Pedro Pena, Joseph Masterjohn, and Ubbo Visser. An Omni-directional Kick Engine for Humanoid Robots with Parameter Optimization. In Hidehisa Akiyama, Oliver Obst, Claude Sammut, and Flavio Tonidandel, editors, RoboCup 2017: Robot Soccer World Cup XXI. Springer Berlin / Heidelberg, 2018. to appear.

An Omni-directional Kick Engine for Humanoid Robots with Parameter Optimization

Pedro Pena and Joseph Masterjohn and Ubbo Visser

University of Miami, Department of Computer Science,
1365 Memorial Drive, Coral Gables, FL, 33146, USA
E-Mail: {pedro|joe|visser}@cs.miami.edu

Abstract. Incorporating a dynamic kick engine that is both fast and effective is pivotal to be competitive in one of the world's biggest AI and robotics initiative: RoboCup. Using the NAO robot as a testbed, we developed a dynamic kick engine that can generate a kick trajectory with an arbitrary direction without prior input or knowledge of the parameters of the kick. The trajectories are generated using cubic splines (two degree three polynomials with a via-point). The trajectories are executed while the robot is dynamically balancing on one foot. When the robot swings the leg for the kick motion, unprecedented forces might be applied on the robot. To compensate for these forces, we developed a Zero Moment Point (*ZMP*) based preview controller that minimizes the ZMP error. Although a variety of kick engines have been implemented by others, there are only a few papers on how kick engine parameters have been optimized to give an effective kick or even a kick that minimizes energy consumption and time. Parameters such as kick configuration, limit of the robot, or shape of the polynomial can be optimized. We propose an optimization framework based on the Webots simulator to optimize these parameters. Experiments of the physical robot show promising results.

1 Introduction

Generating dynamic motions on a robot without explicitly programming the motion is a difficult task and a fairly new research area. Kick engines such as [**?**, **?**, **?**, **?**, **?**, **?**] have been developed for the NAO robot and although they are dynamic, there are static values incorporated into the kicks such as retraction points, foot position from floor, hip/ankle pitch and hip/ankle roll ratio, and shapes of trajectories. These values are usually derived from empirical observations and are not guaranteed to be optimal values. The difficulty of the task is to optimize parameters on the physical robot because the robot is limited by hardware, energy consumption, and most importantly real time execution. Hence, the robot cannot run for thousands or millions of iterations to get a good set of parameters. Other dynamic kick engines such as [**?**] developed a kick engine for the THOR-MANG from Robotis that generate static kick motions. Lengagneua et al. [**?**] have developed a kick engine that incorporates optimization offline and a re-planning step when the kick is executed but do not optimize on the physical

robot or use a simulation software that takes into account hardware properties of the robot. We propose a new kick engine for the NAO robot that can dynamically balance on one foot using a Zero Moment Point (ZMP) based preview controller while also generating kick trajectories using cubic splines. The values of the kick trajectories are optimized in the Webots Simulator to get a good set of parameter values. The overview of the control frame work is presented in **??**.

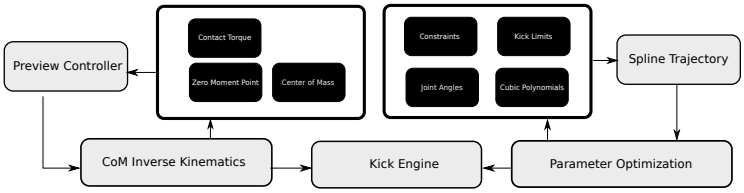

Fig. 1: Overview of the control framework for the kick engine

The major components of the control framework are preview controller, spline trajectory, center of mass (CoM) inverse kinematics, and parameter optimization. The paper is organized as follows: we discuss relevant work in the next section and describe the ZMP based preview controller in Section **??**. Section **??** describes how to move the center of mass to the desired position, and Section **??** describes how the kick trajectory is generated. The model optimization on the kick is discussed in Section **??**, and our experiments and results are explained in Section **??**, followed by the conclusion in Section **??**.

