Efficient stepping strategy for standing bipedal robots under external perturbations

Pages 1–23
2021

Nayan Rawat , Saharsh , Sreejeet Maity , Supriya Mandal , Vidhant Sharma
CP 214 : Foundation of Robotics
Indian Institute of Science, Bangalore
12.12.2021

Keywords: biped, push recovery, trajectory optimization, Euler-Langrangian, orbital energy, Neuro-Fuzzy, Neural Networks, NLP, SQP, Webots

ABSTRACT

The design and control of biped robots are one of the climacteric pillars of modern robotics which need some unified attention. In order to minimize human intervention, biped robots are designed and tested such that they can be deployed in hazardous environments. Even though the robots are generally environment-specific, the problem of balancing under some sudden impulsive perturbation seems to be a genuine problem that needs to be addressed. Inspired by our day-to-day activities, it is expected that in most cases the robot must learn to balance and stop itself from falling by taking a step to counteract the instability. In this project, we estimated the capture point by using forward kinematics and mapped it inversely to evaluate the joint angles needed to transfer the focal foot of the biped robot at the desired capture point to stop the robot from falling. In this project, we have successfully simulated the working model of our biped robot and tested the same over a wide range of forces and the robot showed significantly positive results.Some experimental results and simulations validating our work has been presented in the later section of this project.

1 INTRODUCTION

Biped robots are meant to outperform humans in a wide range of activities, especially in perilous environments. However, the stabilization and control of biped robots are still enigmatic and lack a unified approach. In this project, we tried to address and alleviate one of the major problems that bipeds face while working in real environments. The two-legged robots are often subjected to impulsive forces from random directions causing sudden instabilities that can topple the robot. Once the robot is toppled it is extremely hard to get it back to its original position because of the structural integrity and bulky nature of biped robots. So, it is essential to think of some methods to stop the robot from falling. Based on our day-to-day experiences, we know that under the application of a impulsive force, the body moves in the force direction and tends to topple. This can be counteracted by moving our foot in a specific location such that the balance is restored. The dynamical equations that capture the subtle fundamental picture involved in the above scenario demands copious calculations and thus making the unfiltered model computationally inefficient. Thereby, we need to adopt a simple model that can efficiently mirror the actual dynamics up to a permissible extent. In this project, the model that we will use as a substitute is the linear inverse pendulum model. Since we are concerned about the stepping strategy of a standing robot, this model serves as an excellent replacement and is proved to be fruitful for a wide range of input forces. The first part of this project deals with the analysis of the inverted linear pendulum model and the possible amendments needed to incorporate that in our project. In the second part, when an external force is applied, in order to balance its structure the robot must move its foot to some desired location to counteract the same. This desired location, also called the 'capture point' is the focal region where the foot should be placed and as a result, that balance can be achieved. In order to know the exact location of the capture point, we have applied forward kinematics, and sequentially based on that input, reverse kinematics is applied to retrieve information about the desired joint angles of the biped robot to achieve the same. Apart from the traditional approaches, we have incorporated neuro-Fuzzy model to cross validate our results. The reason for using this model is because the increasing complexity of the dynamical equations can be alleviated by it and it will also be handy in implementing the same in different robots.

2 APPROACH 1: TRAJECTORY PLANNING USING TIME DISCRETE OPTIMIZATION

2.1 Problem statement

We consider a 7-link biped restricted to move in sagittal plane as shown in figure 1. The biped is initially standing still, a push is given and it needs to balance itself. Various techniques such as balance via sticks [2] or by taking a step [3] have been used. We too wish to balance the biped by stepping forward after a push force is applied.

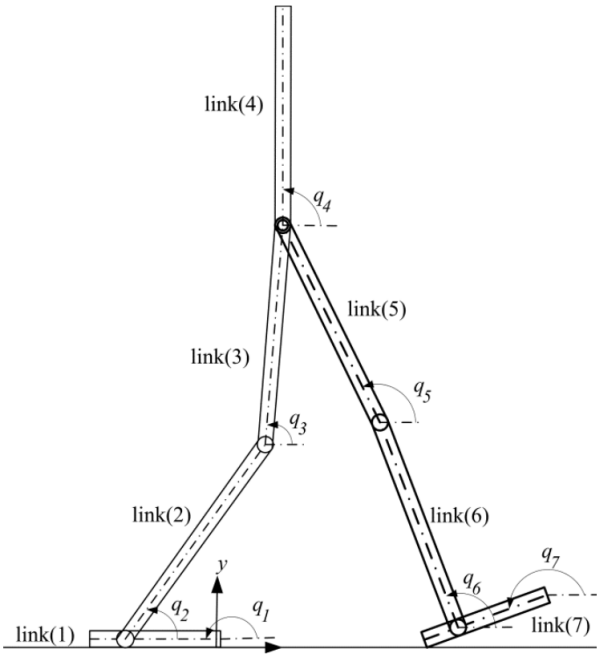

Figure 1: 7-link biped [5]

2.2 Background

The configuration $q = (q_1, q_2, q_3, q_4, q_5, q_6, q_7) \in \mathbb{Q}$ consists of n (=7) joint angles. Therefore, we denote the state $x := (q, \dot{q})$. We will denote the torque input u, which is of dimension n (=7). Given the states, and the inputs, the Euler-Lagrangian dynamics are given by

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = B(u) \tag{1}$$

where $M(q) \in \mathbb{R}^{n\times n}$ is the positive definite inertia matrix, $C(q, \dot{q} \in \mathbb{R}^{n\times n})$ is the Coriolis-centrifugal matrix, and $G(q) \in \mathbb{R}^n$ is the gravity vector, $B \in \mathbb{R}^{n\times m}$ is the one-on-one mapping of the torques to the joints. Direct methods for trajectory optimization are widely used for planning locally optimal trajectories of robotic systems. Most state-of-the-art techniques treat the continuous dynamics that form a trajectory as discrete nodes and restrict the search for a complete path to a specified sequence through these nodes.

2.3 Trajectory Optimization

In this approach, we apply the trajectory optimization strategy developed by Posa, Cantu, and Tedrake [4] to the biped stepping problem. In this formulation, we assume that one leg is fixed and the biped has to move the other leg, so the model becomes fully actuated as opposed to under-actuated and hence the contact forces are not modelled in Lagrangian formulation:
Find

$$q, \dot{q}, \ddot{q}, u, h \tag{2}$$

s.t.

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = B(u) \tag{3}$$

Trajectory optimization involves designing an optimal trajectory via some measure of performance while satisfying a set of constraints. When a sequence can be specified ahead of time, a trajectory optimization problem can be formulated using direct collocation to explicitly constrain the trajectory to obey the dynamics of each mode in the specified sequence [1]. The complexity of the system posed additional problems for the optimization. In this case, additional linear constraints were useful in guiding the solver away from infeasible regions. For the robotic system considered here, the measure of performance can be either power consumption, torque inputs or even the time duration of the trajectory. The set of constraints includes a set of equalities and inequalities that includes limits and terminal conditions.
In this section, we formalize and explain the additional constraints added to the basic formulation in (3) to yield the full optimization problem (4) used to solve the stepping problem. First, we assume a variable-timestep where we discretize time into a fixed number (N = 30) of timesteps, then allow the duration of each timestep h to be an optimization variable.

We do not add a cost to our optimization formulation, as we are interested only in finding feasible trajectories, but additional costs (e.g. a penalty on norm of control effort) can be added easily. Finally, to avoid trajectories involving nonphysical control inputs, we add the bounding box constraints (11) to constrain the minimum and maximum allowable control effort, and to ensure that the trajectory ends at the desired location,we constrain the final configuration of the system (14) so that final foot position is at a desired location.

Find

$$q, \dot{q}, \ddot{q}, u, h \tag{4}$$

s.t.

$$M(q_i)\ddot{q_i} + C(q_i, \dot{q_i})\dot{q_i} + G(q_i) = B(u_i) \tag{5}$$

$$q_{i+1} = q_i + h\dot{q}_{i+1} \tag{6}$$

$$\dot{q}_{i+1} = \dot{q}_i + h\ddot{q}_{i+1} \tag{7}$$

Optimization variables' limits:

$$q_{min} \leq q_i \leq q_{max} \tag{8}$$

$$\dot{q}_{min} \leq \dot{q}_i \leq \dot{q}_{max} \tag{9}$$

$$\ddot{q}_{min} \leq \ddot{q}_i \leq \ddot{q}_{max} \tag{10}$$

$$u_{min} \leq u_i \leq u_{max} \tag{11}$$

$$0 \leq h \leq h_{max} \tag{12}$$

Initial position (angles):

$$q(0) = q_{start} \tag{13}$$

Final foot position:

$$(x7_{CM}(end), y7_{CM}(end)) = (0.2, 0) \tag{14}$$

Additional constraints:
First leg should remain fixed:

$$(x1_{CM_i}, y1_{CM_i}) = (0, 0) \tag{15}$$

Knees should not bend backwards:

$$q2_i \leq q3_i \tag{16}$$

$$q6_i \leq q5_i \tag{17}$$

Both feet should remain horizontal:

$$q1_i = \pi \tag{18}$$

$$q7_i = \pi \tag{19}$$

Torso should remain vertical:

$$q4_i = \frac{\pi}{2} \tag{20}$$

Lifted foot should always move forward:

$$x7_{CM_i} \leq x7_{CM_{i+1}} \tag{21}$$

2.4 Simulation and results

Optimization was done using SQP algorithm in MATLAB using a 2.4 GHz dual core laptop processor, which significantly increased the computational time due to large number of variables as the timesteps increased. Moreover, local optimization needs a good initial guess which was difficult to provide, hence the large converging time. Trajectory can be seen in figure 2, in which 4 timesteps are taken (which translates to 113 decision variables) and it took about 10 mins when using random initial guess. Refeeding the solution as initial guess reduced the computation time, as expected. More timesteps can be done in a HPC.

