Biped Walking Pattern Generation based on Spatial Quantized Dynamics

Dario Zurlo

01 April 2021

1 Introduction

Control a bipedal robot locomotion is a challenging task due to complexity of the system that needs to maintain constantly the balance on its two legs during the walking, many techniques has been studied in the past to generate static and dynamic biped gait on flat and rough terrain. First kind of locomotion developed is static walking, i.e. locomotion in which the robot moves slowly. The reason why this kind of locomotion was the first implemented, is because the joint actuators were big and the computation power was poor. The second one is the dynamic walking that is the non static locomotion where the walking speed is higher. Nowadays the latter can be implemented on a robot more easily thank to the smaller joint actuators and the high computation power. This great computational power has opened the way for other control techniques, some of them are no more based on the kinematic model of the robot such as reinforcement learning techniques or in general techniques based on artificial intelligence, and other are based on optimal control in which the aim is to determine the control law, that applied to the system optimizes its operation with respect to some pre-estabilished evaluation criteria, always taking into account the limit of the system. The latter is the one which the project is based on.

2 Problem Definition

The aim of this project is to reproduce some of the results of bipedal gait generation based on Zero Moment Point (ZMP) with a novel system modeling, named Spatially Quantized Dynamics (SQD) introduced in [1] and show how this technique is more flexible with respect to the classic one, based on time quantization, in order to allow the robot to walk with stretched knees.
The notion of ZMP has never been introduced by a formal definition [2], the notion that we are going to use is that the ZMP is the point with respect to which the moment of the contact force is zero in the horizontal direction. The analysis is performed by considering only horizontal and flat ground both in the sagittal and coronal plane, without considering any obstacle during the walking, this is an enormous simplification but the reason why we do all these simplifications is because we want to show why this technique is suitable in the case of stretched knees.
The ZMP based gait generation is a widely known in the area of walking robot in which the aim is to design a ZMP trajectory such that it always lies in the support polygon (SP), that is a sufficient condition for dynamical balance; this is done by taking the CoM trajectory such that the ZMP moves as planned. The steps followed to design the ZMP based gait are, first of all, design the steps sequence and the relative SP, then design the ZMP reference that it is always inside the SP and then compute the CoM trajectory such that the ZMP moves as planned. Once the CoM trajectory is available then it is possible to move the legs such that the CoM and the steps move as planned. The steps illustrated are independent of the model used, the common one discretized with respect to time or the one discretized with respect to space.
The first problem that arises is due to the complexity of the full humanoid dynamics necessary to directly control the ZMP, for this reason the control of the ZMP is done by controlling the CoM trajectory, that is easier, because it is possible to use a simple linear model called Linear Inverted Pendulum (LIP), that relates the evolution of the CoM, that is the output, with the evolution of the ZMP, that is the input. Where the CoM can be varied by changing the robot configuration through its joints. The dual model is the Cart Table model (CT) suitable to track the ZMP, that in this case is the output and the CoM is the input.
The novelty introduced in the paper on which this project is based on, is in the way in which the dynamics of our robot is discretized. In fact, instead of discretizing the model with respect to time as usual, the model is discretized with respect to the space. This renders the problem more difficult to treat, as we will see later, but the advantage is that it is possible to realize versatile

gait generation.
To solve this problem it has been used nonlinear optimization techniques; differently form the paper, I optimized the cost index on the full horizon using the standard Matlab function, instead of using the the Matlab toolbox of Differential Dynamic Programming (DDP) developed by Tassa, Mansard and Todorov [3] as the authors did in [1]. The results are the same, the difference is in the computation time.
The last step, once the CoM of the robot is computed, is to generate an array of vectors containing all the configurations of the joints; in order to achieve this task it is necessary to compute the inverse kinematics numerically, since the closed form solution can be difficult to find and also because we want to obtain a robust solution against kinematic singularities. The technique used in [1] is the inverse kinematics based on Levenberg-Marquardt method with robust damping [5], because they want also deal with stretched knees that is a singular configuration of the robot legs. We want also to deal with this latter problem firstly comparing three different numeric inverse kinematics techniques and at the end choosing the the best method.
Once the array of configuration is complete, to implement the the walking on the real robot it is necessary to perform a conversion form space domain into time domain. I will give just an idea on how to perform this conversion.

3 Gait Generation based on ZMP

The term gait is used to indicate a sequence of leg and body motions, that propels the robot along some path. In this project we want to study a particular gait generation based on ZMP. In order to find the point with respect to which the moment of the contact forces is zero,i.e. the ZMP, we consider the Euler's rotation equation, that describes the rotation of a rigid body.

$$(\boldsymbol{c}-\boldsymbol{o})\times M\ddot{\boldsymbol{c}}+\dot{\boldsymbol{L}}=\sum_i(\boldsymbol{p_i}-\boldsymbol{o})\times\boldsymbol{f_i}-(\boldsymbol{c}-\boldsymbol{o})\times M\boldsymbol{g} \tag{1}$$

Where $\boldsymbol{c}$ is the position of the CoM, $\boldsymbol{o}$ is the generic position with respect to which we want to compute the moment, M is the total mass of the robot, $\boldsymbol{L}$ is the angular momentum of the robot with respect to its CoM given by the sum of the angular momentum of each robot link, $\boldsymbol{f}_i$ is the force applied at point $\boldsymbol{p_i}$, that is the contact point, and $\boldsymbol{g}$ is the gravitational acceleration. We now consider the torque around the ZMP given by

$$\boldsymbol{\tau}=\sum_i(\boldsymbol{p_i}-\boldsymbol{z})\times\boldsymbol{f_i} \tag{2}$$

That written in terms of vector components is

$$\boldsymbol{\tau}_x=\sum_i(p_{iy}-z_y)f_{iz}-\sum_i(p_{iz}-z_z)f_{iy} \tag{3}$$

$$\boldsymbol{\tau}_y=\sum_i(p_{iz}-z_z)f_{ix}-\sum_i(p_{ix}-z_x)f_{iz} \tag{4}$$

$$\boldsymbol{\tau}_z=\sum_i(p_{ix}-z_x)f_{iy}-\sum_i(p_{iy}-z_y)f_{ix} \tag{5}$$

the position of the ZMP can be obtained by settling (3) and (4) to zero. In fact in case of horizontal and flat plane, all the contact point along the z axis are the same and are also equal

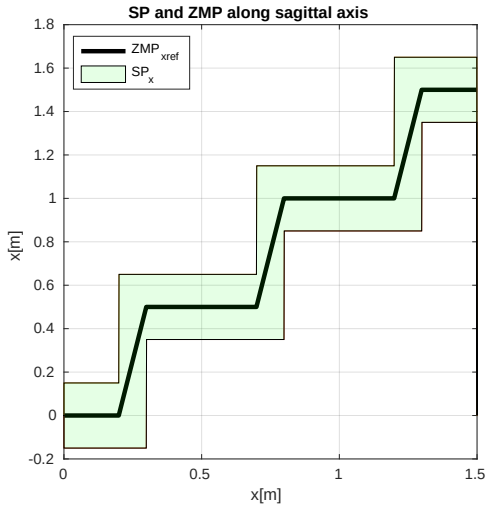

Figure 1: SP and desired ZMP along sagittal axis

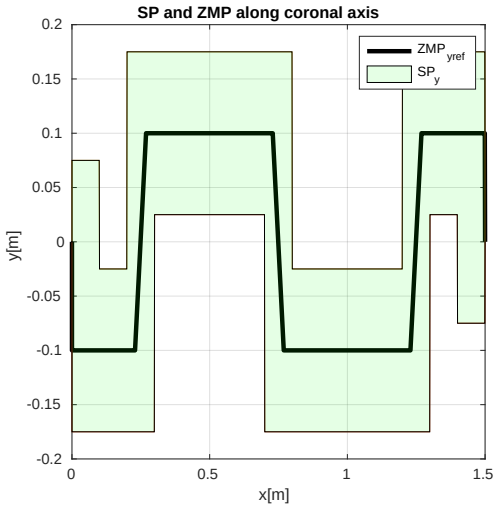

Figure 2: SP and desired ZMP along coronal axis

to the z component of the ZMP, then the second term of (3) and the first term of (4) become zero. This yields

$$\boldsymbol{z} = \frac{\sum_i \boldsymbol{p_i} f_i}{\sum_i f_i} \tag{6}$$

that is the point that we are looking for, under the assumption that the ground is flat and horizontal. The third component of (2) is in general different from zero, this means that there is torque due to the friction between the foot and the ground.

