Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

AUTOMATIC CONTROL LABORATORY

Semester Project

Design of a Dynamic Omnidirectional Kick Engine for NAO Bipedal Robots in RoboCup

Marina Draskovic
mdraskovic@student.ethz.ch

Advisors
Alexandros Tanzanakis
Prof. Dr. John Lygeros

January 2022

Abstract

Creating a smooth, precise and stable kick is essential for success in the RoboCup, robotics football competition. In this thesis, a novel omnidirectional kick engine is proposed and in-silico validated for the NAO, a bipedal robot used in the RoboCup Standard Platform League. The kick engine generates a kick trajectory that can propell the ball in any intended direction without prior input or predefined parameters. The trajectories are generated using Bézier curves with three or four control points. The speed of the kick is adjusted by choosing a number of interpolation points on the curve. The kick is to be performed while the robot balances on the support foot. Apart from remaining stable during the kick, the robot should be able to compensate the external disturbance forces. This is achieved by using a ZMP based preview controller in combination with the proportional arms controller. Simulation studies on the Webots simulator show that the proposed algorithm enables reliable adaptive, stable and robust omnidirectional kicks in a variety of important scenarios.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Design of a Dynamic Omni-Directional Kick Engine for NAO Bipedal Robots in RoboCup

Authored by (in block letters):

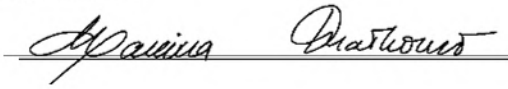
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Draskovic	Marina

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 14.01.2022	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

Chapter 1

Introduction

RoboCup is an annual international robotics football competition. The ultimate goal of the competition is to create a robotic football team that can be superior to the winners of the World Cup in the year 2050 [1]. Although achieving such a goal will not create a remarkable social impact, the RoboCup competition is an exciting challenge from which scientific achievements are arising. As football is a complex game with a dynamic environment and autonomous players, robotic football is driving for the development in a wide area of research, such as bipedal locomotion, intelligent robot control and computer vision. ETH Zürich is participating in the RoboCup under a team name *NomadZ*.

NomadZ RoboCup team is participating in the *Standard Platform League*, where the main focus is on the software behind the robots. As all teams are using *NAO humanoid robots* [2] developed by SoftBank robotics, hardware is not an area of interest in this league. NAO is a 58 cm tall bipedal robot with 25 degrees of freedom and various sensors embedded in the CPU for acquiring information. As such, NAO is able to move around and perceive its environment.

In order to win a football match, robots must be able to keep the ball away from opponents, pass the ball around and eventually score a goal. That is why the kick engine plays a significant role in the success of any team. The faster and the more dynamic the kick is, the higher are the chances of defeating the rivals. Currently, the NomadZ team uses a hard-coded trajectory and the robot is able to shoot only straight ahead. This approach is inflexible and requires a long process of alignment leading to the kick. For this reason the idea of developing an *omnidirectional kick* arose, making the movement fast, accurate and adaptable to various situations.

This thesis aims to design a dynamic, stable and robust omnidirectional kick in the *Webots* simulator. The robot should be able to stabilize on one leg and perform a kick following a trajectory computed in real time, in order to propel the ball in the intended direction.

Figure 1.1: Standard Platform League (left) and NAO robot (right); adopted from [1], [2]

1.1 Related Work

An extensive literature on humanoid robotics proposes using zero moment point based controllers for stability, e.g. [3], [4], [5]. The controller can be designed as a PID ([6]) but this controller has a delayed reaction to disturbances. A better option is a preview controller created by authors in [3] for the walk engine which takes into the account future information. The authors in [5] and [6] suggest that additional stability can be provided by arm movements, though the idea is not formalized and is implemented just as predefined body postures. Some of the methods developed for trajectory planning are using polynomials ([4], [5]), although it is then not straightforward to control the ball's intended direction. The authors in [6] and[7] suggest using Bézier curves instead, yet methods of choosing curve control points are seldom mentioned. In order to transfer the desired positions in the Cartesian coordinates to the required joint motions, much of the literature focuses on the Jacobian based inverse kinematics, for example [8]. Various direct approaches for inverse kinematics based on the geometrical calculations are described in [3] or [4] and are more precise than the former approach. Finally, the authors in [9] suggest that the kick motion can be optimally operated using reinforcement learning techniques, which belongs to a new research direction.

1.2 Contributions

This thesis is advancing the approach for the kick engine currently utilized in the NomadZ team. Overall, the robot is able to adapt to changes in the ball position, kick in different directions, as well as adjust distance covered by the ball. Contributions of the thesis are as follows:

- Maintaining balance by adapting a preview controller. Stability is to be preserved in cases of present external disturbances, for example an opponent pushing the robot. Therefore, we enable enhanced stability by integrating a proportional controller for the arms movement.
- A dynamic and omnidirectional kick trajectory is obtained by using a Bézier curve. The performance is compared when the curve is created with three and four control points and it is described how the control points are chosen.
- Theoretical derivations are confirmed in the Webots simulator, for which a tutorial is provided. The performance of stability controller is tested by applying forces during different time steps of the kick. The kick is tested by placing the ball and target at various positions.

1.3 Thesis Outline

In the next chapter, the overall design of the kick engine is described. Kick phases are introduced and it is explained how the robot chooses whether it kicks with right or left leg. Chapter 3 defines a zero moment point as a reference for stability. Then, two controllers that work in parallel to maintain the stability are presented: the preview controller which affects the robot's body position and proportional controller which affects the position of arms. Next, the motions of swinging back the leg and kicking the ball are described in details. The Bézier curve and its control points used for generating a kicking trajectory are introduced. Moreover, inverse kinematics module that obtains required joint angles is explained.

In chapter 4, a simulation study that demonstartes the performance of the robot is carried out in the Webots simulator. Finally, the last chapter gives a conclusion and a short summary of the gained insights. A tutorial for the Webots simulator is provided in the appendix.

Chapter 2

Kick Engine Design

In this chapter, the general setup of the proposed kick engine is presented. It receives a robot, ball and target positions as inputs from other engines. The output of the kick engine are the joint angles used to perform a robot movement that would kick the ball towards the target. The overall omnidirectional kick is achieved by the cooperation of three modules: *stability*, *trajectory planning* and *inverse kinematics*. Firstly, stability and trajectory planning modules output the required robot position in the natural reference frame for each time step of the motion. This is then used in the inverse kinematics module to calculate necessary joint angles. The general overview of the framework is presented in Figure 2.1.

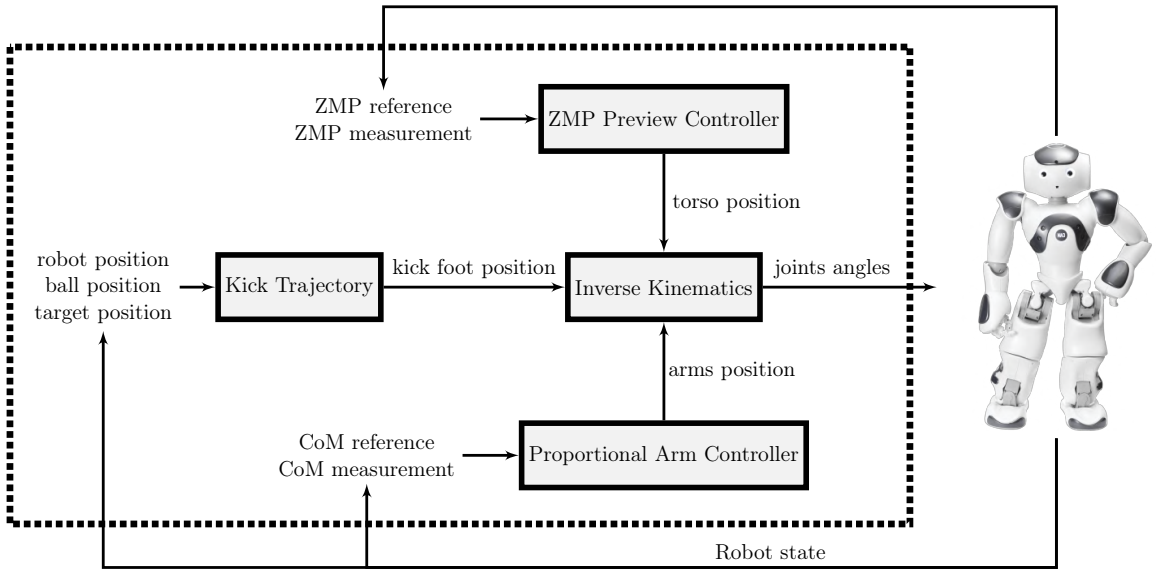


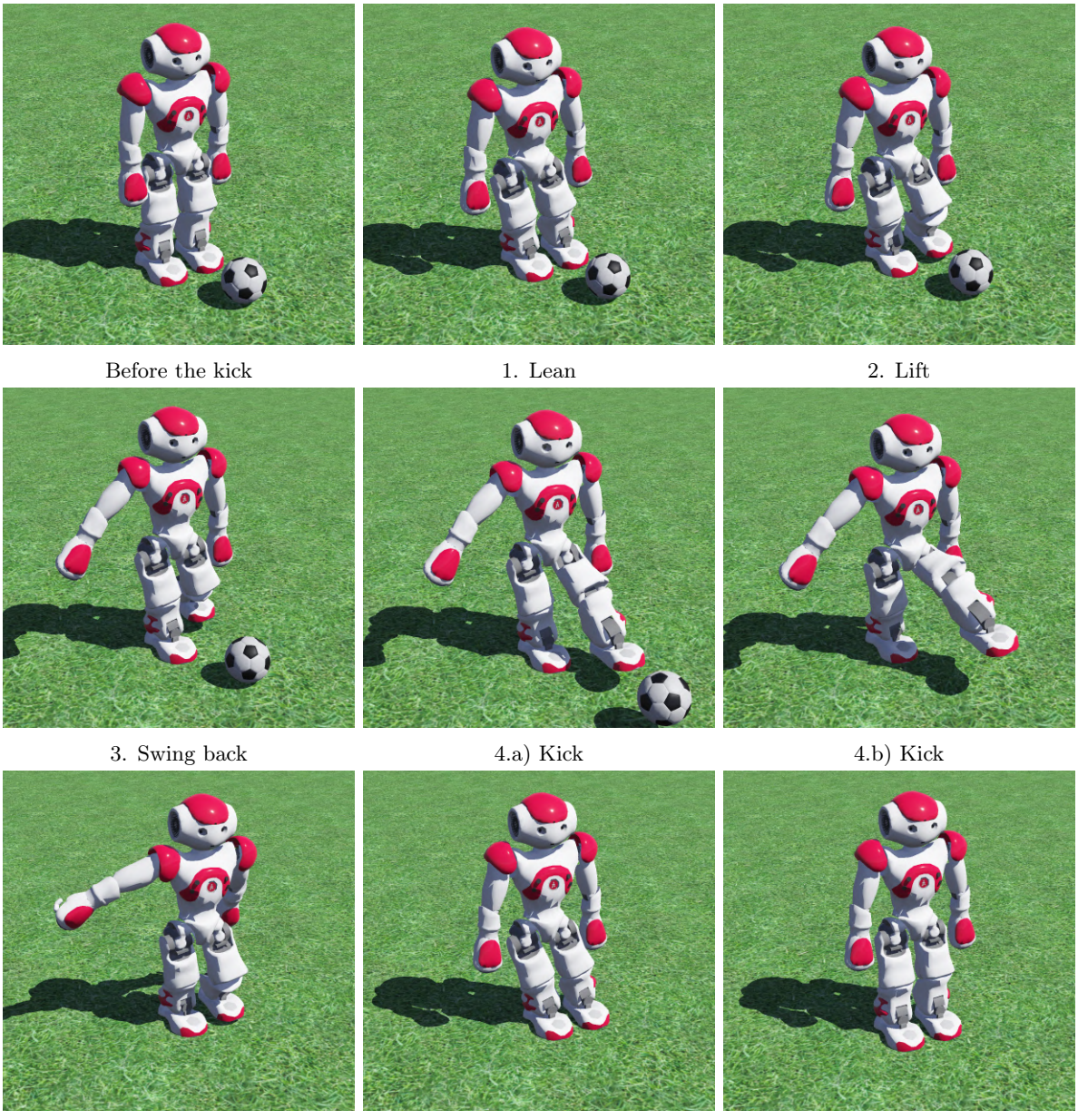
Figure 2.1: General overview of the control framework

Before the actual kick is executed, the robot chooses whether it will kick with its right or left leg based on the current ball and target positions. Namely, if the ball is clearly on the right side of the robot, it can only be reached with its right leg, hence this leg is chosen as the kicking one. The analogous holds when the ball is on the left side. When the ball is right in front (or right behind) the robot, the ball can be reached with both legs. Then, the kicking leg is decided based on the target position. If the target is positioned towards the right side, it is more natural that the robot decides to kick with the left leg, and vice versa. Furthermore, if the target is far away, the robot will aim to shoot the ball as far as possible at the expense of precision. The kick would be more delicate provided that the target lays nearby.

