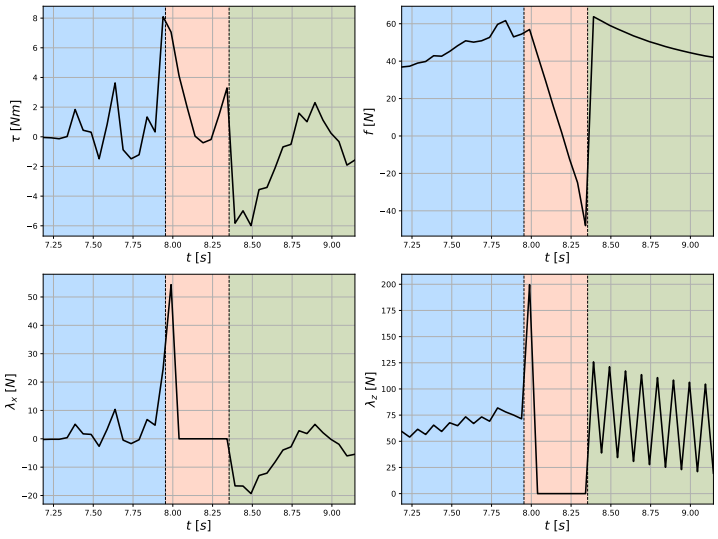

Figure 9: Evolution of the controls during the jumping over a gap phase

Three colors are used to identify the pre-takeoff, flying and landing phases. The robot will contract its

body by restricting the prismatic joint before the takeoff, and then will extend it before approaching again the ground. Finally, the robot will stabilize again to keep balancing.

4.2 PD vs MPC

4.2.1 Simplified model

In this experiment, we are just going to control the robot during the balancing phase, so we do not care about the floating base and we can use a simplified model. In this case, we derive a new dynamic model in which we consider the vector of generalized coordinates as $\mathbf{q} = [\phi, l, \theta]^T$. We have not included x and z (as in the previous model) because now we are going to keep the z value constant and equal to the radius of the wheel. In fact, we always want that the wheel is in contact with the ground. Concerning x, it is not an independent variable since $x = R_w\phi$.

We derive the new dynamic model using the Lagrangian formulation. At first, we define the position x_b, z_b of the mass m_b and its velocity $\dot{x_b}$, $\dot{z_b}$:

$$\begin{aligned} x_b &= R_w\phi + l\sin\theta \\ z_b &= R_w + l\cos\theta \\ \dot{x_b} &= R_w\dot{\phi} + \dot{l}\sin\theta + l\dot{\theta}\cos\theta \\ \dot{z_b} &= \dot{l}\cos\theta - l\dot{\theta}\sin\theta \end{aligned} \tag{9}$$

Then, we define the kinetic and potential energy of the system.
The link will have a rotational and translational kinetic energy component, the wheel will have a rotational term only and the point mass provides only a translational contribution since its inertia is null:

$$T = \frac{1}{2}(I_w\dot{\phi}^2 + m_w\, R_w^2\dot{\phi}^2 + m_b\, \dot{x_b}^2 + m_b\, \dot{z_b}^2) \tag{10}$$

The potential energy is due to the gravity acting on the point mass and on the wheel:

$$U = m_w\, g\, z + m_b\, g\, z_b \tag{11}$$

This led to the following dynamic model:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}) + \mathbf{G} = \mathbf{S}^T\mathbf{u} \tag{12}$$

where $\mathbf{M}(\mathbf{q})$ is the inertia matrix:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} I_w + R_w^2\, m_b + R_w^2\, m_w & R_w\, m_b\, \sin\theta & R_w\, l\, m_b \cos\theta \\ R_w\, m_b\, \sin\theta & m_b & 0 \\ R_w\, l\, m_b \cos\theta & 0 & l^2\, m_b \end{bmatrix} \tag{13}$$

$\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$ is the vector of Coriolis and Centrifugal terms:

$$\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} R_w\, m_b\, \dot{\theta}(2\dot{l}\cos\theta - l\dot{\theta}\sin\theta) \\ -l\, m_b\dot{\theta}^2 \\ 2\, l\, \dot{l}\, m_b\dot{\theta} \end{bmatrix} \tag{14}$$

