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 **2**

2 3D-LIP Model with Heading Angles **3**
2.1 Model Definition 3

3 LIP-MPC: Gait planning with Model Predictive Control **5**
3.1 Heading Angle Preprocessing 5
3.2 Kinematic Constraints 6
3.2.1 Walking Velocities Constraint 6
3.2.2 Leg Reachability Constraint 7
3.2.3 Maneuverability Constraint 7
3.3 Control Barrier Functions 7
3.3.1 Definitions 7
3.3.2 Linear Discrete Control Barrier Functions (LDCBF) 8
3.3.3 Our variation of the LDCBF 9

4 Simulations **11**
4.1 Simple Environment 11
4.1.1 Simulation employing our LDCBF 15

5 Our Improvements **18**
5.1 Subgoals and RRT* 18
5.2 Unknown Environments 22

6 Conclusion **27**

References **28**

1 Introduction

The term *humanoid*, indicates a robot characterized by a structure and kinematics similar to the human body. This type of robots, which are inherently underactuated due to the unilateral ground contacts, are designed to navigate and interact in environments structured for humans. Real-time safe navigation is a crucial task for humanoid robots in real-world applications. A path is considered safe if it does not collide with any obstacle while fulfilling the robot's dynamics and physical constraints. In order to carry out such complex task in real-time, path planning is usually decoupled from gait control, resulting in a significant reduction of the computational load.

The aim of this work is to 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 to be executed in real-time. It models the humanoid's walking dynamics by a Linear Inverted Pendulum, and leverages Model Predictive Control and Control Barrier Functions to deliver a collision-free path while satisfying specific constraints.
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

If the full dynamic model of the humanoid is used to simulate its motion, it becomes computationally impossible to perform joint path and gait planning, due to its high dimensionality and non-linearity. Therefore, a simplifying model must be used. For this scope Peng et al. introduced the "3D-LIP Model with Heading Angle", which describes the discrete dynamics of the Center of Mass (CoM) similarly to the one of an inverted pendulum in three dimensions.

2.1 Model Definition

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

$$\mathbf{x} := \big(p_x,\ v_x,\ p_y,\ v_y,\ \theta\big)^T \in \mathcal{X} \subset \mathbb{R}^5,$$

$$\mathbf{u} := \big(f_x,\ f_y,\ \omega\big)^T \in \mathcal{U} \subset \mathbb{R}^3,$$

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.

The Linear Inverted Pendulum formulation assumes that, the height H of the CoM is constant during the motion. According to [1], we can express the CoM acceleration as:

$$\begin{cases} \dot{v}_x = \dfrac{g}{H}(p_x - f_x) \\ \dot{v}_y = \dfrac{g}{H}(p_y - f_y) \end{cases}, \tag{1}$$

with g as the magnitude of the gravitational acceleration. Based on this expression and the previously defined state $\mathbf{x}$ and input $\mathbf{u}$, we can define the 3D-LIP dynamic model in the continuous time space as:

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\theta} \end{pmatrix} = \mathbf{A_C}\,\mathbf{x}(t) + \mathbf{B_C}\,\mathbf{u}(t) = \begin{pmatrix} \mathbf{A_h} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times1} \\ \mathbf{0}_{2\times2} & \mathbf{A_h} & \mathbf{0}_{2\times1} \\ \mathbf{0}_{1\times2} & \mathbf{0}_{1\times2} & 0 \end{pmatrix} \mathbf{x}(t) + \begin{pmatrix} \mathbf{B_h} & \mathbf{0}_{2\times1} & \mathbf{0}_{2\times1} \\ \mathbf{0}_{2\times1} & \mathbf{B_h} & \mathbf{0}_{2\times1} \\ 0 & 0 & 1 \end{pmatrix} \mathbf{u}(t),$$

$$\text{with:} \quad \mathbf{A_h} := \begin{pmatrix} 0 & 1 \\ \beta^2 & 0 \end{pmatrix}, \quad \mathbf{B_h} := \begin{pmatrix} 0 \\ -\beta^2 \end{pmatrix}, \quad \beta := \sqrt{\frac{g}{H}}.$$

By further assuming that the duration of a single humanoid's step is fixed and equal to T, we can derive the closed-form step-to-step discrete dynamics of the 3D-LIP model:

$$\mathbf{x}_{k+1} = \mathbf{A_L}\,\mathbf{x}_k + \mathbf{B_L}\,\mathbf{u}_k, \tag{2}$$

with:

$$\mathbf{A_L} \coloneqq \begin{pmatrix} \mathbf{A_d} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{A_d} & 0 \\ \mathbf{0} & \mathbf{0} & 1 \end{pmatrix}, \qquad \mathbf{B_L} \coloneqq \begin{pmatrix} \mathbf{B_d} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{B_d} & 0 \\ 0 & 0 & T \end{pmatrix},$$

$$\mathbf{A_d} \coloneqq \begin{pmatrix} \cosh(\beta\,T) & \frac{\sinh(\beta\,T)}{\beta} \\ \beta\,\sinh(\beta\,T) & \cosh(\beta\,T) \end{pmatrix}, \qquad \mathbf{B_d} \coloneqq \begin{pmatrix} 1-\cosh(\beta\,T) \\ -\beta\,\sinh(\beta\,T) \end{pmatrix}.$$

In the following chapter, these discrete 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 (2) 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 in 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:

$$
\begin{aligned}
J^* = \min_{\mathbf{u}_{0:N-1}} & \sum_{k=1}^{N} q(\mathbf{x}_k) \\
\text{s.t.} \quad & \mathbf{x}_k \in \mathcal{X}, \quad k \in [0, N] \\
& \mathbf{u}_k \in \mathcal{U}, \quad k \in [0, N-1] \\
& \mathbf{x}_{k+1} = \mathbf{A_L}\mathbf{x}_k + \mathbf{B_L}\mathbf{u}_k, \quad k \in [0, N-1] \\
& \mathbf{c}_l \leq \mathbf{c}(\mathbf{x}_k, \mathbf{u}_k) \leq \mathbf{c}_u, \quad k \in [0, N-1],
\end{aligned}
\tag{3}
$$

where $q(\mathbf{x}_k)$ is the cost function to minimize along the prediction horizon. It drives the humanoid toward the goal by minimizing the distance between its current position and the target position. It is defined as:

$$
q(\mathbf{x}_k) = \left(p_{x_k} - g_x\right)^2 + \left(p_{y_k} - g_y\right)^2 \qquad \forall k \in [1, N]\,,
$$

where (g_x, g_y) is the goal position. The LIP dynamics is included in the MPC definition to specify how the future states are predicted. Whereas, all the constraints that the optimization problem is subject to are captured by $\mathbf{c}(\mathbf{x}_k, \mathbf{u}_k)$. They are all expressed linearly, and they include the walking velocities, leg reachability, and maneuverability constraints, and the linear control barrier function, which 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, that are kept fixed during the MPC

horizon.
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. Since Peng et al. did not specify how θ and ω were computed, we devised our own approach. We aim to rotate the robot to align its sagittal axis with the goal as soon as possible, and it is achieved with the following formulae:

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

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

where and θ represent the position and orientation of the humanoid at the start of the simulation time step, and T is the robot's sampling time. ω_{min} and ω_{max} are the bounds on the robot turning rate, to consider 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.

3.2 Kinematic 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

$$\begin{pmatrix} v_{x_{\min}} \\ v_{y_{\min}} \end{pmatrix} \leq \begin{pmatrix} \cos\theta_k & \sin\theta_k \\ -\sin\theta_k & \cos\theta_k \end{pmatrix} \begin{pmatrix} v_{x_k} \\ s_v v_{y_k} \end{pmatrix} \leq \begin{pmatrix} v_{x_{\max}} \\ v_{y_{\max}} \end{pmatrix}, \qquad \forall k \in [0,\, N-1]\,. \tag{4}$$

This constraint limits the longitudinal and the lateral velocities: the former is obtained by multiplying the velocities vector by $(\cos\theta_k \sin\theta_k)$, and the latter by multiplying the same vector by $(-\sin\theta_k \cos\theta_k)$. We can also think about this multiplication as the transformation of the velocities from the global to the robot's local reference frame (which is aligned with the current orientation). The term s_v defines which foot is the stance: it is 1 for the right foot, and -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. 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

$$\begin{pmatrix} -l_{\max} \\ -l_{\max} \end{pmatrix} \leq \begin{pmatrix} \cos\theta_k & \sin\theta_k \\ -\sin\theta_k & \cos\theta_k \end{pmatrix} \begin{pmatrix} p_{x_{k+1}} - p_{x_k} \\ p_{y_{k+1}} - p_{y_k} \end{pmatrix} \leq \begin{pmatrix} l_{\max} \\ l_{\max} \end{pmatrix}, \qquad \forall k \in [0,\, N-1]\,. \tag{5}$$

This constraint prevents the inputs computed at time k from moving the CoM too far away from the previous position. It does so by projecting the displacement of the CoM position on the longitudinal and lateral axes, and ensuring that its absolute value is lower than $l_{\max}$, which is the maximum distance reachable by the swing foot in both directions.

3.2.3 Maneuverability Constraint

$$\begin{pmatrix} \cos\theta_{k+1} & \sin\theta_{k+1} \end{pmatrix} \begin{pmatrix} v_{x_{k+1}} \\ v_{y_{k+1}} \end{pmatrix} \leq v_{x_{\max}} - \frac{\alpha}{\pi}|\omega_k| \qquad \forall k \in [0,\, N-1]\,.$$

This constraint is meant to reduce the humanoid's forward velocity while it is turning. It ensures that the component of the CoM velocity in the robot's heading direction (given by the vector projection on the left side) in the next time step will be lower than its bound $v_{x_{\max}}$ diminished by a quantity that depends on the turning rate commanded at the current time step. α is one of the simulation hyperparameters.

3.3 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* S and its boundary ∂S as:

$$S \coloneqq \left\{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}) \geq 0\right\},$$
$$\partial S \coloneqq \left\{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}) = 0\right\}.$$

They 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[1] γ such that

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

[1] A continuous function $\alpha\colon [0, a) \to [0, \infty)$ is said to belong to class $\mathcal{K}$ if:

- it is strictly increasing;
- it is such that $\alpha(0) = 0$.

and the following holds:

$$\begin{aligned} &\Delta h(\mathbf{x}_k,\, \mathbf{u}_k) \coloneqq h(\mathbf{x}_{k+1}) - h(\mathbf{x}_k), \\ \forall \mathbf{x}_k \in S \qquad &\exists \mathbf{u}_k \text{ s.t. } \Delta h(\mathbf{x}_k,\, \mathbf{u}_k) \geq -\gamma(h(\mathbf{x}_k)). \end{aligned} \tag{6}$$

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 (6) becomes:

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

This means that if the state starts from the safety set S, there must exist an input that, when applied, produces a state which will still be in S: namely, S is invariant.

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. 1). The safe region is the latter, and it is defined as:

$$h\left(\vec{x}\right) = \eta^T \left(\vec{x} - c\right) \geq 0, \tag{7}$$

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 (1).

If multiple obstacles are in the environment, the safe region is the intersection of the regions produced by each LDCBF.
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 never collides with the obstacles.

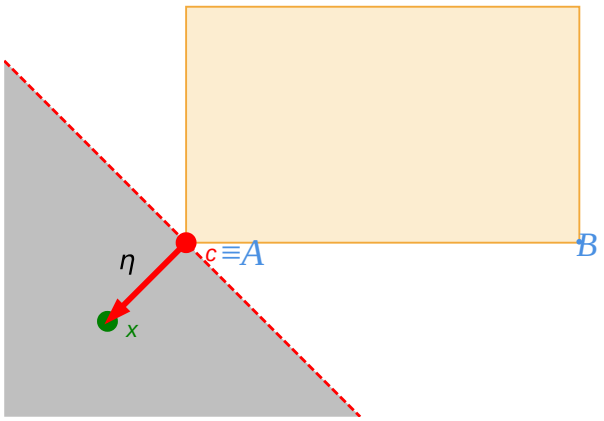


Figure 1: In this figure, the CoM is represented as a green point, and the obstacle is a blue rectangle. The point on the obstacle's boundary that is closest to $\vec{x}$ is the red one, which represents c. The red vector connecting c to $\vec{x}$ is η. The safe region defined by this LDCBF is the gray area. In that half-plane, the humanoid's CoM is free to move without hitting any obstacle.