The complete motion is divided into several phases that are executed in series:

1. **Lean** - The robot shifts center of mass onto the support leg.
2. **Lift** - The robot raises the kicking leg of the ground.
3. **Swing back** - The kicking leg is moved away from the ball.
4. **Kick** - This is the main phase where the robot kicks the ball.
5. **Return** - The leg is returned to the initial lifted position.
6. **Put down** - The leg is returned to the ground.
7. **Center** - The robot returns to the upright position, stabilized on both legs with the center of mass in the middle.

Figure 2.2 shows snapshots of the robot during the above mentioned phases. Theory used to achieve presented movements is detailed in the next chapter.



Before the kick | 1. Lean | 2. Lift

3. Swing back | 4.a) Kick | 4.b) Kick

5. Return | 6. Put down | 7. Center

Figure 2.2: Snapshots of the NAO robot during the kicking phases in the Webots simulator

Chapter 3

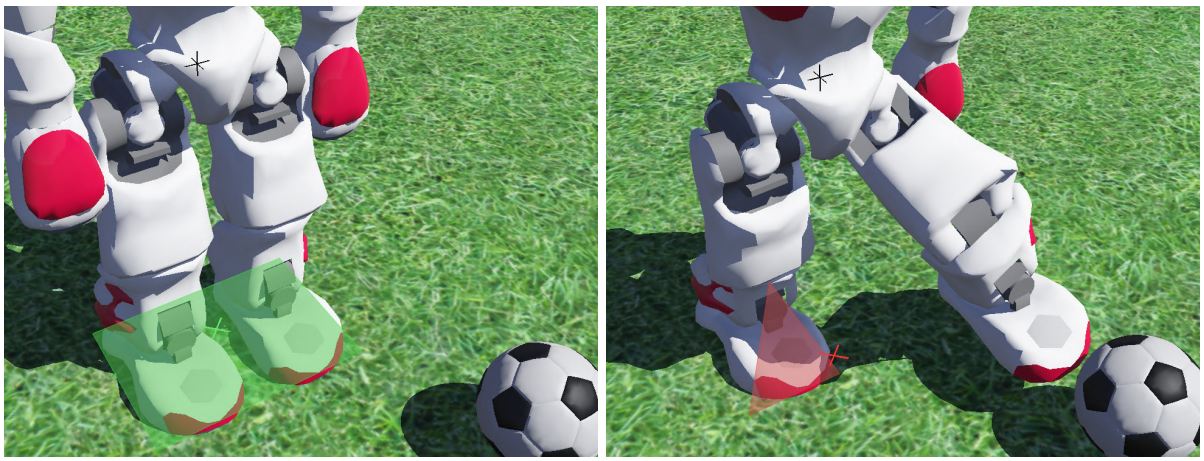
Development of the Kick Engine

The task of the robot is to compute a trajectory for the foot to travel in order to attain a desired kick, while making sure that it does not fall over. Three modules working together in order to accomplish this are stability, trajectory planning and inverse kinematics.

3.1 Stability

Unlike many industrial robots that are able to move without a concern for balance, humanoid robots have to care to maintain the contact between the soles and the ground. Being a humanoid robot, the NAO has an additional concern of having a high *center of mass* (CoM) compared to four-legged or wheeled robots. That is why even standing can be problematic when external disturbances appear. Kicking is an especially challenging task as the motion is complex and fast, and because of additional forces resulting from the interaction with the ball and potentially even opponent robots. Another complication is that during the kick the robot stands on one leg, thus only the sole of the support foot has contact with the ground.

When the robot is at rest or performing motions with small momentum, a sufficient stability criterion is that the projection of the CoM to the ground lays within the *support polygon* ([3]). Support polygon is defined as a convex hull of all parts that contact the ground, i.e. it is an area of contact between the robot and the ground. However, when a robot moves dynamically for example during the kick, the ground CoM projection may exist outside the support polygon, as shown in Figure 3.1. Then, another criterion must be used to verify the stability.



(a) CoM projection inside the support polygon (b) CoM projection outside the support polygon

Figure 3.1

3.1.1 Zero Moment Point

When the robot is standing, there is a *ground reaction* exerted by the ground on the soles. This ground reaction can be represented by three forces, F_x, F_y and F_z, where each displays a force applied in certain direction, following the axis defined in Figure 3.2. Its horizontal components exist due to the friction between the ground and the soles. Three moments that exist at the contact surface can similarly be defined as τ_x, τ_y and τ_z. If there is no sliding, the horizontal forces F_x and F_y compensate the vertical moment τ_z due to the static friction ([10]).

Horizontal moments τ_x and τ_y are examined in order to assess the stability of the robot. The point $\boldsymbol{p}$ of application of the vertical load on the sole of the foot is called the *zero moment point* (ZMP), when the horizontal components of the moment of the ground reaction are zero. The criterion that can be used to establish balance during the dynamic motion is keeping the ZMP inside the support polygon.

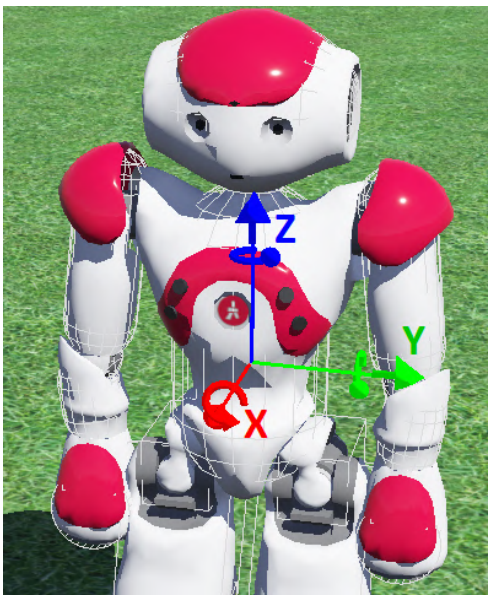


Figure 3.2: The axis of the robot

In order to define the exact zero moment point $\boldsymbol{p} = [p_x, p_y, p_z]^\top$, firstly the moment of the ground reaction force is expressed as

$$\boldsymbol{\tau} = \boldsymbol{p} \times \boldsymbol{f} + \boldsymbol{\tau_p}, \tag{3.1}$$

where $\boldsymbol{f}$ and $\boldsymbol{\tau_p}$ are force and moment about the vertical line containing the ZMP, following the derivations from [10]. Next, the *equation of translational motion* is defined. Namely, the linear momentum $\boldsymbol{\mathcal{P}}$ changes according to the sum of the forces applied to the robot from the outside. As these forces are gravitational and ground reaction force, the equation of translational motion is obtained as

$$\dot{\boldsymbol{\mathcal{P}}} = M\boldsymbol{g} + \boldsymbol{f}. \tag{3.2}$$

The *equation of rotational motion* about the origin is similarly derived. The angular momentum changes according to the sum of the moments applied to the robot. The equation is

$$\dot{\boldsymbol{\mathcal{L}}} = \boldsymbol{c} \times M\boldsymbol{g} + \boldsymbol{\tau}, \tag{3.3}$$

where $\boldsymbol{c} \times M\boldsymbol{g}$ represents the moment generated by the gravitational force, more precisely $\boldsymbol{c}$ is a position of the CoM and M is a total robot's mass. $\boldsymbol{\tau}$ represents the ground reaction moment

applied to the robot.

Substituting 3.1 and 3.2 into 3.3, the moment about the vertical line containing the ZMP $\boldsymbol{\tau_p}$ can be expressed as

$$\boldsymbol{\tau_p} = \dot{\boldsymbol{\mathcal{L}}} - \boldsymbol{c} \times M\boldsymbol{g} + (\dot{\boldsymbol{\mathcal{P}}} - M\boldsymbol{g}) \times \boldsymbol{p}. \tag{3.4}$$

Note that $\dot{\boldsymbol{\mathcal{L}}}$, $\dot{\boldsymbol{\mathcal{P}}}$, $\boldsymbol{\tau_p}$, $\boldsymbol{c}$, $\boldsymbol{g}$ and $\boldsymbol{p}$ in the above equation are actually matrices containing values in x, y and z directions:

$$\begin{aligned}
\boldsymbol{\mathcal{P}} &= [\mathcal{P}_x, \mathcal{P}_y, \mathcal{P}_z]^\top \\
\boldsymbol{\mathcal{L}} &= [\mathcal{L}_x, \mathcal{L}_y, \mathcal{L}_z]^\top \\
\boldsymbol{\tau_p} &= [\tau_{px}, \tau_{py}, \tau_{pz}]^\top \\
\boldsymbol{c} &= [x, y, z]^\top \\
\boldsymbol{g} &= [0, 0, g]^\top \\
\boldsymbol{p} &= [p_x, p_y, p_z]^\top.
\end{aligned} \tag{3.5}$$

Substituting 3.5 into 3.4 and expressing x and y components of $\boldsymbol{\tau_p}$ gives:

$$\begin{aligned}
\tau_{px} &= \dot{\mathcal{L}}_x + Mgy + \dot{\mathcal{P}}_y p_z - (\dot{\mathcal{P}}_z + Mg)\ p_y \\
\tau_{py} &= \dot{\mathcal{L}}_y - Mgx - \dot{\mathcal{P}}_x p_z + (\dot{\mathcal{P}}_z + Mg)\ p_x.
\end{aligned} \tag{3.6}$$

Using the fact that x and y components of the moment about the ZMP are zero, the equations become

$$\begin{aligned}
\dot{\mathcal{L}}_x + Mgy + \dot{\mathcal{P}}_y p_z - (\dot{\mathcal{P}}_z + Mg)\ p_y = 0 \\
\dot{\mathcal{L}}_y - Mgx - \dot{\mathcal{P}}_x p_z + (\dot{\mathcal{P}}_z + Mg)\ p_x = 0.
\end{aligned} \tag{3.7}$$

From the above equation the x and y coordinates of the position of the ZMP can be expressed. Additionally, using the fact that z coordinate denotes the height of the ZMP, and that it lays on the ground, the coordinates of the ZMP are

$$\begin{aligned}
p_x &= \frac{Mgx + p_z\dot{\mathcal{P}}_x - \dot{\mathcal{L}}_y}{Mg + \dot{\mathcal{P}}_z} \\
p_y &= \frac{Mgy + p_z\dot{\mathcal{P}}_y + \dot{\mathcal{L}}_x}{Mg + \dot{\mathcal{P}}_z} \\
p_z &= 0.
\end{aligned} \tag{3.8}$$

When the robot is not moving, the momentums are not changing, namely $\dot{\boldsymbol{\mathcal{P}}} = \dot{\boldsymbol{\mathcal{L}}} = 0$, and the ZMP coincides with the CoM projection: $p_x = x$, $p_y = y$.