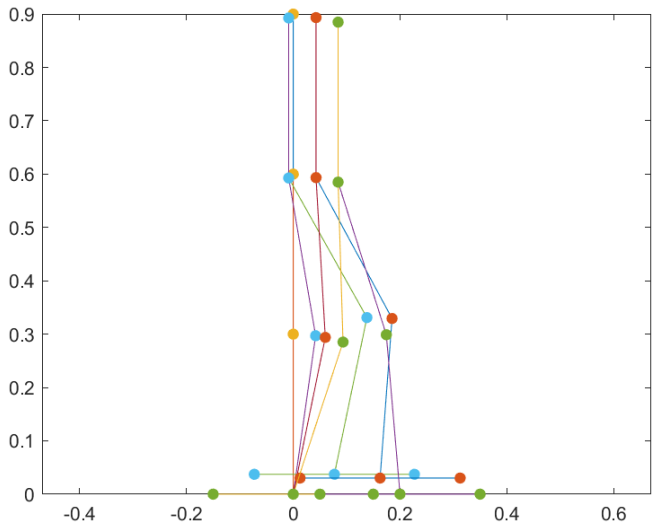

Figure 2: 7-link biped stepping trajectory with 4 timesteps

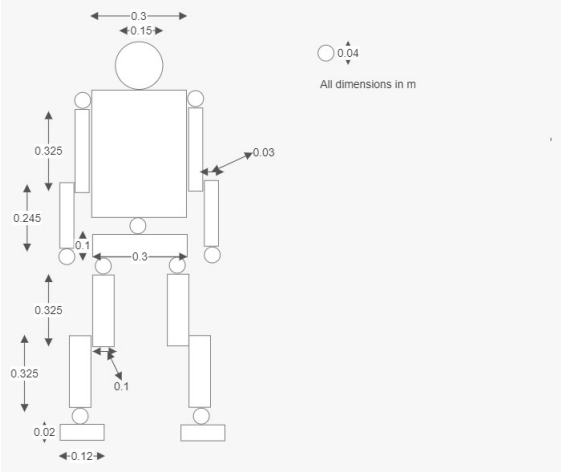

Figure 3: Barebones biped model. (figure not to scale)

3 APPROACH 2: STEPPING STRATEGY USING LINEAR INVERTED PENDULUM MODEL

The dynamical equations governing the exact model of a biped robot is tedious to work with and computationally hard to implement. Even when it is implemented, the motive to do so is significantly hampered because it becomes very hard to control the structure in a desired manner because of the run time complexities of the equations. As a replacement to those dynamical equations, we often take the help of simple models which mimic the behaviour of the robot under a certain class of limits. In our problem of interest we tried to deal with a simplified linear inverted pendulum model for biped robots. We are aware of the fact that, a human when suddenly struck with an external force, it is natural for him/her to maintain his/her balance by swinging the leg to maintain balance. Roughly speaking, a body can maintain its balance when the centre of mass is bounded by the legs . Inspired by this scenario, we considered a linear inverted pendulum with a flywheel model as a valid linear replacement of our biped robot for this specific scenario. Assuming the upper part of our robot does not rotate and the original pose of the body is maintained i.e τ on the flywheel is 0, an orbital energy is defined to take this into account. As a result of that, the orbital energy when taken as 0, gives the sum of kinetic energy and potential energy of the simplified model as 0. The subsequent calculations below shows how we calculated the point of our interest by putting the orbital energy to be 0, which turns out to be a valid assumption.

3.1 Model

Barebones model can be seen in figure 3.

3.2 Control Strategy

Barebones control strategy can be seen in figure 5. Control strategy can be seen in figure 4. Brief explanation:

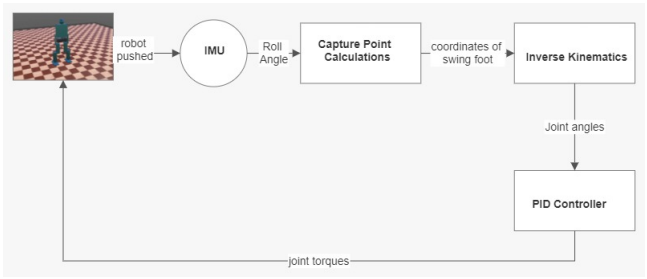

Figure 4: Orbital energy control strategy

- **Robot is pushed in sagittal plane:** An external force will be applied on robot torso. The robot will try to maintain a stable position till the point when it starts toppling about the edge of foot. In such case the stepping strategy starts executing.
- **Inertial moment unit for measures angle of tilt:** The angle measured by IMU is about the x-axis. The measured value is negative for backward fall and positive for forward fall. This measured angle value (in radians) is fed to controller for Capture point calculations.
- **Capture point [11]:** The capture point is calculated by assuming that orbital energy will remain constant, which is true as our angle of tilt is small. The equations for capture point calculation are given in Section 6.4.
- **Inverse Kinematics:** The calculated capture point is used to find the joint angles for swinging leg and non swinging leg using Inverse Kinematics. The equations of inverse kinematics or our robot is given in We have assumed motion of thigh and leg only. The angle of ankle is set so that it is always parallel to ground or swinging leg and perpendicular to leg for non swinging leg. This is done by taking the motion of human as reference which has same motion as above.
- **PID Control:** The calculated joint angles are fed to PID Controller which measures the required torque for achieving those angles. The plot of joint torques is also presented in

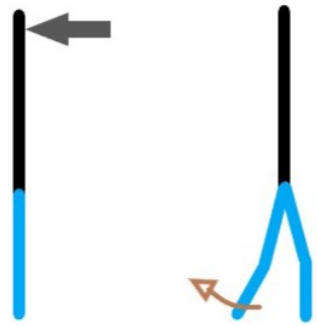

Figure 5: Barebones Open loop control strategy: One leg is taken back when push force is detected.

3.3 Stepping strategy: Forward Kinematics Map for leg

Detailed equations are written in section 6.2

4 SIMULATION AND RESULTS : CAPTURE POINT

Explanation and inferences of simulation results: Simulation was done using capture point calculations in Webots using a 2.4 GHz dual core laptop processor, as is demonstrated in figure 6 and figure 7. Force is applied on the robot manually. It is observed that up to 10 N, there is no tendency of robot to fall due to its weight and PID control. After 10 N, robot starts to tilt and our controller comes into play. Capture point is calculated as given in 6.4 and swing leg (left leg in our case) is moved to compensate for the imbalance cased. The angles of rotation of thigh, knee and ankle joints are calculated using Inverse kinematics using equations given in 6.3. The required angles are achieved using PID control. After the stabilization is achieved, the robot will come back to its original position. It is observed that after 16 N, our robot is unable to balance and falls to ground.

Foot pressure sensor values can be seen in figure 8.

Joint torque values can be seen in figure 9. Joint position values can be seen in figure 10.

IMU sensor values can be seen in figure 11.

5 APPROACH 3: NEURO-FUZZY SYSTEM

Neuro-Fuzzy inference system is an integration of fuzzy inference system and artificial neural networks theory. It is used to determine its parameters (fuzzy sets and fuzzy rules) by processing data samples. Since it based on fuzzy logic, it has learning capability of approximate nonlinear functions.

Modern neuro-fuzzy represented with multilayer feedforward neural networks and follow up to that, it has different models like ANFIS[7], FuNe[6], Fuzzy RuleNet [8]. In this project, we have used ANFIS since it is more efficient. Architecture of ANFIS[9] composed with five layers. The first layer takes the input values and determines the membership functions(MF) belonging to them. And we can assign different number of MFs (such as trimpf, gbellmf, gaussmf, gauss2mf etc.) to the inputs. In the second layer we try to generate the firing strengths for the rules. After that it normalizes the computed firing strengths, by dividing each value for the total firing strength. And after defuzzificated normalized values and the consequence parameter set performed in the fourth layer, last layer return the final output. Since ANFIS converts numeric values into fuzzy values in the preprocessing step, it doesn't need a sigmoid function.

