Five Link Walker [Incomplete Draft]

Oluwami Dosunmu-Ogunbi

January 31, 2022

Acknowledgements

This technical document is a written adaptation follow-up of the "biped bootcamp" offered by Professor Jessy Grizzle. I worked closely with fellow PhD Yukai Gong to learn the material presented in this document. I now share that information with you, my esteemed reader.

Contents

1 Introduction

RABBIT (shown in Figure 1) is a five-link bipedal robot, specifically designed to advance the fundamental understanding of controlled legged locomotion [1]. In this document, you will learn how to model and control RABBIT in Matlab/Simulink, using FROST to generate its kinematics and dynamics.

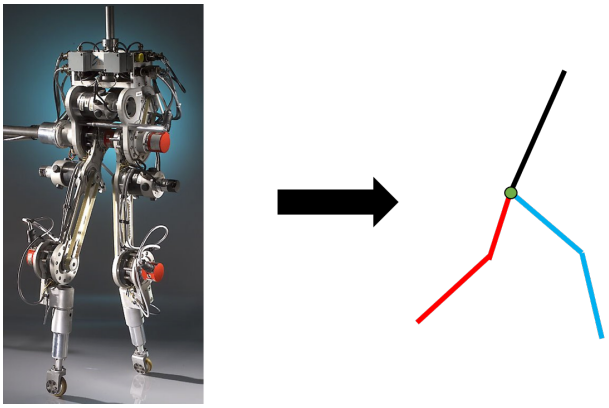

Figure 1: Actual RABBIT Bipedal Robot (left) and its diagram representation (right)

2 Meet RABBIT, the Five Link Walker

Before we can do any controlling, dynamics, or even kinematics, we first need to learn a little bit more about our system. RABBIT consists of seven generalized coordinates:

$$q = \begin{bmatrix} x \\ z \\ \theta_{torso} \\ \theta_{left\ hip} \\ \theta_{left\ knee} \\ \theta_{right\ hip} \\ \theta_{right\ knee} \end{bmatrix} \tag{2.1}$$

where,

- x – refers to the x-position of the hip of RABBIT in the world frame
- z – refers to the z-position of the hip of RABBIT in the world frame
- θ_{torso} – refers to the torso angle of RABBIT with respect to the world reference frame
- $\theta_{left\ hip}$ – refers to the left hip angle of RABBIT with respect to the torso angle reference frame

- $\theta_{left\ \ knee}$ – refers to the left knee angle of RABBIT with respect to the left hip frame
- $\theta_{right\ \ hip}$ – refers to the right hip angle of RABBIT with respect to the torso angle reference frame
- $\theta_{right\ \ knee}$ – refers to the right knee angle of RABBIT with respect to the right hip frame

The order of these coordinates matters, especially when we start using this vector to do things like generate the equations of motion or develop control strategies. One must be careful to be consistent with how they order these coordinates for all calculations. In this document, I will be assuming the order shown in Eq. 2.1.

Figure 2 depicts how these coordinates are defined on the diagram model of RABBIT.

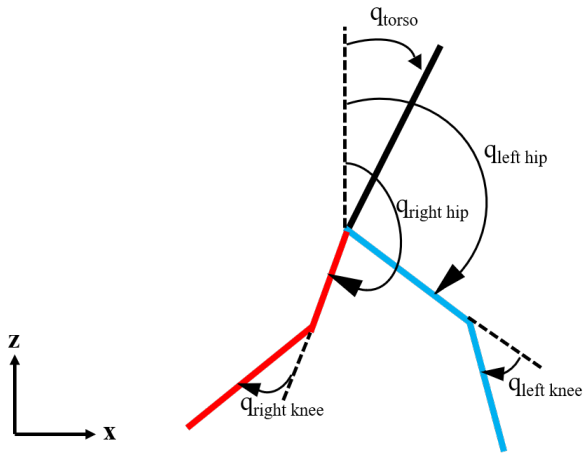

Figure 2: Generalized Coordinates Definitions for RABBIT

Figure **??** shows the names of the links and joints of RABBIT. There is a single torso link that is attached at the hip joints to the left and right thigh links. At the end of each thigh, the knee joint connects the thigh link to the shin link. At the end of each shin link is the foot, defined as a point.

