SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER, CONTROL AND MANAGEMENT ENGINEERING

Real-Time Safe Bipedal Robot Navigation using Linear Discrete Control Barrier Functions

AUTONOMOUS AND MOBILE ROBOTICS

Students:
Eugenio Bugli
Damiano Imola
Salvatore Michele Rago

Academic Year 2024/2025

Contents

1 Introduction

The term *humanoid*, stands for a robot characterized by a structure and kinematics similar to the human body. These type of robots, which are inherently underactuated thanks to the unilateral ground contacts, are designed to navigate and interact in environments structured for humans. One of the most important task for humanoids is Real-time navigation, which represents how these robots should navigate an environment by planning a safe path. With this term, we refer to paths where the dynamics and physical limitation of the robot are respected and that do not involve any collision between the robot and the obstacles. Due to the high complexity of this task, path planning is usually decoupled from gait control, resulting in a significant reduction of the computational load.

The aim of this work is to re-implement the solution proposed by Peng et al. in "*Real-Time Safe Bipedal Robot Navigation using Linear Discrete Control Barrier Functions*", which consists in a unified safe path and gait planning framework that uses Linear Control Barrier Functions (LCBF) to avoid collision with obstacles.
In the following chapters, we will delve into the details of this approach, discuss the results, and propose some improvements.

2 3D-LIP Model with Heading Angles

Small Intro ?

2.1 Local Robot Reference Frame

The state $\mathbf{x}$ and the input $\mathbf{u}$ of the dynamic model are defined as:

$$\mathbf{x} := \begin{bmatrix} p_x & v_x & p_y & v_y & \theta \end{bmatrix}^T \in \mathcal{X} \subset \mathbb{R}^5,$$

$$\mathbf{u} := \begin{bmatrix} f_x & f_y & \omega \end{bmatrix}^T \in \mathcal{U} \subset \mathbb{R}^3,$$

where (p_x, v_x) are the CoM position and translational velocity along the x-axis, f_x is the x-coordinate of the stance foot position, θ and ω are the humanoid's orientation and turning rate, respectively. $\mathcal{X}$ is the set of the allowed states, while $\mathcal{U}$ is the set of the admissible inputs.

Both the state and the input are expressed in the *local coordinates* of the robot, which are time-related. It means that (p_{x_k}, p_{y_k}) represents the position of the CoM at simulation time step k in the k-th reference frame (RF_k) that originates from $(p_{x_{k-1}}, p_{y_{k-1}})$, and is rotated by an angle θ_k around the z-axis with respect to RF_{k-1}. The relation between the vectors in different reference frames is represented in Figure 1. The reference frame at time step 0 is considered the "inertial" or "global" frame. A transformation between the inertial and moving frames is necessary to obtain the position of the humanoid in the global map and to deal with obstacles.

These Transformations are defined by the following matrix:

$$\mathbf{T_k} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_{k,\,\mathrm{glob}} & -\sin\theta_{k,\,\mathrm{glob}} & p_{x,\,k-1,\,\mathrm{glob}} \\ \sin\theta_{k,\,\mathrm{glob}} & \cos\theta_{k,\,\mathrm{glob}} & p_{y,\,k-1,\,\mathrm{glob}} \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

Starting from this, we can express all the quantities in the *global reference frame* as follows:

$$\begin{bmatrix} f_{x,\,k,\,\mathrm{glob}} \\ f_{y,\,k,\,\mathrm{glob}} \\ 1 \end{bmatrix} = \mathbf{T_k} \begin{bmatrix} f_{x,\,k,\,\mathrm{loc}} \\ f_{y,\,k,\,\mathrm{loc}} \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} p_{x,\,k,\,\mathrm{glob}} \\ p_{y,\,k,\,\mathrm{glob}} \\ 1 \end{bmatrix} = \mathbf{T_k} \begin{bmatrix} p_{x,\,k,\,\mathrm{loc}} \\ p_{y,\,k,\,\mathrm{loc}} \\ 1 \end{bmatrix}, \tag{2}$$

$$\begin{bmatrix} v_{x,\,k,\,\mathrm{glob}} \\ v_{y,\,k,\,\mathrm{glob}} \end{bmatrix} = \mathbf{R} \begin{bmatrix} v_{x,\,k,\,\mathrm{loc}} \\ v_{y,\,k,\,\mathrm{loc}} \end{bmatrix} = \begin{bmatrix} \cos\theta_{k,\,\mathrm{glob}} & -\sin\theta_{k,\,\mathrm{glob}} \\ \sin\theta_{k,\,\mathrm{glob}} & \cos\theta_{k,\,\mathrm{glob}} \end{bmatrix} \begin{bmatrix} v_{x,\,k,\,\mathrm{loc}} \\ v_{y,\,k,\,\mathrm{loc}} \end{bmatrix}, \tag{3}$$

The positional vectors are roto-translated using an homogeneous matrix $\mathbf{T_k} \in \mathbb{R}^{3\times3}$, while the velocity vectors are only rotated around the z-axis (indeed, a translation

would change their magnitude) using a rotation matrix $R \in \mathbb{R}^{2\times 2}$.
The global robot's orientation is obtained by summing the latest local variation to the previous global angle and the angular velocity does not need to be transformed because it is along the z-axis, which is fixed.

$$\theta_{k+1,\text{glob}} = \theta_{k,\text{glob}} + \theta_{k+1,\text{loc}}, \qquad \omega_{\text{glob}} = \omega_{\text{loc}}. \tag{4}$$

With the following plot we can represent a change of coordinates between the global frame and the local one.