When the robot is moving, linear and angular momentums need to calculated in order to obtain the ZMP position following 3.8. If the robot is modelled as an *inverted pendulum* as shown in Figure 3.3a, the linear and angular momentums about the origin are given by

$$\begin{aligned}
\boldsymbol{\mathcal{P}} = M\dot{\boldsymbol{c}} &= \begin{bmatrix} M\dot{x} \\ M\dot{y} \\ M\dot{z} \end{bmatrix} \\
\boldsymbol{\mathcal{L}} = \boldsymbol{c} \times M\dot{\boldsymbol{c}} &= \begin{bmatrix} M\ (y\dot{z} - z\dot{y}) \\ M\ (z\dot{x} - x\dot{z}) \\ M\ (x\dot{y} - y\dot{x}) \end{bmatrix},
\end{aligned} \tag{3.9}$$

respectively. Finding the derivative of 3.9 and substituting it into 3.8 gives the approximation of the ZMP position for the inverted pendulum model of the robot:

$$\begin{aligned} p_x &= x - \frac{(z - p_z)\ \ddot{x}}{\ddot{z} + g} \\ p_y &= y - \frac{(z - p_z)\ \ddot{y}}{\ddot{z} + g}. \end{aligned} \tag{3.10}$$

With the assumptions that the ZMP lays on the ground and that the movement of the CoM in z direction is negligible, from 3.10 the coordinates of ZMP finaly become

$$\begin{aligned} p_x &= x - \frac{z}{g}\ \ddot{x} \\ p_y &= y - \frac{z}{g}\ \ddot{y} \\ p_z &= 0, \end{aligned} \tag{3.11}$$

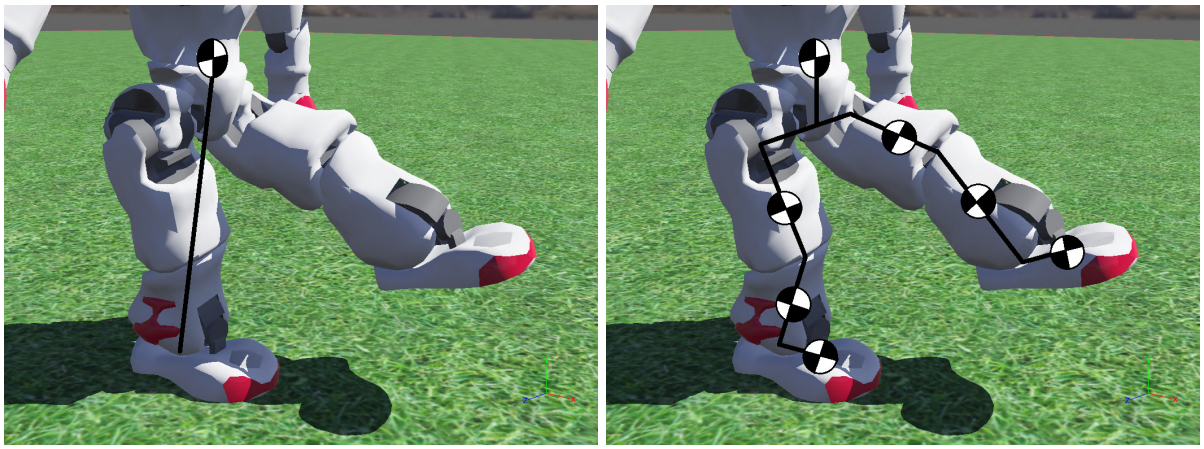
which are together known as the *ZMP equation*. With this assumption, it can be seen that the position of the ZMP depends only on the position of the CoM, $\boldsymbol{c} = [x, y, z]^\top$. To get the robot's CoM coordinates, the robot can be modelled as a *point mass system* (see Figure 3.3b) and the total CoM is calculated by dividing the sum of moments of all links by the total mass:

$$\boldsymbol{c} = \frac{\sum_{j=1}^{N} m_j\ \boldsymbol{c_j}}{M}, \tag{3.12}$$

where m_j and $\boldsymbol{c_j}$ are mass and CoM position vector of the j^{th} link. M is the mass of the whole robot, which is calculated as the sum of masses of all links, $M = \sum_j m_j$. Mass of each link is considered known, and CoM position vectors of links are usually given in the robot's coordinate system ($\hat{\boldsymbol{c_j}}$), and can be transferred to the world's coordinate frame by

$$\boldsymbol{c_j} = \boldsymbol{p_j} + \boldsymbol{R_j}\hat{\boldsymbol{c_j}}, \tag{3.13}$$

where $(\boldsymbol{p_j}, \boldsymbol{R_j})$ denotes the position and orientation of the j^{th} link, which are also treated to be known.

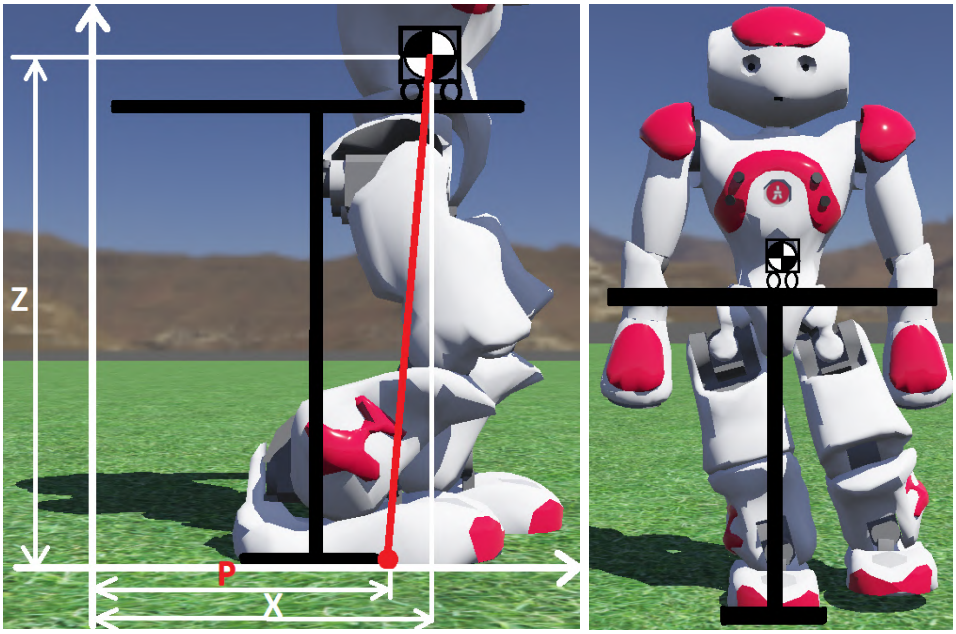


(a) Robot modelled as an inverted pendulum (b) Robot modelled as a point mass system

Figure 3.3

3.1.2 Preview Controller

The authors in [3] suggest using a *cart-table model* to create a ZMP based controller. In the cart-table model, the robot is assumed to be a point mass. A cart with mass M that represents the robot moves on a mass-less table. Height of the cart does not change, or in other words, the height of the CoM of the robot is assumed to be constant. The 3D model is created from two decoupled 2D models in x and y axis. The model is depicted in Figure 3.4.



(a) Model in x axis (b) Model in y axis

Figure 3.4: Cart-table model of the robot

The *preview controller* is based on a cart-table model and uses foot (ZMP) placement information as an input. Overall controller design is based on the ZMP equation, 3.11. The 'preview' in the name comes from the fact that the controller utilizes the future ZMP information. Initially the controller was created by authors of [3] to generate a walking gate and future information used is the walking trajectory. This controller can be used for the kick engine as well. As the robot's support leg is not moving in that case, all preview positions are considered to be constant. Overall, the preview controller is tracking target ZMP using the feedback control and utilizing future information (see Figure 3.5).

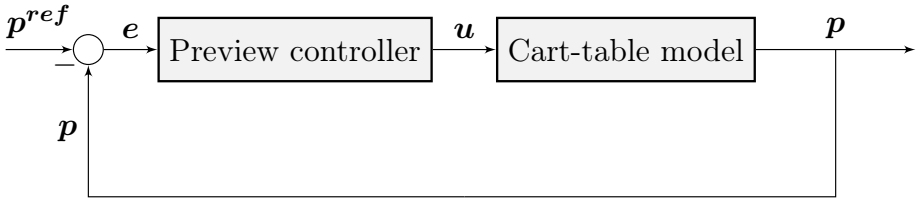


Figure 3.5: Control block diagram of the preview controller

If a derivative of a cart acceleration is treated as an input to the system, namely

$$u = \dddot{x}, \tag{3.14}$$

the ZMP equation 3.11 can be rewritten in the *state space representation* as

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \\ p &= \begin{bmatrix} 1 & 0 & -\frac{z}{g} \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}. \end{aligned} \tag{3.15}$$

The continuous-time system can be discretised using a time step Δt:

$$\begin{aligned} \boldsymbol{x_{k+1}} &= \boldsymbol{A}\ \boldsymbol{x_k} + \boldsymbol{B}\ u_k \\ p_k &= \boldsymbol{C}\ \boldsymbol{x_k}, \end{aligned} \tag{3.16}$$

using that

$$\begin{aligned} &\boldsymbol{x_k} = \begin{bmatrix} x(k\Delta t) & \dot{x}(k\Delta t) & \ddot{x}(k\Delta t) \end{bmatrix}^\top, \quad u_k = u(k\Delta t), \quad p_k = p(k\Delta t) \\ &\boldsymbol{A} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \frac{\Delta t^3}{6} \\ \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 1 & 0 & -\frac{z}{g} \end{bmatrix}. \end{aligned} \tag{3.17}$$

The optimal problem that the controller is trying to solve is minimizing the cost

$$J = \sum_{j=1}^{\infty} \left(Q\ (p_j^{ref} - p_j)^2 + R\ u_j^2 \right) \tag{3.18}$$

with positive weights Q and R. The minimum of the cost J is attained whit the optimal input

$$u_k = -\boldsymbol{K} x_k + \begin{bmatrix} f_1 & f_2 & \dots & f_N \end{bmatrix} \begin{bmatrix} p_{k+1}^{ref} \\ \vdots \\ p_{k+N}^{ref} \end{bmatrix}. \tag{3.19}$$

The input consists of the feedback term that depends on the system state at time k, and feed-forward term that uses ZMP references from N future steps. Feed-forward term consists of an inner product between the future target references p_{k+i}^{ref} and the weights f_i. Future references will all be the same as they are laying in the supporting foot which is not moving. Parameters $\boldsymbol{K}$ and f_i are solutions to

$$\begin{aligned} \boldsymbol{K} &= (R + \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{B})^{-1}\ \boldsymbol{B}^\top P A \\ f_i &= (R + \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{B})^{-1}\ \boldsymbol{B}^\top ((\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})^\top)^{i-1}\ \boldsymbol{C}^\top Q, \end{aligned} \tag{3.20}$$

with $\boldsymbol{P}$ being a solution to *Algebraic Ricatti Equation*

$$\boldsymbol{P} = \boldsymbol{A}^\top \boldsymbol{P} \boldsymbol{A} + \boldsymbol{C}^\top Q \boldsymbol{C} - \boldsymbol{A}^\top \boldsymbol{P} \boldsymbol{B} (R + \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{A}. \tag{3.21}$$

In order to stabilize the robot in the 3D natural reference frame, we create two decoupled controllers following the above equations. One is stabilizing in x direction and the other in y direction. It should also be noted here that this controller, when used for walking, experiences an offset tracking error that appears after a longer distance, as noted by authors in [3]. In that case, an improved preview controller can be designed, which adds to the input an additional, integral action for tracking the error. For the kick engine, this offset is not observed and therefore the two controllers (original and an improved one) behave without any difference.

3.1.3 Proportional Arm Controller

When the robot performs the kick, one leg is raised and it stays in the air for a significant amount of time. Because of the weight of the kicking leg, the cart-table model assumption is broken. In this assumption the robot is modelled as having a center of mass in the torso, and the rest is mass-less. This assumption is not holding when the robot lifts the leg. Therefore, we utilize the arms movement in order to compensate for the weight of the kicking leg.

A simple *proportional controller* is used to controller the arms movement (see Figure 3.6). Input into the controller is the difference between the reference CoM and the current CoM measurement. Based on that difference and the heuristically obtained controller gain K_p, the arms are moved for the certain amount in one or the other direction, based on the error signal sign. To compensate for the CoM movements in both x and y axis, two decoupled controllers should be working in parallel. One controller is affecting the robot's arms pitch movement (movement in x direction), while the other is affecting the arms roll (movement in y direction).

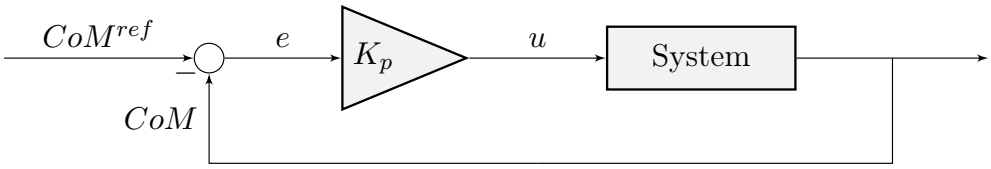


Figure 3.6: Control block diagram of the proportional arm controller

3.2 Trajectory Planning

While robot's torso movement, supporting leg and arms are stabilizing the robot on one leg, the other leg is free to move and perform the kicking motion. Firstly, the kicking leg will be moved away from the ball (swing back phase), then the leg will be extended towards the ball (kick phase), and finally returned to the initial position (return phase).