5.1 Simulation and Results

Here, after pushing the robot, we get the desired positionX,Y,Z of the robot leg with respect to the torso from forward kinematic map[6.2]. From that we create our first data with 3 columns of Y,Z,θ_1 and second data with Y,Z,θ_2. Next, we divide the 70% data into training and 30% into test data. Then using ANFIS, with intial FIS = 7 and 150 epochs, we train our model. Using the trained model, we test our model on our test data, and we compare it with our inverse kinematic approach[3] as shown in figure 12. Also, we plot the error difference in θ_1 and θ_2 as shown in figure 13.

But, as we can see that our model is a 2-link model, inverse kinematic approach would be more suitable in this case. We can use Neuro-Fuzzy system in higher number of link model.

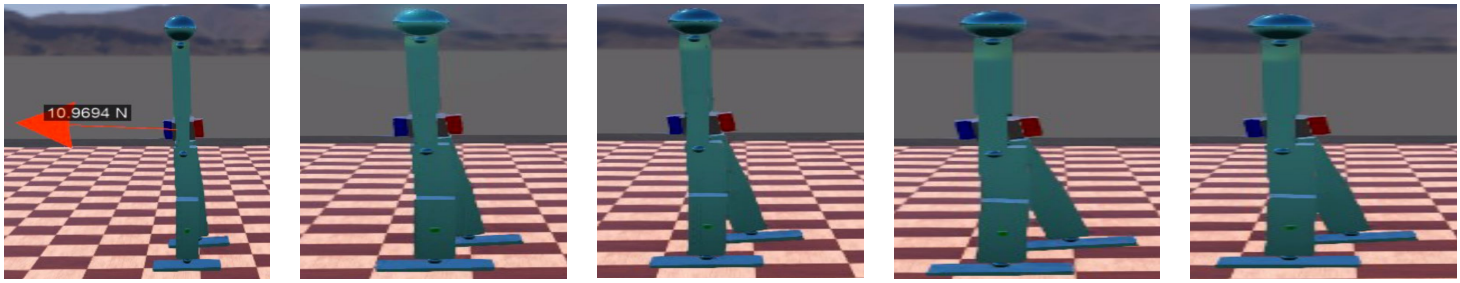

Figure 6: External push applied to biped; it steps backward when push is detected

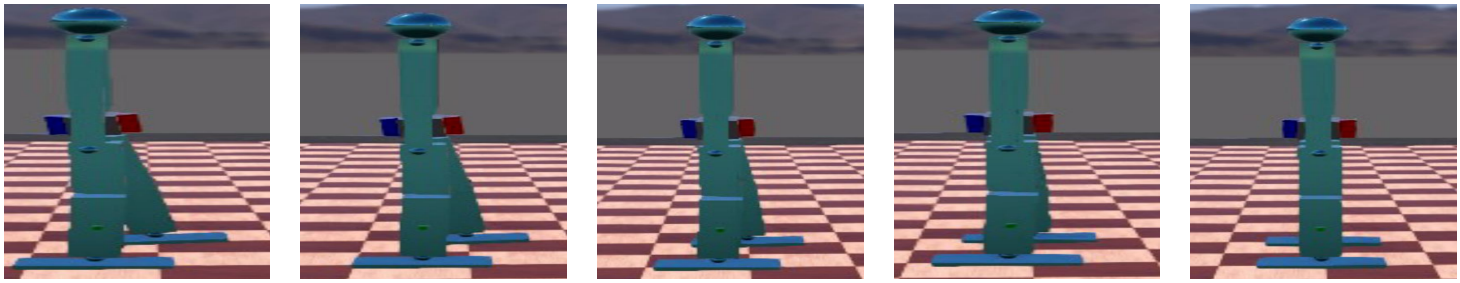

Figure 7: Controller brings the leg back to initial position

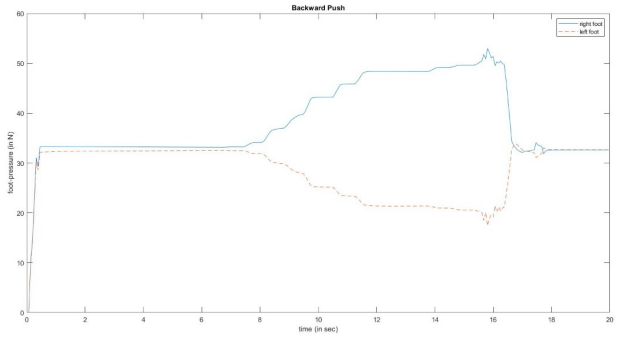

Figure 8: Foot pressure variation. We can see that when the push is detected, the right foot pressure increases and left foot pressure decreases by equal amounts due to weight transfer, and as the biped comes back to initial still position, pressure becomes equal in both the feet.

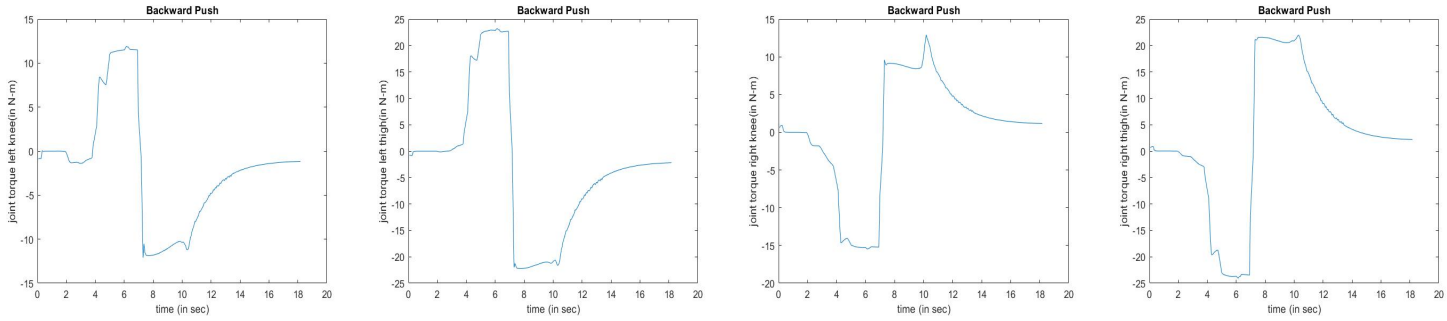

Figure 9: Torques on both knee and thigh joints: In the left knee, on backward push, biped steps backward so there is initial impulse. After the push is released, the controller tries to revert back to the initial position, hence the sharp impulse, and then a near constant torque brings the leg to initial position. Similar pattern can be seen in the other three figures.

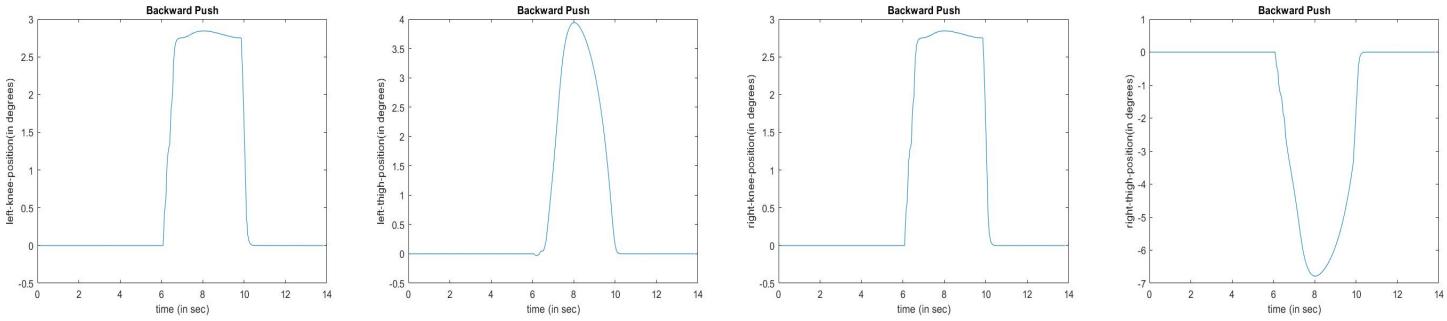

Figure 10: Positions of both knee and thigh joints: Here we can observe that right thigh is rotating anti-clockwise, while the knees are constrained to bend in only one direction-clockwise.

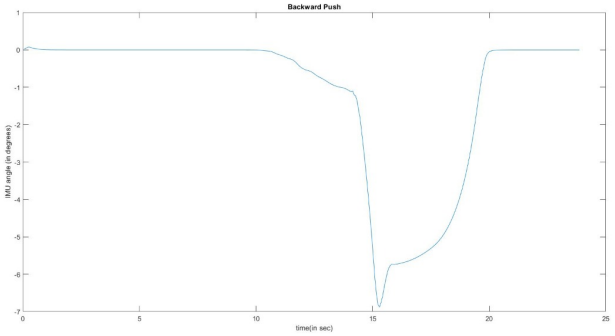

Figure 11: IMU sensor values: roll angle (about x axis): Initial force brings the angle down instantaneously and the controller comes into action just after a threshold is crossed.