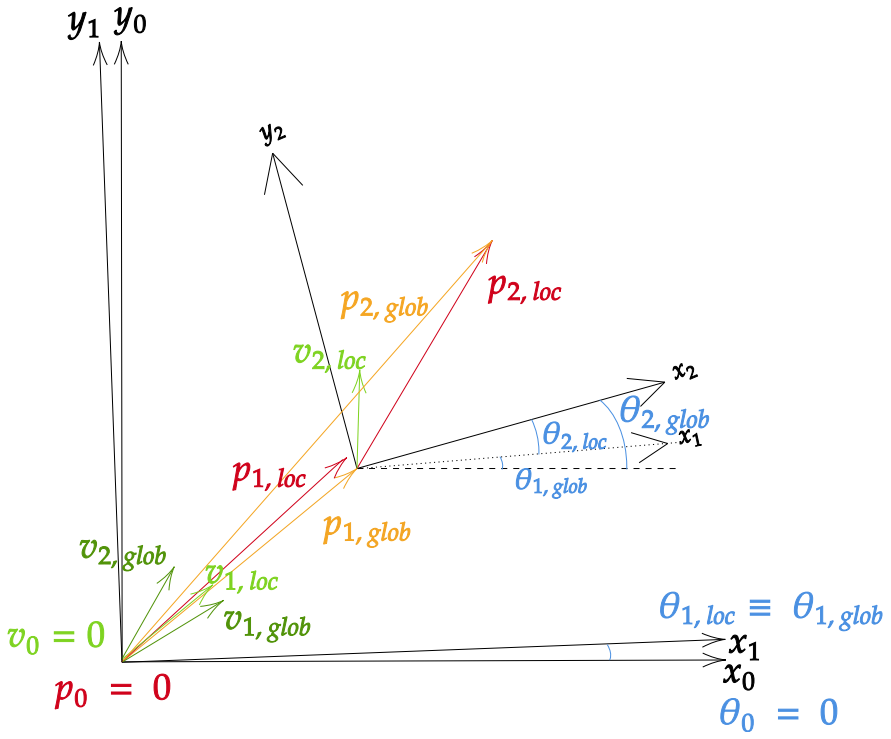


Figure 1: An example of the evolution of the 3D-LIP model's state, highlighting the relationship between local and global coordinates. The initial state is **0** and the RF (x_0, y_0) is the inertial frame. The local RF translates and rotates with the simulation time k: the position of RF_2 (x_2, y_2) has its origin in $p_{1,\text{glob}}$, while its orientation is given by $\theta_{2,glob}$. The pose of the successive frames is computed analogously.

2.2 Model Definition

Due to the high dimensionality and non-linearity of the full dynamic model of a humanoid, we have used a simplified model base on the standard Linear Inverted Pendulum formulation. This reduced model assumes that during the motion, the Center of Mass (CoM) will have a constant height during the motion. According to

(citation), we can express the acceleration of the CoM position as follows:

$$\begin{cases} \dot{v}_x = \dfrac{g}{H}(p_x - f_x) \\ \dot{v}_y = \dfrac{g}{H}(p_y - f_y) \end{cases} \tag{5}$$

By defining the state of our system as $x = [p_x, v_x, p_y, v_y, \theta]^T \in \mathbb{R}^5$ and the control input as $u = [f_x, f_y, \omega]^T \in \mathbb{R}^3$, the dynamics of the model can be written as follows:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{6}$$

This equation can be expanded as:

$$\dot{x} = \begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} A_c & 0 & 0 \\ 0 & A_c & 0 \\ 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} B_c & 0 & 0 \\ 0 & B_c & 0 \\ 0 & 0 & 0 \end{bmatrix} u(t) \tag{7}$$

where $A_c \in \mathbb{R}^{2\times2}$, $B_c \in \mathbb{R}^{2\times1}$ and $\beta = \sqrt{\dfrac{g}{H}}$.
From this continuous model, we can obtain the following discretized system:

$$x_{k+1} = A_L x_k + B_L u_k \tag{8}$$

where the two matrices A_L and B_L are defined as:

$$A_L = \begin{bmatrix} A_d & 0 & 0 \\ 0 & A_d & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad B_L = \begin{bmatrix} B_d & 0 & 0 \\ 0 & B_d & 0 \\ 0 & 0 & T \end{bmatrix} \tag{9}$$

with:

$$A_d = \begin{bmatrix} cosh(\beta T) & \dfrac{sinh(\beta T)}{\beta} \\ \beta sinh(\beta T) & cosh(\beta T) \end{bmatrix} \qquad B_d = \begin{bmatrix} 1 - cosh(\beta T) \\ -\beta sinh(\beta T) \end{bmatrix} \tag{10}$$

In the following chapter, these dynamics will be used as the internal process representation of an MPC. Along with the constraints, they will grant that the found solution is meaningful to the humanoid motion.

3 LIP-MPC: Gait planning with Model Predictive Control

The LIP dynamics defined in (**??**) is used as a model of the process inside a Model Predictive Control (MPC) scheme. The MPC controller uses that model to predict the future output along a *prediction horizon* N, namely the predefined number of time steps to look out in the future. Based on those forecasts and the provided constraints, the MPC computes the sequence of control actions that optimize a given cost function during the prediction horizon. Then, only the first input of that sequence is taken and provided to the process. The real output will be used at the next time step to compute the new predictions and control actions.
In our case, the LIP-MPC is used to respond instantaneously to the changes in the humanoid's state, while providing optimal stepping positions for stable locomotion and safe navigation. It is formulated as follows:

$$\begin{aligned} J^* = \min_{u_{0:N-1}} & \sum_{k=1}^{N} \Big[(p_{x_k} - g_x)^2 + (p_{y_k} - g_y)^2 \Big] \\ \text{s.t.} \quad & x_k \in \mathcal{X}, \quad k \in [1,\ N] \\ & u_k \in \mathcal{U}, \quad k \in [0,\ N-1] \\ & \mathbf{x_{k+1}} = \mathbf{A_L x_k} + \mathbf{B_L u_k}, \quad k \in [0,\ N-1] \\ & \mathbf{c_l} \leq \mathbf{c}(\mathbf{x_k}, \mathbf{u_k}) \leq \mathbf{c_u}, \quad k \in [0,\ N-1]. \end{aligned} \tag{11}$$

The cost function is defined as the distance between the sequence of predicted states over the horizon and the goal position. As a consequence, minimizing the cost function means that the robot is moving towards the goal. This problem is subject to the satisfaction of the robot's dynamics and the kinematics and path constraints, captured in $c(x_k, u_k)$ and defined in the *local coordinate frame*. This constraints are nonlinear due to the presence of the heading angle θ_k. These constraints are linearized by pre-computing the turning rates $\bar{\omega}_k$ for all the horizon length, in order to have them fixed inside the MPC calculation. The constraints include the walking velocities, leg reachability, maneuverability and the linear control barrier function, will be described in details in the following sections.