Now using the definition of the ZMP to guarantee the dynamical balance during walking, we need that the ZMP is at every instant inside the SP, that continuously changes during walking according to the cyclic alternation of 4 phases , if the robot is able to rotate the feet or 2 phases if the robot walks with flat feet.

In this project I consider that the robot is able to rotate its feet and then I consider all the four phases during walking. In figure 1 and 2 are shown the SP and the desired ZMP respectively along the sagittal axis and along the coronal axis.

It is clear that the first step is to define a set of robot steps as shown in the figure 3, through the foot steps planner, in order to define the SP. Once I have the support polygon, I design a suitable ZMP trajectory that is given as a reference of the optimization problem. The result of the optimization problem is the position of the ZMP and also the position of the CoM that makes the ZMP trajectory moves as I planned.

4 Dynamical Model

A full dynamical model of a humanoid robot with many DoF can be very difficult to obtain and above all it can hide the balance of the robot, then a simplified version of the model, that is enough accurate to describe the balance, is necessary. A possible approach, under the assumption that all the mass of robot is concentrated at the CoM and that the robot has massless legs, is to approximate the dynamics of the robot with the dynamics of an inverted pendulum with variable

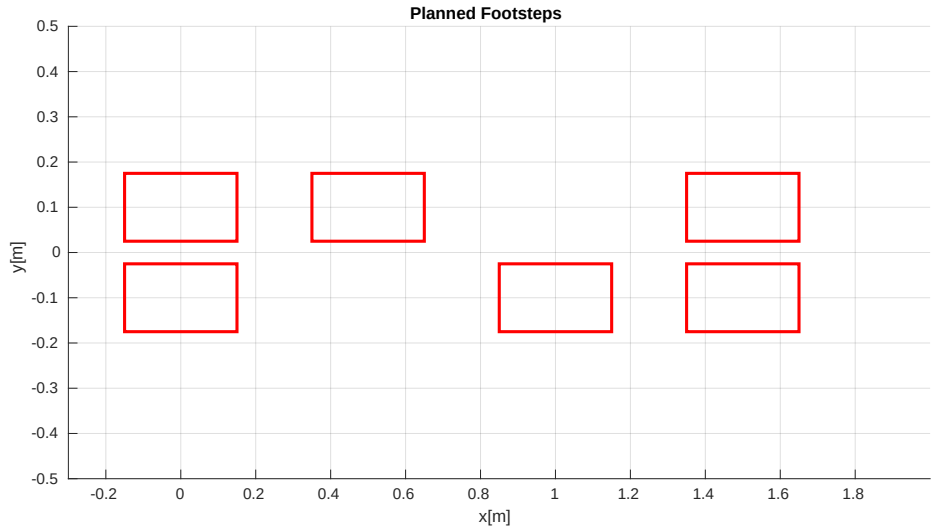

Figure 3: Three footsteps planned

length, as shown in figure 4. Using the Euler-Lagrangian formulation it is possible to obtain the following model for the inverted pendulum

$$\begin{aligned} r\ddot{\theta} + 2r\dot{r}\dot{\theta} - grsin(\theta) &= \tau/M \\ \ddot{r} - r\dot{\theta}^2 + gcos(\theta) &= f/M \end{aligned} \tag{7}$$

Where g is the gravity acceleration, M is the total mass of the robot, r is the length of the pendulum, θ is the angle of inclination with respect to the vertical, f is the force at the prismatic joint along the length and τ is the torque at the pivot, that in general is small and in case of dimensionless foot is zero. The idea to move the CoM along the longitudinal direction is to apply a force f to compensate the gravity, this force is given by the following equation

$$f = \frac{Mg}{cos(\theta)} \tag{8}$$

Then the longitudinal evolution of the CoM is given by

$$M\ddot{c}^x = Mgtan(\theta) = Mg\frac{c^x}{c^z} \tag{9}$$

Where $\boldsymbol{c}$ is the CoM position and the apex indicate the component of the vector x, y or z.

It is possible to obtain a similar model as (9) from eq (1) dividing by the forces along the z-axis and neglecting the variation of the angular momentum $\dot{L}$, that is the variation of the angular momentum of each joint; in this way we obtain the LIP model

$$\ddot{\boldsymbol{c}}^{x,y} = \frac{g^z}{c^z}(\boldsymbol{c}^{x,y} - z^{x,y}) \tag{10}$$

Where $z^{x,y}$ is the ZMP position along the axis x or y, that push away the CoM position and is the input of the model .

Since the CoM position along the x axis has a good correspondence with the horizontal hip displacement, we can expect that the robot dynamics can be expressed by the linear inverted

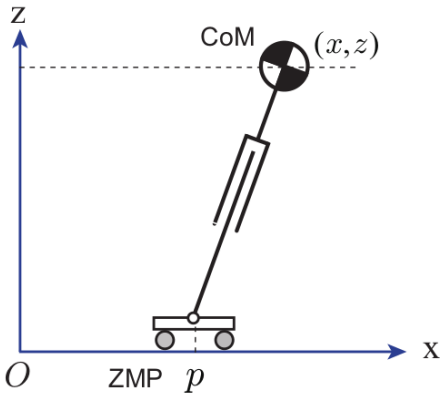

Figure 4: LIP model

pendulum model in which x represent both the hip position and the CoM position. Instead the CoM position along the z axis in the actual robot is not constant but we can assume it as the average of the CoM height during the whole walking. For simplicity I analyze, first of all, the problem along the sagittal axis and the equation (10) becomes

$$\begin{aligned} \ddot{x} &= \omega^2(x-p) \\ \omega &:= \sqrt{g/z} \end{aligned} \tag{11}$$

where in this case x is the hip position that coincides with the CoM position and p is the longitudinal ZMP position .

The model that I'm going to use is a discrete version of (11), but instead of discretizing with respect to time as usual, I discretize with respect to space by using a constant unit length $\Delta x = 0.001m$. This means that the position of the CoM evolves as follows

$$x_i = \Delta x \cdot i \quad (i = 1, 2, ...) \tag{12}$$

where the index i is the index for discretization. Using the variable time step

$$\Delta t_i = \frac{\Delta x}{v_i} \tag{13}$$

we obtain the Spatially Quantized Dynamics (SQD) of the LIP model equation.

$$v_{i+1} = v_i + \omega^2(x_i - p_i)\frac{\Delta x}{v_i} \tag{14}$$

This kind of discretization introduces a difficulty, because the dynamical model becomes nonlinear. As a matter of fact the sample time Δt_i is inversely proportional to the CoM velocity v; this is reasonable because if the CoM velocity is zero, then the time that the robot takes to travel from a point to the adjacent point tends to infinite; for the same reason, if the velocity tends to infinite, the movement is instantaneous. Then the sample time is no more constant.

As it is possible to see from the SQD there is a singularity when the CoM velocity is zero, then to overcome this problem we have to set the constraint on the CoM velocity

$$||v_i|| > \epsilon \tag{15}$$

with $\epsilon = 0.005m/s$.

5 Optimization

The first step to solve the problem with the SQD is to set the constrained optimization problem in order to find the optimal evolution of the ZMP and the CoM position and speed. In the optimization problem we have to identify the constraints and the function to minimize. The dynamical model is one of the constraints because we want that the solution satisfies the SQD. The other constraints are given by the evolution of the CoM (12) and by the bound on the CoM velocity (15), because the SQD has a singularity when the CoM speed is zero, then we have to bound the CoM speed. From this latter constraint, it is also possible to see that the problem is not only non linear but also non convex, in fact if we think about a sphere and we take a point on it, for example w and another point $-w$ then it is possible to connect this two points with a line that is not inside the admissible set. This fact entails that the solution may be not the optimal one, so we started with the optimization problem with a very short horizon length, to check if the solver can find a reasonable solution and then we extended the optimization problem to the whole horizon.
In order to completely define the optimization problem we need to specify the function to minimize and since the aims are to keep dynamic balance during walking and to realize desired walking speed, a suitable cost function is given by

$$J := \sum_{k}^{N} (v_k - v_k^{ref})^2 + \beta (p_k - p_k^{ref})^2 \tag{16}$$

Equation (16) together with (12), (14) and (15) forms the constrained optimization problem that we are going to solve. In the cost function v_k^{ref} is the CoM velocity reference and p_k^{ref} is the ZMP reference, both expressed as a function of the hip position and β is a weight to decide the importance of the ZMP position with respect to the CoM velocity.