- the **torso link** has a mass of 12kg and is 0.48m long
- each of the **thigh links** weigh 6.8kg and are 0.4m long
- each of the **shin links** weigh 3.2kg and are 0.4m long

It is certainly possible to generate the kinematics and dynamics of RABBIT by hand, but this would be a wholly unnecessary and tedious task. If you would like to strengthen your kinematics and dynamics knowledge by practicing generating these equations by hand for a robot, I would recommend that you do it for the three-link walker instead of RABBIT because it is a simpler model. I walk through how to generate the kinematics

of the three-link walker in the Biped Bootcamp technical document [2]. For those of you who have had your fair share of generating math equations by hand, you can join me in this document and use FROST.

3 A Brief Tutorial of FROST

The **F**ast **R**obot **O**ptimization and **S**imulation **T**oolkit (a.k.a. **FROST**) is an open-source MATLAB toolkit created by Dr. Ayonga Hereid for dynamical system modeling, trajectory optimization, and model-based design of robotic systems, with a special focus in hybrid dynamic locomotion [3]. You will need to download the FROST repository from [4] for our implementations.

4 Let's Draw RABBIT Using Matlab

Now that we have generated our kinematics FROST, we can use these equations to animate RABBIT. In this section, I will show you how to create a function in Matlab that will animate RABBIT. Later we will be using Simulink to animate and apply a controller to RABBIT, but for now we will use Matlab to better understand the basics.

In this section, I will walk through how to create a Matlab function called `RABBIT_anim.m` that will produce a plot that draws RABBIT given the coordinate values of an input vector. This function can also take a matrix of generalized coordinates that represent RABBIT's generalized coordinates at a series of time steps and use this matrix to animate RABBIT as it moves through those time steps.

Start by creating a Matlab function file called `RABBIT_anim.m`. Let this function take a single input argument, `q` that represents the generalized coordinate vector. The first line of code in this function should look like this:

```
function RABBIT_anim(q)
```

Now we need to create a figure in Matlab that will be used to plot RABBIT. Use the `figure` function in Matlab to do this. We can define the window size using the `axis` function. Since the input argument `q` can also be a matrix representing the generalized coordinates of RABBIT at a series of time steps, you may find it helpful to define the maximum and minimum x-coordinate axis to be with respect to the x-coordinate hip position of RABBIT (which is defined by the first element in the generalized coordinate vector). The code will look something like this:

```
%% set up plot
figure();
```

```
% define plot window boundaries
xmax = q(1,1) + 3;
xmin = q(1,1) - 3;
ymax = 1;
ymin = -1;

axis([xmin xmax ymin ymax]);
axis equal
axis off
```

The ground is a useful element to also include in our plot. For our purposes, we will keep the ground at a constant value of 0. You can use the `line` and `set` in Matlab to draw the ground as a line. I also color the ground black, though you can use any color here. That code looks like this:

```
% plot ground
ground = line([xmin xmax],[0 0]);
set(ground,'LineWidth',3,'Color','k');
```

To draw RABBIT as the five links (or lines) that it is made up of, you need to know the positions of the points of interest. Specifically, you need to know the positions of the head, the hip, the left and right knee, and the left and right feet. This is where you can use the position functions that you created in FROST. Call those functions using the first element of the `q` input argument to get the positions of the points of interest. The code will look something like this:

```
%% Find and plot initial positions of links
% initial joint positions
p_left_foot = p_left_foot_func(q(1,:));
p_right_foot = p_right_foot_func(q(1,:));
p_head = p_head_func(q(1,:));
p_hip = p_left_hip_func(q(1,:));
p_left_knee = p_left_knee_func(q(1,:));
p_right_knee = p_right_knee_func(q(1,:));
```

Given these positions, you can use the `line` and `set` functions to draw each of the five links. For clarity, I color the left thigh blue, the left shin cyan, the right thigh red, the right shin magenta, and the torso green. That code looks like this:

```
% define links
% left thigh
```

```
link_left_thigh = line([p_left_knee(1) p_hip(1)],...
    [p_left_knee(2) p_hip(2)]);
set(link_left_thigh,'LineWidth',2,'Color','b');

% left shin
link_st_shin = line([p_left_foot(1) p_left_knee(1)],...
    [p_left_foot(2) p_left_knee(2)]);
set(link_st_shin,'LineWidth',2,'Color','c');

% right thigh
link_right_thigh = line([p_hip(1) p_right_knee(1)],...
    [p_hip(2) p_right_knee(2)]);
set(link_right_thigh,'LineWidth',2,'Color','r');

% right shin
link_right_shin = line([p_right_knee(1) p_right_foot(1)],...
    [p_right_knee(2) p_right_foot(2)]);
set(link_right_shin,'LineWidth',2,'Color','m');

% torso
link_torso = line([p_hip(1) p_head(1)], [p_hip(2) p_head(2)]);
set(link_torso,'LineWidth',2,'Color','g');
```