3.1 Heading Angle Preprocessing

Many of the constraints imposed in the MPC make use of θ in a non-linear way: for example, the kinematic constraints often show sinusoidal terms having θ as argument. This hinders real-time computation. The solution proposed by Peng et al. in [1] consists in using precomputed values for θ and ω, which will become fixed inside the MPC calculation. This means that the values of θ and ω are not determined by the optimization of the previously defined cost function performed by the MPC, but their

values throughout the prediction horizon are computed at the beginning of each time step with the following formulae:

$$\omega_k = \max\left\{\min\left\{\mathbf{atan2}(g_y - p_y,\, g_x - p_x) - \theta,\ \omega_{max}\right\},\ \omega_{min}\right\} \qquad \forall k \in [0, N-1]\,,$$

$$\theta_0 = 0, \qquad \theta_k = \theta_{k-1} + \omega_k\, T \qquad \forall k \in [1, N]\,,$$

where p_x, p_y and θ are the pose of the humanoid at the start of the simulation time step, T is the sampling time, and the goal position $(g_x,\ g_y)$ is expressed in local coordinates. With ω_{min} and ω_{max} we define the bounds on the robot turning rate, due to the actuation limits and to avoid sharp turns, which would threaten the stability of the humanoid. With the specified formulae, θ rotates with a velocity ω until the robot points to the goal. The initial value of the heading angle θ is set to zero since we are in a local frame, while for each step over the horizon is computed as the sum between the previous angle and the product between the current turning rate and the timestep. The turning rate ω, during each timestep is clamped to avoid sharp turns as follows: $|\omega_k| \leq 0.156\pi$ rad/s.

3.2 Safety Constraints

The stepping positions provided by the MPC solution must comply with specific kinematic constraints in order to be physically feasible. In this work, the authors of the paper decided to enforce the walking velocities, leg reachability, and maneuverability constraints as follows.

3.2.1 Walking Velocities Constraint

The velocity constraints, which limit the longitudinal and lateral velocities, are defined in the *local coordinate frame* as follows:

$$\begin{bmatrix} v_{x_{\min}} \\ v_{y_{\min}} \end{bmatrix} \leq \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} \begin{bmatrix} v_{x_k} \\ s_v v_{y_k} \end{bmatrix} \leq \begin{bmatrix} v_{x_{\max}} \\ v_{y_{\max}} \end{bmatrix}, \qquad \forall k \in [0,\ N-1]\,. \tag{12}$$

The pre-multiplying matrix rotates the velocities vector by an angle $-\theta_k$ around the z-axis: namely, it takes the velocity vector $\mathbf{v}_{k+1}$ back to the orientation of RF_k. In the vector resulting from the transformation, the first component is the longitudinal velocity, while the second is the lateral velocity, both in RF_k. At this point, the constraint imposes that the velocities are within the limits, $v_{y_{\{\min,\,\max\}}}$ and $v_{x_{\{\min,\,\max\}}}$ respectively.

With the term s_v, we define the stance foot, which is 1 if the foot is the right one, while it is -1 otherwise. This is used to limit the lateral velocity in such a way that, at the end of each step, the stance foot is on the opposite side but the humanoid does not lose balance.

decide whether this text should be removed

For instance, if the right foot is the stance, since the body tends to fall on the left side, we want the lateral velocity not to increase too much toward the direction of the positive y. Therefore, we set $s_v = 1$, pushing the MPC to find a solution which is closer to $v_{y_{\min}}$.

3.2.2 Leg Reachability Constraint

Since there might be the possibility of having an over-extension of the swing leg during motion, a reachability constraint is introduced to prevent that.

$$\begin{bmatrix} -l_{\max} \\ -l_{\max} \end{bmatrix} \leq \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} \begin{bmatrix} p_{x_k} \\ p_{y_k} \end{bmatrix} \leq \begin{bmatrix} l_{\max} \\ l_{\max} \end{bmatrix}, \qquad \forall k \in [0,\ N-1]\,. \tag{13}$$

This type of constraint prevents the inputs computed at time k from moving the CoM too far away from $(p_{x_{k-1}},\ p_{y_{k-1}})$. Indeed, the distance between the previous and the new position cannot be farther than $l_{\max}$, which is the maximum reachable distance of the swing foot in both directions on the ground. As in the previous case, the constraint is defined locally, so we need to rotate everything accordingly.

3.2.3 Maneuverability Constraint

This constraint is used to prevent fast turns during motion by decelerating the walking speed while turning.

$$\begin{bmatrix} \cos\theta_k & \sin\theta_k \end{bmatrix} \begin{bmatrix} v_{x_k} \\ v_{y_k} \end{bmatrix} \leq v_{x_{max}} - \frac{\alpha}{\pi}|\omega_k|, \qquad \forall k \in [0,\ N-1]\,. \tag{14}$$

This ensures that the component of the CoM velocity in the robot's heading direction (given by the vectors projection on the left side) is lower than a quantity that depends on the turning rate and some constant values. The parameter α is empirically set to 1.44 by the authors of the original paper.