2 Related Work

Wenk et al. [**?**] developed a kick engine that generates online kick motions using trajectories generated by Bzier curves, but in order to create such motions, the humanoid robot has to dynamically balance on one foot so it can handle any force generated by the kick. In order to find these forces, Wenk et al. used inverse dynamics to calculate the ZMP. Inverse dynamics can be computed such that given the acceleration, we work out the forces. The Inverse Dynamics problem was solved using a variation of the Recursive Newton Euler Algorithm (RNEA). For the kick trajectories the authors used Bzier curves that are continuously differentiable to generate a kick.

Böckmann et al. [**?**] provided a mass spring damper model to model motor behavior, and modified the ZMP equation to account for this behavior to get the actual motor position rather than a believed state. The authors also adapted Dynamic Motion Primitives (DMP) to generate kick trajectories. Kick trajectories are usually generated from Bzier curves or via-point kick trajectories, but here the authors used a PD controller with a forcing term in the transformation system to control the shape of the trajectory.

Sung et al. [**?**] used full body motion planning and via-point representation to generate joint angle trajectories. These trajectories are generated using five degree polynomials and via points are specified to constrain the swing trajectory.

For the balancing the authors also used ZMP. In order to create efficient full body motion trajectories the author used optimization techniques such as Semi-Infinite Programming (SIP) to specify constraints such as minimal energy and torque. The optimization also dealt with joint redundancy.

Yi et al. [**?**] used THOR-OP (Tactical Hazardous Operations Robot - Open Platform) full sized humanoid robot to generate kick and walk motions for the AdultSize League in RoboCup 2014. The robot had a hybrid walking controller that used two types of controllers: ZMP preview controller and a ZMP based reactive controller. The ZMP based reactive controller used techniques such as Central Pattern Generators to create motions that require less computation than the ZMP preview controller. The kick motions generated were handled by the hybrid walking controller to create smooth transitions between the dynamic walk and strong kick.

The kick engine of Xu et al. [**?**] is separated into four phases: preparation, retraction, execution, and wrap-up phase. The authors use a grid space to find the kick that maximizes the distance in the retraction point and minimizes the angle between the direction of the foot and the direction of the ball. The maximum distance of the retraction point of the foot is assumed to create the greatest impulse. The stabilization of the robot is done with a Body Inclination Control that controls the torso angle to maintain the center of mass in the support polygon. The authors also found the reachable space through experimentation. The kick engine was experimented on the NAO robot.

Although these approaches use a dynamic kick engine that utilizes a controller for balancing and generate kicks online or offline, they all lack a framework that uses model optimization to find kicks under certain constrains such as strongest kick or fastest kick. In our approach, we use a controller for balancing and cubic splines for kick trajectory and then we use optimization to find a good set of parameters for the kick. Next, we will discuss each component of the kick engine.

3 ZMP based Preview Controller

This section describes how the robot stabilizes on the supporting foot. A ZMP based Preview Controller is used to keep the Zero Moment Point (ZMP) in the support polygon. The first section describes the definition of ZMP and how it is derived. The following section will illustrate how the Preview Controller uses the ZMP to stabilize the robot.

3.1 Zero Moment Point

In order for the humanoid robot to balance, it has to maintain contact with the ground. When the robot is at rest, the Center of Mass (CoM) has to be inside the support polygon. The CoM is defined by:

$$CoM = \frac{m_j \mathbf{c}_j}{M} \tag{1}$$

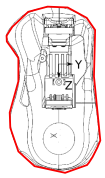

Fig. 2: Support Polygon

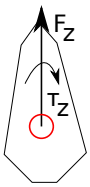

Fig. 3: ZMP location

where $(m_j, \mathbf{c}_j)$ is the weight and position vector respectfully of the j^{th} link, and $M = \sum m_j$. The support polygon is the area touched by the robot and the ground. When the robot is kicking, the area touched by the robot in the ground is the supporting foot. The support polygon is the convex hull of the foot touching the ground and it is defined by

$$CoS = \left\{ \sum \alpha_j \mathbf{p}_j | \alpha_j \geq 0, \sum \alpha_j = 1, \mathbf{p}_j \in S(j = 1, ..., N) \right\}$$

where $S(j = 1, ..., N)$ is the set of edges of the supporting foot. When the robot is at rest, the CoM criterion is enough to stabilize the robot, but if the robot is in motion, the CoM might leave the support polygon. In that case, we can use another criterion to verify the supporting foot still has contact with the ground.