If you wish, you may use the function that you created in FROST to calculate the position of the center of mass to draw the center of mass as a circle. That code looks like this:

```
% CoM
pCOM = p_COM_func(q(1,:));
r = .03; % radius
th = 0:pi/50:2*pi;
xunit = r * cos(th) + pCOM(1);
yunit = r * sin(th) + pCOM(2);
CoM = line(xunit, yunit);
set(CoM,'LineWidth',3,'Color','k');
```

Great! Your function should be ready to create at least a still image. Try calling `RABBIT_anim.m` with a set of generalized coordinates that makes sense. I used:

```
RABBIT\_anim([0, 0.7, 0, -pi/6, pi/3, -pi/6, pi/4])
```

**NOTE: Make sure that you have added the folders where you have stored your FROST functions to your path or Matlab will throw an error saying that that those functions are not defined! You can do this by right clicking on the folder in your Matlab directory, hover over "Add To Path", and choosing "Selected Folders and SubFolders".

**NOTE: The last line in your function should be the `return` statement.

Now that you are able to generate a static image, let's add code to the `RABBIT_anim.m` function to give it the ability to animate.

You need to first know how many images that you will be producing to create the animation. You can do this by looking at the length of the input vector (which will actually be a matrix in this case). Using this number, you will create a `for loop` that will draw the five lines that make up RABBIT for every set of generalized coordinate in the input matrix `q`. The method is the same as what you did to produce the first image before. Make sure to also update the position of the center of mass if you are also drawing that. You will also need to update the plot window axes as well as the ground to make sure that it stays in frame. Finally, use the `drawnow` and `pause` functions to have draw re-draw the lines at each iteration and pause at each image to ensure an appropriate frame rate. The code looks like this:

```
%% Animate
% Define inputs
[length, ~] = size(q); % number of data points we are dealing with

for j = 1:length % iterate through each timestep
    % Calculate points for each iteration
    p_left_foot = p_left_foot_func(q(j,:));
    p_right_foot = p_right_foot_func(q(j,:));
    p_head = p_head_func(q(j,:));
    p_hip = p_left_hip_func(q(j,:));
    p_left_knee = p_left_knee_func(q(j,:));
    p_right_knee = p_right_knee_func(q(j,:));

    % left leg
    set(link_left_thigh,'XData',[p_hip(1) p_left_knee(1)], ...
        'YData',[p_hip(2) p_left_knee(2)]);
    set(link_st_shin,'XData',[p_left_knee(1) p_left_foot(1)], ...
        'YData',[p_left_knee(2) p_left_foot(2)]);
```

```
        % right leg
        set(link_right_thigh,'XData',[p_hip(1) p_right_knee(1)], ...
            'YData',[p_hip(2) p_right_knee(2)]);
        set(link_right_shin,'XData',[p_right_knee(1) p_right_foot(1)], ...
            'YData',[p_right_knee(2) p_right_foot(2)]);
        % torso
        set(link_torso,'XData',[p_hip(1) p_head(1)], ...
            'YData',[p_hip(2) p_head(2)]);
        % COM
        pCOM = p_COM_func(q(j,:));
        xunit = r * cos(th) + pCOM(1);
        yunit = r * sin(th) + pCOM(2);
        set(CoM,'XData',xunit,'YData',yunit);

        % update plot window
        xmax = q(j,1) + 3;
        xmin = q(j,1) - 3;
        axis([xmin xmax ymin ymax]);
        axis equal
        axis off

        % ground
        set(ground,'XData',[xmin xmax],'YData',[0 0]);

        drawnow;
        pause(0.0001);
    end
```

**NOTE: the last line in your function code should be the `return` statement.

Now your function is ready to animate a time series of generalized coordinates for RABBIT. But to test to see if your code is actually working properly, we need to give it a matrix of values. Sure, you could define these positions by hand; however, it would be easier to generate some sort of reference trajectory and send those corresponding generalized coordinate values to your animation function instead. The next section will walk you through how to generate reference trajectories for RABBIT.