3.2.1 Swing Back

In the beginning of the swing back phase (see Chapter 2 and Figure 2.2 for the explanation of phases) the kicking leg is lifted above the ground and is in the neutral position. This initial lifted position is treated as a center of the circle of all positions that the leg can reach during the swing back phase. Using the information about the ball position coming from a computer vision engine (or directly from the simulation), the robot will swing the leg to the final position which is on the opposite side of the circle from where the ball lays, (see Figure 3.7). The trajectory from the initial lifted position to the final point is created with a linear interpolation.

This circle is heuristically obtained as the largest possible that does not make the robot fall. With the large circle, the swing back will be a motion with the large displacement. This will then lead to a long kicking trajectory and therefore a strong kick with better direction control. It should be noted that in the experiments that are used to acquire the circle size, it is crucial that the robot does not fall when reaching the points on the circle that are in front or behind the robot. On the other hand, if, for example, the robot is kicking with the left leg, it should not be considered a problem if the robot is falling when reaching the points on the far left side of the circle, as the swing back motion will anyway never be in that direction. Moreover, the circle should be cropped on the side where lays the support leg, so that the robot does not kick

itself when performing the motion. A few centimeters offset is left from the support leg surface as a safety margin so that collisions between the kick foot and the support leg are avoided.

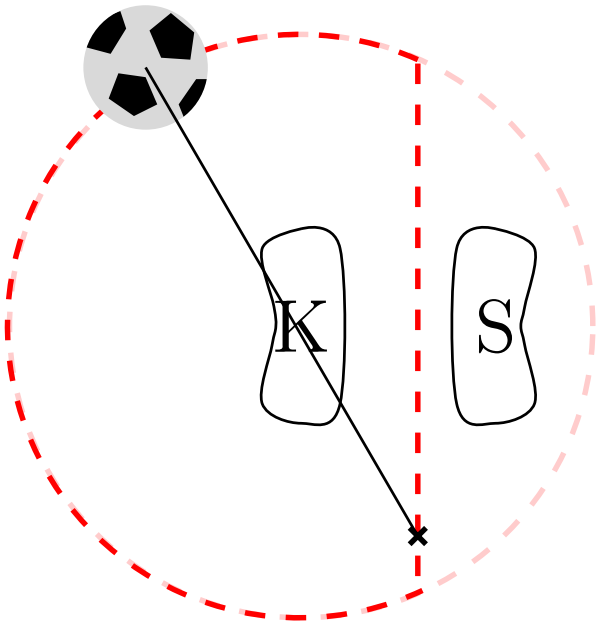


Figure 3.7: Geometric construction of the swing back: Kick and support feet are labeled as 'K' and 'S', respectively. Red cropped circle is the circle of attainable swing back positions with the kicking leg. Black cross marks the final swing back point when the ball is positioned as drawn.

3.2.2 Kicking the Ball

In this phase the goal is to create a trajectory for the foot to follow in order to impose a motion to the ball in the desired direction. *Bézier curves* are used to determine a path between two points: swing back point described in the previous subsection, and the point of contact with the ball.

The Bézier curve is a *parametric polynomial* derived from a set of *control points*. For $n+1$ control points denoted as $P_i,\ i \in [0, n]$, the curve is defined as

$$C(t) = \sum_{i=0}^{n} B_{i,n}(t)\ P_i, \tag{3.22}$$

where $B_{i,n}(t)$ is the *Bernstein polynomial* defined for the i^{th} control point as

$$B_{i,n}(t) = \binom{n}{i}\ t^i\ (1-t)^{n-i}. \tag{3.23}$$

From above equations it can be deducted that a curve with $n+1$ control points is of degree n. Generally, no control points except from the first and the last one lay on the curve, unless all points are collinear and the curve is a straight line. Nonetheless, the middle points affect the shape of the curve. For example, the tangent of the curve at the first control point will be a line containing control points P_0 and P_1.

Reasonable number of control points used to create a curve for the kick trajectory is three or four. Using only two control points would mean that the curve would always be a straight line, and using higher order curve would be impractical for the purpose of computational speed. With the appropriate curve, a kick motion can be adapted dynamically to strike a ball at any reachable position in a desired direction.

As already mentioned, the first control point P_0 is the final retraction point (cross in Figure 3.7). The last control point is the point of the contact between the foot and the ball. It lays on the surface of the ball, in the line that connects the ball and the target that the ball should reach. Middle control points are adjusting the trajectory curvature. Example of the Bézier curves created with three or four control points are presented in Figures 3.8 and 3.9. The curve with three control points is simpler, slightly faster to compute and resembles a straight line more which is better for the stability of the robot. On the other hand, the curve with four control points makes it easier to push a ball towards a custom direction.

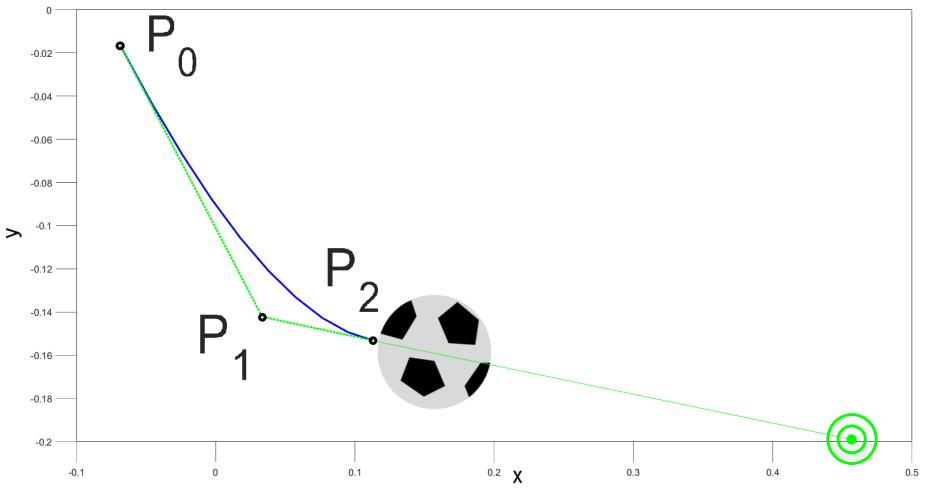
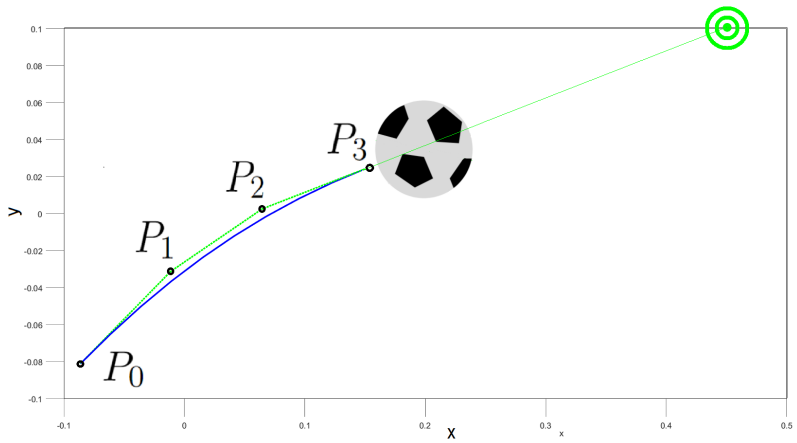


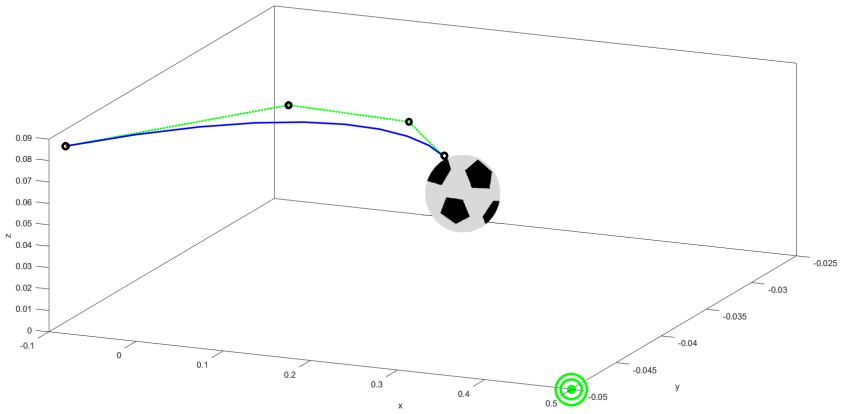
Figure 3.8: The Bézier curve (blue) created with three control points. P_0 is the point where the trajectory starts. Line that contains P_1 and P_2 is a tangent of the curve. P_1 is picked to be on the line that connects the ball coordinates and the target where the ball should aim (green target). P_2 is on the surface of the ball.

Once the curve is created, the leg will follow the curve by going through interpolation points. In one time step, the leg moves from one interpolation point to the other. If the number of interpolation points is greater, the movement will be smooth but slow, as the leg will cover only a small distance in one time step. Conversely, when the curve is interpolated only with a few points, the movement will be faster at the cost of sophistication (see Figure 3.10). Hence, the speed of the kick, which affects how far the ball is kicked, can be adjusted by choosing the number of interpolation points on the Bézier curve.

After the trajectory is created and the kick is performed, the robot will return the kicking leg back to the initial lifted position, using the linear interpolation between two end points. Afterwards, the robot will return the leg back to the ground and finally shift the center of mass back to the middle. With this, the whole kick motion is finished. It should be noted here that all points defined in this section are given in Cartesian coordinates. In order to transform these points into the robot's joint angles, the inverse kinematics module is created.

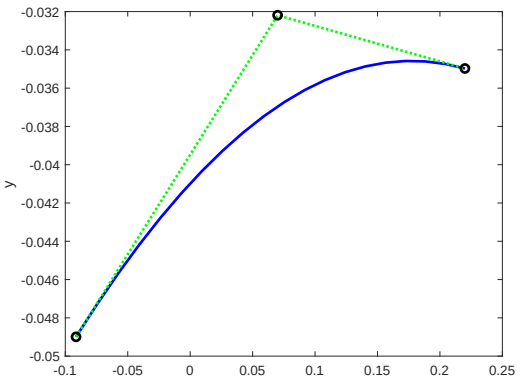


(a) The Bézier curve in xy plane

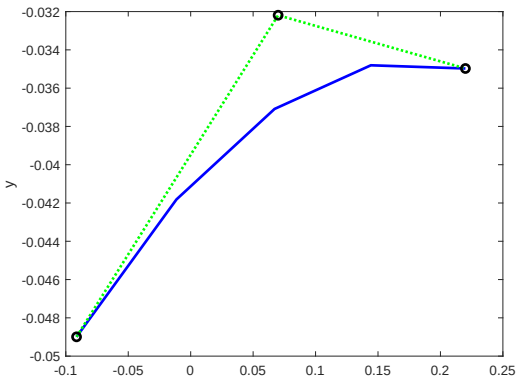


(b) The Bézier curve in 3D

Figure 3.9: The Bézier curve created with four control points (P_0, P_1, P_2, P_3). P_0, P_2 and P_3 are obtained as in Figure 3.8. The second control point is added such that it creates a symmetric curve, i.e. triangles $P_1P_0P_3$ and $P_2P_3P_0$ are similar.



(a) The Bézier curve interpolated with 20 points



(b) The Bézier curve interpolated with 5 points

Figure 3.10

3.3 Inverse Kinematics

In this section we will discuss how to obtain joint angles from the information about the robot's posture in Cartesian coordinates. Robot's torso (CoM) and support leg positions are coming from the stability module, whereas the kick leg position is obtained from the trajectory planning. The *inverse kinematics* is necessary here for determining the amount of joint rotation for the hips, knees and ankles required for achieving a certain movement. The analytical calculation is based on a geometry described in [3].

A few characteristics of a NAO robot should be taken into account when designing an inverse kinematics module. As opposed to most bipedal robots, NAO has hip joints inclined for 45° that make its pelvis similar to that of a human. Each joint has a motor that can be independently controlled. The only exception is that the hip yaw-pitch of both legs are mechanically connected.