$\mathbf{G}(\mathbf{q})$ is the gravity vector:

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} 0 \\ g\, m_b \cos\theta \\ -g\, l\, m_b \sin\theta \end{bmatrix} \tag{15}$$

and $\mathbf{S}$ is a selection matrix:

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{16}$$

The control input is still the vector:

$$\mathbf{u} = \begin{bmatrix} \tau \\ f \end{bmatrix} \tag{17}$$

so the actuation torque and the linear force in the prismatic joint, but this time we have not considered explicitly the contact forces.

4.2.2 Tests

After testing the MPC controller, we decided to use another kind of controller in order to make a comparison; in particular, we implemented a PD control law. For this experiments we simplify the robot's task and we consider only a balancing operation. The gains are tuned in order to obtain behaviors that are as similar as possible between the MPC and the PD controller. We run both the simulations for 12 seconds; the initial configuration of the robot is $\mathbf{q}_i = [0, 0.5, \frac{\pi}{8}]$, while the target one is $\mathbf{q}_f = [-\frac{2}{R_w}, 0.8, 0]$.

Even if the generalized coordinates of the model are only ϕ, l and θ, we plot also the values of x and z; the former is obtained through the relation $x = R_w\phi$, while the latter is always constant and equal to R_w.

We report the results obtained with the PD controller:

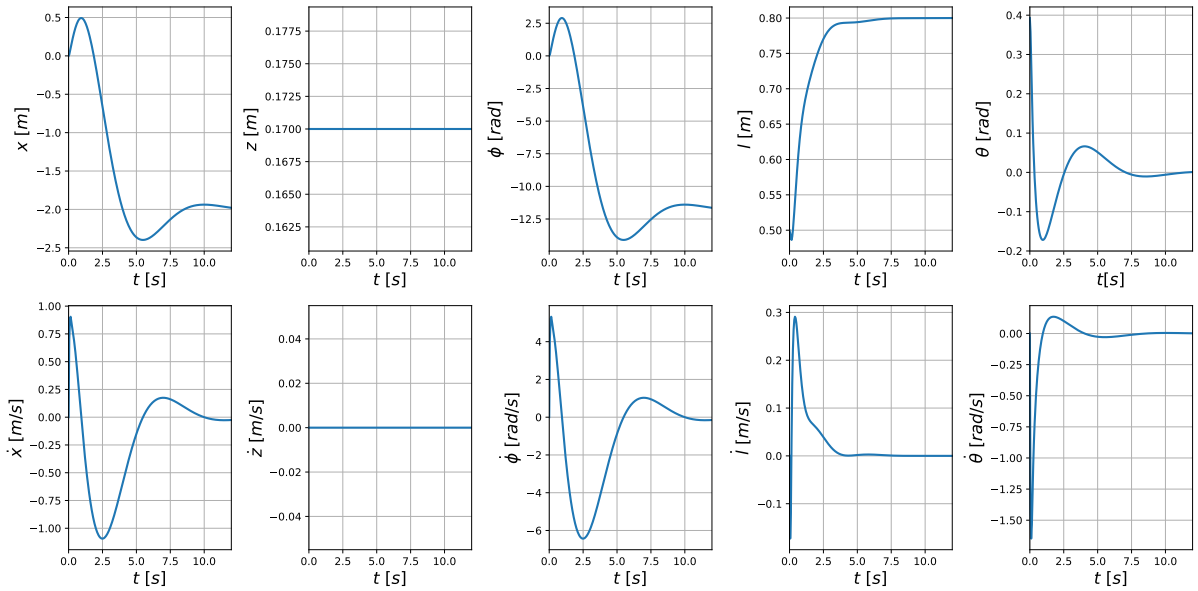

Figure 10: State variables – PD

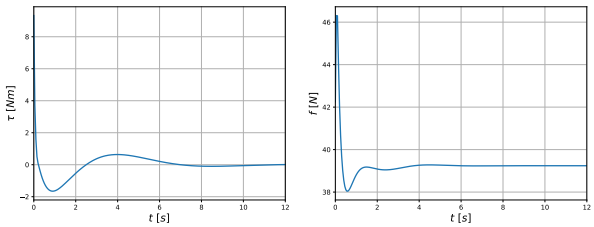

Figure 11: Control variables – PD

And the ones obtained with the MPC:

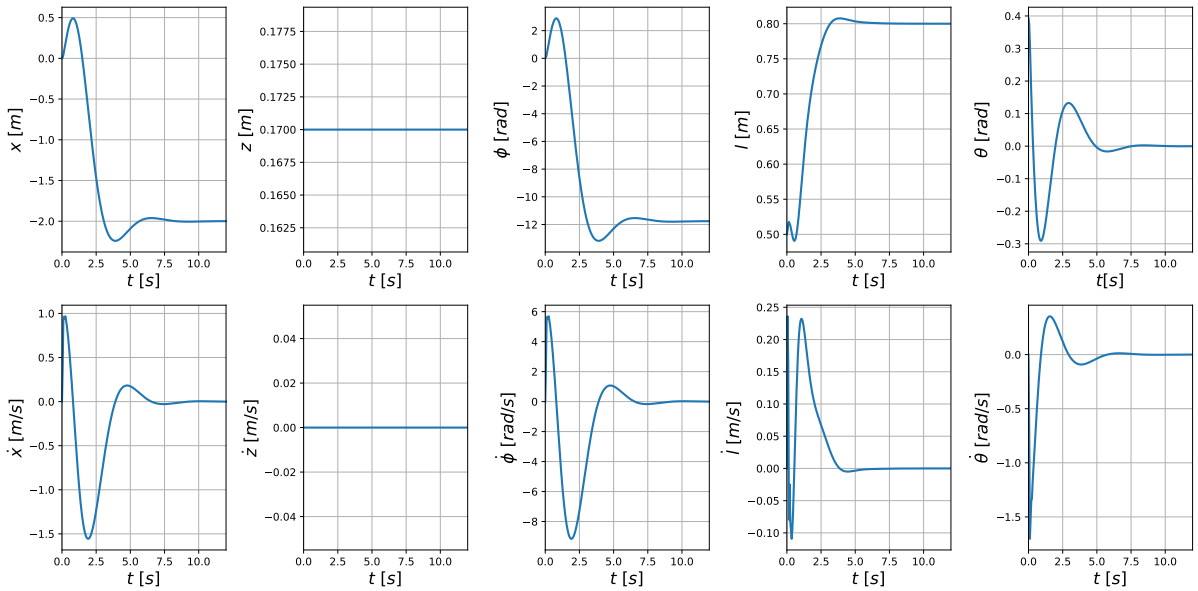

Figure 12: State variables – MPC

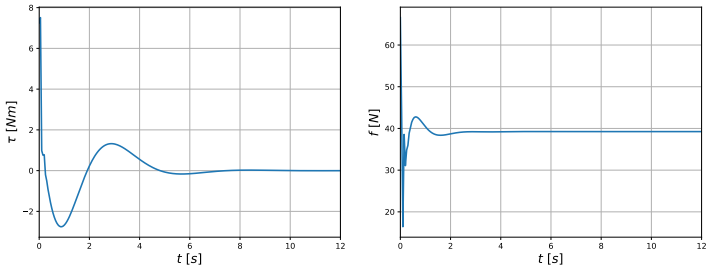

Figure 13: Control variables – MPC

From these plots we can see that the MPC controller is faster in reaching the target; in fact, the state values stabilize after about 7 seconds of simulation, while for the PD controller we would have had to run the simulation longer in order to reach a steady state.

As expected, since the behavior with the PD controller is slower, the values of the control input τ are a little smaller compared to the ones for the MPC. However, in both cases l reaches the desired value in few seconds; this is reflected in the values of the control input f, which are similar in the two performances.