3.3 Obstacle Avoidance with Control Barrier Functions

3.3.1 Definitions

Given a continuous and differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, we can define the *safety set* $\mathcal{C}$ and its boundary $\partial\mathcal{C}$ as:

$$\begin{aligned} \mathcal{C} &\coloneqq \left\{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}_k) \geq 0\right\}, \\ \partial\mathcal{C} &\coloneqq \left\{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}_k) = 0\right\}. \end{aligned} \tag{15}$$

These are the region and its perimeter where the robot can freely move without colliding with obstacles. According to [2], assume that there exists a class $\mathcal{K}$ function γ such that

$$0 < \gamma(h(\mathbf{x})) \leq h(\mathbf{x}),$$

and the following holds:

$$\begin{aligned} &\Delta h(\mathbf{x}_k,\, \mathbf{u}_k) \coloneqq h(\mathbf{x}_{k+1}) - h(\mathbf{x}_k), \\ \forall \mathbf{x}_k \in \mathcal{C} \qquad &\exists \mathbf{u}_k \text{ s.t. } \Delta h(\mathbf{x}_k,\, \mathbf{u}_k) \geq -\gamma(h(\mathbf{x}_k)). \end{aligned} \tag{16}$$

Then, $h(\cdot)$ is a discrete control barrier function (DCBF). We can also take γ as a scalar such that $0 < \gamma \leq 1$, and (16) becomes:

$$\forall \mathbf{x}_k \in \mathcal{C} \qquad \exists \mathbf{u}_k \text{ s.t. } \Delta h(\mathbf{x}_k,\, \mathbf{u}_k) \geq -\gamma * h(\mathbf{x}_k).$$

It means that, if the state starts from the safety set $\mathcal{C}$, there exists an input that, when applied, produces a state which will sill be in $\mathcal{C}$: namely, $\mathcal{C}$ is invariant.
Not sure about it

3.3.2 Linear Discrete Control Barrier Functions (LDCBF)

Assume that for each obstacle there is a function of the robot's configuration such that $F(\mathbf{x}) = 0$ when the robot touches the boundary of the obstacle, $F(\mathbf{x}) > 0$ when it does not collide with the obstacle, and $F(\mathbf{x}) < 0$ if it is inside the obstacle. It is convenient to choose $h(\cdot) = F(\cdot)$. This is the case in our project, where the DCBF is a function of $\vec{x}$, the Cartesian position of the CoM: therefore, $h(\cdot)\colon \mathbb{R}^2 \to \mathbb{R}$. Moreover, we assume that either the obstacle associated with $F(\cdot)$ is convex or $F(\cdot)$ describes the convex shape that encloses the non-convex obstacle. However, if $F(\cdot)$ is a non-linear function, we obtain a non-linear DCBF constraint, which should be avoided for the reasons exposed before.
Peng et al. in [1] linearized the DCBF by approximating the safe region. The plane is partitioned in two halves: one containing the obstacle and the other containing the robot (as shown in Fig. 2). The safe region is the latter, and it is defined as:

$$h\left(\vec{x}\right) = \eta^T \left(\vec{x} - c\right) \geq 0, \tag{17}$$

where $c \in \mathbb{R}^2$ is the point on the obstacle's edge that is closest to the CoM position $\vec{x}$, while $\eta \in \mathbb{R}^2$ is the normal vector that connects $\vec{x}$ to the boundary. Hence, η is also the vector orthogonal to the line that defines the half-planes, pointing toward the safe region. All of these components are shown in Figure (2).

If multiple obstacles are in the environment, the safe region is the intersection of the regions produced by each LDCBF.

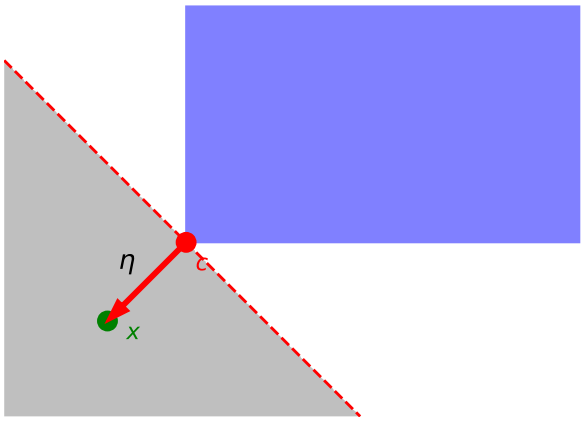


Figure 2: In this plot, the CoM is represented as a green point, and the obstacle is a blue rectangle. The point on the obstacle's boundary that is closest to $\vec{x}$ is the red one, which represents c. The red vector connecting c to $\vec{x}$ is η. The safe region defined by this LDCBF is the gray area. In that half-plane, the humanoid's CoM is free to move without hitting any obstacle.

By enforcing that the states $\mathbf{x}_{1\ldots N+1}$ produced by the optimal inputs $\mathbf{u}_{0\ldots N}$ computed by the MPC satisfy $h(\vec{x}) > 0$, we guarantee that, during the motion, the humanoid's CoM does never collide with the obstacles.

Vectors η and c are computed as follows. Indicating with A and B the endpoints of the edge of the obstacle that is closest to $\vec{x}$ we have:

$$\overrightarrow{AX} \coloneqq \vec{x} - A, \qquad \overrightarrow{AB} \coloneqq B - A,$$
$$t \coloneqq \frac{\overrightarrow{AX} \cdot \overrightarrow{AB}}{\left\|\overrightarrow{AB}\right\|^2}, \qquad \tilde{t} = \max\left\{0,\ \min\left\{1,\ t\right\}\right\},$$
$$c \coloneqq A + \tilde{t} * \overrightarrow{AB},$$
$$\eta \coloneqq (-1)^{\gamma} * (\vec{x} - c),$$

where $\gamma = 1$ if $\vec{x}$ is inside the obstacle, otherwise it is 0. Due to the presence of min and max, the specified expressions of c and η are non-linear. However, the constraint enforced on the MPC solution at simulation time k is based on the values of c and η computed at the beginning of the time step. Therefore, they are included in the constraint as constants, and the LDCBF becomes a linear combination of $\vec{x}$.