Firstly, in order to simplify the following calculations, a *transformation matrix* is defined. Generally, the transformation matrix defines position and orientation of a limb b with respect to a limb a, i.e. the coordinate origin is in the limb a. It is written as

$$ {}^{a}\boldsymbol{T}_b = \begin{bmatrix} {}^{a}\boldsymbol{R}_b & {}^{a}\boldsymbol{p}_b \\ \mathbf{0} & 1 \end{bmatrix}. \tag{3.24} $$

${}^{a}\boldsymbol{R}_b \in \mathbb{R}^{3\times3}$ and ${}^{a}\boldsymbol{p}_b \in \mathbb{R}^{3\times1}$ are rotation and position matrices, respectively. $\mathbf{0}$ and 1 are added so that the dimensions are compatible. The matrix that contains the absolute position and orientation of the link b relative to a world frame is

$$ \boldsymbol{T}_b = \begin{bmatrix} \boldsymbol{R}_b & \boldsymbol{p}_b \\ \mathbf{0} & 1 \end{bmatrix}. \tag{3.25} $$

A coordinate transformation can be done by multiplying two transformation matrices, i.e.

$$ {}^{c}\boldsymbol{T}_b = {}^{c}\boldsymbol{T}_a \cdot {}^{a}\boldsymbol{T}_b. \tag{3.26} $$

We consider the information about the torso ($\boldsymbol{p}_{torso}$, $\boldsymbol{R}_{torso}$) and legs ($\boldsymbol{p}_{rightfoot}$, $\boldsymbol{R}_{rightfoot}$), ($\boldsymbol{p}_{leftfoot}$, $\boldsymbol{R}_{leftfoot}$) (and therefore $\boldsymbol{T}_{torso}$, $\boldsymbol{T}_{rightfoot}$ and $\boldsymbol{T}_{leftfoot}$ following 3.25) to be known inputs into the module. It should be noted that the rotation of any limb is not affected throughout the kicking motion. Limb lengths used in calculations can be found in [2] and are considered constant: D is the distance between the body torso and the hip, A is the upper leg length and B is the lower leg length (see Figure 3.11).

First step in the calculation is to obtain position of the hips. The calculations will be derived for the right leg. If the vertical distance between the torso and the hip is given as $D_{vertical}$ and the horizontal distance as $D_{horizontal}$, then the position of the hip relative to the frame located at the robot's torso is ${}^{torso}\boldsymbol{p}_{hip} = [0, D_{horizontal}, D_{vertical}]^\top$. Then, the hip position in the world frame is obtained as

$$ \boldsymbol{p}_{hip} = \boldsymbol{T}_{torso} \cdot {}^{torso}\boldsymbol{p}_{hip}. \tag{3.27} $$

Next, a relative pose of the hip node with respect to the foot node ($\boldsymbol{r}$ in Figure 3.11) is calculated as

$$ \boldsymbol{r} = \boldsymbol{R}_{\boldsymbol{rightfoot}}^{\top} (\boldsymbol{p}_{hip} - \boldsymbol{p}_{rightfoot}) = [r_x \; r_y \; r_z]^\top. \tag{3.28} $$

The first angle to be obtained is a knee pitch. The *cosine rule* applied to the triangle created by the thigh and the calf gives

$$ |\boldsymbol{r}|^2 = A^2 + B^2 - 2AB\cos(\pi - \angle\text{knee pitch}). \tag{3.29} $$

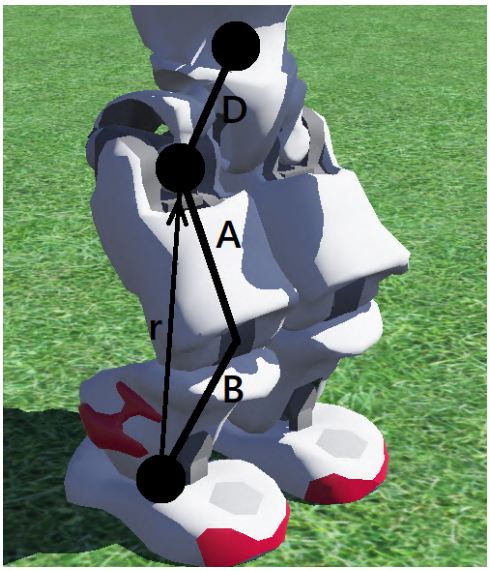


Figure 3.11: Definition of distances in a robot

From this ∠knee pitch is calculated as

$$\angle\text{knee pitch} = \pi - \arccos \frac{A^2 + B^2 - |\boldsymbol{r}|^2}{2AB}. \tag{3.30}$$

Similarly, applying the cosine rule to the angle near the foot in the same triangle gives

$$A^2 = B^2 + |\boldsymbol{r}|^2 - 2B|\boldsymbol{r}|\cos(\angle \boldsymbol{r}B) \tag{3.31}$$

which leads to

$$\angle \boldsymbol{r}B = \arccos \frac{B^2 + |\boldsymbol{r}|^2 - A^2}{2B|\boldsymbol{r}|}. \tag{3.32}$$

Apart from 3.32, the foot pitch is affected from the inclination of $\boldsymbol{r}$. This also affects the foot roll. Consequently, foot angles are obtained as

$$\begin{aligned} \angle\text{foot pitch} &= -\angle \boldsymbol{r}B - \arctan \frac{r_x}{\sqrt{r_y^2 + r_z^2}} \\ \angle\text{foot roll} &= \arctan \frac{r_y}{r_z}. \end{aligned} \tag{3.33}$$

It remains to obtain hip pitch, yaw and roll angles. The kinematic chain for the whole right leg is written as

$$\boldsymbol{R}_{rightfoot} = \boldsymbol{R}_{torso}\ \boldsymbol{R}_z(\angle\text{hy})\ \boldsymbol{R}_x(\angle\text{hp})\ \boldsymbol{R}_y(\angle\text{hr})\ \boldsymbol{R}_y(\angle\text{kp} + \angle\text{fp})\ \boldsymbol{R}_x(\angle\text{fr}), \tag{3.34}$$

where ∠hy stands for ∠hip yaw, ∠kp stands for ∠knee pitch, ∠fr stands for ∠foot roll, and so on. In order to obtain the angles related to the hip joint, all the terms should be rearranged such that the terms containing hip angles are on one side,

$$\boldsymbol{R}_z(\angle\text{hy})\ \boldsymbol{R}_x(\angle\text{hp})\ \boldsymbol{R}_y(\angle\text{hr}) = \boldsymbol{R}_{torso}^{\top}\ \boldsymbol{R}_{rightfoot}\ \boldsymbol{R}_x(-\angle\text{fr})\ \boldsymbol{R}_y(-\angle\text{kp} - \angle\text{fp}). \tag{3.35}$$

Right side of the equation

$$\boldsymbol{R} := \boldsymbol{R}_{torso}^{\top}\ \boldsymbol{R}_{rightfoot}\ \boldsymbol{R}_x(-\angle\text{fr})\ \boldsymbol{R}_y(-\angle\text{kp} - \angle\text{fp}) = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{3.36}$$

is calculated using that

$$\boldsymbol{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \qquad \boldsymbol{R}_y(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}. \tag{3.37}$$

By expanding the left side of the equation 3.35 and considering that elements in matrix $\boldsymbol{R}$ are known from 3.36 and 3.37, we can write

$$\begin{bmatrix} c_{\angle\text{hy}}c_{\angle\text{hp}} - s_{\angle\text{hy}}s_{\angle\text{hr}}s_{\angle\text{hp}} & -s_{\angle\text{hy}}c_{\angle\text{hr}} & c_{\angle\text{hy}}s_{\angle\text{hp}} + s_{\angle\text{hy}}s_{\angle\text{hr}}c_{\angle\text{hp}} \\ s_{\angle\text{hy}}c_{\angle\text{hp}} + c_{\angle\text{hy}}s_{\angle\text{hr}}s_{\angle\text{hp}} & c_{\angle\text{hy}}c_{\angle\text{hr}} & c_{\angle\text{hy}}s_{\angle\text{hp}} - c_{\angle\text{hy}}s_{\angle\text{hr}}c_{\angle\text{hp}} \\ -c_{\angle\text{hr}}s_{\angle\text{hp}} & s_{\angle\text{hr}} & c_{\angle\text{hr}}c_{\angle\text{hp}} \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}, \tag{3.38}$$

where $c_{\angle\text{hy}} \equiv \cos(\angle\text{hip yaw})$, and $s_{\angle\text{hr}} \equiv \sin(\angle\text{hip roll})$, and extract desired angles as

$$\begin{aligned} \angle\text{hip yaw} &= \arctan\frac{-R_{12}}{R_{22}} \\ \angle\text{hip pitch} &= \arctan\frac{-R_{31}}{R_{33}} \\ \angle\text{hip roll} &= \arcsin(R_{32}). \end{aligned} \tag{3.39}$$

Above equations are written for the right leg. For the left leg, the sign of D in the definition for ${}^{torso}\boldsymbol{p}_{hip}$ is inverted and the same calculations then apply. With these calculations, all required joint angles are calculated that will put the robot in the desired position. When using the calculations in a program on a real robot, in each iteration it needs to be checked whether the joint angles exceed their limits, so that the robot is not damaged. Joint limits can be found in [2].

Chapter 4

Simulation Study

Various omnidirectional kicks were generated and the data for the stability and trajectory planning was recorded, in order to verify the methodology presented in previous chapters. The experiments were carried out in *Webots simulator* created by the Cyberbotics [11]. The simulator has a development environment to model, program and simulate a wide range of mobile robots. The NAO robot can be simulated using the Webots C, C++, Python, Java or MATLAB programming languages. In this thesis the simulation study was conducted in MATLAB [1].

The simulations is created following the kick motion phases explained in Chapter 2. In Webots, there is a predefined loop that executes the simulations steps every couple milliseconds. Before the robot starts executing these iterations for the kick engine, it should choose whether it kicks with right or left leg, as well as how strong the kick should be. Namely, if the kick should be very precise, then the Bézier curve will be cretaed with 4 control points and interpolated with 10 points. If the kick should be strong as the target is far away, then it is better to have less control and interpolation points, for example 3 control and 5 interpolation points. Once this is decided, in the loop iterations the robot's position and orientation will be acquired through the sensors, and the CoM will be derived. The next kick foot position will be obtained from the trajectory planning module. After this, the stability module will come into the play, followed by the inverse kinematics. Lastly, robot's angles will be updated and the loop will be executed for the next time step. The overview of the algorithm can be seen in Figure 4.1.

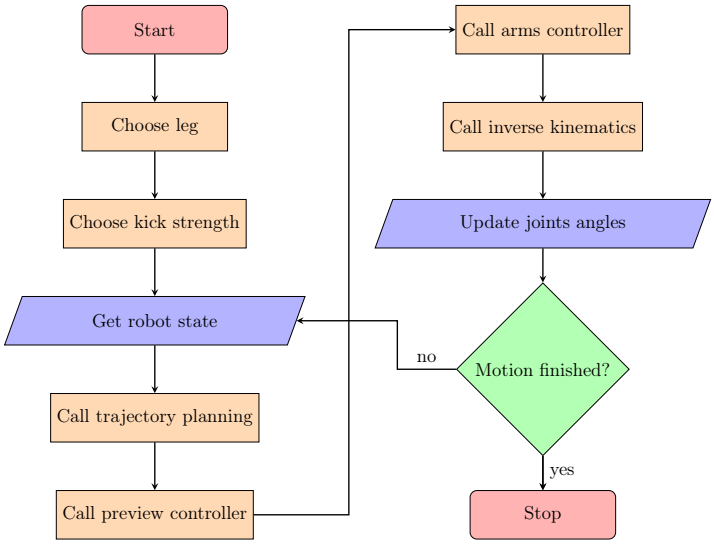
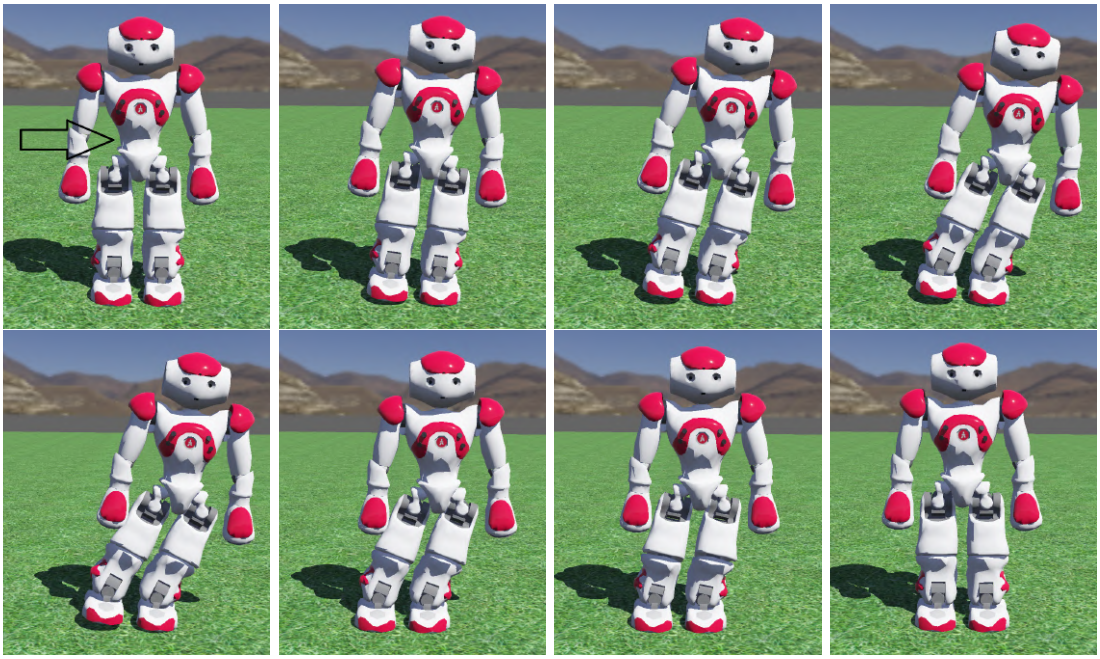