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:

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

where $\xi = 1$ 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 (1) 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:

$$h\left(\vec{x}\right) = \eta^T \left(\vec{x} - c\right) - \delta \geq 0, \tag{8}$$

where c and η are the previously defined vectors, while δ is the distance the CoM must keep from c (i.e., from the obstacle's boundary) to satisfy the constraint. The result is shown in Figure 2

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.

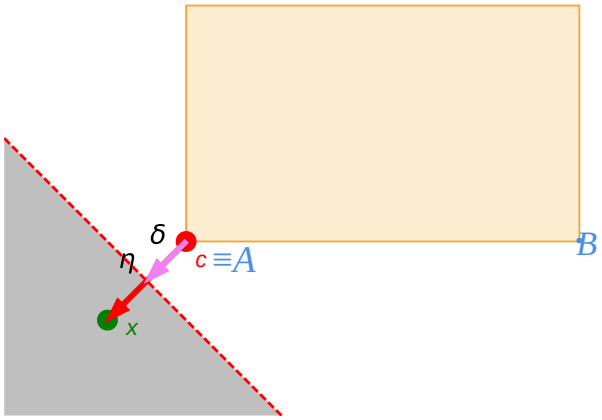


Figure 2: This plot shows the same environment of Figure 1, but here expression (8) with $\delta > 0$ was used as LDCBF. It is clear that the safe zone is not adjacent to the obstacle, and $\vec{x}$ maintains a distance δ from the obstacle.

4 Simulations

In this chapter, we will test the previously described MPC in a virtual environment. 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 hyperparameters 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
$\left(v_{x_{min}},\, v_{y_{min}}\right)$	$(-0.1,\, -0.1)$ m s^{-1}
$\left(v_{x_{max}},\, v_{y_{max}}\right)$	$(0.8,\, 0.4)$ m s^{-1}

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 3 and 4.

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 4), 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.

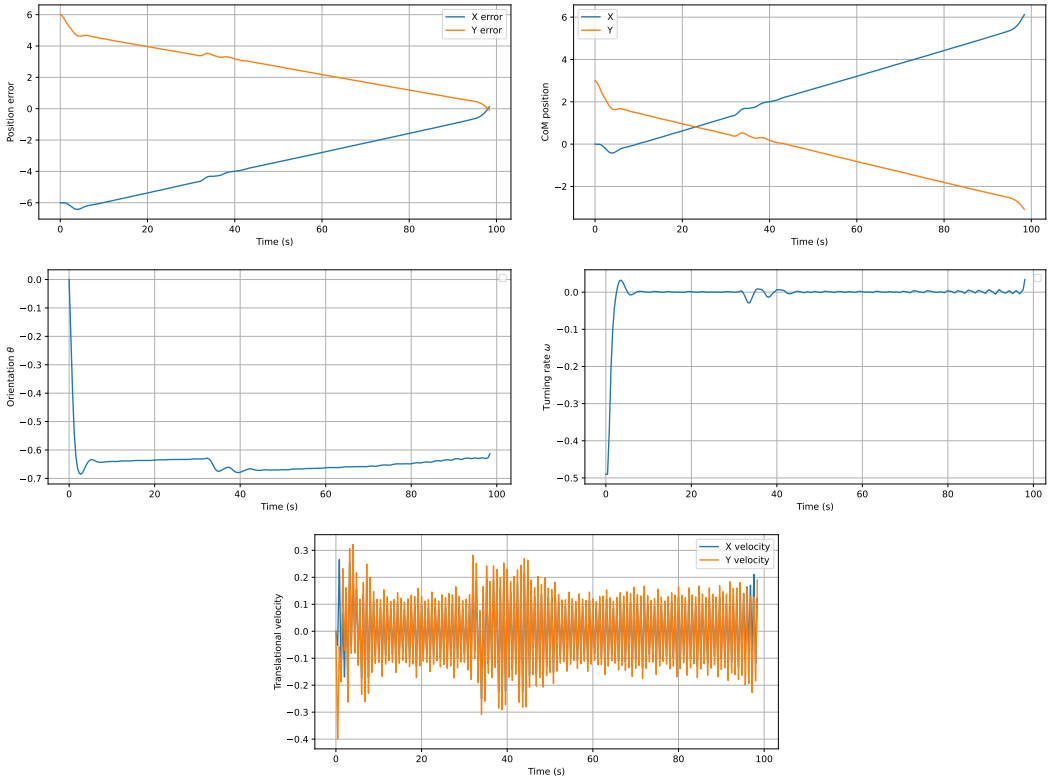


Figure 3: These figures depict the evolution of the humanoid's state and the input throughout the simulation in the base environment employing our LDCBF. The top-left plot shows the error between the CoM and the goal position, while the top-right represents how the CoM evolves along the x- and y-axis. The middle left and right plots show the theta and omega evolution, respectively. The bottom plot represents the translational velocities along the x- and y-axis.