3.3.3 Our variation of the LDCBF

From Figure (2) it is evident that a such defined LDCBF constraint allows the humanoid to get very close to the obstacle's boundary. And, even though the CoM will not collide with it, the footstep positions provided by the MPC will likely overlap with the obstacle. Therefore, we developed a variant of the LDCBF proposed by Peng et al., defined as follows:

$$h\left(\vec{x}\right) = \eta^T \left(\vec{x} - c\right) - \delta \geq 0, \tag{18}$$

where c and η are the previously defined vectors, while δ is the distance that the CoM must keep from the c (i.e., from the obstacle's boundary) to satisfy the constraint. The result is shown in Figure 3

Forcing the control barrier function not to take values below a threshold allows us to take into account the footstep positions, and to generate trajectories that are not only feasible for the CoM, but for the motion of the whole humanoid.

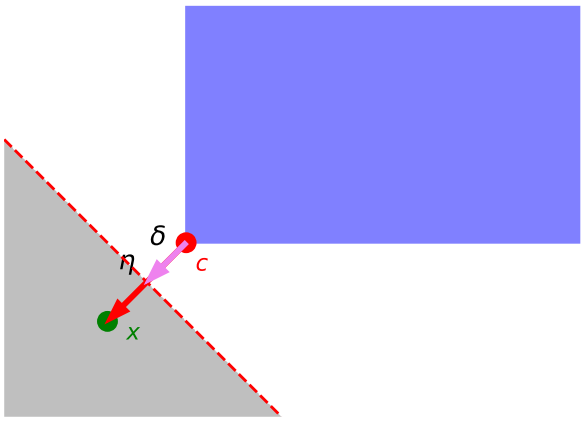


Figure 3: This plot shows the same environment of Figure 2, but here expression (18) with $\delta > 0$ was used as LDCBF. It is clear that the safe zone is not adjacent to the obstacle, and $\vec{x}$ maintains a distance δ from the obstacle.

4 Simulations

In this chapter, we will test the previously described MPC in a virtual enviroment. Throughout the simulations, the whole complexity of the humanoid will be condensed in a single point, representing its CoM.
The objective of this set of tests is to prove that, using the inputs provided by the MPC proposed by Peng et al. in [1] and developed by us, a robot is able to reach a user-defined goal, while avoiding collisions with all the obstacles in the environment.
The hyper-parameters used in the following simulations are summarized in Table 1.

Simulation Parameters	
Parameter	**Value**
α	1.44
CoM height	1 m
Step duration (T)	0.4 s
$l_{\max}$	$0.1\sqrt{3}$ m
$\left(v_{x_{max}},\, v_{y_{max}}\right)$	$(-0.1,\, 0.1)\ \mathrm{m\,s^{-1}}$
$\left(v_{y_{min}},\, v_{y_{min}}\right)$	(0.8, 0.4)

Table 1: The MPC and LIP parameters used for all the simulations in this chapter.

4.1 Simple Environment

The first simulation takes place in a basic environment: the humanoid is positioned at (0, 0) with orientation $\theta = 0$, and the goal position is (5, 5). In between, we placed 3 obstacles with a quasi-circular shape. The choice of approximating circles with polygons was made to comply with the CBF computation described in 3.3.2. The results are shown in Figures 4 and 5.

By inspecting the plots, we see that, at the beginning of the simulation, a high turning velocity ω is commanded in order to make the robot point toward the goal. In the meantime, the MPC provides new footstep positions but, due to the maneuverability constraint, the high turning rate limits the velocity along the sagittal axis. Therefore, the robot is encouraged to move laterally, and the first inputs generate a large v_y that induces a *side-walking* motion.
Then, there is an overshoot in the plots of θ and ω. This behavior derives from the large sampling time of the robot. To lighten the computational load, in our simulations, we set equal sampling times for the MPC and the humanoid. This means that every T seconds a new footstep position and angular velocity are computed, and they are provided as input to the robot until new inputs are available, even though it could have a lower sampling time and, then, several different inputs could be commanded during T. Therefore, the ω needed to point to the goal is provided as input and executed for T

seconds. But, during that interval, the robot misses the right orientation and continues rotating. Hence, a new turning velocity is commanded in the opposite direction, and the sagittal axis is successfully aligned with the goal.

At this point, the robot moves forward to the goal, the orientation is kept constant, and the positional error decreases smoothly. Nevertheless, there is some chattering in the v_y plot. This is a regular behavior: at the end of each step, the humanoid changes the stance foot by *falling* on the opposite side, and it results in the lateral velocity changing sign.

Around the second 35 of the simulation (4th and 5th frames of Figure 5), the humanoid approaches one of the obstacles. It is unable to go forward, as this would lead the CoM into the circle. To keep the robot in the safe area defined by the LDCBF, the new MPC solution makes the humanoid move laterally. Thus, it can successfully avoid the obstacle. In this case, *side walking* occurs because the precomputed value of ω only aims at aligning the sagittal axis of the robot with the goal, and, once it is achieved, the turning rate is kept constant. Consequently, the humanoid can only escape the obstacle while facing the goal. This movement explains the minor variations in the position error and translational velocity plots around the second 35.

The humanoid can, then, realign itself (producing a minimal oscillation in the θ and ω plots) and continue navigating to the goal, which it ultimately reaches successfully.