Figure 4.1: Flowchart for the simulation study

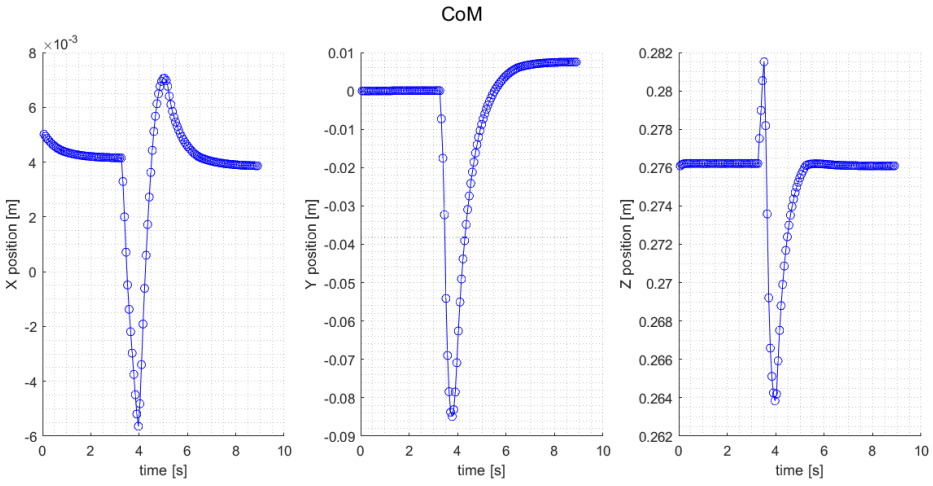
[1] Code used for the study available at `https://gitlab.ethz.ch/mdraskovic/semester-project-2`

4.1 Stability

The first experiment conducted to evaluate the stability is testing the preview controller when the robot is stabilizing on both legs. In that case the ZMP reference is the CoM projection to the ground, i.e. the point on the ground between two feet. The force that is simulating a push from the opponent is applied to the NAO in Webots simulator from various directions. In Figure 4.2 it is presented how the robot with a preview controller reacts when the force of 60N is applied from the side. With such force the robot falls without the controller. It is possible, however, that without the controller the robot does not fall if the force is weaker, for example 30N. Nevertheless, in that case the preview controller makes the deviation of the CoM 6 times smaller and overall improves robot's stability (Figure 4.3). Moreover, in Figure 4.4 is is presented how the preview controller stabilizes when the force is applied from the front or back.

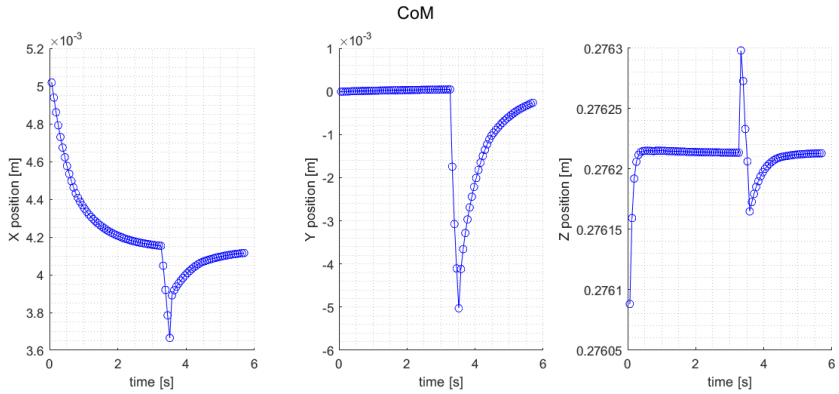


(a) Snapshots from Webots simulator

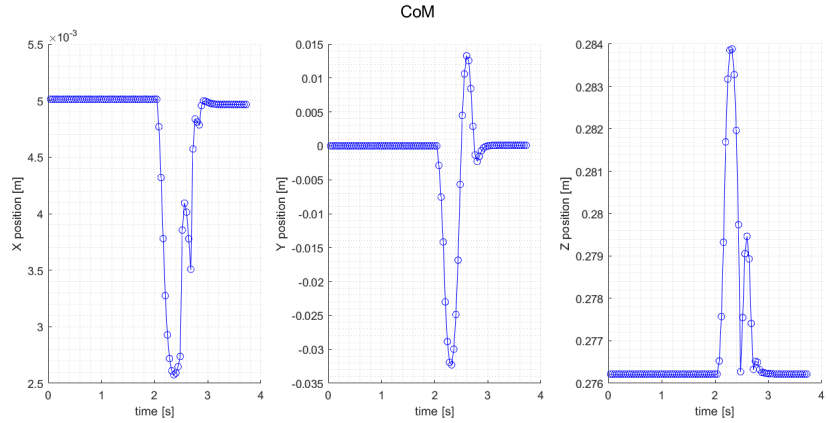


(b) Robot's CoM in world coordinate frame

Figure 4.2: NAO robot reacting to force of 60N applied from the side direction while stabilizing on both legs with the preview controller

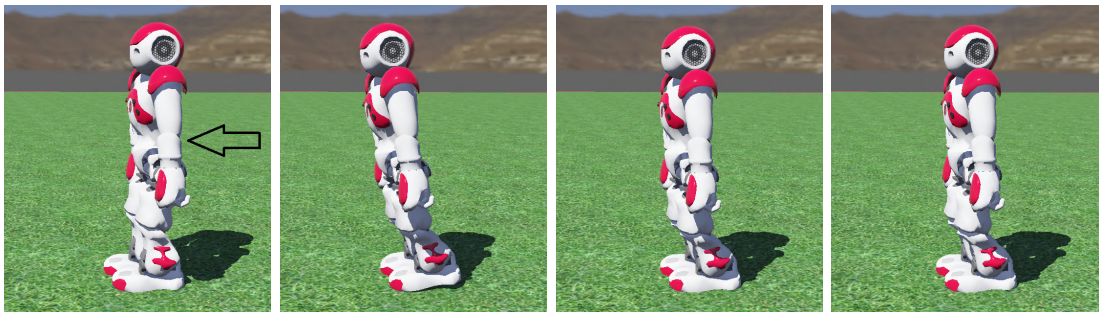


(a) With preview controller stabilized on both legs

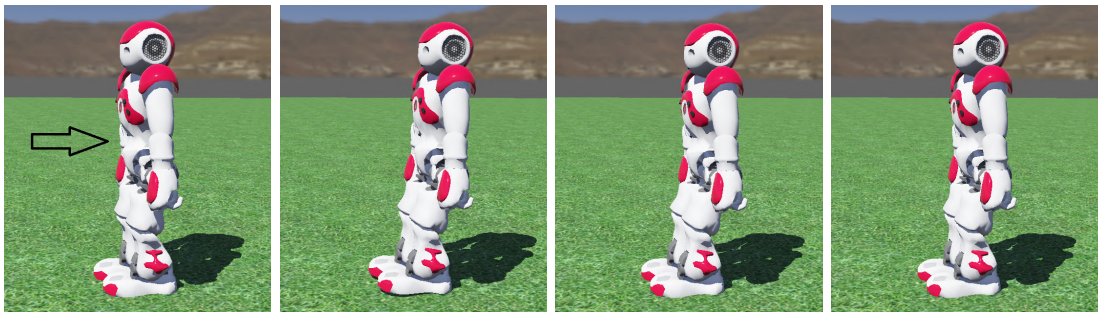


(b) Without preview controller

Figure 4.3: Robot's CoM in world coordinate frame when the robot is pushed with 30N force to the side



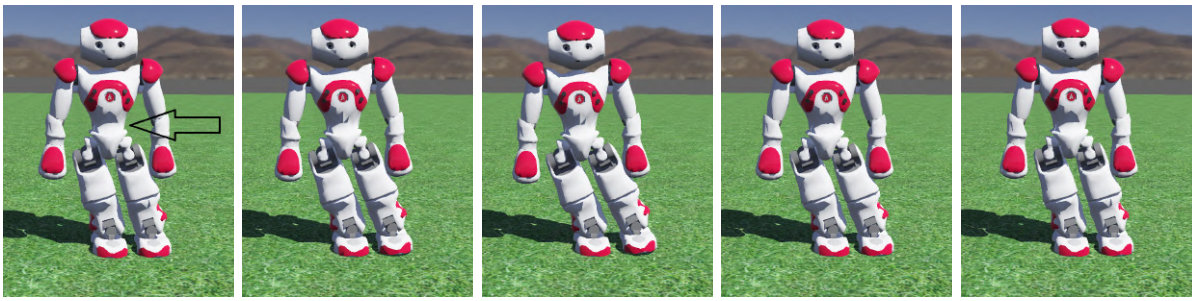
(a) Force of 55N applied from the back



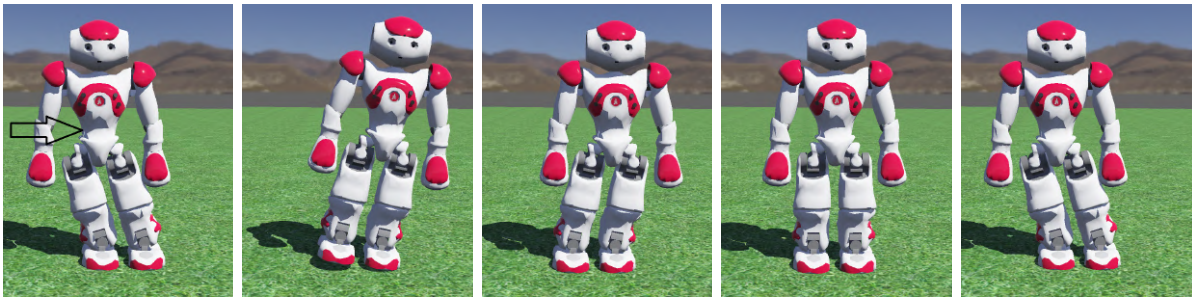
(b) Force of 30N applied from the front

Figure 4.4: NAO robot stabilized on both legs reacting to force

Now we have have shown that the robot stabilizes well on both legs and that it is able to remain upright when pushed in any direction. During the kick, however, the robot's weight should be shifted onto the supporting leg. In order to shift a CoM to the support leg, the ZMP reference in the preview controller is moved to be in the center of the support foot. The experiments from before are repeated (Figure 4.5).



(a) Force of 30N applied from the side



(b) Force of 90N applied from the side

Figure 4.5: NAO robot stabilized on the support foot reacting to side force

Even now when the CoM is shifted to the support foot, both feet are still on the ground. Thus, the preview controller alone is managing to stabilize the robot nicely. When the robot lifts the leg, the cart-table assumption around which the preview controller is built is violated and the robot falls under the weight of the kicking leg (Figure 4.6). To compensate for the leg weight, the proportional arm controller is added and the robot can successfully lift the leg (Figure 4.7). We also carried out experiments on how the robot stabilizes when it is balancing on one leg and the external force is applied. The results are presented in Figure 4.8. It can be seen that the arm motions contribute to the stability significantly.