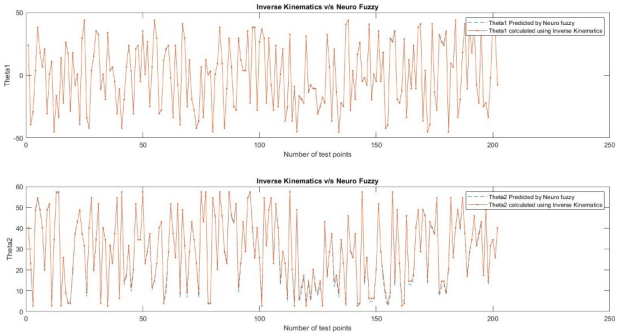

Figure 12: Comparing Inverse kinematics and Neuro fuzzy method for thigh and knee joint angles of one leg. As can be sen that the values predicted from Neuro Fuzzy and calculated values from inverse kinematics closely match and hence NF can be a good predictive model

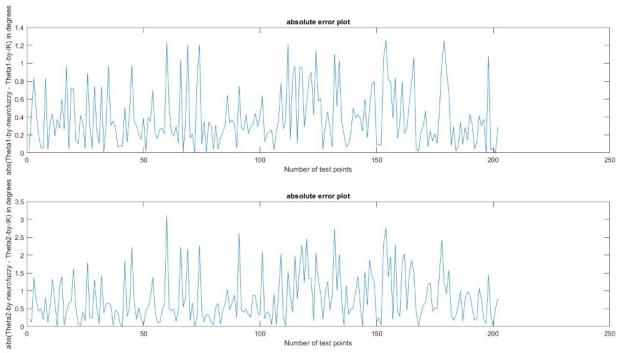

Figure 13: Absolute error between calculated and predicted values: Mean errors are 0.4627° and 0.8916° respectively, hence both models closely agree with each other.

REFERENCES

[1] S. Kolathaya, W. Guffey, R. W. Sinnet and A. D. Ames, "Direct Collocation for Dynamic Behaviors With Nonprehensile Contacts: Application to Flipping Burgers," in IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3677-3684, Oct. 2018, doi: 10.1109/LRA.2018.2854910.

[2] Tam, Benjamin and Navinda Kottege. "Fall avoidance and recovery for bipedal robots using walking sticks." (2016).

[3] Faraji, S., Razavi, H. and Ijspeert, A. J. (2019) 'Bipedal walking and push recovery with a stepping strategy based on time-projection control', The International Journal of Robotics Research, 38(5), pp. 587–611. doi: 10.1177/0278364919835606.

[4] Posa M, Cantu C and Tedrake R (2014) A direct method for trajectory optimization of rigid bodies through contact. The International Journal of Robotics Research 33(1): 69-81. doi: 10.1177/0278364913506757

[5] Al-Shuka, Hayder F. N., Burkhard Corves and Wen-Hong Zhu. "Dynamic Modeling of Biped Robot using Lagrangian and Recursive Newton-Euler Formulations." International Journal of Computer Applications 101 (2014): 1-8.

[6] S. K. Halgamuge and M. Glesner, Neural networks in designing fuzzy systems for real world applications, Fuzzy Sets and Systems 65 (1994) 1–12.

[7] J.-S.R. Jang, ANFIS: Adaptive-Network-Based Fuzzy Inference Systems, IEEE Trans. Systems, Man Cybernetics 23 (1993) 665–685

[8] N. Tschichold-Gürman, RuleNet – A New Knowledge–Based Artificial Neural Network Model with Application Examples in Robotics, PhD thesis (ETH Zürich, 1996).

[9] Wikipedia contributors. "Adaptive neuro fuzzy inference system." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 30 Nov. 2021. Web. 12 Dec. 2021.

[10] Murray, Richard M., Zexiang Li, and Shankar Sastry. A Mathematical Introduction to Robotic Manipulation. Boca Raton: CRC Press, 1994.

[11] J. Pratt, J. Carff, S. Drakunov and A. Goswami, "Capture Point: A Step toward Humanoid Push Recovery," 2006 6th IEEE-RAS International Conference on Humanoid Robots, 2006, pp. 200-207, doi: 10.1109/ICHR.2006.321385.