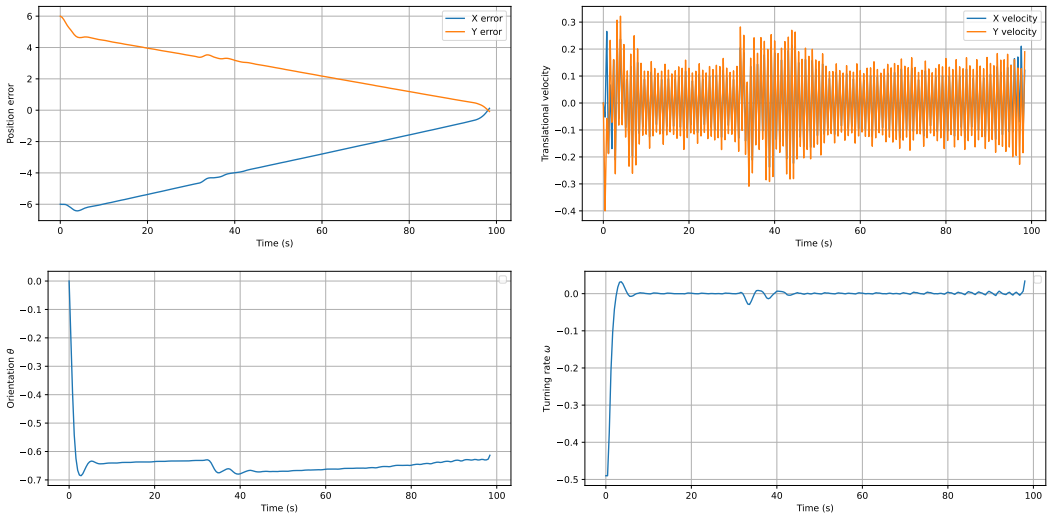


Figure 4: These figures depict the evolution of the humanoid's state and the input throughout the simulation. The top-left plot shows the error between the CoM and the goal position, while the top-right represents the translational velocities along the x- and y-axis. The bottom left and right plots show the theta and omega evolution, respectively. All the quantities are expressed in the global RF.

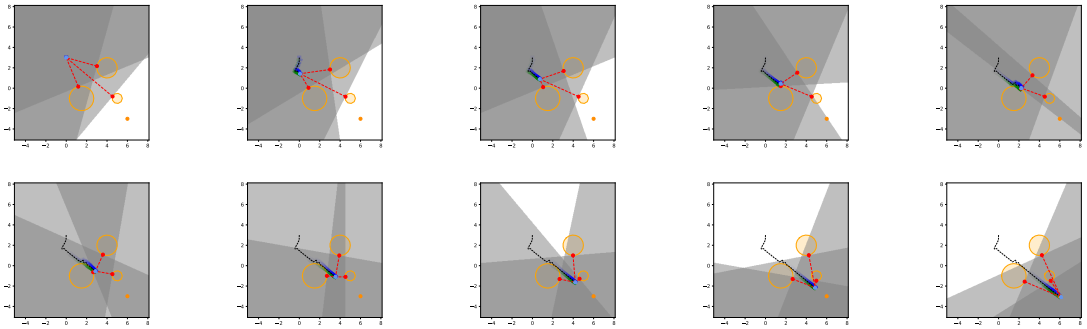


Figure 5: This sequence of frames illustrates the robot's trajectory to the goal. Obstacles are represented with orange shapes. The humanoid is represented by an isosceles triangle whose sagittal axis is aligned with the robot's. Green rectangles mark right footsteps, while blue rectangles mark left ones. Each frame shows the vectors η and c along with the safe area associated with each obstacle. The darkest region represents the intersection of these semi-planes, which is the overall safe area.

5 Our Improvements

In this chapter, we introduce some modifications or additions to the solution proposed by Peng et al. in [1], aiming to improve the robot's behavior in more general and complex environments.

5.1 Subgoals and RRT*

As described in Section 3, the solution provided by the MPC is the input vector that minimizes the cost function while satisfying all the constraints. When an obstacle lies between the start and the goal positions (as illustrated in Figure 6a), the humanoid must get around it in order to approach the goal. However, it would require the robot to deliberately increase its distance from the goal before reducing it. It implies that the MPC should return a sub-optimal solution, though one that minimizes the cost function exists. Consequently, an approach that relies solely on the MPC cannot guarantee goal attainment in such cases.
To address this, we compute a path from the start to the goal position using RRT*. Then, we request the humanoid to reach sequentially all the *subgoals*, namely the nodes of the tree in the path from the start to the goal. In this way, we ensure that the MPC produces feasible control inputs while adhering to the original cost-minimization framework, and the humanoid can successfully reach the goal.
The RRT* algorithm is used to rapidly compute a collision-free path while minimizing its cost, i.e. the sum of the weights on the edges connecting the start to the goal node. Whenever a new node j is added to the tree, the cost of the edge $e_{i,j}$ connecting it to node i is defined as:

$$cost(e_{i,\,j}) \coloneqq cost(path_{i,\,start}) \;+\; dist(i,\;j) * e^{-clearance(j)}.$$

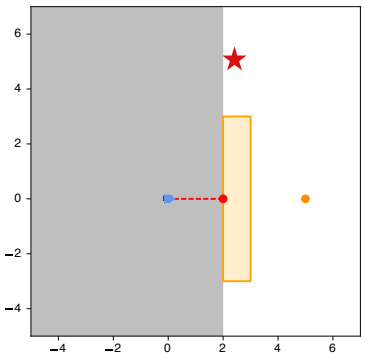
It is the sum of two addends: the cost of the path from i to the start node, and the Euclidean distance between the position represented by i and the one represented by j, multiplied by the exponential of the negative clearance of the new node (namely, the distance between j and the closest obstacle). Hence, the resulting path will minimize the travelled distance while maximizing the clearance.

