Planar Bipedal Robot

A Major Qualifying Project Report submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the Degree of Bachelor of Science in

Mechanical Engineering

August, 2018

Authors

Sahawat Amonlikitsin

Project Advisor;

Professor Marko B. Popovic, Advisor

Professor Pradeep Radhakrishnan, Co -Advisor

Table of Contents

Table of Figures

Table of Tables

Abstract

The basis of this project is to create a walkable planar bipedal robot. Walkable gait is designed using concept of zero-moment point. The generated gait will be test on MATLAB/Simulink and later on a bipedal robot with a constrained in transverse plane. At the end of the project, we can produce a walkable gait which works on both simulation and the robot.

Acknowledgement

First of all, I would like to thank you of our advisors, Professor Marko Popovic and Professor Pradeep Radhakrishan. With your support and guidance that the project is made possible.

I would also like to acknowledge and thank Thananart Piyajarawong and Zhengxing Yang. Although we cannot accomplish this project together, their contribution in this project become my inspiration to pursuit greater depth in robotics field.

Chapter 1: Introduction

Bipedal robot have been a challenge for many year. Not only complexity from high DOF (degrees of freedom), but also its underactuated nature where not every DOF is directly controllable.

For bipedal robot, its underactuated nature comes from where its foot interacts with environment: one cannot directly control how the foot will either slip or rotate while it's attracting to the floor. Therefore, whether robot will fall or stand cannot be directly controlled. Placing foot in wrong position causes unstable position whether it stand with one leg (Single Support Phase) or both leg (Double Support Phase)

To determine stability of biped robot, concept of Zero-Moment Point (ZMP) was introduced in 1986 by Vukobratovi´c and Juriˇci´c. ZMP, in short, is a point where ground force reaction acting robot in order to balance robot. If ZMP is in support polygon, then robot is stable and vice-versa. Concept of ZMP plays an important role in our project as we use it to design robot walking pattern.

The goal of this project is to build a walkable biped robot with constraint in transverse plane. There's 4 phases in our project. The first phase introduces design and manufacturing process of robot mechanism. In this phase we also pick a suitable servo for our project. In the second phase, we import and create simulation of our robot in MATLAB/SIMULINK. For the third phase, we use concept of ZMP and knowledge of mechanism and electronic limitation of our robot to design walking gait. This gait will be used in the last phase where we test the gait in the simulation and the robot respectively.

Chapter 2: Background

2.1 Passive and Powered Bipedal Robot

In general, depending on reliability on power supply, we can categorize bipedal robot into two: passive, and powered. Passive bipedal do not draw power from a supply. Instead, the robots rely on potential energy: the robot must be placed on inclined slope for it will be able to walk. Because design of passive bipedal robot heavily exploit dynamics nature of mechanism and geometry of the robot, one requires deep understanding in robot dynamics. An example of passive bipedal robot can be seen in figure below:

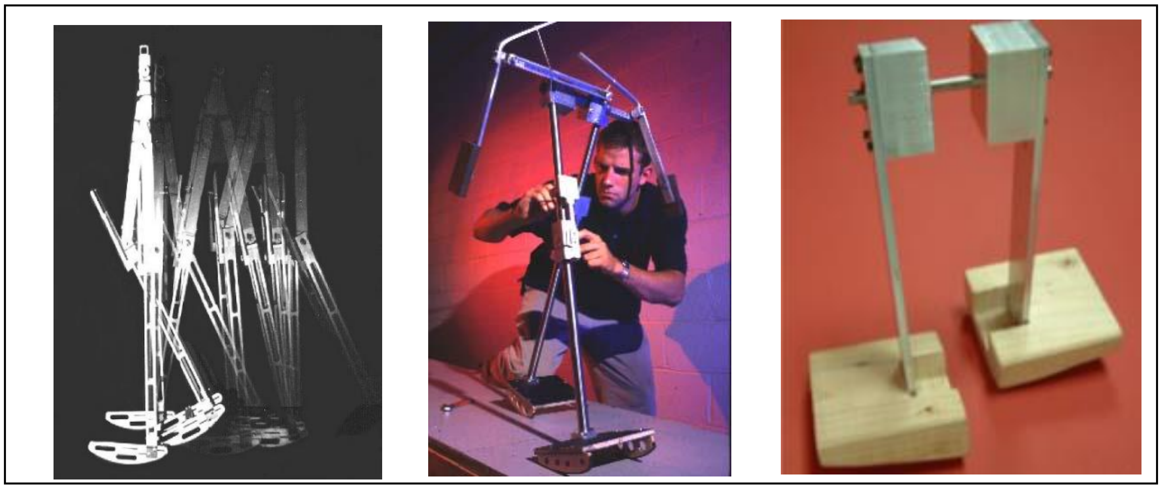


Figure 1: Examples of Passive Bipedal Robot. From left to right are 1) Imitation of Tad Mcgeer's machine by Biorobotics and Locomotion Lab, Cornell University 2) Passive walker by Steven H. Collins from Cornell University 3) Passive Walker by Russ Tedrake from MIT Media Lab.