Once the optimization problem is solved, we obtain the optimal evolution of the input p_k^* and the optimal evolution of both position and speed of the CoM x_k^* and v_k^*, respectively. All these trajectories are in function of the hip position, but it is possible to convert them from space domain into time domain by computing each instant of time for the $i-th$ spatial reference data by using the equation

$$t_i = \sum_{k=1}^{i-1} \frac{\Delta x}{v_k} \tag{17}$$

in this way we obtain the time at step i, that collected in an array can be used to express all the trajectory in the time domain

6 Kinematic Walking Pattern

Eventually, to completely define the Spatial Walking Pattern (SWP), we have to find all the joint angle values, then we have to solve the inverse kinematics from the CoM to the foot for each SQD step. The SWP is an off-line walking pattern composed by the ZMP position p_k^*, by the CoM speed v_k^* ,resulting form the optimization problem, and the joint values q_k^* resulting from the inverse kinematics.

$$\{p_k^*, v_k^*, q_k^*\} \quad (k = 0 \dots N) \tag{18}$$

The goal of the inverse kinematics is to compute the joint angle values that will cause the end effector to reach the desired position. While the forward kinematics is relatively easy to compute, because it requires just to evaluate a nonlinear function, the inverse kinematics requires to invert a nonlinear relationship. There are two possible ways to perform the inverse kinematics: one is to use analytical methods but it is possible only on simple chains; the other one is the numerical method that uses local approximation and iteration to converge on a solution. This method is more expensive and sensitive to the initial conditions, but more general and can be used on a generic chain. The method that I used is the latter because I want to find values also close to the singularities. I tried three different numerical methods. The first one is the Gradient method in which the aim is to minimize the error function , where r_d is the desired Cartesian position and $f(q)$ is the forward kinematics.

$$H(q) = \frac{1}{2}||r_d - f(q)||^2 \tag{19}$$

$$q_{k+1} = q_k - \alpha \nabla_q H(q_k) \tag{20}$$

In order to minimize this error, in the next step the joint values are computed by updating the current joint values in the direction in which the gradient decreases moving on a step α, as it is possible to notice from (20). The step $\alpha \in (0, 1]$ is an important parameter that must be chosen properly, in fact if we chose α too close to 1 the method may miss the solution; on the other hand, if α is close to 0 the convergence is quite slow. In order to choose this value it is necessary to trade off between accuracy and convergence rate.

With simple math it is possible to get the final update law

$$q_{k+1} = q_k + \alpha J^T(q_k)(r_d - f(q_k)) \tag{21}$$

As it is possible to see from (21), the update law does not require the Jacobian inversion, as a consequence this method is more robust against singularities.

The second method that I tried is the Newton method, that instead to solve a minimization problem it tries to solve the equation

$$r_d - f(q) = 0 \tag{22}$$

from this equation and using the Taylor expansion of the nonlinear function $f(q)$ it is possible to write

$$r_d = f(q) = f(q_k) + J(q_k)(q - q_k) + o(||q - q_k||) \tag{23}$$

If in equation (23) we consider only the first order approximation, it is possible to obtain the following updating law

$$q_{k+1} = q_k + J^{-1}(q_k)(r_d - f(q_k)) \tag{24}$$

However, since in the updating law there is the Jacobian inverse, when the solution approaches the singularity the inverse matrix becomes quite large and then this method it is not suitable in the neighbourhood of these points. In case of non square matrix it is possible to use the pseudo-inverse $J^{\#}(q)$ instead of the inverse. The advantage of this method is that it has quadratic convergence rate when close to a solution.

The third method, that is the one used by the authors in [1], is the Levenberg-Marquardt

method with robust damping, in particular it exploits the advantages of the gradient methods, i.e. robustness close to singularities, and the Newton method, i.e. fast convergence rate.

$$q_{k+1} = q_k + H_k^{-1} J^T W_e (r_d - f(q)) \tag{25}$$

$$H_k = J_k^T W_e J_k + W_n \tag{26}$$

The updating law is given by (25) where H_k is defined in (26) that is guaranteed to be regular and positive-definite, $W_e = diag\{w_{e,i}\}$ with $(w_{e,i} > 0 \ \forall i)$ and $W_n = diag\{w_{n,i}\}$ with $(w_{n,i} > 0 \ \forall i)$ is the damping factor. The damping factor is the key point of this method because it allows the updating law to switch from the Newton method if W_n tends to zero to the gradient method if W_n tends to infinity. A possible technique in order to chose the damping parameter dynamically, as proposed in [4], is based on the manipulability measure of the leg given by

$$w_k = \sqrt{det(J(q_k) \cdot J(q_k)^T)} \tag{27}$$

so that the damping parameter becomes quite large near the singularities and quite small apart from the singularities . The damping parameter is given by $W_n I$ where $W_n$is given by the following equation

$$Wn = \begin{cases} \lambda_0 (1 - \frac{w_{k+1}}{w_k}) & if \ \frac{w_{k+1}}{w_k} < \mu \\ 0 & otherwise \end{cases} \tag{28}$$

where k, μ is the threshold to decide the singularity neighborhood and λ_0 is a constant that permits to choose among a more robust method but less exact, in case of larger values, or less robust but more exact, in case of smaller values. Another possible way to chose the damping factor,discussed from a viewpoint of convergence performance, is described in [5] where the authors define the damping as

$$W_n = E_k I + \bar{W}_n \tag{29}$$

$$E_k = \frac{1}{2}(r_d - f(q))^T W_e (r_d - f(q)) \tag{30}$$

where E_k is the cost function (30), I is the identity matrix and $\bar{W}_n$ is a small constant bias.

I performed some tests for the three different inverse kinematics methods, using the 2R planar robot with links length $l = 0.41$, with base in $[0, 0.82]$ and the Cartesian point in which we want to place the end effector is $(0, 0)$, so that the the robot reaches this point in a singular configuration, since this point is on the workspace boundary. In the table 1 are reported the final error, the number of iteration and the time elapsed to compute the solution for the three different methods. As it is possible to see the Newton method performs better than the other methods in terms of Cartesian error and elapsed time to get to the solution. It is important to notice that in this test, the leg did not cross the singularity but it just approached it, and I accepted the error of $9.568 \mu m$. The reason why it is possible to get this low error is because Matlab is able to represent very high numbers in absolute value, without generating memory overflow. As a matter of fact Matlab as default works with double precision representation that occupies 64 bits with sign, then 63 bits to represent the absolute value of the number and 1 bit to represent the sign. Surely if we have to cross the singularity, Newton method could not be used and a singularity robust method would have been necessary such as Levembuerg-Marquardt method. Since also during

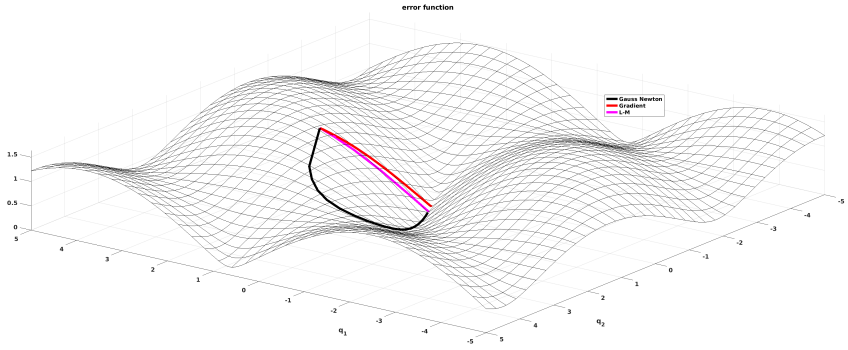

Figure 5: Comparison among the three inverse kinematic methods

the walking it is never necessary to cross singularities, then we can use Newton without any problems.

In the figure 5 it is shown the evolution of the error for the three different methods.