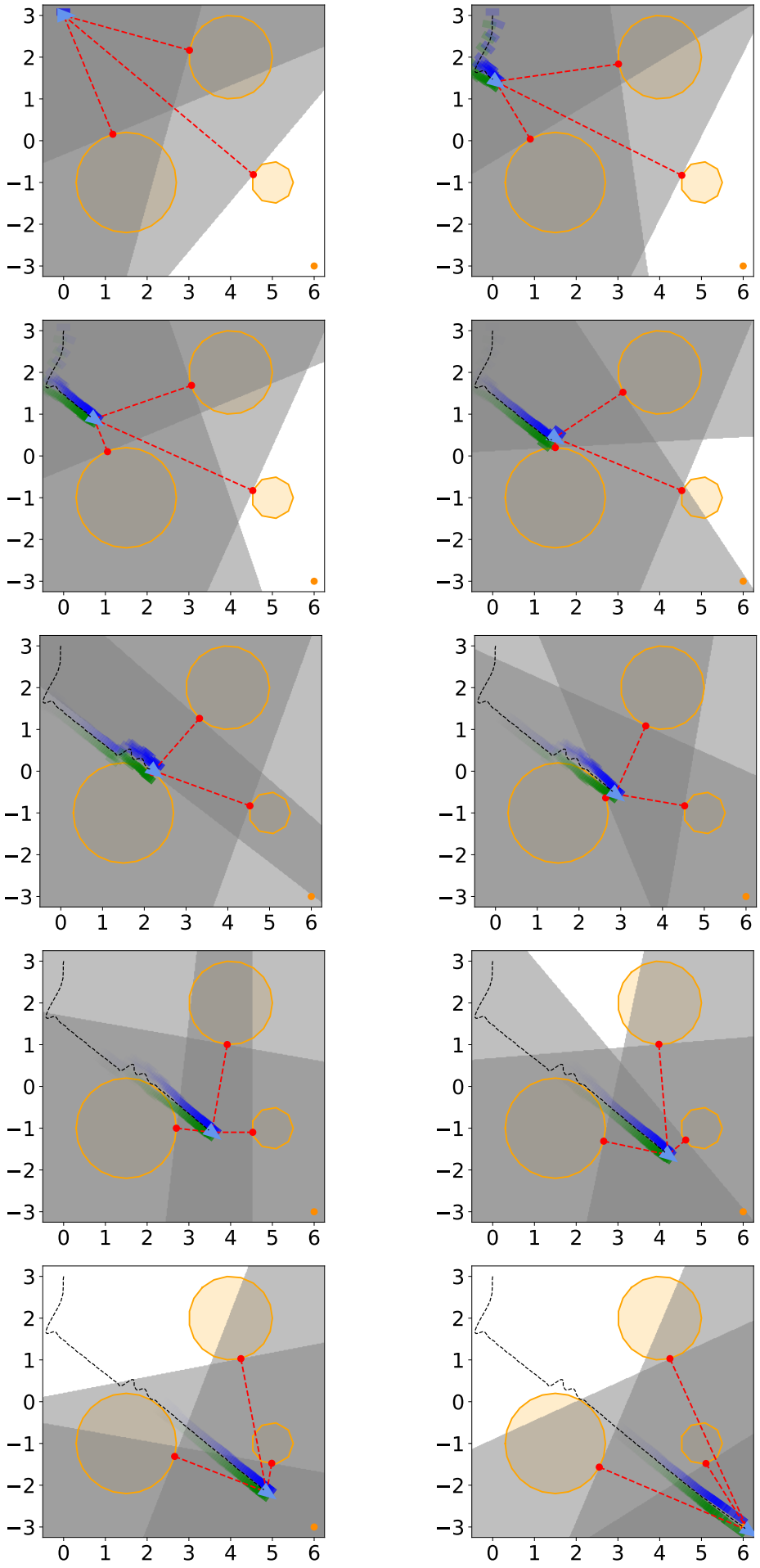


Figure 4: Simulation with circular obstacles.

4.1.1 Simulation employing our LDCBF

From an accurate analysis of the previous simulation's frames, it can be observed that, despite the robot accomplishing the mission of reaching the goal while satisfying all the constraints, when it gets very close to the obstacles, its feet overlap with the obstacles. This occurs due to the LDCBF defined by Peng et al., which only ensures that the CoM never collides with the obstacles. However, this strategy produces a path that is impractical for the full-body motion of the humanoid (as already pointed out in 3.3.3). Therefore, we proposed our variant of their LDCBF, and in Figures 5 and 6 we show the results of the simulation that constraints the MPC with the LDCBF defined at 8, setting $\delta = 0.3$.

From the plots, we notice that the robot reaches the boundary of the safe area – no longer coincident with the obstacle's edge – earlier than the previous simulation, and it reorients itself with the modality exposed before. The same situation arises around the second 70 of the simulation, when a turning velocity is commanded to keep a distance δ from the last obstacle.

The simulation frames in Figure 6 reveal that the humanoid's footsteps never intersect the obstacles. Subsequently, the LDCBF contributed to designing a motion plan suitable for the humanoid in real-world conditions.

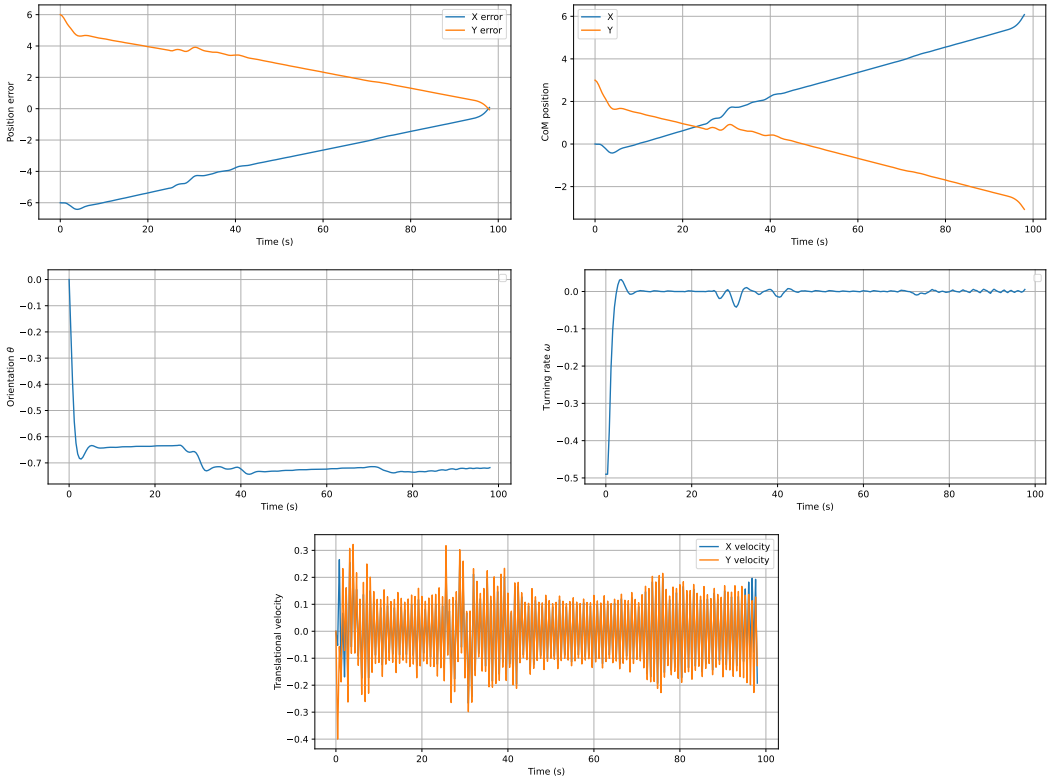


Figure 5: These figures depict the evolution of the humanoid's state and the input throughout the simulation in the base environment employing our LDCBF. The top-left plot shows the error between the CoM and the goal position, while the top-right represents how the CoM evolves along the x- and y-axis. The middle left and right plots show the theta and omega evolution, respectively. The bottom plot represents the translational velocities along the x- and y-axis.