The dynamic criterion for the robot to maintain contact with the ground is the Zero Moment Point (ZMP) [**?**]. In order to correctly balance the robot, roboticists use ZMP to keep the robot from falling. The ZMP is the point where the robot's contact point with the ground does not generate a moment about the y and x axis. This is defined by

$$ZMP = \{ p_o \times f + \tau = 0 | \tau_x, \tau_y = 0 \} . \tag{2}$$

Therefore the robot does not rotate about these axes and tip over. Notice that the ZMP criterion does not say anything about the moment about the z axis. This rotation is allowed in the ZMP but it does not fall but rotate about the z axis. In order to derive the ZMP, the x and y components of (**??**) need to be solved and approximated by discretized points:

$$ZMP_{approx.} = \{ (p_i - p_o) \times f_i + \tau_i = 0 | \tau_{ix}, \tau_{iy} = 0 \} .$$

Therefore,

$$p_x = \frac{\sum_{i=1}^{N} -\tau_{iy} - (p_{iz} - p_z) f_{ix} + p_{ix} f_{iz}}{\sum_{i=1}^{N} f_{iz}} \; and \tag{3}$$

$$p_y = \frac{\sum_{i=1}^{N} \tau_{ix} - (p_{iz} - p_z) f_{iy} + p_{iy} f_{iz}}{\sum_{i=1}^{N} f_{iz}} . \tag{4}$$

This is a very simple model, the full dynamics of the physical world can not be captured. Hence, a Kalman filter was used to add noise to the model. The measurement model used for the Kalman filter was the foot sensors from the

NAO robot. The data from the foot sensors were used to solve for (**??**) and (**??**). The foot sensors give us values for the normal ground force exerted at the contact points. Therefore all the terms in (**??**) and (**??**) except f_{iz} are zero, resulting in

$$p_x = \frac{\sum_{i=1}^{N} p_{ix} f_{iz}}{\sum_{i=1}^{N} f_{iz}} \quad p_y = \frac{\sum_{i=1}^{N} p_{iy} f_{iz}}{\sum_{i=1}^{N} f_{iz}}. \tag{5}$$

The NAO robot has four sensors in the each foot, and it can be used to measure the magnitude of the force. Using (**??**) and the magnitude and position of each sensor, the empirical ZMP can be computed. For the belief model of the Kalman filter, we use (**??**) and we approximate the ZMP using a linear inverted pendulum [**?**, **?**]. The dynamics of the inverted pendulum is well understood in physics and it is used to model the dynamics of balancing for the humanoid robot. The top of the pendulum is assumed to be the CoM of the robot and the bottom part is assumed to be negligible. Therefore, we need to assume that the legs of the robot do not weigh much compared to the center of mass of the robot. Therefore, the linear and angular momentum of the center of mass is

$$\mathcal{P} = M\dot{\mathbf{c}} \quad \mathcal{L} = \mathbf{c} \times M\dot{\mathbf{c}} \tag{6}$$

To compute the approximated ZMP, we use (**??**) and (**??**) and get,

$$p_x = \frac{(z-p_z)\ddot{x}}{\ddot{z}+g} \quad p_y = \frac{(z-p_z)\ddot{y}}{\ddot{z}+g} \tag{7}$$

With a belief and measurement model, we can get a close approximation to the ZMP. We can now use a controller that will control the ZMP to maintain it in the polygon of the supporting foot. This will be discussed in the next section.