Method	error [m]	iterations [#]	elapsed time [s]
Gradient	8.486842e-03	500	50.189070
Newton	9.567838e-06	117	9.881217
L-M	9.710972e-05	29	15.078741

Table 1: comparison among Gradient methood, Newton method and Levenberg-Marquardt method

7 Space to Time conversion

The values that I found through the inverse kinematics cannot be passed directly to the robot, since these values are function of the hip displacement, then it is necessary to perform a conversion during the robot motion. The idea to perform this conversion is to define a constant control cycle T and compute the conversion of the joints angles; however this is not a trivial problem, because T is different from the SQD time step Δt that is also variable, depending on the robot speed. The problem that we have to address is that if $T < \Delta t$ then one SQD step passes in many control cycles, on the other hand if $T > \Delta t$ then many SQD steps pass in one control cycle. In order to compute the joint values at the current control cycle, it is necessary to compute the next time node t_{k+1} using Δt given by (13)using the actual robot speed. If $t_{k+1} > t_{now}$, where t_{now} is the current time incremented at every control cycle of T, then it is possible to compute the joint values making an interpolation between two adjacent SWP node. Otherwise increase the node index k and it is necessary to compute again the time until it is larger than t_{now}.

8 Simulation and Results

The simulations that I performed was done using Matlab, on a Linux Ubuntu 20.04.2 LTS with Intel Core™ i7-8550U CPU @ 1.80GHz × 8 . The first problem that I addressed is to solve the optimization problem given by the cost function (16) and constraints (12), (14) and (15). In order to define the optimization problem in Matlab I used Yalmip, that is an external library of Matlab that allows us to define the cost function, the constraints and to call the solver. This latter can be chosen among all the available solvers, otherwise if one does not specify any solver it chooses the suitable one. In my case I used the solver "`fmincon`", because it is available on Matlab for free; it is not the best solver, as a matter of fact the computation time to find the optimal solution is quite long.
I solved this problem for two different values of β: $\beta = 1.5$ and $\beta = 50$. In case of $\beta = 1.5$ it is possible to see from the figure (15 that the ZMP trajectory obtained does not track the reference well, but if we look at the velocity profile in figure (16), it follows the desired profile as much as possible except for the unavoidable oscillations. On the other hand, if we use $\beta = 50$, we can see in figure (19) that the obtained ZMP tracks well the reference but the velocity profile it is worse with respect to the previous case as it is possible to see from figure (20). Under the above discussion, I plot the simulation of three steps having chosen $\beta = 50$ both in space discretization and time discretization (33).

Although in a real robot the ZMP coincides with the ZMP measured from the force/torque sensors at the foot and never exists outside the support polygon [7], when calculating the ZMP from the simulated motion of the robot, usually it is used a "precondition" such that the sole keeps the surface contact with the ground since it is fixed to the ground. Instead in my case I do not use this precondition and then it could happen that the ZMP goes outside the SP, as it is shown in figure 27. To overcome this problem it is necessary to tune properly the weight in the cost function of the optimization problem, such that the ZMP always lies in the SP.

Converting the result from space to time (17) it is also possible to visualize the same results expressed in function of time in figures (17),(18),(21) and (22).

For what concerns the lateral motion, it is not possible to solve the problem with SQD since the motion along the y axis is oscillatory so the velocity is bounded between a certain positive value and a negative value, so it passes trough $v_i = 0$ and that is a singularity for the model. As a consequence I solved the optimization problem directly in the time domain that is also simpler to solve, since I have a linear convex problem, and it does not affect the flexibility of the SQD along the longitudinal axis. The result of the motion along the coronal axis are reported in figure 25 for the case with $\beta = 1.5$ and in figure 26 for the case with $\beta = 50$ After these simulations, I analyzed the same problem but directly in the time domain, with a fixed time discretization, we can see the results from figure (27) to figure (33). Comparing these two techniques, we notice that we have similar results. The main difference is that using the SQD we have a quite high computation time, i.e about 11 hours, versus few seconds in the time discretization case; probably this time could have decreased by using a powerful solver.

Once solved the previous optimization problem I considered the kinematics of the robot by computing the joint values at each discretization step, after that I visualized the result on a biped walker that I developed in Matlab. In particular I developed two different kind of biped walker, the first one has two legs with 3 DoF on each leg, as shown in figure 10, instead the second one has two leg with 5 DoF on each leg, as shown in figure 11. In order to compute the inverse kinematics I considered the CoM trajectory along x and y resulting from the optimization problem, and also I defined the trajectory of the CoM along z using a composition of two kind of

i	α_i	a_i	d_i	θ_i
1	0	0.41	0	$\theta_1^\star$
2	0	0.41	0	$\theta_2^\star$
3	0	0.15	0	$\theta_3^\star$

Table 2: DH table 3 DoF model

i	α_i	a_i	d_i	θ_i
1	0	0.025	0.2	$\theta_1^\star$
2	$\pi/2$	0	0	$\theta_2^\star$
3	$-pi/2$	0.075	0	$\theta_3^\star$
4	0	0.41	-0.39	$\theta_4^\star$
5	0	0.41	-0.41	$\theta_5^\star$
6	0	0.15	-0.15	$\theta_6^\star$

Table 3: DH table 5 DoF model

functions, a parabola and a constant function, as it is possible to see from the following equation.

$$f(x) = \begin{cases} -0.3750x^2 + 0.81 & if\ x \in [0, 0.2) \\ 0.7951 & if x \in [0.2, 0.3) \\ -0.3750x^2 + 0.375x + 0.7162 & if\ x \in [0.3, 0.7) \\ 0.7951 & if x \in [0.7, 0.8) \\ -0.3750x^2 + 0.75x + 0.4350 & if\ x \in [0.8, 1.2) \\ 0.7951 & if x \in [1.2, 1.5) \end{cases} \tag{31}$$

In order to solve the inverse kinematics problem I design a trajectory for the heel. Since this trajectory has a particular shape, figure 12, it is not very easy to define it by functions, then I have designed this trajectory by hand using the Matlab toolbox called *singEditor*, after that I extracted the data and saved as an array of samples in the work-space of Matlab. At the end I used this data as a reference trajectory of the heel for one footstep, taking care to adjust it during the inverse kinematics computation. In fact, since this trajectory has been designed by hand, it could happen that in a certain point the distance from the CoM and the heel is larger than the length of the leg, so I force the leg to pass through a singularity and as I said in section 6 this is a problem for the Newton method. To overcome this problem at each discretization step I check the distance between the CoM and the heel and if this distance is larger then the length of leg I increase the values of the heel along the z axis, so that this distance reaches, at most, the value of the length of the leg. In analogy of a generic manipulator the CoM trajectory along x,y and z represent the base frame position, instead the trajectory of the heel is the end effector position. Using this two trajectories, i.e. the base trajectory and the end effector trajectory, I computed the inverse kinematics using the Newton method for the two kinds of biped walkers. But first I defined the forward kinematic by assigning the reference frame of each joint for the 3 DoF model as shown in fig 6 and for the 5 DoF model as shown in fig 7, and then I computed the DH tables respectively tab 2 and 3.

I collected all the values computed with the inverse kinematics, of one leg, in an array of vectors where the dimension of the array is equal to the number of the SQD steps, then using the values of one leg I computed the values of the other leg by translating the vectors in the array, figure 8 and 9 show the evolution in space of the joints respectively of the left leg and the right leg, where the point in which the knee angle crosses the zero value is a singular configuration.

Additionally I have measured some performance indexes to evaluate the inverse kinematics computation, that I reported in table 4 for the 3 DoF model and in the table 5 for the 5 DoF model. Clearly the performance of the 3 DoF model are better than the 5 DoF one because the model is simpler, but both have good performance in terms of convergence in time, number of iterations and Cartesian errors, even if the method used has some problems near singularities.