Figures 6a and 6b show the behavior of the humanoid in a tricky environment employing the framework without the RRT* extension. To reach the goal, the robot must first overcome the obstacle by passing through the via-point represented by the red star. However, for the reasons explained above, this is not possible. Thus, the MPC only tries to reduce the distance from the goal, and leads the humanoid toward the obstacle, where it gets stuck because it cannot get closer to the goal without colliding.
Following the approach that integrates RRT*, the robot is able to reach the goal (Figure 8). The MPC is requested to provide the inputs that drive the humanoid

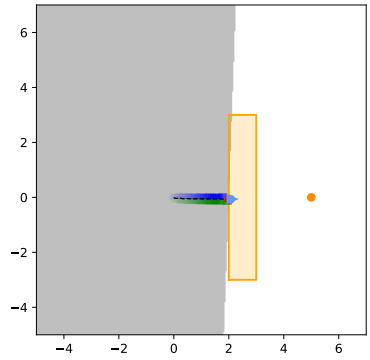
through the sequence of subgoals. In this way, the robot can pass by the red star, overcome the obstacle, and finally reach the goal.

From the evolution of the state (illustrated in Figure 9) emerges something that never occurred in the previous simulations. When the robot relies only on the MPC solutions to approach the goal, the error curves monotonically decrease (except for minimal variations). On the other hand, as described at the beginning of this section, in this case, it is necessary to overcome the obstacle, and hence increase the error. When the approach integrating RRT* is used, the humanoid moves toward the first subgoal, and initially increases the magnitude of the error with the final goal. By traveling through the RRT*-defined subgoals, the error is progressively reduced until the goal is reached.

The *step line* in the orientation plot indicates that, whenever a subgoal is reached, a turning rate is commanded to point toward the next subgoal. This behavior explains the peaks in the ω plot. The high lateral velocity generated while turning arises for the same reason outlined in Section 4.1.



(a) This figure illustrates an environment where the humanoid cannot reach the goal unless it employs the framework integrating RRT*. The robot starts from position $(0, 0)$ with orientation $\theta = 0$, and the goal is at $(5, 5)$. The red star represents an ideal via-point to arrive at the goal.



(b) This figure shows what happens when the robot tries to reach the goal in a tricky environment with the base framework.

Figure 6

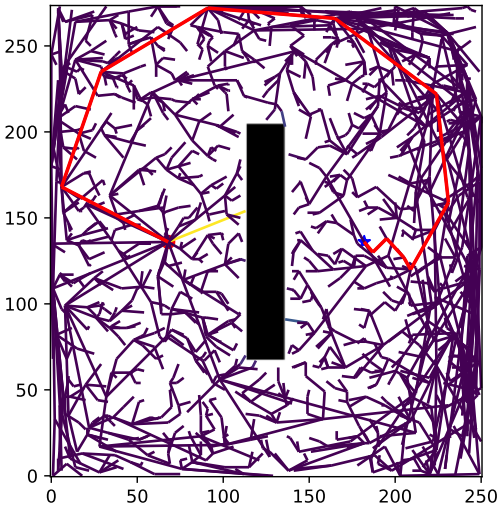


Figure 7: This figure illustrates the tree computed by the RRT* algorithm. The path is represented as a red line, the start position as a red point, and the goal as a blue star. The RRT is executed and illustrated inside an occupancy grid representing the original (continuous) workspace.

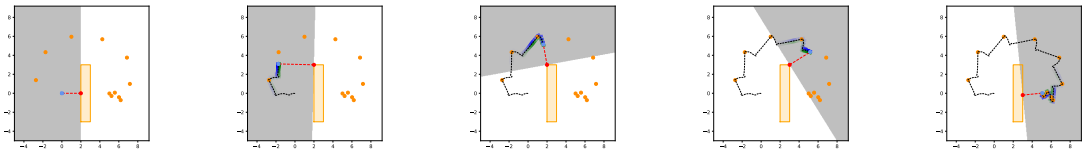


Figure 8: This sequence of frames illustrates the robot's trajectory to the goal passing through the RRT*-defined subgoals (represented as orange points).

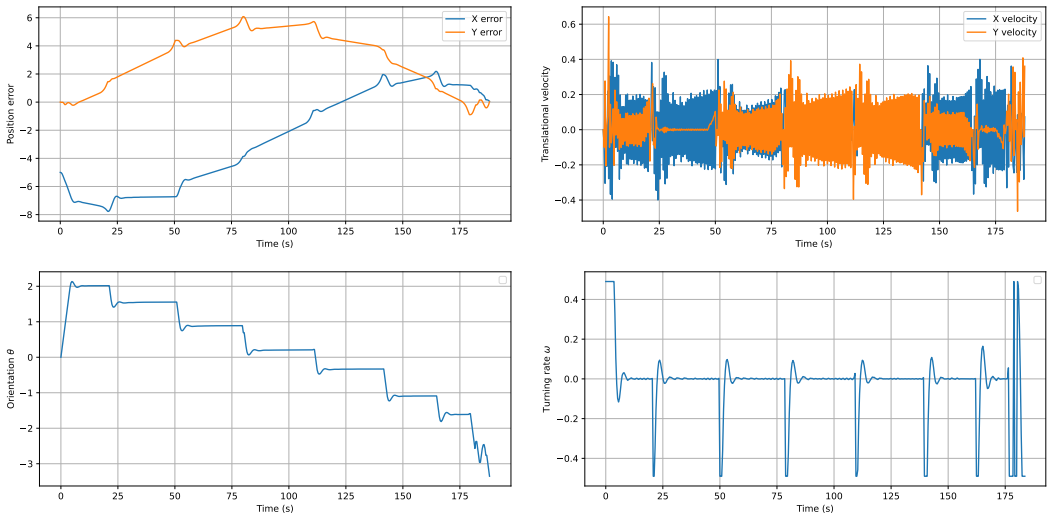


Figure 9: These figures depict the evolution of the humanoid's state and the input throughout the simulation with RRT*. The top-left plot shows the error between the CoM and the goal position, while the top-right represents the translational velocities along the x- and y-axis. The bottom left and right plots show the theta and omega evolution, respectively. All the quantities are expressed in the global RF.