Unlike passive robot, powered bipedal robot is less reliable on nature of its mechanism and geometry. Unlike passive walker, powered bipedal robot is more adaptive to its environment: it walking ability do not limit only to inclined slope. This Powered bipedal robot is so typical that you commonly see them in commercial. One of the most notable powered bipedal robot is AZIMO by Honda Company, which have been seen as a representative of bipedal robot community since 2001.

2.2 Concept of Zero-Moment Point (ZMP)

Though there’s no formal definition of ZMP given before, in work of M. Vukobratovic “Zero-Moment Point – 35 years of its life” have provided explanation zero-moment point which is summarized as following:

During stance phase and under non-slip condition (friction force is enough to hold robot foot from sliding), zero-moment Point (ZMP) is a point where ground force reaction (GRF) acting on a foot in order to counter balance external moment.

The reader can do experiment with yourself to feel ZMP: while you are standing on one leg and moving upper body without using your hand to help balancing yourself, in some stance, there’s a place on your feet where you can feel intense force acting on. ZMP is such a point.

During the experiment, the reader can observes that when you tries to lean to any direction, GRF tends to acting toward to those side that you are leaning on. For example, if you are leaning toward right side, GRF is acting toward around right side of the foot. As you are leaning until you are off balance, you can feel intense force on an edge of your foot. This phenomena occurs as when you leaning, your weight generate moment. The more you lean, the greater gravity moment and the further GRF move away from the rotation point as it tries to counter moment from your weight. When you are off-balance, GRF is already at the edge of your foot and it cannot move further beyond this point; therefore, external moment overcomes GRF moment. In short, during single support phase, one stays balance as long ZMP is exist inside his/her foot area.

Unlike single support phase, during double phase, ZMP can exist outside of feet area. Vukobratovic call a region where ZMP can exist as “Support Polygon”. Mathematically, support polygon is defined as convex hull of all of feet contact points.

To define convex hull, we must understand concept of convex set first. Convex set is a set such that if one draw a line between two elements in the set. Every point on the line must be an element. And convex hull of set S is minimum convex set that contain set S. An example of support polygon can be shown below:

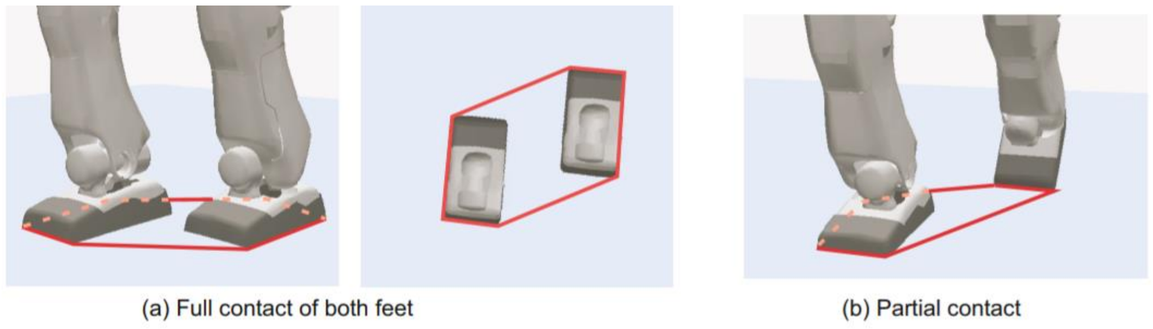


Figure 2: An Example of Support Polygon (Kajita, S., 2016 Page 70)

Generally, there are two ways to derive position of ZMP: using foot sensors or deriving from position and velocity of center of mass of the robot. Using sensor is considered more effective in controlling, but also more expensive as well. Due to constraint in budget, we choose to estimate ZMP from derived equation.