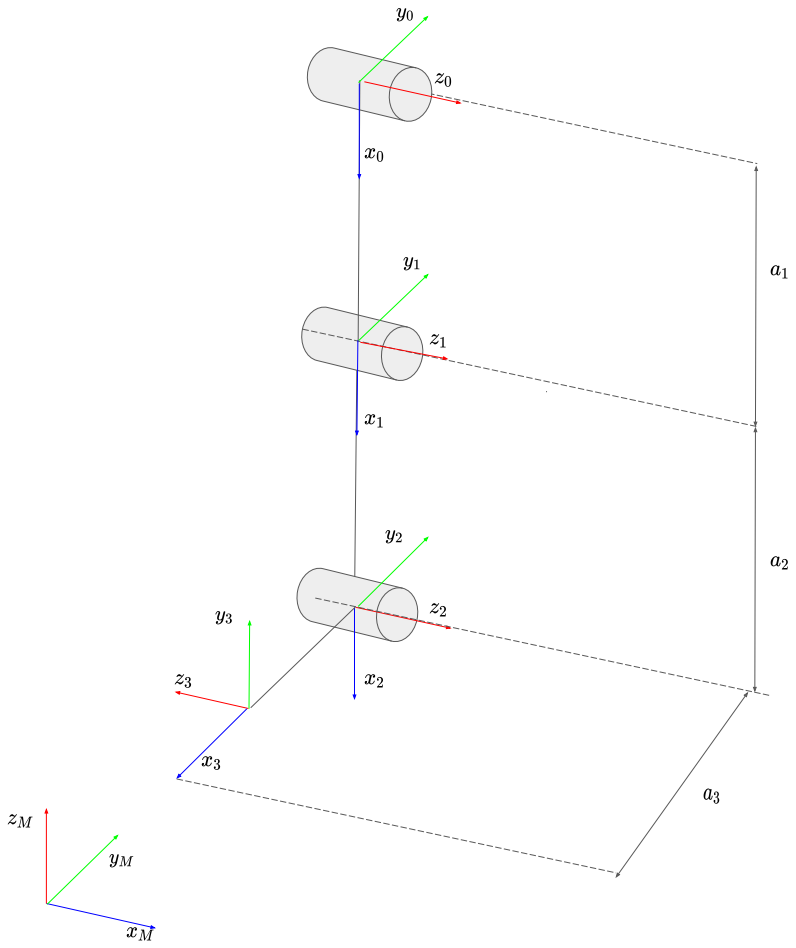

Figure 6: Frame assignment to the 3 DoF leg

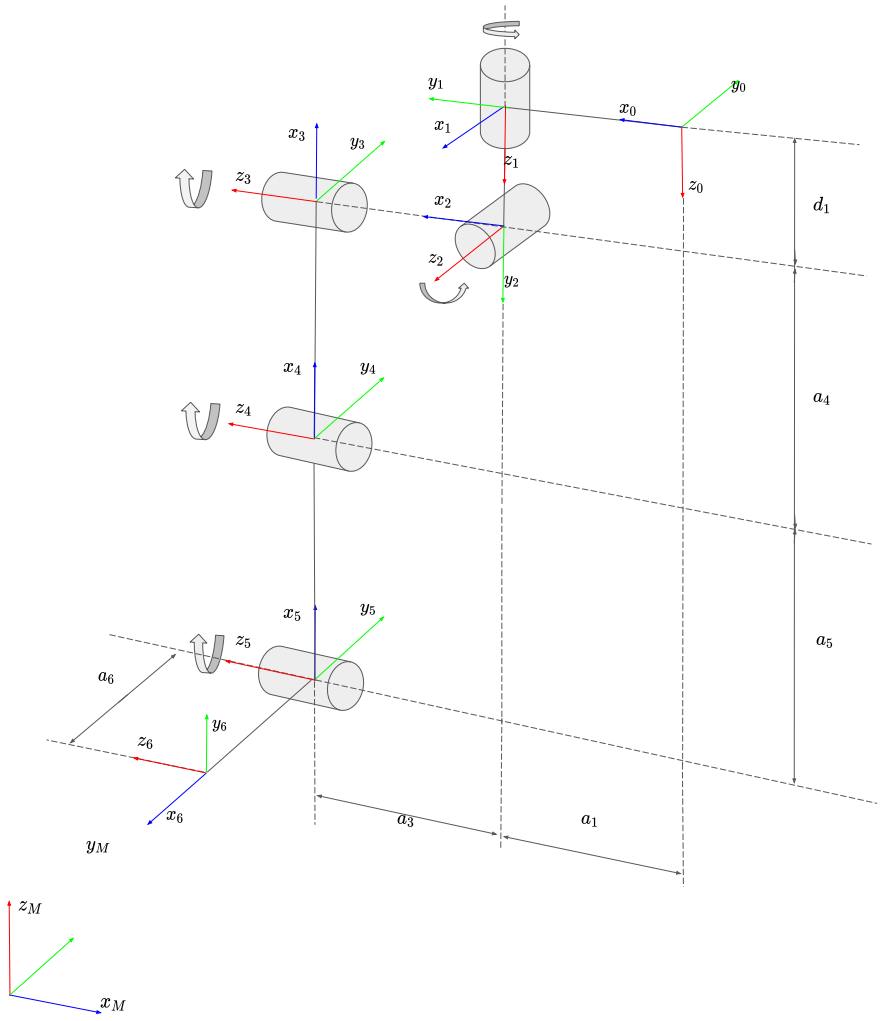

Figure 7: Frame assignment to the 5 DoF leg

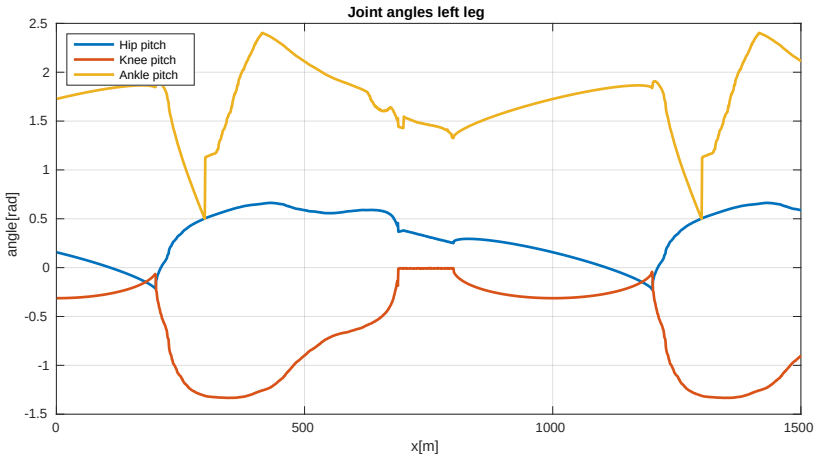

Figure 8: Left leg joint values

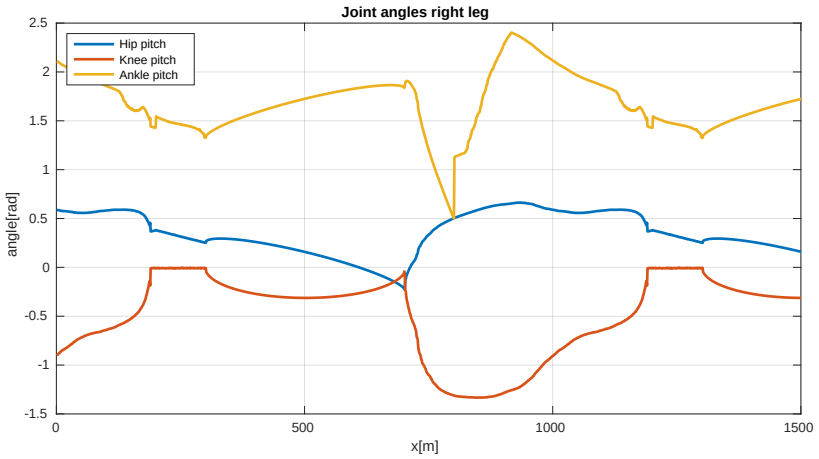

Figure 9: Right leg joint values

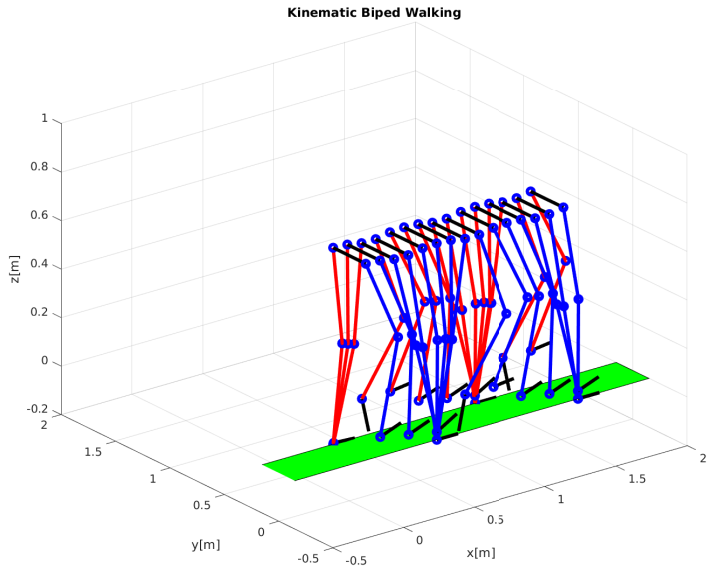

Figure 10: Some poses of the 5 DoF biped walking

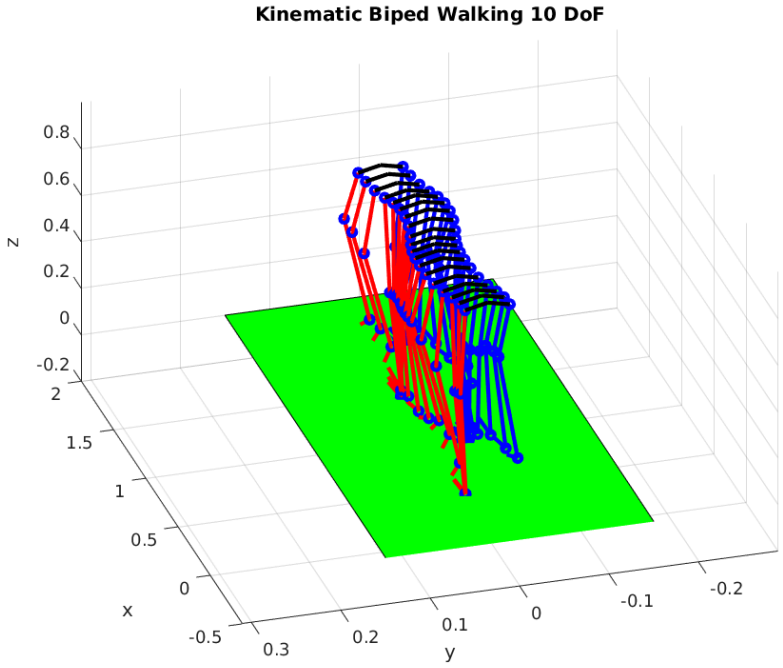

Figure 11: Some poses of the 10 DoF biped walker

Time [ms]	max	3.053
	min	0.011
	avg	0.065
Iterations [#]	max	6
	min	1
	avg	1.48
Error [m]	max	9.982e-07
	min	1.434e-12
	avg	1.434e-12

Table 4: Inverse kinematics performance index for the 3 DoF model

Time [ms]	max	5.705
	min	0.008
	avg	3.54
Iterations [#]	max	31
	min	1
	avg	9.29