3.2 Preview Controller

The ZMP Preview Controller is based on the cart table model and it is a closed-loop system [**?**, **?**, **?**]. This model assumes that given the cart on the table, the Center of Pressure (CoP) (foot of the table) is affected, and in turn, affects the stability of the table (ZMP is affected). The ZMP Preview Controller is based on future ZMP positions of the robot. Since the robot is not walking, it is assumed that the robot will have the same future preview gains. This is an important simplification because it helps with the computation time. The Preview Controller uses a Linear Quadratic Regulator (LQR) to find the optimal values of the gains,

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

where R and Q are weights. The LQR tries to solve a Riccati differential equation for the optimal values,

$$\mathbf{P} = \mathbf{A}^T\mathbf{P}\mathbf{A} + \mathbf{c}^T Q\mathbf{c} - \mathbf{A}^T\mathbf{P}\mathbf{b}(R + \mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T\mathbf{P}\mathbf{A}$$

where,

$$\mathbf{A} = \begin{bmatrix} 1 & \delta t & \delta t^2/2 \\ 0 & 1 & \delta t \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \delta t^3/6 \\ \delta t^2/2 \\ \delta t \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 & 0 & -z/g \end{bmatrix}$$

and tries to minimize the cost function $\mathbf{J}$ by using the input and $\mathbf{K}$ such that,

$$u_k = -\mathbf{K}x_k + \begin{bmatrix} f_1 & f_2 & \dots & f_N \end{bmatrix} \begin{bmatrix} p^{ref}_{k+1} \\ \vdots \\ p^{ref}_{k+N} \end{bmatrix}$$

where

$$\mathbf{K} = (R + \mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T\mathbf{P}\mathbf{A}$$
$$f_i = (R + \mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T(\mathbf{A} - \mathbf{b}\mathbf{K})^{T*(i-1)}\mathbf{c}^T Q$$

This calculation is only done once and it is used for the Preview Controller. When all the optimal values have been calculated, the state transition equation can be used to compute the next CoM position, velocity and acceleration,

$$\mathbf{x_{k+1}} = \mathbf{A}\mathbf{x_k} + \mathbf{b}u_k$$

It is important to note that $p^{ref} = \left\{ p^{ref}_{k+1}, \dots p^{ref}_{k+N} \right\}$ will always be the same because p^{ref} will always be in the supporting foot of the kick and it will not move. Therefore, it will save computation time since p^{ref} is a constant when computing u_k. When u_k is computed and we find the next CoM, then we need to move the CoM to next position. This will be discussed in the next section.

4 Analytical Approximation for Inverse Kinematics

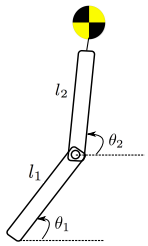

Fig. 4: An approximation of the center of mass kinematic for Inverse Kinematic model. l_1 is the length from the ankle to the hip and l_2 is the length from the hip to the center of mass. This is approximation is done to the frontal plane and saggital plane. The hip pitch ($\theta_{2,pitch}$), hip roll ($\theta_{2,roll}$), ankle pitch ($\theta_{1,pitch}$) and ankle roll ($\theta_{1,roll}$) are adjusted according to the Inverse Kinematic model.

In order to move the CoM to the position that was given by the Preview Controller, an analytical approximation using Inverse Kinematics can be used

to move the joints to the correct position. Since the Center of Mass is not directly actuated by a specific joint, an analytical approximation can be used. The position of the CoM can be affected by the leg joints. Let's assume the CoM is above the leg joints, then we can approximate the CoM using the hip joints and ankle joints. In the frontal plane, the CoM can be affected by the hip and ankle roll. If we approximate the joints and the CoM using a two-link arm [**?**], then we can use an analytical solution to move the CoM. The first link of the arm is the ankle roll joint to the hip roll joint. The second arm is the hip roll joint to the center of mass. The link of the first and second arm are revolute joints, and the end effector which is the CoM does not have a joint. The length of the first arm is the distance from the ankle roll to the hip roll, l_1 and the length of the second arm, l_2, is the distance from the hip roll to the CoM. In the frontal plane, we want our end-effector to get to (y, z). y is the difference between the next CoM's y position and the current CoM's y position. z is the plane that CoM is constrained on. Now, that the problem is defined, we can use these values to get the two angles we need, but we first need to get the range from the first link to the next CoM position in the frontal plane, (y, z),

$$r = \sqrt{(y - \alpha_{ankle,y}) + (z - \alpha_{ankle,z})} \tag{6}$$

With r, the second angle, θ_2, can be found. But before computing θ_2, we need to find the angle α that corresponds to the angle formed by l_1, l_2, and r. Therefore, we can use the cosine law to get the following,

$$\alpha = \cos^{-1} \frac{l_1^2 + l_2^2 - r^2}{2l_1 l_2}, \quad \theta_2 = \alpha \pm \pi \tag{7}$$