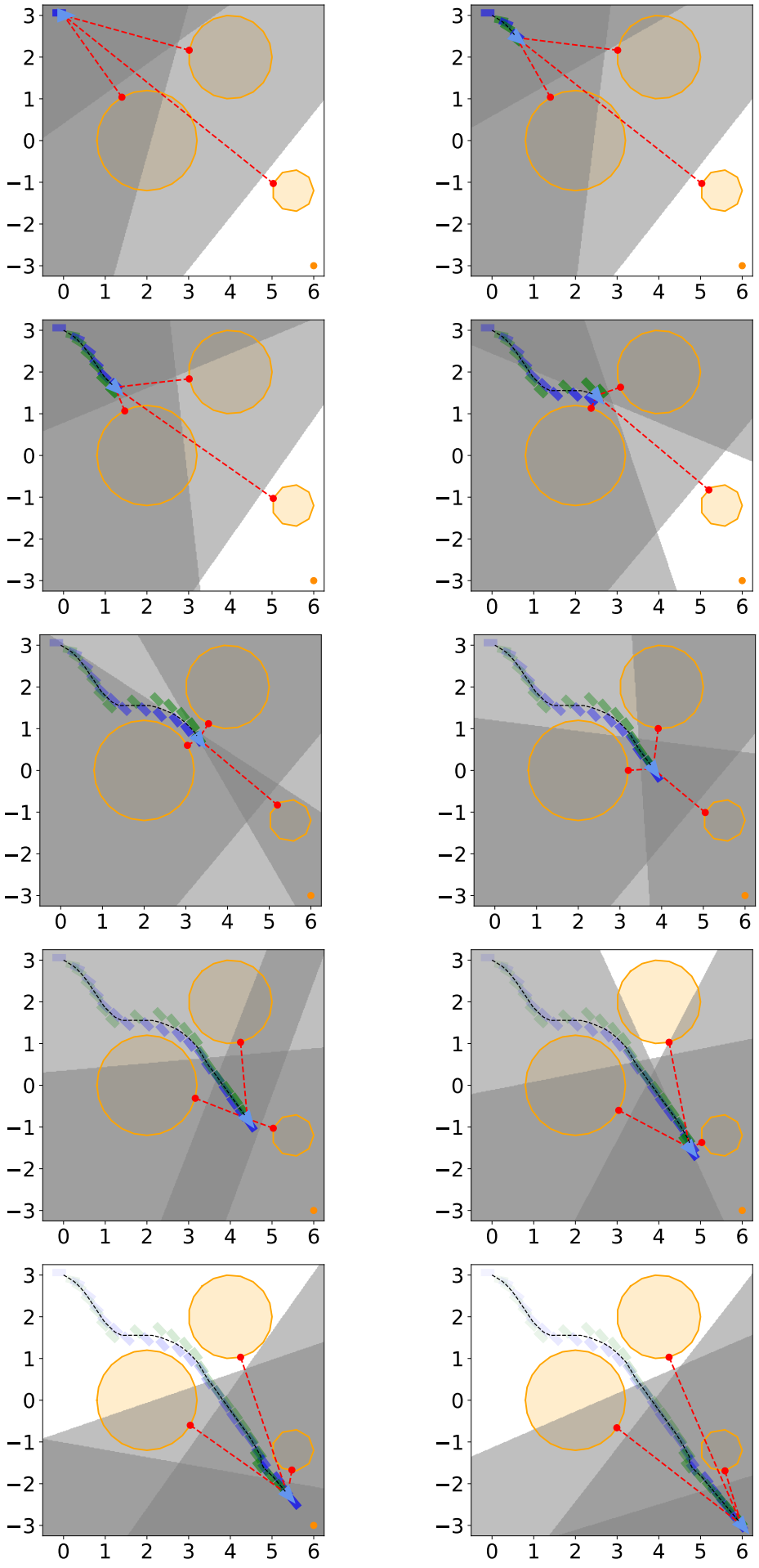


Figure 6: Simulation with circular obstacles employing our LDCBF (with $\delta = 0.3$).

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 7a), 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:

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

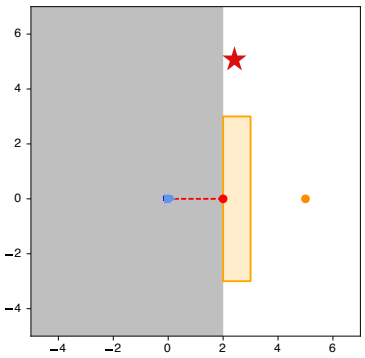
It is the sum of two addends: the cost of the path from i to the start node, and the Euclidean distance between the position represented by i and the one represented by j, multiplied by the exponential of the negative clearance of the new node (namely, the distance between j and the closest obstacle).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

Figures 7a and 7b 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 9). The MPC is requested to provide the inputs that drive the humanoid

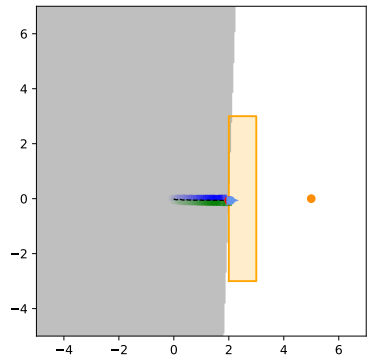
through the sequence of subgoals. In this way, the robot can pass by the red star, overcome the obstacle, and finally reach the goal.