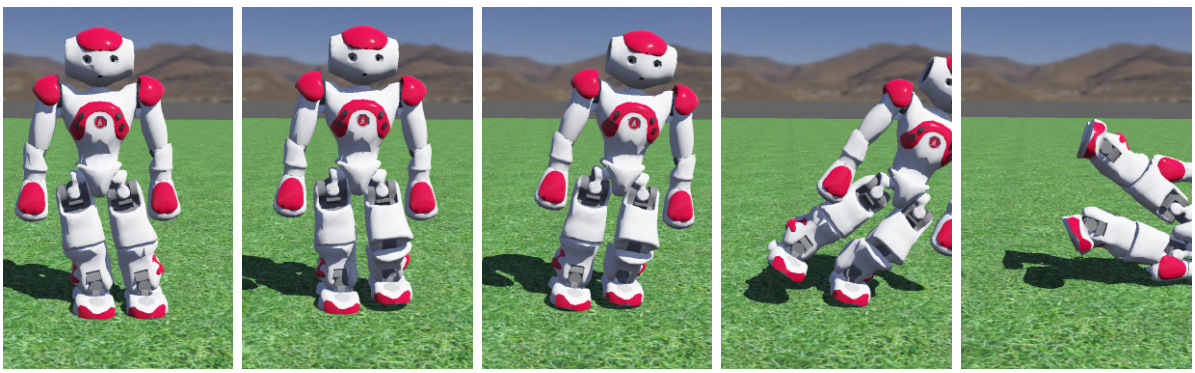


Figure 4.6: Robot trying to lift the leg only with a preview controller and falling

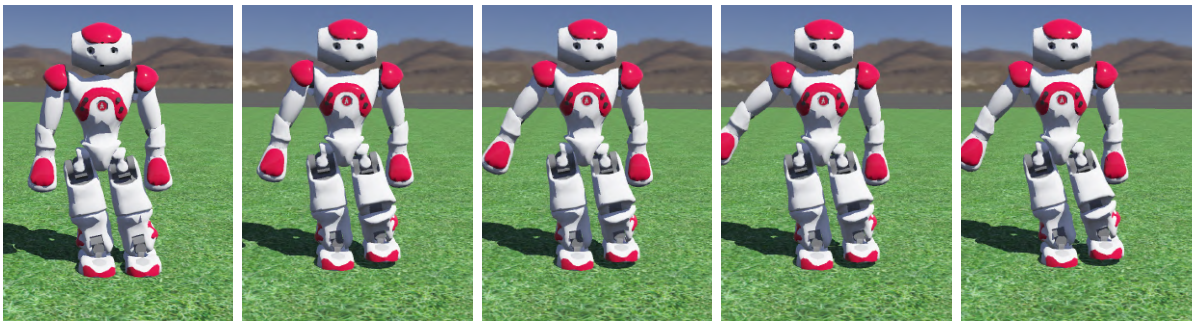


Figure 4.7: Robot successfully lifting the leg with preview and proportional arm controllers

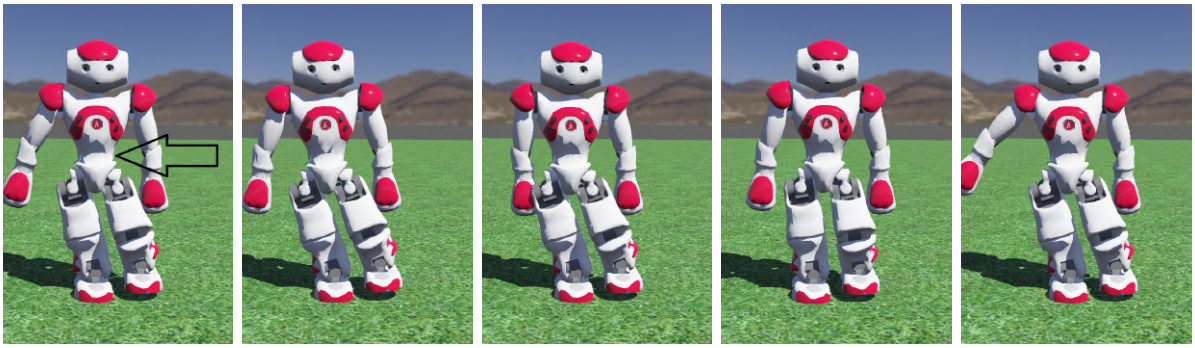


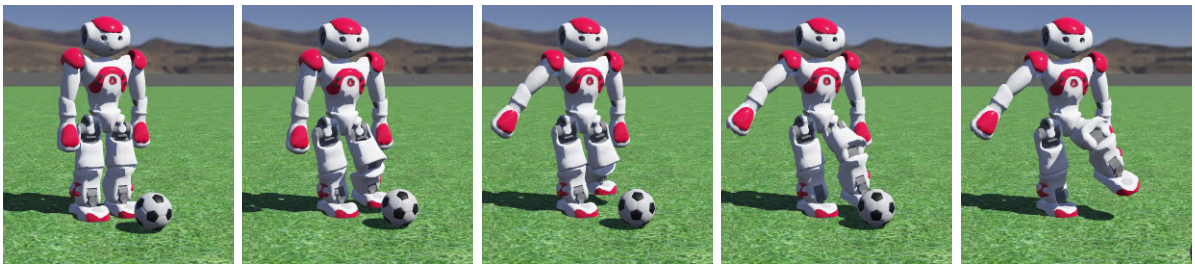
Figure 4.8: NAO robot balancing on one leg reacting to side force

4.2 Trajectory Planning

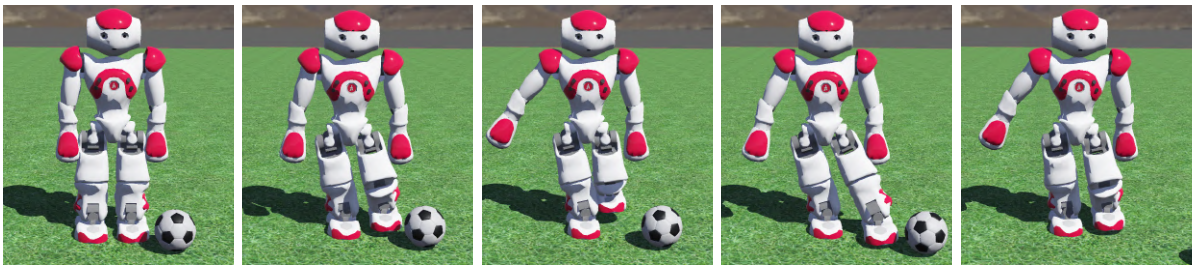
Once the stability has been established, we focus the experiments on trajectory planning. To test the accuracy of the kick, five precise kicks were performed at increasing intervals of 20°. In these experiments, the NAO robot kicks a standard RoboCup ball towards a target. Figure **??** visualizes these five kicks. The whole motion lasts for around 10s. The ball reaches distance between 0.5 and 1m, depending on the kick direction. Next, we show the cross kick, where the robot hits the ball with the right leg even though the ball is in front of the left leg. This is because the target is on the far left (Fig. 4.10). Then we show the less precise but stronger kick (Fig. 4.11) that can be used for the penalty kicks as the ball covers the distance of 1.6m.

A standard way of operating any motor in Webots is to control the end effector position directly (*position control*). In this type of control the user specifies target positions for the kick leg and then the *predefined PID controller* from the Webots manages the velocity, acceleration and motor force in order to move the leg between two target positions in two time steps. Good side of this predefined controller is that it makes sure that the motion is smooth and the user does not need to worry about this. However, the downside is that in the position control it is not possible to directly affect the velocity of the motors. The only way left to impact the speed of the kick is to provide targets further away from each other and this is done by interpolation a Bézier curve with less points, as explained in Chapter 3. Instead of position control the motors can also be operated with *velocity control*. This is used for continuous motors, such as the ones in wheeled robots, and the position cannot be controlled. Therefore, this is not an option for us.

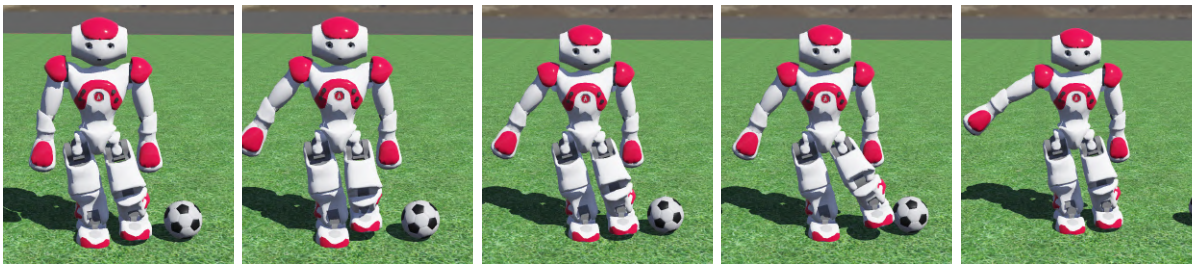
Lastly, we note that the Bézier curve is created between the swing back and kick phases, meaning that if the ball has moved since the motion started, the robot will be able to compensate. Thus, the requirement for the dynamic and adjustable kick is fulfilled (Fig. 4.12).



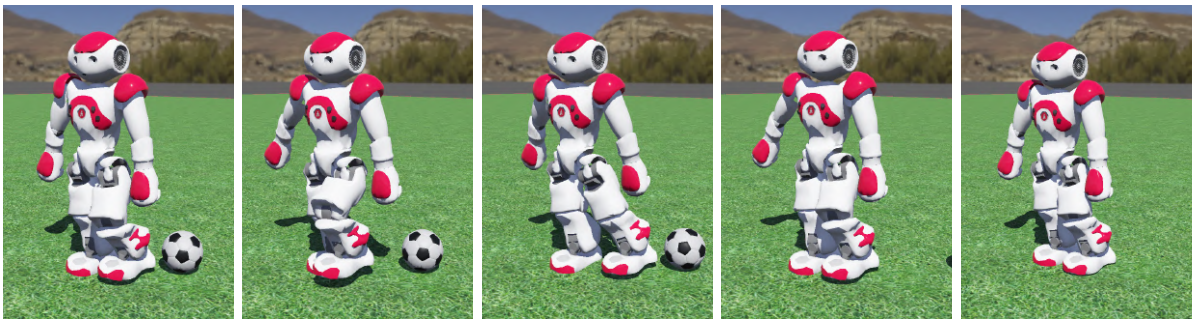
(a) Front kick



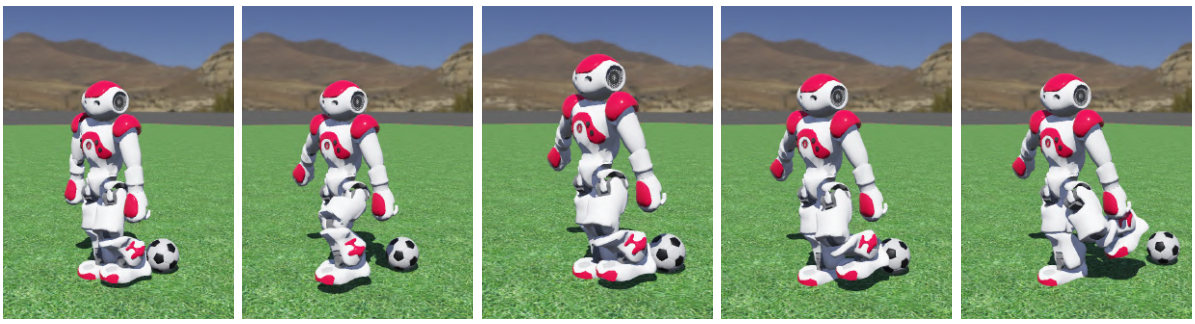
(b) Front-diagonal kick



(c) Side kick



(d) Back-diagonal kick



(e) Back kick

Figure 4.9: Precise kicks: Bézier curve created with 4 control points 10 interpolation points