Therefore there are two solutions for θ_2 but only one can give us the correct solution because of the balancing constraint placed on the robot. Hence the correct solution is as follows

$$\theta_2 = \begin{cases} \alpha - \pi, & \text{if right kick} \\ \alpha + \pi, & \text{else left kick} \end{cases}$$

To compute θ_1, the angle opposite of l_2, β, needs to be calculated first. To calculate β, the law of cosines can be reused and we end up with,

$$\beta = \cos^{-1} \frac{r^2 + l_1^2 - l_2^2}{2l_1 r}, \tag{8}$$

β gives the angle in the triangle, but the θ_1 is the angle between y-axis and l_1. To get this angle, the bearing of the first link to the the next CoM position is needed,

$$\arctan 2(z, y), \quad \theta_1 = \arctan 2(z, y) \pm \beta \tag{8}$$

There are also two solutions for θ_1, and a condition is given to decide which on to use,

$$\theta_1 = \begin{cases} \arctan 2(z, y) - \beta, & \text{if right kick} \\ \arctan 2(z, y) + \beta, & \text{else left kick} \end{cases}$$

The same needs to be do in the sagittal plane. The only difference in the sagittal plane, is that instead of hip and ankle roll, hip and ankle pitch will be used, and the trigonometric solution is done for point (x, z). When both calculations are done for the frontal and saggital plane, the current CoM position can now be inferred from these four joint angles.

5 Kick Trajectory using Cubic Spline

In order to generate kick trajectory, the most trivial case is to change the leg joint angles until a desired kick configuration has been reached. This is tedious work and will be suboptimal since the joint space is very large. Another approach is to use optimization to find key-frame values, but this only works for a set of kicks and it is not dynamic enough to create any kick trajectory. Therefore, a more efficient solution is to generate motions using polynomials [**?**]. Polynomials are a great solution in robot motion because they can be configured to generate smooth curves. To generate a polynomial for the kick motion requires constraints such that the motion generated by the polynomial does not conflict with any unwanted configuration. The polynomials will not be used in the joint space, but rather will be used to determine the next position of the swinging foot. When the position of the foot is determined by the cubic polynomial, the Inverse Kinematic module will provide the angles for the foot position requested. For the kick motion, two cubic polynomials are generated. The point where the two polynomials meet is called the via-point. The purpose of this point will be discussed later. The cubic polynomial is as follows,

$$\alpha_1(t) = a_{13}t^3 + a_{12}t^2 + a_{11}t + a_{10} \quad \alpha_2(t) = a_{23}t^3 + a_{22}t^2 + a_{21}t + a_{20}$$

In order to generate an arbitrary motion, specific constraints need to be put upon the polynomials. Since there are two cubic polynomials (i.e. 8 coefficients / degrees of freedom), there are 8 constraints. The first constraint is the point of the first polynomial at $t = 0$. At $t = 0$, the kick motion will swing the leg back. This is called the retraction point. This is the point farthest from the ball. The second constraint is that the velocity of the first point at $t = 0$ which is zero. The third constraint is the the position of the via-point where both cubic polynomials meet. The via-point is used to determine the height of the kick trajectory. It is also a very important point because it is where both polynomials meet. Hence, both polynomials need to have the same position at this point and their velocities need to match. Moreover, the acceleration at the via point for both polynomials need to also match. This guarantees a smooth trajectory with C^2 continuity. The last two constraints define the position and velocity of the second cubic spline at $t = t_f$. The constraints are summarized as follows,