From the evolution of the state (illustrated in Figure 10) 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 7

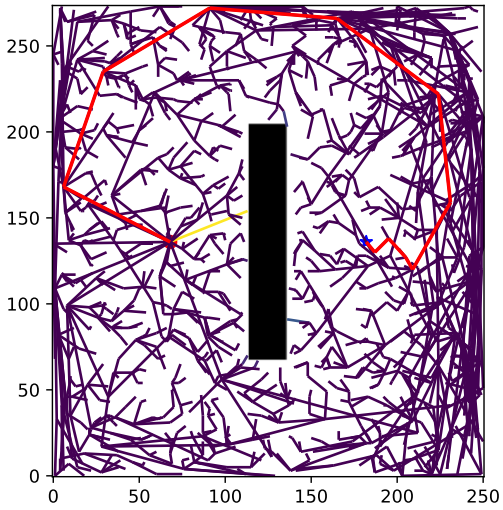


Figure 8: This figure illustrates the tree computed by the RRT* algorithm. The path is represented as a red line, the start position as a red point, and the goal as a blue star. The RRT is executed and illustrated inside an occupancy grid representing the original (continuous) workspace.

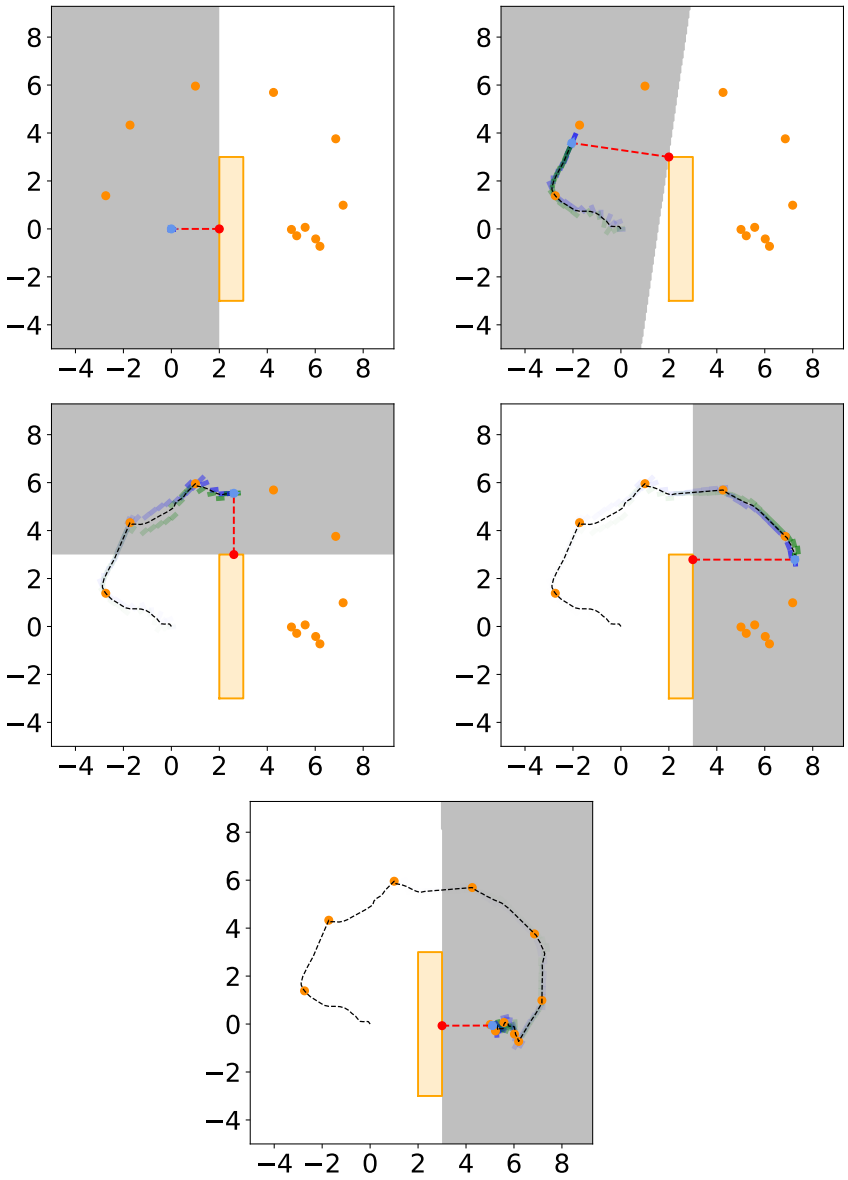


Figure 9: Simulation with RRT*-defined subgoals (represented as orange points).

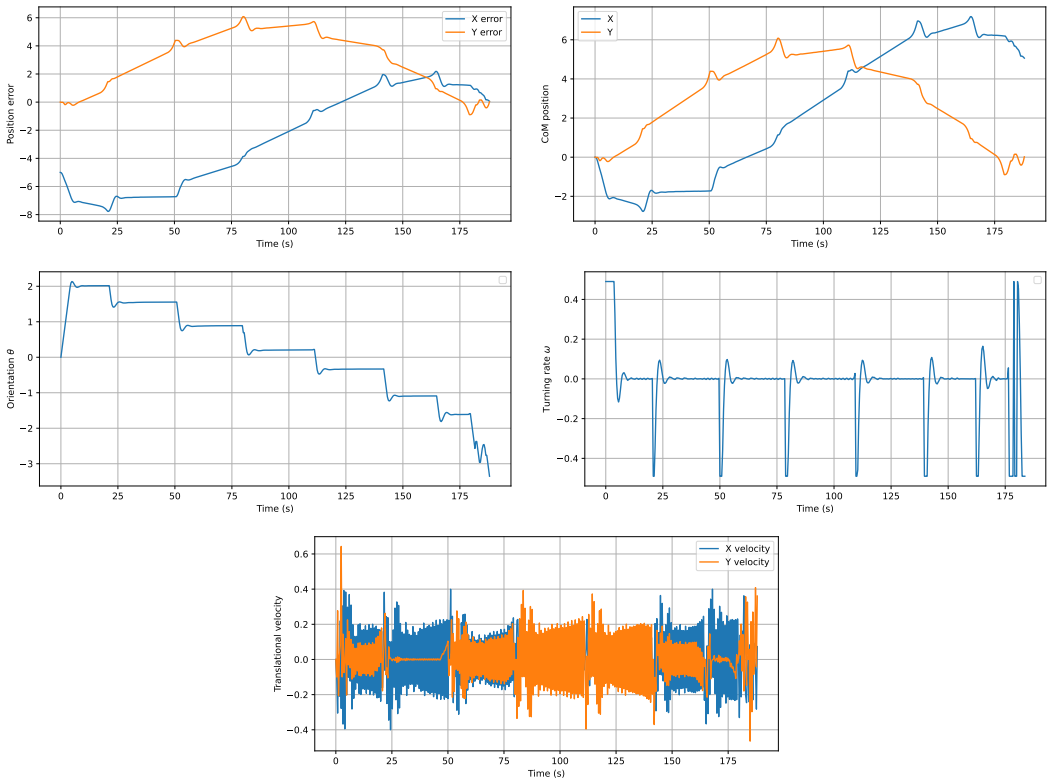


Figure 10: These figures depict the evolution of the humanoid's state and the input throughout the simulation in the base environment employing our LDCBF. The top-left plot shows the error between the CoM and the goal position, while the top-right represents how the CoM evolves along the x- and y-axis. The middle left and right plots show the theta and omega evolution, respectively. The bottom plot represents the translational velocities along the x- and y-axis.