6 APPENDIX

6.1 Trajectory planning code: MATLAB

```
% 7 link model with variable step h but no optimizing criteria

%clear

[n,N]=constants; %contains no . of links and no. of time steps

lb=-5*ones(1,N*n*4);
ub=5*ones(1,N*n*4);
for k=1:1:N   %all angles limited from 0 to pi
    lb(4*n*(k-1)+1:4*n*(k-1)+7) = zeros(1,7); % Lower bound constraint on theta
    ub(4*n*(k-1)+1:4*n*(k-1)+7) = pi*ones(1,7); % Upper bound constraint on theta
end
lb=[lb,0];
ub=[ub,10];

rng default
x0 = [rand(1,4*n*N),1.5]; % initial guess

opts = optimoptions(@fmincon,'Algorithm','sqp','Display','iter');
tic
figure
x = fmincon(@(x)0,x0,[],[],[],[],lb,ub,@fminconstr,opts);
time_taken=toc;

%Collecting results
th=[];thdot=[];thdotdot=[];u=[];
for k=1:1:N
    th=[th, x(4*n*(k-1)+1:4*n*(k-1)+7)];
    thdot=[thdot, x(4*n*(k-1)+8:4*n*(k-1)+14)];
    thdotdot=[thdotdot, x(4*n*(k-1)+15:4*n*(k-1)+21)];
    u=[u, x(4*n*(k-1)+22:4*n*(k-1)+28)];
end
h=x(end);

%plots
figure
for k=1:1:N
    plotfigure(x(4*n*(k-1)+1:4*n*(k-1)+7))
    axis equal
    hold on
end
%plot(thdot)
%plot(thdotdot)
%plot(u)

function [n,N] = constants()
% constants needed
N=5;  % time instants (generally from 2 to 30)
n=7; %no. of links
end

function O=optimizing_criteria(x)
% norm of control input u is set as criteria
[n,N]=constants;
u=[];
for k=1:1:N
    u=[u, x(4*n*(k-1)+22:4*n*(k-1)+28)];
end
O=norm(u);
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [c,ceq] = fminconstr(x)   %constraints function

c = ineq(x); % No nonlinear inequality
ceq = fbnd(x); % fsolve objective is fmincon nonlinear equality constraints
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function G = ineq(x)
% function containing all inequality constraints only
[n,N]=constants;
l2=0.3;

c1=[];  %knee 1 should not bend backwards
c2=[];  %knee 2 should not bend backwards
for k=1:1:N
    c1= [c1, x(4*n*(k-1)+2)-x(4*n*(k-1)+3)];  %angle2 <= angle3
    c2= [c1, x(4*n*(k-1)+6)-x(4*n*(k-1)+5)];  %angle6 <= angle5
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c3=[]; % minimum height of 2nd foot >=0
c4=[]; % 2nd foot should always move forward
c5=[]; % torso (link 4) should always move forward
% Positions of CoM of each link
for k=1
    th1=x(4*n*(k-1)+1);
    th2=x(4*n*(k-1)+2);
    th3=x(4*n*(k-1)+3);
    th4=x(4*n*(k-1)+4);
    th5=x(4*n*(k-1)+5);
    th6=x(4*n*(k-1)+6);
    th7=x(4*n*(k-1)+7);

    xc1 = 0;
    yc1 = 0;
    xc2 = xc1 + l2/2*cos(th2);
    yc2 = yc1 + l2/2*sin(th2);
    xc3 = xc2 + l2/2*(cos(th2) + cos(th3)) ;
    yc3 = yc2 + l2/2*(sin(th2) + sin(th3)) ;
    xc4 = xc3 + l2/2*(cos(th3) + cos(th4)) ;
    yc4 = yc3 + l2/2*(sin(th3) + sin(th4)) ;
    xc5 = xc4 + l2/2*(-cos(th4) + cos(pi - th5)) ;
    yc5 = yc4 + l2/2*(-sin(th4) - sin(pi - th5)) ;
    xc6 = xc5 + l2/2*(cos(pi - th5) + cos(pi - th6)) ;
    yc6 = yc5 + l2/2*(-sin(pi - th5) - sin(pi - th6)) ;

    xc7 = xc6 + l2/2*cos(pi - th6) ;
    yc7 = yc6 - l2/2*sin(pi - th6) ;
end

for k=2:1:N
    th1=x(4*n*(k-1)+1);
    th2=x(4*n*(k-1)+2);
    th3=x(4*n*(k-1)+3);
    th4=x(4*n*(k-1)+4);
    th5=x(4*n*(k-1)+5);
    th6=x(4*n*(k-1)+6);
    th7=x(4*n*(k-1)+7);

    xc1 = 0;
    yc1 = 0;
    xc2 = xc1 + l2/2*cos(th2);
    yc2 = yc1 + l2/2*sin(th2);
```

```
    xc3 = xc2 + l2/2*(cos(th2) + cos(th3)) ;
    yc3 = yc2 + l2/2*(sin(th2) + sin(th3)) ;
    xc4 = xc3 + l2/2*(cos(th3) + cos(th4)) ;
    yc4 = yc3 + l2/2*(sin(th3) + sin(th4)) ;
    c5 = [c5, xc5-(xc4 + l2/2*(-cos(th4) + cos(pi - th5)))];
    xc5 = xc4 + l2/2*(-cos(th4) + cos(pi - th5)) ;
    yc5 = yc4 + l2/2*(-sin(th4) - sin(pi - th5)) ;
    xc6 = xc5 + l2/2*(cos(pi - th5) + cos(pi - th6)) ;
    yc6 = yc5 + l2/2*(-sin(pi - th5) - sin(pi - th6)) ;
    c4 = [c4,  xc7+0.03-(xc6 + l2/2*cos(pi - th6))];   % 2nd foot should always move forward
    xc7 = xc6 + l2/2*cos(pi - th6) ;
    yc7 = yc6 - l2/2*sin(pi - th6) ;
    c3=[c3, -yc7];          % minimum height of 2nd foot >=0
end

G=[c1, c2, c3, c4, c5];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function F = fbnd(x)
% function containing all equality constraints only
[n,N]=constants;
B=eye(n);
F=[];
syms th1 th2 th3 th4 th5 th6 th7 th1dot th2dot th3dot th4dot th5dot th6dot th7dot th1dotdot th2dotdot th3c
thdotdot = [th1dotdot th2dotdot th3dotdot th4dotdot th5dotdot th6dotdot th7dotdot];
thdot= [th1dot th2dot th3dot th4dot th5dot th6dot th7dot];
th= [th1 th2 th3 th4 th5 th6 th7];
l2=0.3;

for k=1:1:N
    % M*thetadotdot +N+G
    Equations=lageqn();
    u=x(4*n*(k-1)+22:4*n*(k-1)+n*4);
    F=[F, (Equations.'-B*u.').'];
    F= subs(F,[th thdot thdotdot],x(4*n*(k-1)+1:4*n*(k-1)+3*n));
end

%1 time step is from x=1 to 28

% more constraints
%% Foot angles at all times
c1=[]; % 1st foot angle = pi always
c2=[]; % 2nd foot angle = pi always
c3=[]; % torso angle   = pi/2 always
for k=1:1:N
    c1=[c1, x(4*n*(k-1)+1)-pi];
    c2=[c2, x(4*n*(k-1)+7)-pi];
    c3=[c3, x(4*n*(k-1)+4)-pi/2];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%initial angle of link 2 to 6
c4=x(2:6)-pi/2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% final position of 2nd foot = (0.2,
c5=[]; % final position of 2nd foot = (0.2,0)
% Positions of CoM of each link
for k=N   % last time instant
    th1=x(4*n*(k-1)+1);
    th2=x(4*n*(k-1)+2);
    th3=x(4*n*(k-1)+3);
    th4=x(4*n*(k-1)+4);
    th5=x(4*n*(k-1)+5);
    th6=x(4*n*(k-1)+6);
```

```
    th7=x(4*n*(k-1)+7);

    xc1 = 0;
    yc1 = 0;
    xc2 = xc1 + l2/2*cos(th2);
    yc2 = yc1 + l2/2*sin(th2);
    xc3 = xc2 + l2/2*(cos(th2) + cos(th3)) ;
    yc3 = yc2 + l2/2*(sin(th2) + sin(th3)) ;
    xc4 = xc3 + l2/2*(cos(th3) + cos(th4)) ;
    yc4 = yc3 + l2/2*(sin(th3) + sin(th4)) ;
    xc5 = xc4 + l2/2*(-cos(th4) + cos(pi - th5)) ;
    yc5 = yc4 + l2/2*(-sin(th4) - sin(pi - th5)) ;
    xc6 = xc5 + l2/2*(cos(pi - th5) + cos(pi - th6)) ;
    yc6 = yc5 + l2/2*(-sin(pi - th5) - sin(pi - th6)) ;
    xc7 = xc6 + l2/2*cos(pi - th6) ;
    yc7 = yc6 - l2/2*sin(pi - th6) ;
    c5=[c5 xc7-0.2 yc7-0];        % final position of 2nd foot = (0.2,0)
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k=1   % last time instant
    th1=x(4*n*(k-1)+1);
    th2=x(4*n*(k-1)+2);
    th3=x(4*n*(k-1)+3);
    th4=x(4*n*(k-1)+4);
    th5=x(4*n*(k-1)+5);
    th6=x(4*n*(k-1)+6);
    th7=x(4*n*(k-1)+7);
    th1dot=x(4*n*(k-1)+8);
    th2dot=x(4*n*(k-1)+9);
    th3dot=x(4*n*(k-1)+10);
    th4dot=x(4*n*(k-1)+11);
    th5dot=x(4*n*(k-1)+12);
    th6dot=x(4*n*(k-1)+13);
    th7dot=x(4*n*(k-1)+14);
    th1dotdot=x(4*n*(k-1)+15);
    th2dotdot=x(4*n*(k-1)+16);
    th3dotdot=x(4*n*(k-1)+17);
    th4dotdot=x(4*n*(k-1)+18);
    th5dotdot=x(4*n*(k-1)+19);
    th6dotdot=x(4*n*(k-1)+20);
    th7dotdot=x(4*n*(k-1)+21);
end

c6=[];
h=x(end);
for k=2:1:N   % last time instant
    c6=[c6, x(4*n*(k-1)+1)-th1-h*x(4*n*(k-1)+8)];
    th1=x(4*n*(k-1)+1);
    c6=[c6, x(4*n*(k-1)+2)-th2-h*x(4*n*(k-1)+9)];
    th2=x(4*n*(k-1)+2);
    c6=[c6, x(4*n*(k-1)+3)-th3-h*x(4*n*(k-1)+10)];
    th3=x(4*n*(k-1)+3);
    c6=[c6, x(4*n*(k-1)+4)-th4-h*x(4*n*(k-1)+11)];
    th4=x(4*n*(k-1)+4);
    c6=[c6, x(4*n*(k-1)+5)-th5-h*x(4*n*(k-1)+12)];
    th5=x(4*n*(k-1)+5);
    c6=[c6, x(4*n*(k-1)+6)-th6-h*x(4*n*(k-1)+13)];
    th6=x(4*n*(k-1)+6);
    c6=[c6, x(4*n*(k-1)+7)-th7-h*x(4*n*(k-1)+14)];
    th7=x(4*n*(k-1)+7);
    c6=[c6, x(4*n*(k-1)+8)-th1dot-h*th1dotdot];
    th1dot=x(4*n*(k-1)+8);
```

```
    c6=[c6, x(4*n*(k-1)+9)-th2dot-h*th2dotdot];
    th2dot=x(4*n*(k-1)+9);
    c6=[c6, x(4*n*(k-1)+10)-th3dot-h*th3dotdot];
    th3dot=x(4*n*(k-1)+10);
    c6=[c6, x(4*n*(k-1)+11)-th4dot-h*th4dotdot];
    th4dot=x(4*n*(k-1)+11);
    c6=[c6, x(4*n*(k-1)+12)-th5dot-h*th5dotdot];
    th5dot=x(4*n*(k-1)+12);
    c6=[c6, x(4*n*(k-1)+13)-th6dot-h*th6dotdot];
    th6dot=x(4*n*(k-1)+13);
    c6=[c6, x(4*n*(k-1)+14)-th7dot-h*th7dotdot];
    th7dot=x(4*n*(k-1)+14);
    th1dotdot=x(4*n*(k-1)+15);
    th2dotdot=x(4*n*(k-1)+16);
    th3dotdot=x(4*n*(k-1)+17);
    th4dotdot=x(4*n*(k-1)+18);
    th5dotdot=x(4*n*(k-1)+19);
    th6dotdot=x(4*n*(k-1)+20);
    th7dotdot=x(4*n*(k-1)+21);
end



F=[F,c1,c2,c3,c4,c5,c6];
F=double(F);
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Equations=lageqn()
%returns 7 lagrangian equations
syms th1 th2 th3 th4 th5 th6 th7 th1dot th2dot th3dot th4dot th5dot th6dot th7dot th1dotdot th2dotdot th3d
thdotdot = [th1dotdot th2dotdot th3dotdot th4dotdot th5dotdot th6dotdot th7dotdot];
thdot= [th1dot th2dot th3dot th4dot th5dot th6dot th7dot];
th= [th1 th2 th3 th4 th5 th6 th7];
Equations=[(3*th1dotdot)/400, (9*sin(th2 - th3)*th3dot^2)/200 + (3*th2dotdot)/25 + (32373*cos(th2))/2000 +
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Equations=lagran()
% fuction to find lagrange eqns for 7 link model (not used when running
% optimization)

l1=0.3; l2=0.3; l3=0.3; l4=0.3; l5=0.3; l6=0.3; l7=0.3;
syms th1 th2 th3 th4 th5 th6 th7 th1dot th2dot th3dot th4dot th5dot th6dot th7dot
m1=1; m2=1; m3=1; m4=1; m5=1; m6=1; m7=1;      %Inertia & masses
g=9.81;      % gravitional acceleration
syms th1dotdot th2dotdot th3dotdot th4dotdot th5dotdot th6dotdot th7dotdot
I1=m1*l1^2/12;I2=m2*l2^2/12;I3=m3*l3^2/12;I4=m4*l4^2/12;I5=m5*l5^2/12;I6=m6*l6^2/12;I7=m7*l7^2/12;

% Positions of CoM of each link



xc1 = 0;
yc1 = 0;
xc2 = xc1 + l2/2*cos(th2);
yc2 = yc1 + l2/2*sin(th2);
xc3 = xc2 + l2/2*(cos(th2) + cos(th3)) ;
yc3 = yc2 + l2/2*(sin(th2) + sin(th3)) ;
xc4 = xc3 + l2/2*(cos(th3) + cos(th4)) ;
```

```
yc4 = yc3 + l2/2*(sin(th3) + sin(th4)) ;
xc5 = xc4 + l2/2*(-cos(th4) + cos(pi - th5)) ;
yc5 = yc4 + l2/2*(-sin(th4) - sin(pi - th5)) ;
xc6 = xc5 + l2/2*(cos(pi - th5) + cos(pi - th6)) ;
yc6 = yc5 + l2/2*(-sin(pi - th5) - sin(pi - th6)) ;
xc7 = xc6 + l2/2*cos(pi - th6) ;
yc7 = yc6 - l2/2*sin(pi - th6) ;



%%
% The linear velocity of the center of the mass of each link
% link1, fixed foot
xc1dot=0;
yc1dot=0;
vc1=[xc1dot;yc1dot];

%link2
xc2dot=diff(xc2,th2)*th2dot;
yc2dot=diff(yc2,th2)*th2dot;
vc2=[xc2dot;yc2dot];

%link3, thigh1
xc3dot=diff(xc3,th2)*th2dot+diff(xc3,th3)*th3dot;
yc3dot=diff(yc3,th2)*th2dot+diff(yc3,th3)*th3dot;
vc3=[xc3dot;yc3dot];

%link4, torso
xc4dot=diff(xc4,th3)*th3dot+diff(xc4,th4)*th4dot;
yc4dot=diff(yc4,th3)*th3dot+diff(yc4,th4)*th4dot;
vc4=[xc4dot;yc4dot];

% link5, thigh2
xc5dot=diff(xc5,th4)*th4dot+diff(xc5,th5)*th5dot;
yc5dot=diff(yc5,th4)*th4dot+diff(yc5,th5)*th5dot;
vc5=[xc5dot;yc5dot];

% link6
xc6dot=diff(xc6,th5)*th5dot+diff(xc6,th6)*th6dot;
yc6dot=diff(yc6,th5)*th5dot+diff(yc6,th6)*th6dot;
vc6=[xc6dot;yc6dot];

% link7, free foot
xc7dot=diff(xc7,th6)*th6dot;
yc7dot=diff(yc7,th6)*th6dot;
vc7=[xc7dot;yc7dot];

%%
% Equations of motion
Mt=[m1,m2,m3,m4,m5,m6,m7];
THDOTt=[th1dot,th2dot,th3dot,th4dot,th5dot,th6dot,th7dot];
% THDOTDOTt=[th1dotdot,th2dotdot,th3dotdot,th4dotdot,th5dotdot];
It=[I1,I2,I3,I4,I5,I6,I7];
YCt=[yc1,yc2,yc3,yc4,yc5,yc6,yc7];
VCt=[(vc1(1).^2+vc1(2).^2),(vc2(1).^2+vc2(2).^2),(vc3(1).^2+vc3(2).^2),(vc4(1).^2+vc4(2).^2),(vc5(1).^2+v

for i=1:7
    k(i)=simplify(0.5*Mt(i)*VCt(i)+0.5*It(i)*THDOTt(i).^2);
    p(i)=simplify(Mt(i)*g*YCt(i));
end
K=sum(k);
P=sum(p);
L=K-P;
```

```
Equations=Lagrange(L,[th1 th1dot th1dotdot th2 th2dot th2dotdot th3 th3dot th3dotdot th4 th4dot th4dotdot
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotfigure(th)
%used to plot biped figure
l2=0.3;
theta=th;
xc(1)= 0;
yc(1) = 0;
xc(2) = xc(1) + l2/2*cos(theta(2));
yc(2) = yc(1) + l2/2*sin(theta(2));
xc(3) = xc(2) + l2/2*(cos(theta(2)) + cos(theta(3)));
yc(3) = yc(2) + l2/2*(sin(theta(2)) + sin(theta(3))) ;
xc(4) = xc(3) + l2/2*(cos(theta(3)) + cos(theta(4))) ;
yc(4) = yc(3) + l2/2*(sin(theta(3)) + sin(theta(4))) ;
xc(5) = xc(4) + l2/2*(-cos(theta(4)) + cos(pi - theta(5))) ;
yc(5) = yc(4) + l2/2*(-sin(theta(4)) - sin(pi - theta(5))) ;
xc(6) = xc(5) + l2/2*(cos(pi - theta(5)) + cos(pi - theta(6))) ;
yc(6) = yc(5) + l2/2*(-sin(pi - theta(5)) - sin(pi - theta(6))) ;
xc(7) = xc(6) + l2/2*cos(pi - theta(6)) ;
yc(7) = yc(6) - l2/2*sin(pi - theta(6)) ;
for i=1:7
    xcendb(i)=(xc(i)-l2/2*cos(theta(i)));
    xcendf(i)=(xc(i)+l2/2*cos(theta(i)));
    ycendb(i)=(yc(i)-l2/2*sin(theta(i)));
    ycendf(i)=(yc(i)+l2/2*sin(theta(i)));
end
vec1=[xcendb' xcendf']';
x=vec1(:)';
vec2=[ycendb' ycendf']';
y=vec2(:)';
plot([x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8)],[y(1) y(2) y(3) y(4) y(5) y(6) y(7) y(8)]);
hold on
plot([x(7) x(9) x(11) x(13) x(14)],[y(7) y(9) y(11) y(13) y(14)]);
hold on
scatter(x,y,'filled');
end
```