Error [m]	max	9.999e-07
	min	3.852e-08
	avg	9.198e-07

Table 5: Inverse kinematics performance index for the 5 DoF model

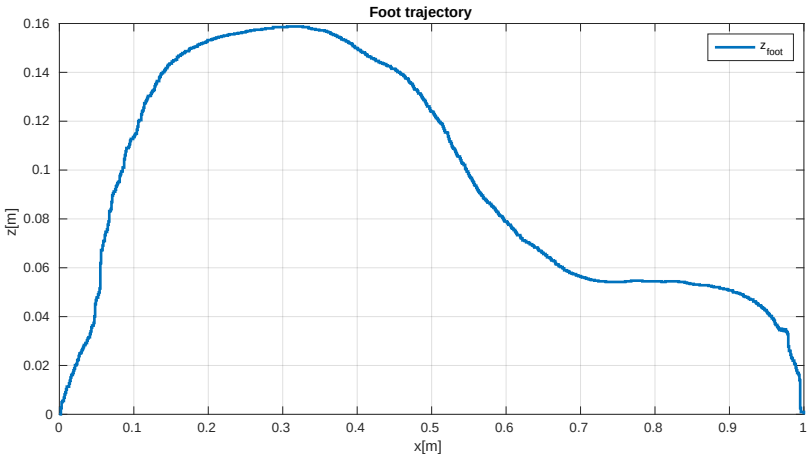

Figure 12: Foot trajecotry in function of the hip position

To visualize the two kind of biped walker, I defined a Matlab function that plots a leg , given the base position and its configuration. In particular this Matlab function plots each part of the leg by computing the forward kinematic from the Matlab reference frame to the tip of each link. After that, in order to plot the two legs I called twice this function inside a loop of N iteration, where N is the number of SQD steps, and at each iteration I passed the configuration found in the inverse kinematic to plot the left leg and the translated one to plot the right leg.