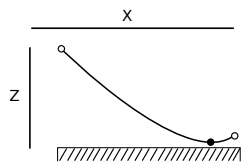

Fig. 5: These two polynomials are generated for a front kick. Since the front kick does not vary on the y-plane, it is two dimensional in the x and z plane. The white circles are the beginning and end points of the kick respectively. The black circle is the via-point that constraints the height of the polynomial. The first polynomial is from the first white circle to the black circle, and the second polynomial is from the black circle to the last white circle.

$$\begin{aligned}
\alpha_1(0) &= [x_0, y_0, z_0] && \text{(retraction point)} \\
\dot{\alpha}_1(0) &= [0,0,0] \\
\alpha_1(t_{via}) &= [x_{t_{via}}, y_{t_{via}}, h_{floor}] && \text{(constraint on foot from floor)} \\
\alpha_2(0) &= [x_{t_{via}}, y_{t_{via}}, h_{floor}] && \text{(constraint on foot from floor)} \\
\dot{\alpha}_1(t_{via}) &= \dot{\alpha}_2(0) \\
\ddot{\alpha}_1(t_{via}) &= \ddot{\alpha}_2(0) \\
\alpha_2(t_f) &= [x_f, y_f, z_f] && \text{(contact point)} \\
\dot{\alpha}_2(t_f) &= [0,0,0]
\end{aligned}$$

Moreover, as shown above, the constraints are defined as vectors because there needs to be polynomials for the x, y, and z plane. The six cubic polynomials (two for each plane) will form a parametric curve in $\mathbb{R}^3$. To solve the coefficients of the polynomials we solve the following system which was created by inputting the constraint in the polynomial and rearranging terms,

$$\begin{bmatrix} 1 & t_{via} & 0 & 0 & 0 \\ 0 & 0 & 1 & t_f & t_f^2 \\ 0 & 0 & 1 & 2t_f & 3t_f^2 \\ 2t_{via} & 3t_{via}^2 & -1 & 0 & 0 \\ 2 & 6t_{via} & 0 & -2 & 0 \end{bmatrix} \begin{bmatrix} a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} \frac{\alpha_1(t_{via}) - \alpha_1(0)}{t_{via}^2} \\ \frac{\alpha_2(t_f) - \alpha_2(t_{via})}{t_{via}^2} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{9}$$

As seen in (**??**), a_{10}, a_{11}, and a_{20} do not need to be a part of the system of equations because their answer is trivial:

$$\alpha_1(0) = a_{10} \quad \dot{\alpha}_1(0) = a_{11} \quad \alpha_2(0) = a_{20}$$

Solving (**??**), gives us the coefficients for our polynomials, but we still are missing one step. Before we begin solving (**??**), we need to determine the optimal via point position. This is discussed in section **??**.

6 Model Optimization

Generating a good parameter set is important to attain a good kick. Although these values can be found empirically, it is a tedious task. We therefore used

the Covariance Matrix Adaptation Evolution Strategy (*CMA-ES*) for model optimization. The values were optimized and visualized in Webots, which can be seen as the standard simulation software for NAOqi. The first parameter set optimized was the time length of the kick and the speed towards the via point. The second run on the model optimization was done on the time length of the kick, the speed of the via point, and the via point location in the x, y, z plane. Although this gave good results, the via point was moved equally proportionally in 3D space, which resulted in a polynomial where the cubic and quadratic components were negligible in the specified interval of t. Therefore for the last optimization run, the via point was moved disproportional in the x, y, and z plane to generate curves in 3D space.

Run	Dimensions	Distance (Meters)	Num. Eval.	Time(ms)	Percent	X Via	Y Via	Z Via
seed	0	2.008	0	1180	0.5	0.5	0.5	0.5
1	2	2.124	6516	853	0.46	0.5	0.5	0.5
2	3	2.230	10633	1316	0.53	0.49	0.49	0.49
3	5	2.696	320	2273	0.77	0.1	0.42	0.44

Table 1: Optimization results. The Time column is the duration of the kick. The Percent column shows the percent of the time spent on the first polynomial. X, Y, and Z Via is the Via point in 3D space.