6.2 Stepping strategy: Forward Kinematics Map[10] for leg

Figure for forward kinematic analysis can be seen in 14.

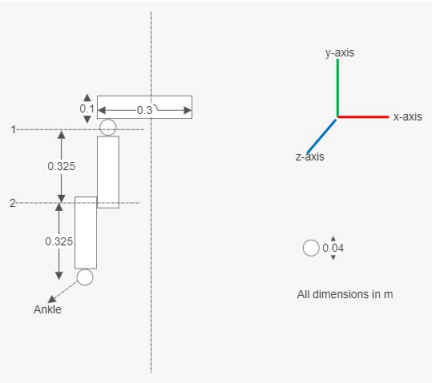

Figure 14: Forward kinematics analysis

ω_1 and ω_2 are axis of rotation of knee 1 joint and knee 2 joint respectively

$$\omega_1 = \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \omega_2 = \begin{pmatrix}1\\0\\0\end{pmatrix} \tag{22}$$

q_1 and q_2 are points on the axes

$$q_1 = \begin{pmatrix}-0.13\\-0.07\\0\end{pmatrix} \quad q_2 = \begin{pmatrix}-0.14\\-0.38\\0\end{pmatrix} \tag{23}$$

Twist coordinates are

$$\xi_1 = \begin{pmatrix}V_1\\W_1\end{pmatrix} = \begin{pmatrix}-W_1 \times q_1\\W_1\end{pmatrix} = \begin{pmatrix}0\\0\\0.07\\1\\0\\0\end{pmatrix} \tag{24}$$

$$\xi_2 = \begin{pmatrix}V_2\\W_2\end{pmatrix} = \begin{pmatrix}-W_2 \times q_2\\W_2\end{pmatrix} = \begin{pmatrix}0\\0\\0.38\\1\\0\\0\end{pmatrix} \tag{25}$$