5 Generating Reference Trajectories

In this section, I will show you how to generate trajectories for your RABBIT model to follow. We will use a Matlab script to get the logic down in this section, but in the next section we will be using the logic implemented in this section for a function that will apply a controller to control RABBIT.

We will be creating the following three reference trajectories:

- Pinning RABBIT's hip two meters in the air, move RABBIT's left foot up and down
- Pinning RABBIT's hip two meters in the air, alternate moving RABBIT's left and right feet up and down
- Placing RABBIT on the ground, alternate moving RABBIT's left and right feet up and down

Start by creating a new Matlab script called `TrajectoryGeneration.m`. It should be noted that this script will only be used for learning purposes, and thus has no functional value for later implementations in this technical document. We will use this script to generate all three of our desired reference trajectories.

5.1 Trajectory 1: RABBIT Pinned, Left Foot Moving

For the first reference trajectory, we will be pinning RABBIT's hip two meters in the air and moving its left foot up and down. Incidentally, this desired trajectory already tells us that five of the seven values in the generalized coordinate vector will be constant values.

5.1.1 Defining the Constant Values

Recall the generalized coordinate vector:

$$q = \begin{bmatrix} x \\ z \\ \theta_{torso} \\ \theta_{left\ hip} \\ \theta_{left\ knee} \\ \theta_{right\ hip} \\ \theta_{right\ knee} \end{bmatrix}$$

If we only care about the left foot position, then only the $\theta_{left\ hip}$ and $\theta_{left\ knee}$ angles should change. The rest of the coordinates will remain constant.

The first two generalized coordinates x and z are RABBIT's x- and z- hip position respectively in the world frame. Since we desire to pin RABBIT two meters in the air, we want to hold both of these coordinates at the following constant values:

- $x = 0$
- $z = 2$

NOTE: The x position could be set to any real number value. The "pinned hip" constraint technically only applies to the z coordinate since we just want RABBIT to be two meters in the air. For simplicity however, I am setting x to 0. Remember, the purpose of this first trajectory is to just move the left foot up and down. The x-hip position does not need to move to accomplish this purpose.

The third generalized coordinate θ_{torso} is RABBIT's torso angle. This angle does not need to change to move the left foot up and down, and so we can also hold it constant. The angle value that you choose to use does not matter; however, for simplicity and clarity I will set this value to 0.

The sixth and seventh generalized coordinates $\theta_{right\ hip}$ and $\theta_{right\ knee}$ are the right hip and knee angle respectively. The left foot position is not affected by either of these two generalized coordinates, so we can also set these to constant values. You can choose any values for these coordinates that make sense (given how these angles are defined in Figure 2). I am choosing to use the following to values:

- $\theta_{right\ hip} = 0$
- $\theta_{right\ knee} = \pi/3$

5.1.2 Defining the Left Foot Position

This just leaves us with two generalized coordinates left to be defined, the fourth and fifth coordinates, which are $\theta_{left\ hip}$ and $\theta_{left\ knee}$ respectively.

To define these coordinates, I will first use a sine function to define the z-position of the left foot and hold the x-position of the left foot to be a constant value. This will generate the effect of the left foot moving up and down. Using inverse kinematics, I will determine the values of $\theta_{left\ hip}$ and $\theta_{left\ knee}$ at each time step, given the foot position.

I will define the foot position with respect to RABBIT's hip coordinates. Recall from Section 2 that the length of thigh and shin links of RABBIT are both 0.4m long. I also know that I am holding RABBIT two meters in the air at its hip. Given this information, I will choose to have RABBIT oscillate its left foot from 1.7m to 1.3m with respect to the hip in the z-direction. (NOTE: You can choose to use different values, as long as those

values are within the physical reach for RABBIT.) The sine function to accomplish this z-position displacement is:

$$P_z = 1.5 + 0.2 \sin(t) \tag{5.1}$$

where P_z is the left foot's z-position and t is the time variable.

Since I am only trying to move the left foot in the z-direction, I will hold the left foot's x-position at a constant 0.

Using the left foot position coordinates, I can implement inverse kinematics to find the corresponding $\theta_{left\ hip}$ and $\theta_{left\ knee}$ values.