5.2 Unknown Environments

In unknown environments, 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 affected by a 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 having $\varepsilon = 0.3$, to group nearby points into clusters that represent inferred obstacles (see Figure 11). With the integration of a LiDAR system, our framework is capable of replicating the previously described performance in real-word scenarios as well. The ε parameter needed by DBSCAN specifies the radius of the neighborhood of each point, that is needed for correctly assign each point to a cluster. In other words, ε is the maximum distance between two samples for one to be considered as in the neighborhood of the other. The other parameters are left as default. Table 2 summarizes the additional parameters of the system in unknown environment settings.

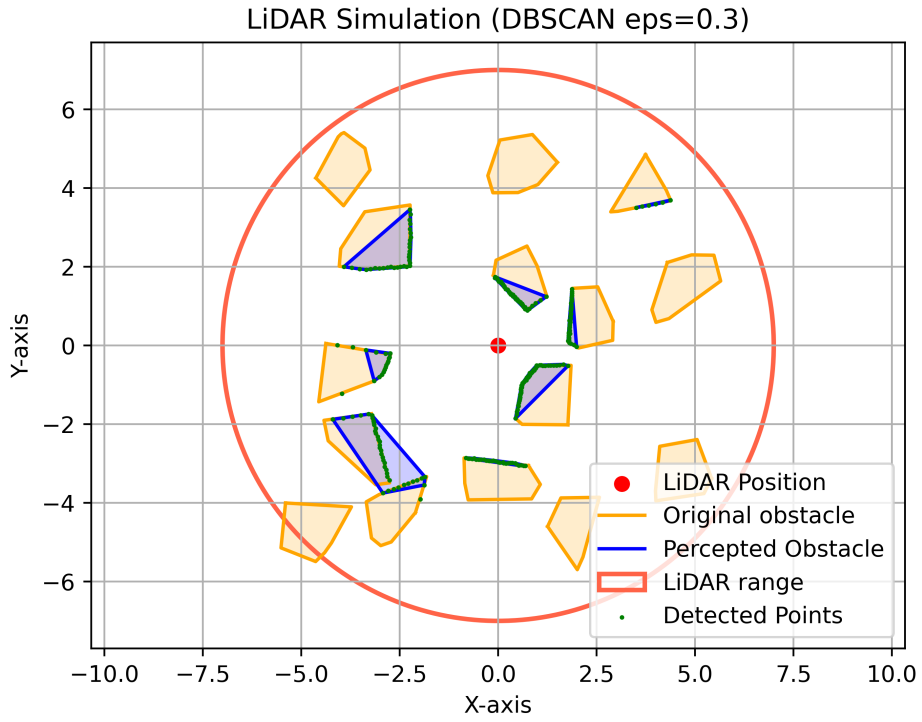


Figure 11: Percepted obstacles using 2D LiDAR measurements and DBSCAN. Here $\varepsilon = 0.3$, a resolution of 360° and LiDAR readings are affected by a zero-mean Gaussian noise.

Unknown environments additional Parameters	
Parameter	**Value**
ε	0.3
minPts	5 (default)
Metric	Euclidean

Table 2: The Unknown Environment additional parameters used for the following simulation. The *minPts* parameter is left as default, and in the library we are using for DBSCAN (i.e. Sklearn) it is called `min_samples`.

The additional parameters in Table 2 need a further tuning of the system, in order to achieve desired performance in real world scenarios. 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 MPC computes one LDCBF for each of these inferred obstacles. In this way, the navigation system can mirror the behavior of the base simulation while accounting for realistic environmental complexity; and the humanoid is able to navigate safely

despite not knowing the environment a priori, relying solely on the obstacles inferred dynamically from its sensor data.

The frames evolution in Figure 12 shows that using this approach, the robot is able to reach the goal also in highly packed environments. The evolution of the state variables (see Figure 13) shows almost no difference with respect to the base simulation; the main difference occurs on the overall behaviour of the humanoid in the environment: its flexibility in handling only a subset of obstacles, allows the robot to navigate in challenging crowded environments easily; moreover, the robot relies only on its perception, allowing for a safe navigation in real-world scenarios.