Finally, it is interesting to analyse data about the computational times in the two cases:

	mean time	*std dev*	*max time*	*total time*
PD	0.00008	0.00002	0.0003	0.0195
MPC	0.0025	0.0006	0.0054	0.6082

where *mean time* is the mean time value among all the iterations, *max time* is the time that the slowest iteration takes in order to finish, *std dev* is the standard deviation among the iteration times and *total time* is the time needed to execute 12 seconds of simulation.

We can recognize that the MPC is slower than the PD controller; in fact, in the table, we can see longer computational times for the MPC case.

4.2.3 Robustness to Sensor Noise

To test our work further, we added sensor noise on both MPC and PD controller. We considered a Gaussian noise, therefore, the sensor reading at each step is:

$$x_{read} = x + \mathcal{N}(\mathbf{0}, \sigma\mathbb{I}) \tag{18}$$

so we assume to have Gaussian noise with zero mean and σ standard deviation. $\mathbb{I}$ is the identity matrix and x is the real state in which the robot is.
Noise varied in range $0 - 0.4$ with a step size of 0.02. For each value of σ, we run 40 experiments. We repeated the experiments on the two controller considering the setting defined in 4.2.2.

To make a comparison between the two controllers, we computed the L2-distance between the final state that we reach at each simulation and the desired value. We obtain:

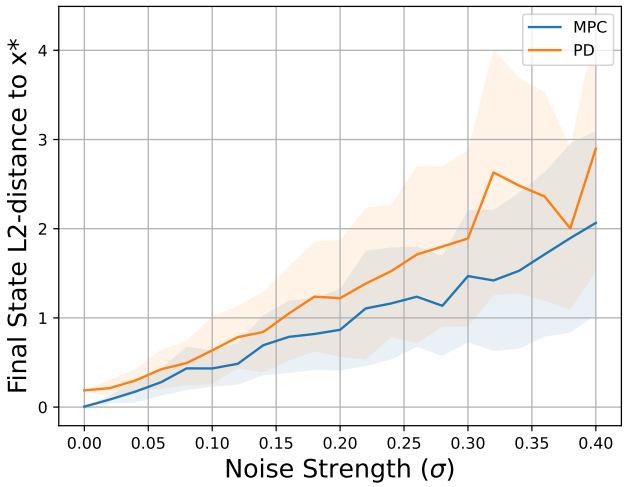

Figure 14: Noise comparison

When the noise is small, the PD controller and the MPC behave in a quite similar way, in fact, the gap between them is very small. Anyway, the difference in the two controller will increase as soon as σ increases. In fact, the PD controller will have a much larger variance with respect to the MPC, so we can state that the latter is more robust. Anyway, both controllers are very sensible to the introduction of noise, so we should try to reject noise as much as possible.

5 Conclusion

The goal of the project was to control the Variable-Length Wheeled Inverted Pendulum for a balancing and jumping task. We have demonstrated that, despite its very simple dynamics, this system is able to generate complex motions. With the Model Predictive Control, we have been able to control it in a very precise way, obtaining high accuracy motions during all the phases.

We have also compared the results of the Model Predictive Control with the one obtained with a Proportional Derivative controller in the balancing phase. Also in this case, we are able to perform the desired task. Anyway, with the PD controller, we had to tune the gains in an empirical way by finding out which was the best configuration for our task. On the other hand, with the MPC, we were able to obtain even more difficult tasks (as jumping) in an easier way by just including constraints in the formulation of the problem. One of the drawbacks of MPC was that it needs more time to solve the same task with respect to PD controller. In fact, the control action found by the MPC is the results of an optimization problem.

Finally, to test robustness of both controllers, we have included sensor noise. The obtained results

show that MPC is more robust than PD controller since it was less sensible to noise. In fact, the future of MPC is bright due to its wide field of applications and to the very good results that it is able to achieve.

References

[1] Traiko Dinev, Songyan Xin1, Wolfgang Merkt, Vladimir Ivan,and Sethu Vijayakumar. *Modeling and Control of a Hybrid Wheeled Jumping Robot*

[2] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Jonathan Frey, Thivaharan Albin, Rien Quirynen, Moritz Diehl *acados – a modular open-source framework for fast embedded optimal control*