5.1.3 Inverse Kinematics

One can calculate inverse kinematics using either a numerical or an analytical approach. Both approaches have their own pros and cons. For example, a numerical approach tends to be more easily applicable to a wider range of systems with minimal change in the mathematical derivation, however can be computationally expensive and run the risk of falling to a local minima. An analytical approach allows for quick calculations of solutions, however, generating the equations necessary for this approach can be difficult or tedious to solve. See [5] for more information on inverse kinematics.

In this document, we use an analytical approach to solve the inverse kinematics problem. I will skip the derivation in this section and write the final equations for $\theta_{left\ hip}$ and $\theta_{left\ knee}$. To see the full derivation, see Appendix A. The equations for $\theta_{left\ hip}$ and $\theta_{left\ knee}$ are defined as follows:

$$\theta_{left\ hip} = \tag{5.2}$$

and

$$\theta_{left\ knee} = \tag{5.3}$$

where x is the position of the hip in the x-direction, z is the position of the hip in the z-direction, L_1 is the length of the thigh link, L_2 is the length of the shin link, $\theta_{left\ hip}$ is the left hip angle, θ_{torso} is the torso angle, and $\theta_{left\ knee}$ is the left knee angle.

5.1.4 Writing the Code

We now have all the information that we need to write the code to create Trajectory 1.

We will start the `TrajectoryGeneration.m` file by clearing our workspace and adding the paths to the kinematics generated from FROST. That code looks like this for me:

```
clear all
close all
```

```
clc

addpath('kin/m','kin/mex','kin/src')
```

Next, we want to define a time vector and set some constraints.

```
%%  Define time vector and set constraints
% time vector
time = 1:0.01:20;

% set constraints
L1 = 0.4; % thigh length
L2 = 0.4; % shin length

xref = 0; % hip x-coord
zref = 2; % hip z-coord
thetaref = 0; % torso angle

% right leg constraints (to hold right leg stationary)
qRH = 0;
qRK = pi/3;
```

Next, we want to create some empty vectors to store trajectory of the generalized coordinates as well as the desired foot and calculated (from inverse kinematics) left foot positions.

```
%% Trajectory 1: Move left leg up and down
q1 = zeros(length(time),14); % vector to store trajectory
Pdes1 = zeros(length(time),2); % stores desired foot placement
Pref1 = zeros(length(time),2); % stores calculated ref foot placement
```

Now, we want to create a `for loop` that will iterate through the time vector, determine the left foot position based on the sinusoidal trajectory defined in Section 5.1.2, compute the corresponding inverse kinematics to determine $\theta_{left\ hip}$ and $\theta_{left\ knee}$, and populate the generalized coordinate vector as well as the desired and calculated foot positions. That code looks like this:

```
% generate trajectory
for i = 1:length(time)
    t = time(i); % time
```

```
    % left leg
    % foot position, based on desired trajectory
    Px = 0;
    Pz = 1.5 + 0.2*sin(t);

    Pdes1(i,1) = Px;
    Pdes1(i,2) = Pz;

    % Inverse kinematics to find hip and knee angle
    tempy = sqrt(1 - ((L1^2+L2^2-Px^2-(Pz-2)^2)/(2*L1*L2))^2);
    tempx =((L1^2+L2^2-Px^2-(Pz-2)^2)/(2*L1*L2));
    qLH = -(pi - atan2(tempy,tempx))/2;
    qLK = pi -  atan2(tempy,tempx);

    % populate trajectory vector
    q1(i,1) = xref;
    q1(i,2) = zref;
    q1(i,3) = thetaref;
    q1(i,4) = qLH;
    q1(i,5) = qLK;
    q1(i,6) = qRH;
    q1(i,7) = qRK;

    % forward kinematics to verify that foot positions based on hip and
    % knee angles found with inverse kinematics is correct
    Phip = p_left_hip_func(q1(i,1:7));
    Pknee =  p_left_knee_func(q1(i,1:7));
    Pfoot = p_left_foot_func(q1(i,1:7));
    Pref1(i,1) = Pfoot(1);
    Pref1(i,2) = Pfoot(2);
end
```

Finally, we can generate some useful plots as well as call our animation function so we can see our trajectory being applied to a simulated model of RABBIT.

```
% Plot
figure
plot(time,q1(:,[1 2 3 4 5 6 7]),'LineWidth',2)
legend('x','z','theta','qLh','qLK','qRH','qRK')
title('Time versus Generalized Positions')
```

```
xlabel('time (s)')
ylabel('Angle (rad)')

figure
plot(time,Pdes1,'LineWidth',2)
hold on
plot(time,Pref1,'LineWidth',2)
legend('PxDes','PzDes','PxRef','PzRef')
title('Px and Pz vs Time')
xlabel('time (s)')
ylabel('position (m)')

% Animate trajectory
RABBIT_anim(q1(:,1:7));
```

Congratulations! You have generated your first trajectory for RABBIT. Run your `TrajectoryGeneration.m` script to see your hard work in action!