Although we got promising results from the optimization, we believe we can get better results for the optimization by increasing the dimension of the optimization. As you can see in table **??**, as we include more parameters, there is an increase in the distance of the optimal kick as well as a decrease in the number of generations until convergence. The additional parameters that can be added are: kick limit configuration (exploring a set of limits on the polynomials that can generate a good kick), end points of the polynomial, and the time it takes to get to the retraction point.

7 Experiments

For the validation of our approach, various kicks were generated and data for the balancing and spline trajectory generation was recorded. Figure **??** (a) shows the ZMP reference point for a right front kick was 0.03 for x and 0.052 for y for all preview reference points in the preview controller. Although the x and y ZMP values varies around the reference points, the error is less than 1 centimeter which is less than the radius of the foot or support polygon. Furthermore, the small error accumulated is due to the simplification of the inverse kinematic model, but it still gives a good approximation to the CoM position. The CoM position in **??** (b) is very close to the ZMP value because when the robot is balancing on one foot, it does not generate a force in the x or y axis. This can be verified in figure **??** (c), which is very close to zero. The graph in **??** (c) is bounded between -1.5×10^{-4} and 2×10^{-4}. Hence, the difference between the estimated ZMP and the estimated CoM is negligible within working levels of precision. Although we might be tempted to control the CoM alone and disregard

ZMP because they coincide, we need to notice that if a force is applied to the robot by another robot, it will generate a force in the x or y axis, which in turn might make the CoM leave the support polygon, but we can still use the ZMP criterion to stabilize the robot. In figure **??** (d), and (e), the polynomials in the x, y, and z plane have been generated and projected in 3D space. The blue polynomial is the first polynomial and the orange polynomial is the second polynomial generated, and the union is the via point. As can be observed, the polynomials are very close to a straight line. This is due to the configuration limit of the robot. Since the robot operates in a very small space, the coefficients of the polynomials are very small (less than 1). Therefore, the quadratic and cubic terms are negligible and the linear term is dominant. In order to get better shapes for the kick trajectories, we used CMA-ES optimization to generate shapes by moving the via point from being halfway. The configuration limit of the kick trajectory can be visualized in **??** (f). A video of the kick engine can be found at: https://www.youtube.com/watch?v=4fmuqI_CpQw.

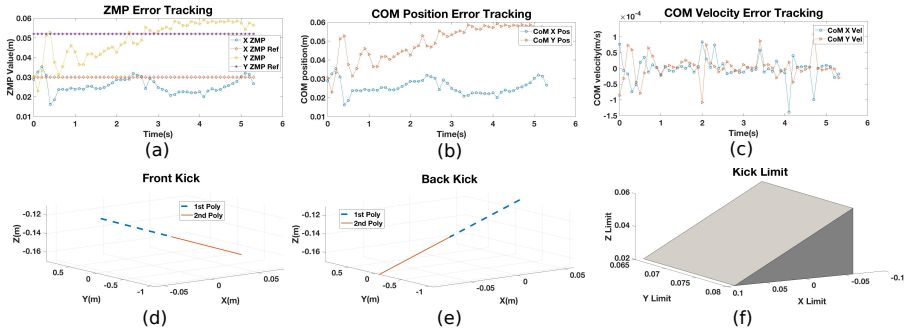

Fig. 6: (a) The ZMP error between the reference point and the actual ZMP. (b) The position of the center of mass in x and y. (c) The center of mass velocity. (d)-(e) The dashed polynomial is generated first. (d) The front kick trajectory in 3D space. (e) The back kick trajectory in 3D space. (f) The kick limit of the NAO in 3D space.

8 Conclusion

We have developed and implemented a dynamic kick engine that dynamically balances using a ZMP based preview controller. The preview controller is given the center of mass position based of the error in the ZMP. In order to move the center of mass to the correct position, we use a simplified model that consists of a two link arm. The end effector of the arm is denoted as the center of mass. The kick trajectory is generated using cubic splines. The knot in the cubic splines is called the via point, and it is used to control the shape of the polynomial. The kick was optimized using a simulation software called Webots. On Webots, we optimized the via point position, the speed to the via point, and the time duration of the kick. The optimization attained optimal set of

parameters without having to experiment on the robot to find them. These values were used on the physical robot to verify results.