Initial transformation matrix:

$$g_{\delta T}(0) = \begin{bmatrix}I & q_2\\0 & 1\end{bmatrix} \tag{26}$$

Forward kinamatics map:

$$g_{\delta T} = e^{\hat{\xi_1}\theta_1} e^{\hat{\xi_2}\theta_2} g_{\delta T}(0), \tag{27}$$

where θ_1 and θ_2 are angles of rotation. Rodriguez formula:

$$e^{\hat{\xi_1}\theta_1} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & c\theta_1 & -s\theta_1 & 0.07(c\theta_1 - 1)\\0 & s\theta_1 & c\theta_1 & 0.07(s\theta_1)\\0 & 0 & 0 & 1\end{bmatrix} \tag{28}$$

$$e^{\hat{\xi_2}\theta_2} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & c\theta_2 & -s\theta_2 & 0.38(c\theta_2 - 1)\\0 & s\theta_2 & c\theta_2 & 0.38(s\theta_2)\\0 & 0 & 0 & 1\end{bmatrix} \tag{29}$$

$$g_{\delta T}(\theta_1, \theta_2) = \begin{bmatrix}1 & 0 & 0 & -0.14\\0 & c(\theta_1+\theta_2) & -s(\theta_1+\theta_2) & -0.32(c\theta_1 - 0.07)\\0 & s(\theta_1+\theta_2) & c(\theta_1+\theta_2) & -0.32(s\theta_1)\\0 & 0 & 0 & 1\end{bmatrix} \tag{30}$$

So for the left leg, $P_A = g_{\delta T} P_{2A}$ from this we can find foot coordinates X, Y, Z as follows,

$$X = -0.14 \tag{31}$$

$$Y = -0.32cos(\theta_1 + \theta_2) - 0.32cos\theta_1 - 0.07 \tag{32}$$

$$Z = -0.32sin(\theta_1 + \theta_2) - 0.32sin\theta_1 \tag{33}$$

, where,

$$P_{A2} = \begin{bmatrix} 0 \\ -0.32 \\ 0 \\ 1 \end{bmatrix} \tag{34}$$

Also, the for the right leg,

$$X = 0.14 \tag{35}$$

$$Y = -0.32cos(\theta_1 + \theta_2) - 0.32cos\theta_1 - 0.07 \tag{36}$$

$$Z = -0.32sin(\theta_1 + \theta_2) - 0.32sin\theta_1 \tag{37}$$

6.3 Inverse kinematic for leg

$$Y + 0.07 = -0.32[c(\theta_1 + \theta_2) + c\theta_1] \tag{38}$$

$$Z + 0.07 = -0.32[s(\theta_1 + \theta_2) + s\theta_1] \tag{39}$$

Now we can get the $\theta_1$and θ_2 from the above equations
(1) By squaring and adding them, we can get θ_2 as follows,

$$\theta_2 = c^{-1}[\frac{(Y + 0.07)^2}{2(0.32)^2} + \frac{(Z)^2}{2(0.32)^2} - 1] \tag{40}$$

(2) By dividing both the equations, we can get θ_2 as follows,

$$\theta_1 = tan^{-1}[\frac{Z}{(Y + 0.07)} - 0.5\theta_2] \tag{41}$$

6.4 Orbital energy and capture point calculation

Linear Inverted pendulum model :

LIPM model can be seen in figure 15.

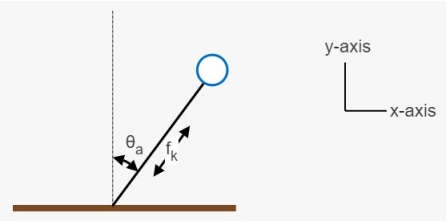

Figure 15: LIPM

We know that, $m\ddot{x} = f_x s\theta_a$ and $m\ddot{y} = -mg + f_x cos\theta_a$,
where f_X = linear actuation force on the leg and
θ_a = vertical leg angle
l is the distace of C.o.M. from ground.

Now, with the assumption of negligible variation in vertical direction = 0 and at t = 0, y(0) = l
thus $f_x = \frac{mg}{c\theta_a} = \frac{mgl}{y}$, since $c\theta_a = \frac{y}{l}$

$\therefore m\ddot{x} = \frac{mgx}{y}$, which gives, $d\dot{x} = \frac{gx}{y}$

We know that, orbital energy is defined as,

$$E_{LIP} = \frac{1}{2}\dot{x}^2 + \frac{g}{2l}x^2 \tag{42}$$

For final stable position, $E_{LIP} = 0$

$\therefore \dot{x} = \mp x\sqrt{\frac{g}{l}}$

And, we can get the capture point from this equation, $x_{capture} = \dot{x}\sqrt{\frac{l}{g}}$

Since, $\dot{x} = lcos(\theta_a)(\dot{\theta_a})$

We can find the $(\dot{\theta_a})$ from enery conservation law, since,

Kinetic energy(K.E.) = Potential energy(P.E.),

$\frac{1}{2}ml^2\dot{\theta_a}^2 = mgl(1 - c\theta_a)$

$$\therefore \dot{\theta_a} = \sqrt{\frac{2g}{l}(1 - c\theta_a)} \tag{43}$$

Therefore, We get calculate our capture point from following equation, $x_{capture} = lcos(\theta_a)\sqrt{\frac{2g}{l}(1 - c\theta_a)}\sqrt{\frac{l}{g}}$

We will measure the θ_a using IMU sensor in webots.

6.5 Side push: balancing strategy

We assume that a wall is placed along beside the biped as shown in figure 16 . As the biped is pushed sideways, one hand is raised as shown in figure 16. x is the distance from the wall as measured by the distance sensor and d_{wall} is the distance between the edge of the arm and wall. We rotate the arm by θ so that the hand rests on the wall to balance it. Some trigonometric calculations:

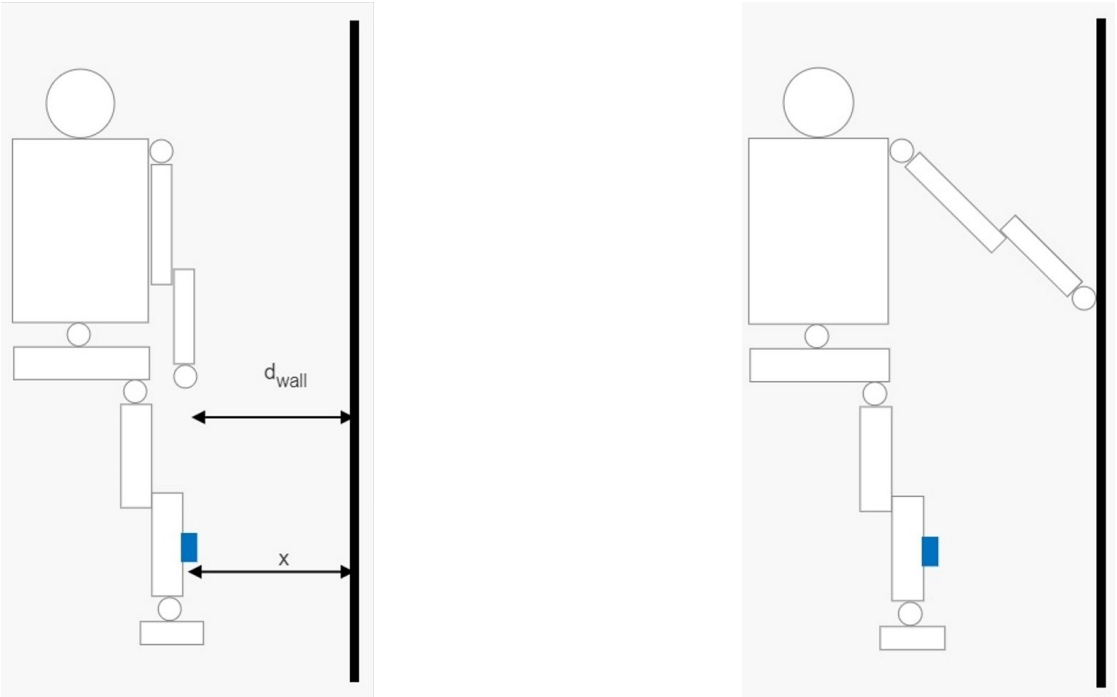

Figure 16: Configuration of wall and biped

$$d_{wall} = x + 0.09 \tag{44}$$

$$sin(\theta) = \frac{d_{wall}}{l_{arm}} = \frac{x + 0.09}{0.61} \tag{45}$$

6.6 Webots Kinematics code: C

```
#include <webots/robot.h>
#include <webots/motor.h>
#include <webots/position_sensor.h>
#include <stdio.h>
#include <webots/inertial_unit.h>
#include <webots/distance_sensor.h>
#include <webots/touch_sensor.h>
#include <math.h>

#define TIME_STEP 64
#define pi 3.14

int i = 0;


int main() {
  /* necessary to initialize webots stuff */
  wb_robot_init();

  WbDeviceTag left_thigh_FB = wb_robot_get_device("left thigh joint forward backward");
  WbDeviceTag left_thigh_SW = wb_robot_get_device("left thigh joint sideways");
  WbDeviceTag left_knee_FB = wb_robot_get_device("left knee joint forward backward");
  WbDeviceTag left_ankle_FB = wb_robot_get_device("left ankle joint forward backward");
  WbDeviceTag left_ankle_SW = wb_robot_get_device("left ankle joint sideways");

  WbDeviceTag right_thigh_FB = wb_robot_get_device("right thigh joint forward backward");
  WbDeviceTag right_thigh_SW = wb_robot_get_device("right thigh joint sideways");
  WbDeviceTag right_knee_FB = wb_robot_get_device("right knee joint forward backward");
  WbDeviceTag right_ankle_FB = wb_robot_get_device("right ankle joint forward backward");
  WbDeviceTag right_ankle_SW = wb_robot_get_device("right ankle joint sideways");

  WbDeviceTag middle_body_FB = wb_robot_get_device("middle body forward backward");
  WbDeviceTag middle_body_SW = wb_robot_get_device("middle body sideways");

  WbDeviceTag left_shoulder_FB = wb_robot_get_device("left shoulder forward backward");
  WbDeviceTag left_shoulder_SW = wb_robot_get_device("left shoulder sideways");
  WbDeviceTag left_elbow_FB = wb_robot_get_device("left elbow forward backward");

  WbDeviceTag right_shoulder_FB = wb_robot_get_device("right shoulder forward backward");
  WbDeviceTag right_shoulder_SW = wb_robot_get_device("right shoulder sideways");
  WbDeviceTag right_elbow_FB = wb_robot_get_device("right elbow forward backward");

  WbDeviceTag right_thigh_joint_sensor_FB = wb_robot_get_device("position sensor right thigh joint forwar
  WbDeviceTag right_knee_joint_sensor_FB = wb_robot_get_device("position sensor right knee joint forward
  WbDeviceTag left_thigh_joint_sensor_FB = wb_robot_get_device("position sensor left thigh joint forward
  WbDeviceTag left_knee_joint_sensor_FB = wb_robot_get_device("position sensor left knee joint forward ba


  WbDeviceTag distance_sensor_right_lower_leg = wb_robot_get_device("distance sensor right lower leg");
  WbDeviceTag distance_sensor_left_lower_leg = wb_robot_get_device("distance sensor left lower leg");
  WbDeviceTag IMU_middle_body = wb_robot_get_device("middle body inertial unit");
  WbDeviceTag touch_sensor_right_foot = wb_robot_get_device("touch sensor right foot");
  WbDeviceTag touch_sensor_left_foot = wb_robot_get_device("touch sensor left foot");

  wb_position_sensor_enable(right_thigh_joint_sensor_FB ,TIME_STEP);
  wb_position_sensor_enable(right_knee_joint_sensor_FB ,TIME_STEP);
  wb_position_sensor_enable(left_thigh_joint_sensor_FB ,TIME_STEP);
  wb_position_sensor_enable(left_knee_joint_sensor_FB ,TIME_STEP);

  wb_inertial_unit_enable(IMU_middle_body ,TIME_STEP);
```

```
wb_distance_sensor_enable(distance_sensor_right_lower_leg , TIME_STEP);
wb_distance_sensor_enable(distance_sensor_left_lower_leg , TIME_STEP);
wb_touch_sensor_enable(touch_sensor_right_foot ,TIME_STEP);
wb_touch_sensor_enable(touch_sensor_left_foot ,TIME_STEP);

wb_motor_enable_force_feedback(left_thigh_FB ,TIME_STEP);
wb_motor_enable_force_feedback(left_knee_FB ,TIME_STEP);
wb_motor_enable_force_feedback(left_ankle_FB ,TIME_STEP);

wb_motor_enable_force_feedback(right_thigh_FB ,TIME_STEP);
wb_motor_enable_force_feedback(right_knee_FB ,TIME_STEP);
wb_motor_enable_force_feedback(right_ankle_FB ,TIME_STEP);


const double *rpy;
double ds_right;
double ds_left;
float middle_body_angle;
float right_thigh_angle;
double ts_right_foot;
double ts_left_foot;

while (wb_robot_step(TIME_STEP) != -1)
{
  rpy = wb_inertial_unit_get_roll_pitch_yaw(IMU_middle_body);
  ds_right = wb_distance_sensor_get_value(distance_sensor_right_lower_leg);
  ds_left = wb_distance_sensor_get_value(distance_sensor_left_lower_leg);
  ts_right_foot = wb_touch_sensor_get_value(touch_sensor_right_foot);
  ts_left_foot = wb_touch_sensor_get_value(touch_sensor_left_foot);
  double right_thigh_ps = wb_position_sensor_get_value(right_thigh_joint_sensor_FB);
  double right_knee_ps = wb_position_sensor_get_value(right_knee_joint_sensor_FB);
  double left_thigh_ps = wb_position_sensor_get_value(left_thigh_joint_sensor_FB);
  double left_knee_ps = wb_position_sensor_get_value(left_knee_joint_sensor_FB);

  float l_arm = 0.61;

  /* Process sensor data here */
  printf("roll is %f\n",rpy[0]);
  printf("pitch is %f\n",rpy[1]);
  printf("distance of right leg & right wall is %f\n",ds_right);
  printf("distance of left leg & left wall is %f\n",ds_left);
  printf("yaw is %f\n",rpy[2]);

   /* Set intitial postion as all zero joint angles*/
   wb_motor_set_position(left_thigh_FB , 0);
   wb_motor_set_position(left_knee_FB , 0);
   wb_motor_set_position(right_thigh_FB , 0);
   wb_motor_set_position(right_knee_FB , 0);
   wb_motor_set_position(left_ankle_FB , 0);
   wb_motor_set_position(right_ankle_FB , 0);
   wb_motor_set_position(middle_body_FB , 0);
   wb_motor_set_position(right_shoulder_SW , 0);
   wb_motor_set_position(left_shoulder_SW , 0);

    if((rpy[0]<-0.02 && rpy[0]>-1.5) || (rpy[0]>0.02 && rpy[0]<1.5)) //if roll angle > 1 degee and less
    {
      /*Compensate for direction of all*/
      if(rpy[0]>0.02 && rpy[0]<1.5)
        {
          i = 2;
        }
        else
```

```
    {
      i = 1;
    }

  //Capture point calculation for swinging leg
  float l_com = 0.71; // Distance between ground and COM of body
  float g = 9.81;
  float phi_dot = sqrt(2*(g/l_com)*(1-cos(rpy[0])));
  float z_torso =0.95*pow(-1,i)*(l_com*cos(rpy[0])*phi_dot*(sqrt(l_com/g)));
  float Y = -0.71*cos(rpy[0]); //distance from robot frame to ankle joint
  float Z = z_torso;
  float T = atan(Z/(Y+0.07));
  // Joint angles calculation using Inverse Kinematics
  float Theta2 = acos(4.88*(Y+0.07)*(Y+0.07) + 4.88*Z*Z - 1);
  float Theta1 = T - 0.5*Theta2;


  //for non swinging leg
  float Y_ns = -0.71*cos(rpy[0]);
  float Z_ns = -Z;
  float T_ns = atan(Z_ns/(Y_ns+0.07));
  // Joint angles calculation using Inverse Kinematics
  float Theta2_ns = acos(4.88*(Y_ns+0.07)*(Y_ns+0.07) + 4.88*Z_ns*Z_ns - 1);
  float Theta1_ns = T_ns - 0.5*Theta2_ns;


wb_motor_set_position(left_thigh_FB , Theta1);
wb_motor_set_position(left_knee_FB , Theta2);
wb_motor_set_position(middle_body_FB , -rpy[0]);
wb_motor_set_position(right_thigh_FB ,Theta1_ns) ;
wb_motor_set_position(right_knee_FB , Theta2_ns);
wb_motor_set_position(right_ankle_FB , -(Theta1_ns+Theta2_ns));

//foot stopping criteria
  if (Theta1_ns<-0.15 || Theta1_ns>0.15)
          {
          wb_motor_set_position(left_ankle_FB , -(Theta1+Theta2));
          }
  else
          {
          wb_motor_set_position(left_ankle_FB , -(Theta1+Theta2)- rpy[0]);
          }

}


//Side falling
if(rpy[1]<-0.02 && rpy[1]>-1.5)
{
  float d_wall_right = ds_right + 0.09;
  if(d_wall_right<l_arm)
  {
  float theta_right_arm = asin(d_wall_right/l_arm);
  wb_motor_set_position(right_shoulder_SW , theta_right_arm - rpy[1]);
  }
  else
  {
  wb_motor_set_position(right_shoulder_SW , 1.57 - rpy[1]);
  }
}

if(rpy[1]>0.02 && rpy[1]<1.5)
{
```

```
        float d_wall_left = ds_left + 0.09;
        if(d_wall_left<l_arm)
        {
        float theta_left_arm = asin(d_wall_left/l_arm);
        wb_motor_set_position(left_shoulder_SW, -(theta_left_arm + rpy[1]));
        }
        else
        {
        wb_motor_set_position(left_shoulder_SW, -(1.57 + rpy[1]));
        }
      }

  };

  wb_robot_cleanup();

  return 0;
}
```