5.2 Trajectory 2: RABBIT Pinned, Both Feet Moving

For the second reference trajectory, we will still be pinning RABBIT's hip two meters in the air; however, instead of moving just the left foot up and down, we will alternate moving RABBIT's left and right feet up and down one at a time. You will be happy to know that much of the logic used here is the exact same as what we used for Trajectory 1. To move the right foot up and down, we need to define the inverse kinematics that translates the right foot position to the joint angles for $\theta_{right\ hip}$ and $\theta_{right\ knee}$. Since we defined the inverse kinematics of the left foot with respect to the hip, the equations translate well to use for the right foot, as is shown in the following subsection. The right foot positions will also be defined using the same logic as the left foot discussed in Section 5.1.2.

The only significant difference between Trajectory 1 and Trajectory 2 is the switch logic that tells the code when to switch from moving the left foot to moving the right foot and vice versa. To accomplish this switch, we will need to define a step period–that is, the time it takes for a single foot to move up and down. From there, the sinusoidal functions that we use to define the left and right feet position will move the appropriate foot at the correct time. The methodology to do so is shown in the following subsection.

5.2.1 Writing the Code

You can append the code that you will use to generate Trajectory 2 to the end of the same Matlab file that you used to generate Trajectory 1.

Start by creating empty vectors to store the trajectory as well as the desired and calculated feet placement. You will also need to create a variable to store the desired period length.

```
%% Trajectory 2: Alternate moving left/right foot up and down in the air

q2 = zeros(length(time),14); % vector to store trajectory
Pdes2_left = zeros(length(time),2); % stores desired foot placement
Pref2_left = zeros(length(time),2); % stores calculated ref foot placement
Pdes2_right = zeros(length(time),2); % stores desired foot placement
Pref2_right = zeros(length(time),2); % stores calculated ref foot placement

period = 0.4;
```

As in the previous trajectory, you will want to create a `for loop` that will iterate through the time vector (which should still be defined at the top of the code) to (i) determine the left and right feet positions based on the sinusoidal trajectory defined in Section 5.1.2 while implementing logic to switch the feet based on period, (ii) compute the corresponding inverse kinematics to determine $\theta_{left\ hip}$, (iii) $\theta_{left\ knee}$, $\theta_{right\ hip}$, and $\theta_{right\ knee}$, (iv) populate the generalized coordinate vector as well as the desired and calculated feet positions.

```
for i = 1:length(time)
    t = time(i); % time

    stance = floor(t/period); %  right stance if even, left stance if odd

    % if left foot moving
    if (mod(stance, 2) == 0)
        Px_left = 0;
        Pz_left = -0.4 + 0.2*sin(2*pi/period * t - pi/2);

        Px_right = 0;
        Pz_right = -0.6;
    end

    % if right foot moving
    if (mod(stance, 2) ~= 0)
        Px_left = 0;
        Pz_left = -0.6;
```

```
    Px_right = 0;
    Pz_right = -0.4 + 0.2*sin(2*pi/period * t - pi/2);
end

Pdes2_left(i,1) = Px_left;
Pdes2_left(i,2) = Pz_left;
Pdes2_right(i,1) = Px_right;
Pdes2_right(i,2) = Pz_right;

% left leg
% Inverse kinematics to find hip and knee angle
% w.r.t. hip coords
phi = acos((L1^2+L2^2-Px_left^2-Pz_left^2)/(2*L1*L2));
qLH = -(pi - phi)/2;
qLK = pi -  phi;

% right leg
% Inverse kinematics to find hip and knee angles
 % w.r.t. hip coords
phi = acos((L1^2+L2^2-Px_right^2-Pz_right^2)/(2*L1*L2));
qRH = -(pi - phi)/2;
qRK = pi -  phi;

% forward kinematics to varify that foot positions based on hip and
% knee angles found with inverse kinematics is correct
% left leg
Phip = [xref; zref];
Pknee = -[L1*sin(thetaref+qLH); L1*cos(thetaref+qLH)]; % w.r.t. hip coords
Pfoot = Pknee - [L2*sin(thetaref+qLH+qLK); L2*cos(thetaref+qLH+qLK)];
Pref2_left(i,1) = Pfoot(1);
Pref2_left(i,2) = Pfoot(2);

% right leg
Phip = [xref; zref];
Pknee = -[L1*sin(thetaref+qRH); L1*cos(thetaref+qRH)]; % w.r.t. hip coords
Pfoot = Pknee - [L2*sin(thetaref+qRH+qRK); L2*cos(thetaref+qRH+qRK)];
Pref2_right(i,1) = Pfoot(1);
Pref2_right(i,2) = Pfoot(2);
```

```
    % populate trajectory vector
    q2(i,1) = xref;
    q2(i,2) = zref;
    q2(i,3) = thetaref;
    q2(i,4) = qLH;
    q2(i,5) = qLK;
    q2(i,6) = qRH;
    q2(i,7) = qRK;
end
```