5.2 Unknown Environment

In unknown environments, the robot does not rely on a pre-loaded map but rather builds its understanding on the fly.

proposta cambiare con "the robot navigation is not based on a pre-loaded map, but on real-time perceptions."

To achieve this, a 2D LiDAR range finder is employed to capture real-time measurements of the surroundings. These LiDAR readings are getting perturbed with zero-mean Gaussian noise to better simulate the uncertainty found in real-world sensor data. The noisy measurements are then processed using the DBSCAN clustering algorithm (with an epsilon of 0.3) to group nearby points into clusters that represent inferred obstacles.

This leads to a further tuning of the system, in order to achieve desired performance in real world scenarios;

proposta cambiare con: "With the integration of a LiDAR system, our framework is capable of replicating the previously described performance in real-word scenarios as well."

in fact provided that every real obstacle is convex, and assuming the obstacles are sufficiently distant, the LiDAR resolution is not too high (here 360°), and the DBSCAN epsilon remains small enough (here 0.3); these inferred obstacles will also be convex.

proposta cambiare con: "These LiDAR readings are affected by a zero-mean Gaussian noise"???

$\epsilon = 0.3$?

In fact, provided that every obstacle in the workspace is convex, and assuming that the obstacles are sufficiently distant, the LiDAR resolution is about 360°, and the DBSCAN ϵ is small enough, the obstacles inferred by our system will be convex too.

By considering only these clustered obstacles—rather than every single detected point—the navigation system mirrors the behavior of the base simulation while accounting for realistic environmental complexity.

proposta cambiare con: (continuando nello stesso paragrafo) "Then, the MPC computes one LDCBF for each of those obstacles. In this way, the navigation system can mirror ..."

In this way, the humanoid robot is able to navigate safely despite not knowing the environment a priori, relying solely on the dynamically inferred obstacles from its sensor data.

aggiungerei una breve descrizione di quello che accade nella simulazione in basso.

", and the humanoid" continuando da precedente paragrafo

obstacles inferred dynamically

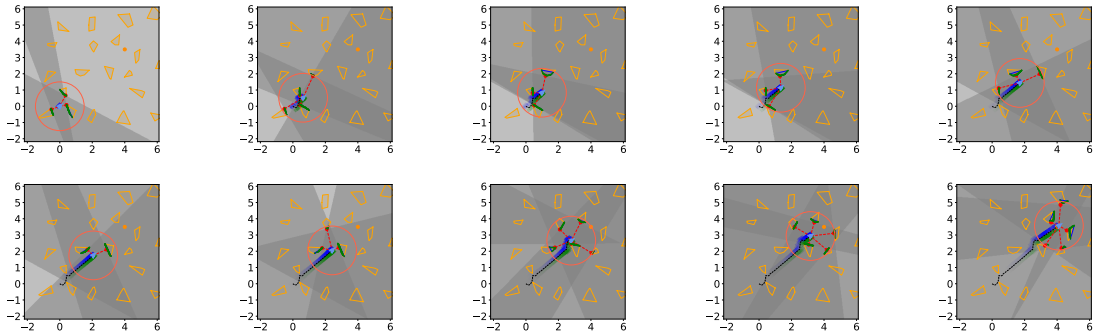


Figure 10: This is a description.

aggiungerei descrizione degli elementi nelle figure (es. punti verdi, cerchio)

IMPORTANTE: credo che bisogna aggiungere i plot sullo stato e l'input, e dire se hanno qualcosa di particolare o sono analoghi alle simulaizoni precedenti

6 Conclusion

In this project, we successfully reproduced the methodology presented in Peng et al. [1], implementing a real-time safe navigation framework for bipedal robots using Linear Discrete Control Barrier Functions (LDCBFs). Our implementation validated the feasibility of using a Linear Inverted Pendulum (LIP) model combined with Model Predictive Control (MPC) to achieve stable and dynamically feasible locomotion in cluttered environments.

Minding about social robotics, beyond reproducing the original work, we introduced an additional *safety margin* in the computation of LDCBF constraints. This modification aimed to enhance obstacle avoidance robustness, ensuring a more conservative approach to collision prevention while maintaining real-time performance. Our simulations demonstrated that this adjustment effectively increased the reliability of navigation without introducing significant computational overhead; anyway this novelty results in additional system tweaking, so to obtain the best safety margin over the environment.

Further refinement of the safety margin parameter could provide an optimal trade-off between conservatism and maneuverability; an environment-based dynamic safety margin will be the best suited for that task. For example, someone can tweak it based on the number of obstacles inside the FOV, or based on the width of the available walking space.

Moreover, for safe gait planning with limited FOV, we leveraged LiDAR scans of the surrounding scene and applied DBSCAN algorithm so to obtain a convex obstacle dynamically computed. In this way, due to the flexibility of our system, this can be deployed in dynamic environments with walking humans, enhancing its usage in social robotics environments.

Despite these improvements, certain limitations remain. The precomputed turning rates, while simplifying real-time computation, may reduce flexibility in highly constrained environments. Future work could explore adaptive strategies for obstacle avoidance, integrating learning-based methods to enhance decision-making in dynamic environments.

Overall, this project provided valuable insights into safe gait planning for bipedal robots and reinforced the effectiveness of LDCBF-based navigation. Our contributions offer a promising direction for improving safety-critical real-time locomotion, with potential applications in both simulation and real-world robotic deployments.

References

[1] Chengyang Peng, Victor Paredes, Guillermo A. Castillo, and Ayonga Hereid1. Real-time safe bipedal robot navigation using linear discrete control barrier functions. *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[2] Jun Zeng, Bike Zhang, and Koushil Sreenath. Safety-critical model predictive control with discrete-time control barrier function, 2021.

[3] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Inverse Differential Kinematics*, chapter 3.5, pages 123–128. Springer, 2009.