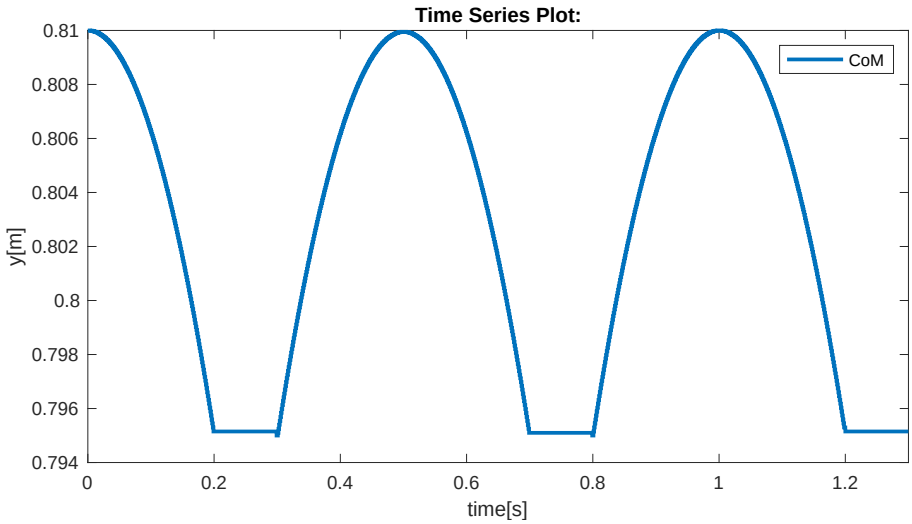

Figure 13: CoM height trajectory

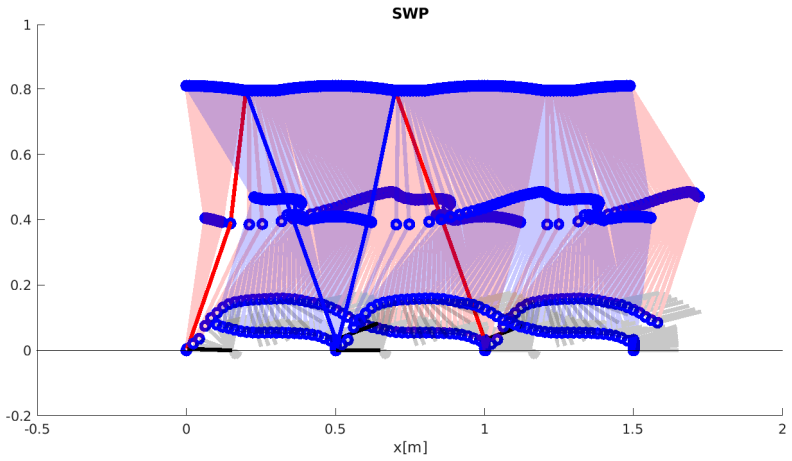

Figure 14: Spatial Walking Pattern

9 Conclusions

In this project I have reproduced the fundamental results of the paper [1]. I started from the classical optimization problem in time domain , to get the evolution of the CoM and the ZMP in time, which I later used as comparison with respect to the new method introduced in the paper on which the project is based on. Then I have formalized the same problem in the space domain by introducing the SQD, that is the LIP model discretized in the space domain, and I have solved the optimization problem along the sagittal axis, instead for the lateral motion I solved the optimization problem directly in the time domain because it is not possible to solve in the space domain. Using this kind of discretization I also transform the linear convex optimization problem into a nonlinear non-convex optimization problem. As a consequence the computation time to get the solution increased remarkably and also due to the non convexity the solution that I got may be not the optimal one, but the results are acceptable. This is the first big drawback of this new technique, in fact for how I implemented the optimization problem, i.e I optimized on the whole horizon at once, it is not possible to use it as an online technique, but this is not a problem because it is conceived as an off line technique. After obtaining the evolution of the CoM along x and y axis and the evolution of the ZMP with respect to the hip position I focused on the kinematic part. The aim of the kinematic part is to find the values of the legs joints to allow the robot to walk with stretched knee, so first of all I compared three different numeric inverse kinematic algorithms that are Gradient method, Newton method and Levenberg-Marquardt method and I choose Newton method because it has better performances with respect to the other methods, and also for the reason explained at the end of section 6. In order to tackling the problem gradually I started with a simplified leg of 3 DoF and I computed all the joint values along the path, in which I made an analogy with a manipulator, where the hip is the moving base and the heel is the end effector. In this way I have completed the the spatial walking pattern SPW, that is an array generated offline in which there are the evolution of the ZMP, the evolution of CoM velocity and the values of the joint at every discretization step expressed in the space domain. After that, I increased the difficulty by considering the 5 DoF leg that allowed us to add the lateral motion to the robot, in fact these two extra DoFs are two revolute joints at the hip of the robot. Once I have generated these spatial walking patterns, for the 3 DoF and for the 5 DoF I implemented a kinematic biped walker in order to visualize the results. So I defined the DH tables and the homogeneous transformation between the hip and the Matlab frame and I plotted the leg by computing the forward kinematic for reach the joints and then I linked them with a segment. Additionally I measured some performance indexes to evaluate the inverse kinematic performance along the whole path. At the end I gave an idea on how to perform the conversion from the space domain to the time domain that is necessary to implement this method for a dynamic simulation, that I did not perform, or to implement on a real robot.

This new version of the biped walking generation based on the control of the ZMP has many drawbacks. The most crucial are that the optimization problem is more difficult to solve with respect to the classic case in the time domain, it is not implementable in real time because the SPW is generated offline, the discretized model is nonlinear and it has a singularity when the CoM speed is zero. The advantage is that it is more flexible, in the sense that it is possible to realize versatile gait generation and in fact I allow the robot to walk with stretched knee, instead in the classical formulation of the problem in time domain the robot walks in a crouch position due to the more conservative gait design. Even if this method add this important advantage the benefit of the SQD compared with conventional methods is still unclear.

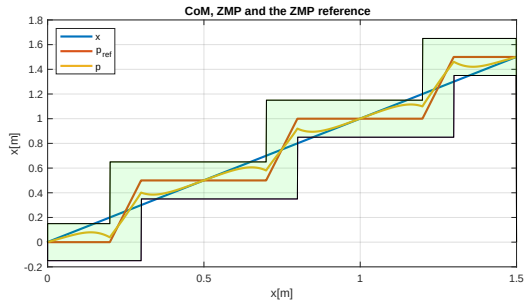

Figure 15: ZMP and CoM profile resulting from the optimization with $\beta = 1.50$

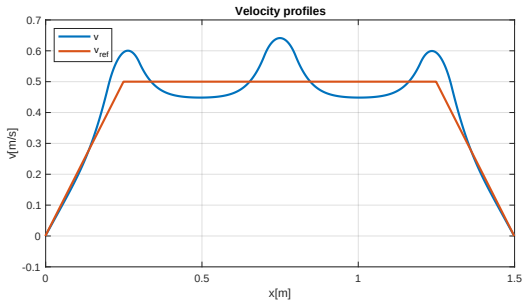

Figure 16: CoM speed profile resulting from the optimization with $\beta = 1.50$

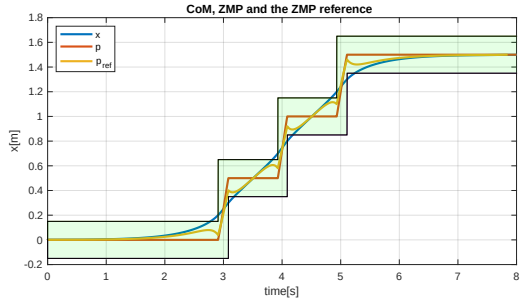

Figure 17: ZMP and CoM profile resulting from the optimization in time domain with $\beta = 1.50$

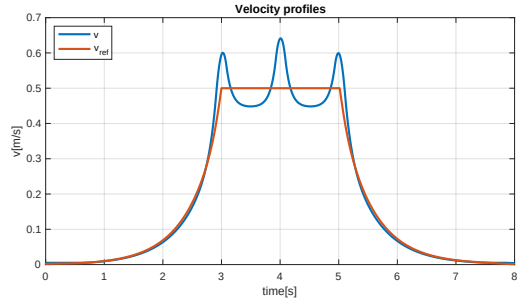

Figure 18: ZMP and CoM profile resulting from the optimization in time domain with $\beta = 1.50$

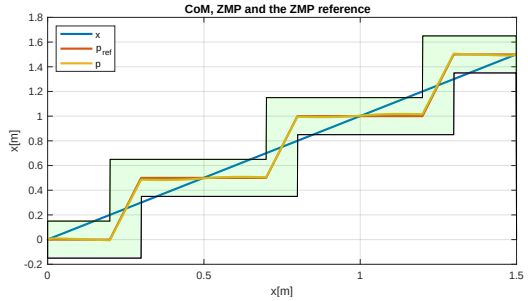

Figure 19: ZMP and CoM profile resulting from the optimization with $\beta = 50$

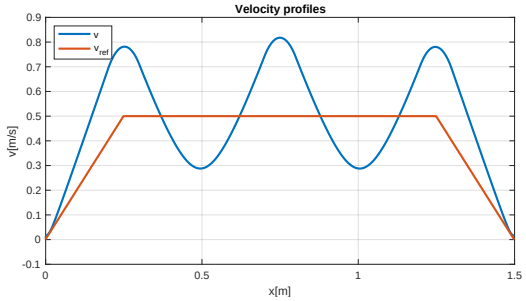

Figure 20: CoM speed profile resulting from the optimization with $\beta = 50$

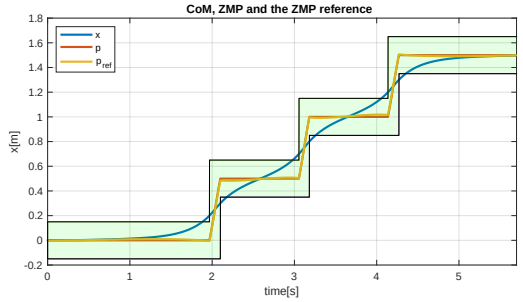

Figure 21: ZMP and CoM profile resulting from the optimization in time domain with $\beta = 50$

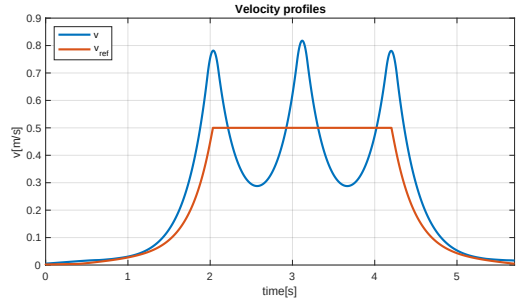

Figure 22: ZMP and CoM profile resulting from the optimization in time domain with $\beta = 50$

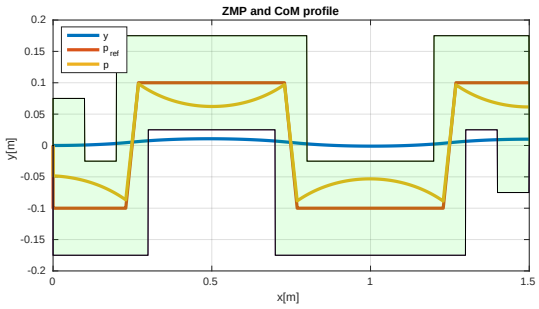

Figure 23: ZMP and CoM lateral motion resulting from the optimization in space domain with $\beta = 1.50$

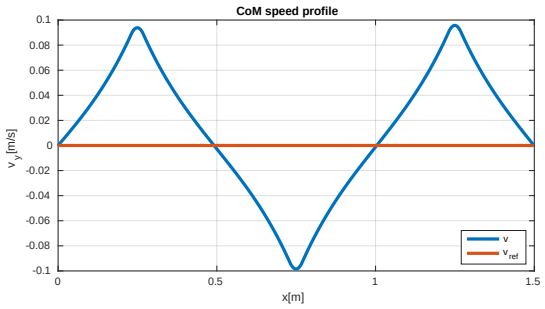

Figure 24: CoM speed profile lateral motion resulting from the optimization in space domain with $\beta = 1.50$