Finally, we can generate some useful plots as well as call our animation function.

```
% Plot
figure
plot(time,q2(:,[1 2 3 4 5 6 7]),'LineWidth',2)
legend('x','z','theta','qLh','qLK','qRH','qRK')
title('Time versus Generalized Positions (My Approach)')
xlabel('time (s)')
ylabel('Angle (rad)')

figure
plot(time,Pdes2_left,'LineWidth',2)
hold on
plot(time,Pref2_left,'LineWidth',2)
legend('PxDes','PzDes','PxRef','PzRef')
title('Px and Pz vs Time Left Leg')
xlabel('time (s)')
ylabel('position (m)')

figure
plot(time,Pdes2_right,'LineWidth',2)
hold on
plot(time,Pref2_right,'LineWidth',2)
legend('PxDes','PzDes','PxRef','PzRef')
title('Px and Pz vs Time Right Leg')
xlabel('time (s)')
ylabel('position (m)')

% animate trajectory
```

```
RABBIT_anim(q2(:,1:7));
```

Congratulations! You have generated your second trajectory for RABBIT. Run your `TrajectoryGeneration.m` script to see your hard work in action!

5.3 Trajectory 3: RABBIT Grounded, Left and Right Feet Moving

In the final reference trajectory, you will be placing RABBIT on the ground and alternate moving the left and right feet up and down. The code for this trajectory is exactly identical to Trajectory 2. You only need to change the `zref` variable–which was defined at the top of the code, before generating Trajectory 1. Instead of setting `zref` to 2, use 0.6 instead.

6 Controlling RABBIT

You have now generated the kinematics and dynamics of RABBIT, animated RABBIT in Matlab, and generated three reference trajectories for RABBIT. Now it is time to actually control RABBIT. In this section, I will walk you through how to control RABBIT using an input/output feedback linearization controller. We will be creating two Matlab files to implement and call the controller. We will use a file called `dynamics.m` to apply the controller to the dynamics of the system, and another file called `RABBIT_sim.m` to be the main script which you will run directly as the top-level to call all of your functions.

7 Foot Placement

8 Moving to Simulink

Appendices

A Inverse Kinematics Derivation for Trajectory 1

About the Author

Oluwami Dosunmu-Ogunbi, also known as Wami Ogunbi to her peers, is a second year Robotics PhD student at the University of Michigan. She is advised by Professor Jessy

Grizzle in the Biped Robotics Lab. Her current research focus is in controls with applications in bipedal locomotion.

She has an interest in effectively disseminating complex engineering and robotics concepts to a wide audience, and so one of her long-term goals is to become a professor.

Learn more about Wami by visiting her website at wamiogunbi.com.

Figure 3: Wami standing next to Agility Robotics' Digit

References

[1] Chevallereau, Christine, et. al, "RABBIT: A Testbed for Advanced Control Theory." 2003.

[2] Dosunmu-Ogunbi, Oluwami. "Biped Bootcamp." 2021.

[3] https://ayonga.github.io/frost-dev/pages/introduction.html

[4] https://ayonga.github.io/frost-dev/

[5] http://motion.pratt.duke.edu/RoboticSystems/InverseKinematics.html