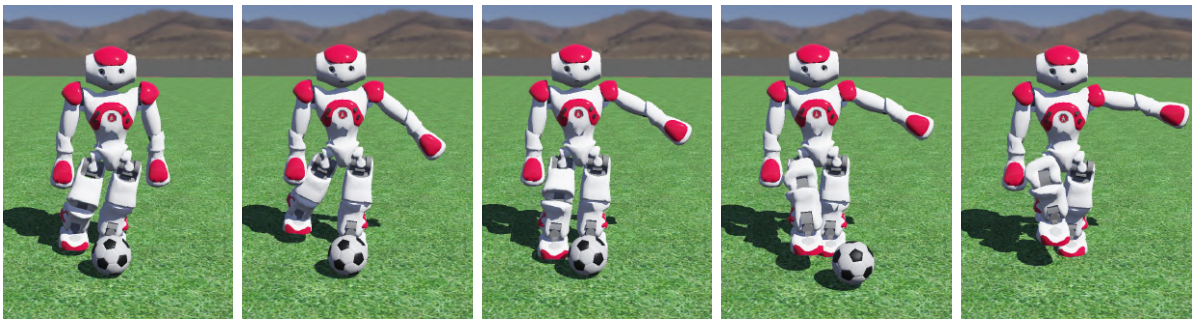


Figure 4.10: Cross kick

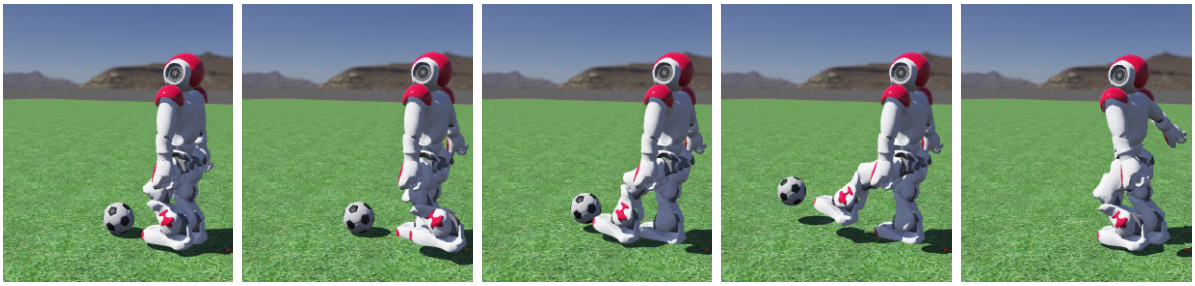


Figure 4.11: Strong kick: Bézier curve created with 3 control points 5 interpolation points

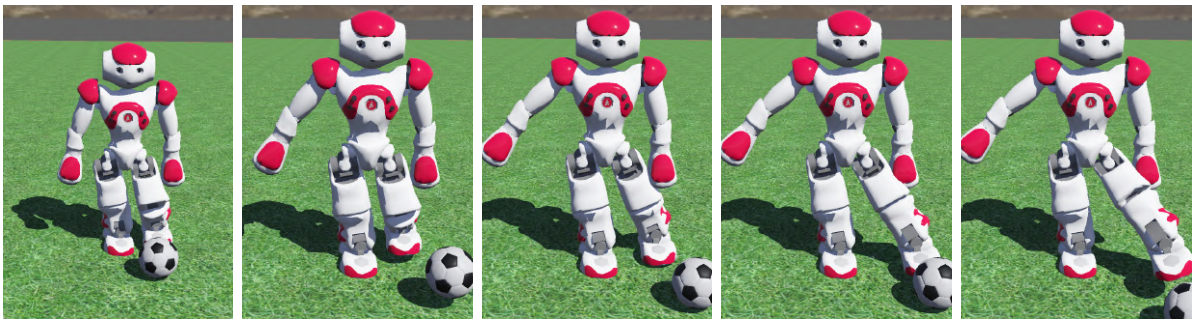


Figure 4.12: During the swing back phase the ball position is changed and the robot adjust the movement

Chapter 5

Conclusion

In this thesis, we propose a dynamic omnidirectional kick engine that will improve the performance of the NomadZ team in the RoboCup competition. We firstly described that the complete kick motion consists of seven phases performed in series. Namely, in the beginning the NAO robot will shift its CoM to the support leg and lift the kick leg. The leg will be swung away from the ball, and then the actual kick will be performed, after which the kick leg and CoM are returned to the initial positions. To achieve these movements, three modules are cooperating: stability, trajectory planning and inverse kinematics. ZMP based preview controller and proportional arms controller are working in parallel to balance the robot on the support leg. The kick trajectory is created using a Bézier curve. Number of control and interpolation points on the curve depend on the requirements for the kick. If the kick needs to be strong, then less points are utilized than when the weaker precise kick is demanded. Analytical inverse kinematics module will then transform the target position points from the Cartesian coordinates to the joint angles.

We carried out a simulation study in Webots simulator that displays the behaviour of the robot. The stability controllers were tested with applying force to the robot from the various directions. The purpose and performance of arms controller is especially noted. Then we tested the kick itself by placing the ball in several positions around the robot.

We note several interesting directions for future work. To improve the behaviour, optimization can be used in order to perform faster kicks and drive the ball further. Furthermore, it would be interesting to try the combined motions. For example, instead of walking towards the ball, then stopping and performing the kick, the walk-kick could be implemented, where the kick would be just one special step with changed control points withing the walk engine. Also, the omnidirectional kick can be achieved with the alignment process during the walk towards the ball. Another idea is instead of inventing the behaviour ourselves, to use reinforcement learning so that the robot learns actions that can be taken from own experience. Finally, the next step is to to embed the created kick engine into the rest of the framework and test it on the real NAO robot.

Appendix A

Webots Tutorial

We provide a short tutorial with some information about the basic concepts of Webots simulator (see Figure A.1).

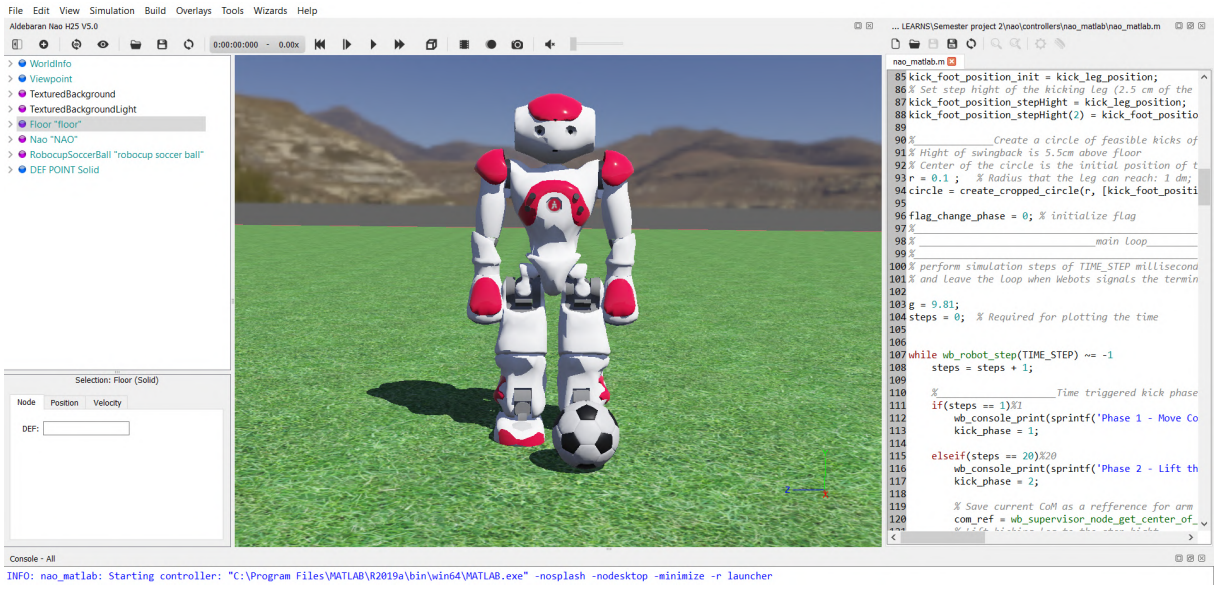


Figure A.1: Webots simulator

On the left side of the window, a *scene tree* can be seen, where all the *nodes* are listed. Nodes represent different objects existing in the simulation and they are enumerated as 0, 1, 2.... Each node can have sub-nodes and adjustable properties called *fields*. With these, we could for example set the floor to look like grass, or set the size of the ball to fit the RoboCup standard. Information about the nodes, for example where the objects are and their appearance, is written in the *world file*. This file also contains the initial state of a simulation.

The right side of the window is where the *controller* is. A controller is a program which defines the behavior of a robot. The robot node contains the controller field, where the desired controller should be specified. The controller can be written in C, C++, Java, Python or MATLAB (which we used). Once the controller file is created, a `while` loop that will execute every few milliseconds will be already defined. `TIME STEP` is the variable which defines how fast the loop will be executed. This is also predefined, but can be modified. Everything that needs to be defined before the kick motion should be written before the `while` loop, for example loading the instances, enabling robot's devices, etc. Everything that is inside the loop will be continuously executed until the signal for termination is received.

To create instances of nodes in the code, they need to be accessed from the scene tree:

```
wb_robot_init();
root = wb_supervisor_node_get_root(); % Returns the scene tree
children = wb_supervisor_node_get_field(root,'children'); % Returns list of nodes

% If 5th, 6th and 7th node in the scene are robot, ball and target
robot = wb_supervisor_field_get_mf_node(children,5);
ball = wb_supervisor_field_get_mf_node(children,6);
target = wb_supervisor_field_get_mf_node(children,7);
```

In order to be able to access robot's joints, we need to see how they are defined in the robot's *PROTO file*. This files contains definitions of all motors. Following code creates the instance of the robot's right foot roll joint if that joint is defined in the PROTO file as 'RAnkleRoll':

```
RAnkleRoll = wb_robot_get_device('RAnkleRoll');
```

Robot's links are defined in PROTO file as end points to the corresponding joints. To access the position of the right foot, the following code is used:

```
RAnkleRoll_node = wb_supervisor_node_get_from_device(RAnkleRoll);
RAnkleRoll_hinge_joint = wb_supervisor_node_get_parent_node(RAnkleRoll_node);
RAnkleRoll_endPoint_ref =wb_supervisor_node_get_field(RAnkleRoll_hinge_joint,'
    endPoint');
RAnkleRoll_end_point = wb_supervisor_field_get_sf_node(RAnkleRoll_end_point_ref);
Right_foot_position = wb_supervisor_node_get_position(RAnkleRoll_end_point);
```

Position, orientation and center of mass of the nodes (or links) can be obtained as

```
position = wb_supervisor_node_get_position(robot);
orientation = wb_supervisor_node_get_orientation(robot);
com = wb_supervisor_node_get_center_of_mass(robot);
```

Although center of mass positions is not a value usually accessible to the robot, we assumed here that the CoM calculation already exists and can be used from other engines. Positions of the motors (linear or angular, depending on the motor type) is obtained as

```
RAnkleRoll_position = wb_motor_get_target_position(RAnkleRoll);
```

After the inverse kinematics module outputs desired joint angles, they can be updated as

```
wb_motor_set_position(RAnkleRoll, desired_angle);
```

Lastly, one can notice that *supervisor node* is used in some of the above mentioned functions. A supervisor is a special type node that has additional powers. Any node can be turned into a supervisor when its field 'supervisor' is set to true. Some of the possibilities of the supervisor node is that it can modify the environment or acquire measurements about the state of the node.

Bibliography

[1] https://www.robocup.org/.

[2] https://www.softbankrobotics.com/emea/en/nao.

[3] Shuuji Kajita et al. *Introduction to Humanoid Robotics*. Springer Publishing Company. 2016.

[4] Pedro Pena, Joseph Masterjohn, and Ubbo Visser. *An Omni-Directional Kick Engine for Humanoid Robots with Parameter Optimization*. Springer Nature Switzerland AG 2018; H. Akiyama et al. (Eds.): RoboCup 2017, LNAI 11175, pp. 385–397. 2018.

[5] Emile Bahdi. *Development of a Locomotion and Balancing Strategy for Humanoid Robots*. University of Denver, Electronic Theses and Dissertations. 1406. 2018.

[6] Rui Ferreira et al. *Development of an Omnidirectional Kick for a NAO Humanoid Robot*. Springer Berlin Heidelberg, pp. 571-580. 2012.

[7] Judith Muller, Tim Laue, and Thomas Rofer. *Kicking a Ball – Modeling Complex Dynamic Motions for Humanoid Robots*. J. Ruiz-del-Solar, E. Chown, and P.G. Ploeger (Eds.): RoboCup 2010, LNAI 6556, pp. 109–120. 2010.

[8] Dimitri Eckert. *A Parametrized, Dynamically Balanced Kick for NAO Robots*. Semester project. 2019.

[9] Amin Rezaeipanah, Parvin Amiri, and Shahram Jafari. *Performing the Kick During Walking for RoboCup 3D Soccer Simulation League Using Reinforcement Learning Algorithm*. International Journal of Social Robotics. 2020.

[10] Miomir Vukobradovic and Branislav Borovac. *Zero-Moment Point - Thirty Five Years on its Life*. International Journal of Humanoid Robotics 1(1), pp.157–173. 2004.

[11] https://www.cyberbotics.org/.