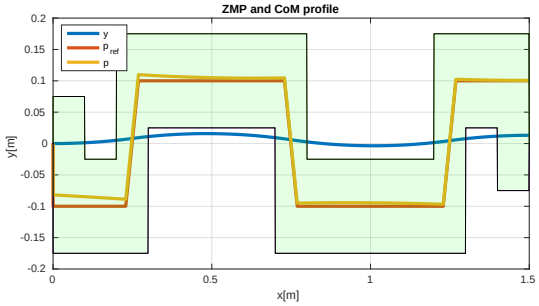

Figure 25: ZMP and CoM lateral motion resulting from the optimization in space domain with $\beta = 50$

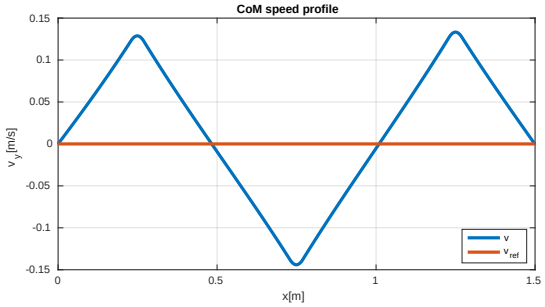

Figure 26: CoM speed profile lateral motion resulting from the optimization in space domain with $\beta = 50$

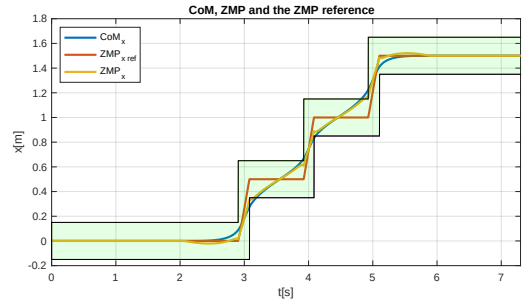

Figure 27: ZMP and CoM profile resulting from the optimization with $\beta = 1.50$

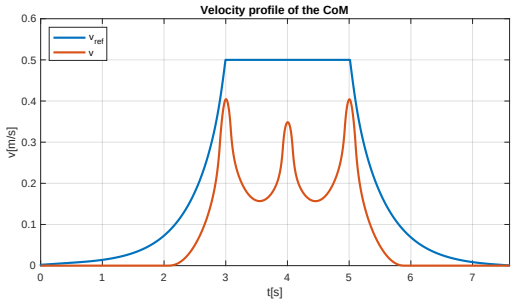

Figure 28: CoM speed profile resulting from the optimization with $\beta = 1.50$

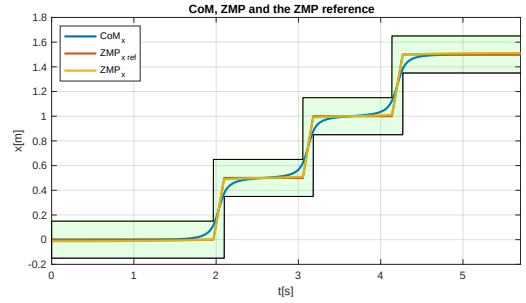

Figure 29: ZMP and CoM profile resulting from the optimization in time domain with $\beta = 50$

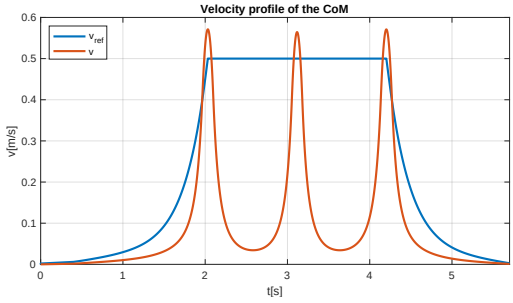

Figure 30: CoM speed profile resulting from the optimization with $\beta = 50$

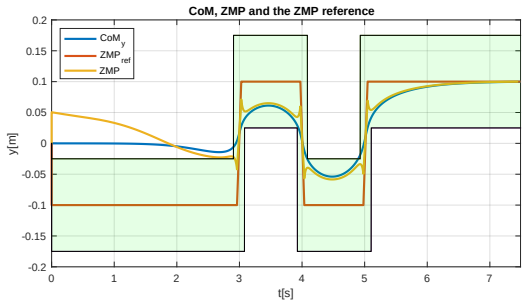

Figure 31: ZMP and CoM lateral motion resulting from the optimization with $\beta = 1.50$

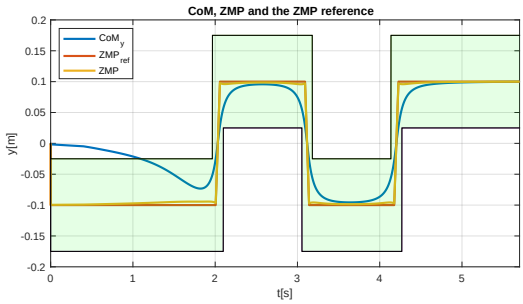

Figure 32: CoM speed profile lateral motion resulting from the optimization with $\beta = 50$

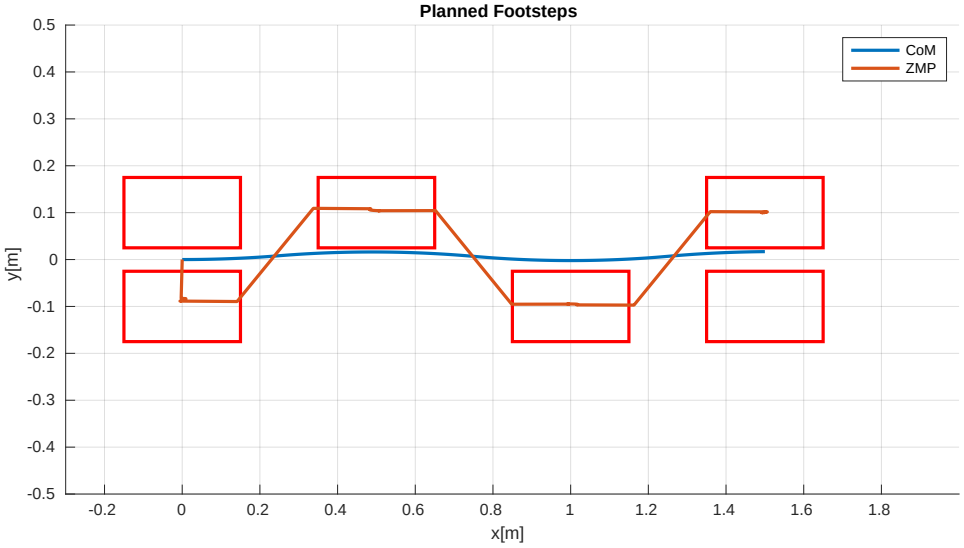

Figure 33: Simulation of three steps with $\beta = 50$

References

[1] S. Kajita et al., "Biped walking pattern generation based on spatially quantized dynamics," 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), Birmingham, UK, 2017, pp. 599-605, doi: 10.1109/HUMANOIDS.2017.8246933.

[2] Vukobratovic, Miomir Borovac, Branislav. (2004). Zero-Moment Point - Thirty Five Years of its Life.. I. J. Humanoid Robotics. 1. 157-173. 10.1142/S0219843604000083.

[3] Y. Tassa, N. Mansard and E. Todorov, "Control-limited differential dynamic programming," 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014, pp. 1168-1175, doi: 10.1109/ICRA.2014.6907001.

[4] Kelmar, L. Khosla, Pradeep. (1988). Automatic generation of kinematics for a reconfigurable modular manipulator system. 663 - 668 vol.2. 10.1109/ROBOT.1988.12135.

[5] T. Sugihara, "Solvability-unconcerned inverse kinematics based on Levenberg-Marquardt method with robust damping," 2009 9th IEEE-RAS International Conference on Humanoid Robots, Paris, France, 2009, pp. 555-560, doi: 10.1109/ICHR.2009.5379515.

[6] Optimization Problem - MATLAB & Simulink https://www.mathworks.com/help/mpc/ug/optimization-problem.html

[7] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, Introduction to Humanoid Robotics. Springer Publishing Company Inc., 2014.