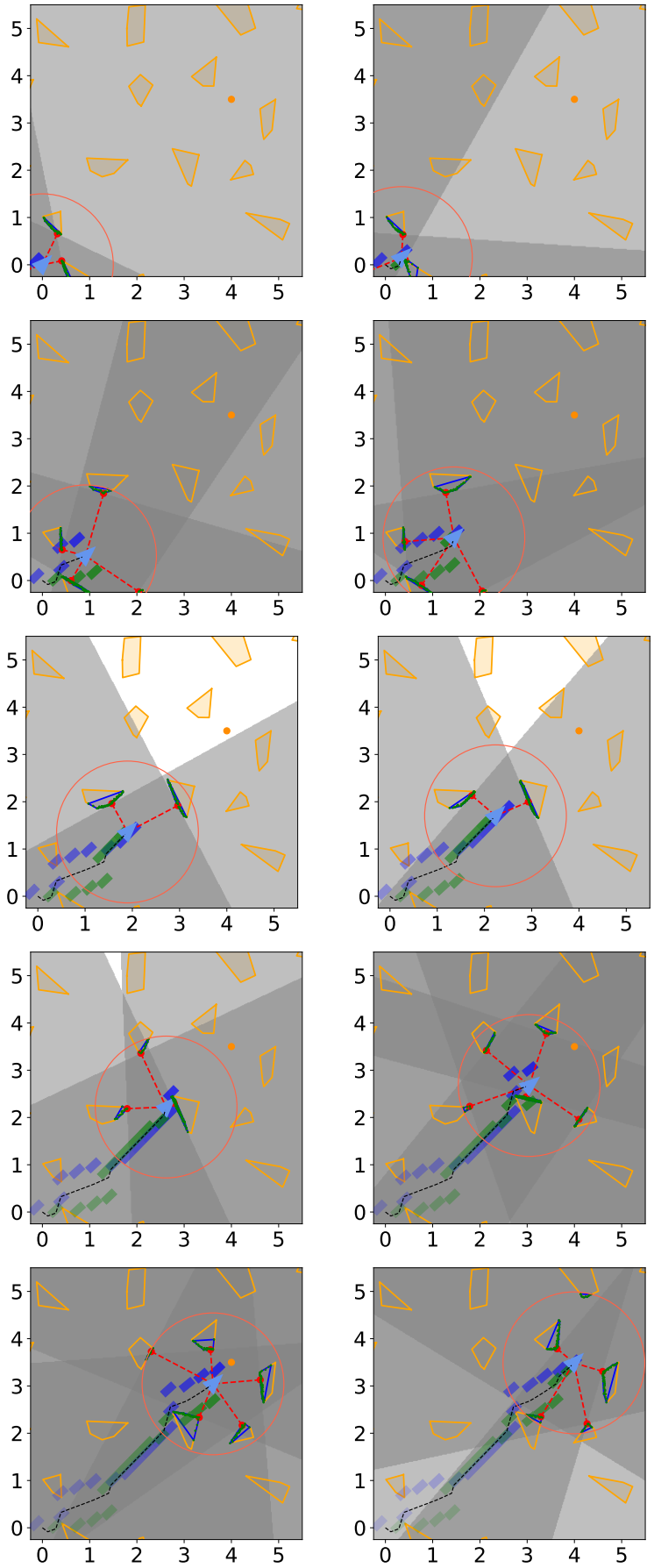


Figure 12: Simulation in unknown environment. The small green dots are the 2D LiDAR readings, the red circle that surrounds the robot is the LiDAR range, while the blue convex polygons are the inferred obstacles.

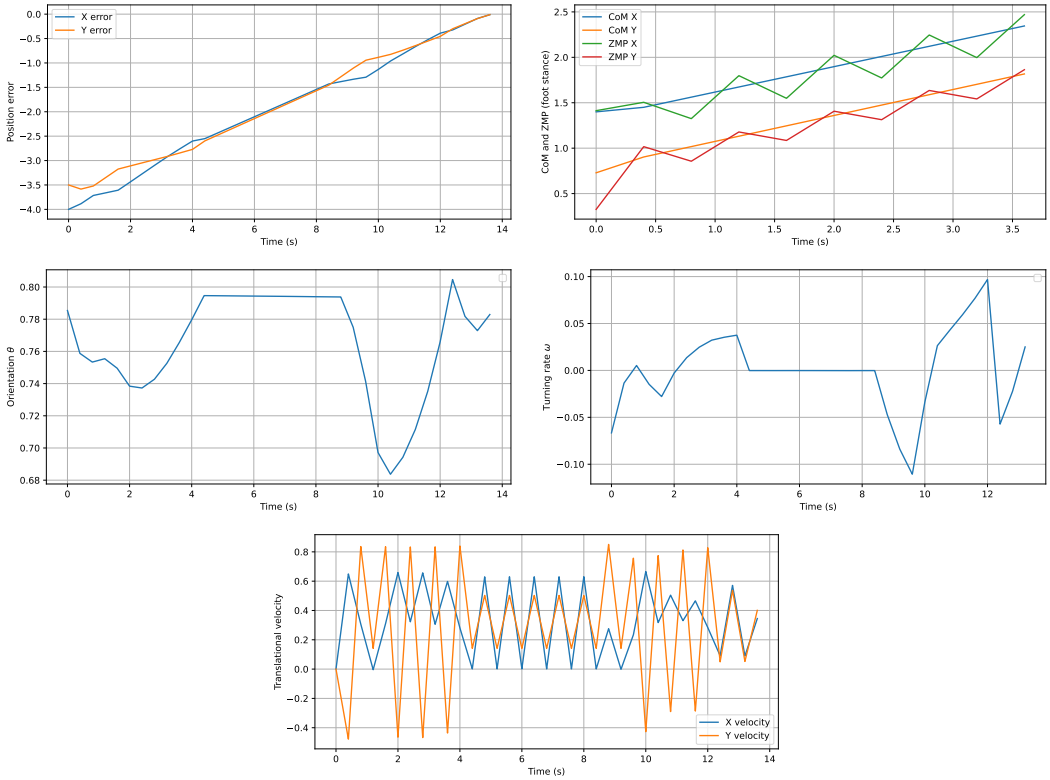


Figure 13: These figures depict the evolution of the humanoid's state and the input throughout the simulation in the base environment employing our LDCBF. The top-left plot shows the error between the CoM and the goal position, while the top-right represents how the CoM evolves along the x- and y-axis. The middle left and right plots show the theta and omega evolution, respectively. The bottom plot represents the translational velocities along the x- and y-axis.

6 Conclusion

In this project, we successfully reproduced the methodology presented by Peng et al. [1], implementing a real-time safe navigation framework for bipedal robots using Linear Discrete Control Barrier Functions (LDCBFs). We 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 real-world situations, 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 and to ensure the feasibility of the generated path, while maintaining real-time performance. Our simulations proved that this adjustment effectively increased the reliability of navigation without introducing significant computational overhead.

Moreover, for safe gait planning with limited FOV, we leveraged LiDAR scans of the surrounding scene and applied DBSCAN algorithm to obtain dynamically computed convex obstacles. Thanks to its flexibility, our system can be deployed in real-world environments (e.g., in a room with walking humans). However, the MPC alone is not enough to lead the robot to the goal in every situation. For this reason, we integrated in our framework an algorithm to reach the goal by travelling through a sequence of subgoals computed by RRT*.

Despite these improvements, certain limitations remain. The precomputed turning rates, while simplifying real-time computation, 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 proved the effectiveness of MPC-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.