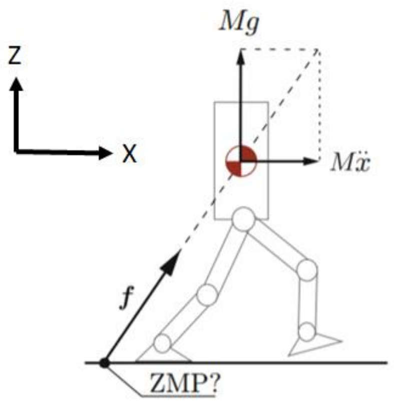


Figure 3: Picture of a planar robot and its coordinate reference (Modified from Kajita, S., 2016 Page 99)

Using coordinate reference as shown above, equation for location of ZMP is shown below: (Kajita, S., 2016 Page 100)

$$P_x = x - \frac{z\ddot{x}}{\ddot{z}+g} \quad - \ (1)$$

Where P_x : Position of ZMP on x-axis

x : Position of CM on x-axis

$\ddot{x}$: Acceleration of CM on x-axis

z : Position of CM on z-axis

$\ddot{z}$: Acceleration of CM on z-axis

g : Gravity acceleration

If acceleration of the robot in both x and z axis is so low that we can neglect, equation (1) can be reduced into:

$$P_x = x \quad - \ (2)$$

Therefore if robot moves with constant speed or low acceleration, ZMP point is estimated to be a projection of center of mass of the robot to the ground

Chapter 3: Robot Mechanical Design

3.1 Design of Robot Constraint Mechanism

The robot we are building need to be constrained in transverse plane. Therefore, there are two spate important part: robot constraint and robot body. As this project mainly focus on generating gait pattern for a robot, we will neglect for most of mechanism design. Instead we will adapt design of already existed bipedal robot with constraint instead of designing one ourselves.

For robot constraint, there are exists many bipedal robots with a constraint in transverse plane: RABBIT and MABLE from Michigan University, Raptor from KAIST etc. Those robots is constrained by attaching to a pole. A rod that connects robot with the pole is directly pass robot main body. The constraint must maintain distance between the robot and the pole but not limit robot movement to moving either vertically or horizontally or rotating around the rod connecting the robot. This type of mechanism requires low effort to build as there required small number of parts. A disadvantage of the pole design is the rod always require to maintain parallel to the ground during support phase; otherwise, both leg is not touch the ground as it supposed to.

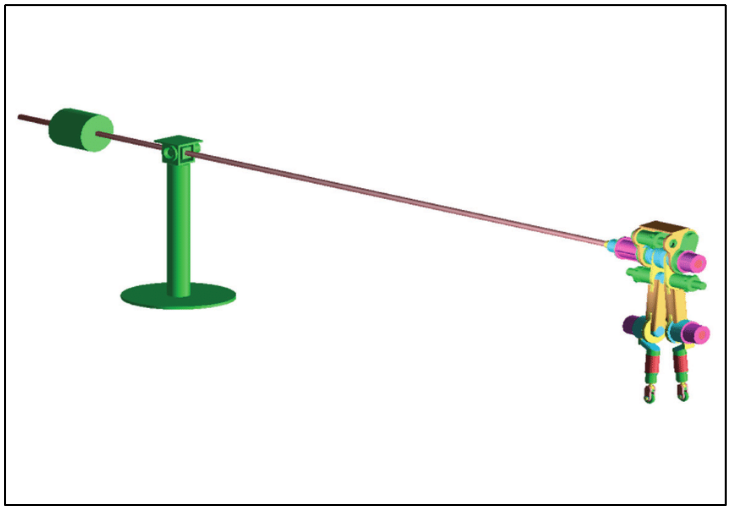


Figure 4: Design of Pole Constraint for RABBIT, a robot from University of Michigan. Retrieved from https://www.researchgate.net/figure/RABBIT-Walking-in-a-Circle-While-Looking-Like-a-Planar-Biped-The-radial-bar-is-attached_fig1_3207484

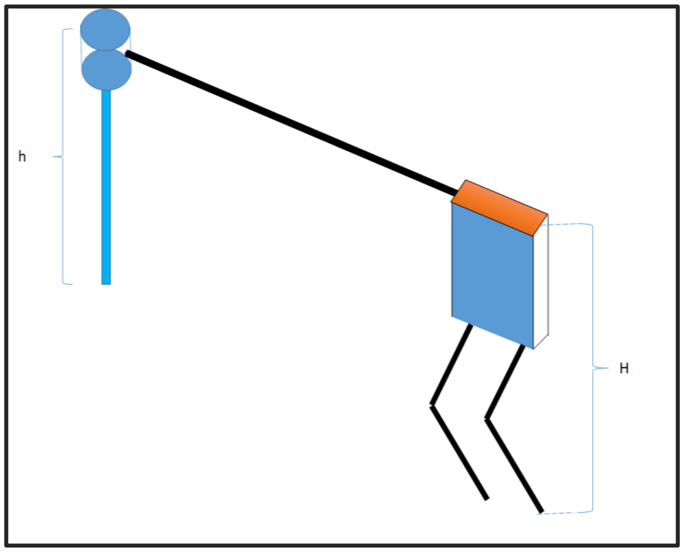


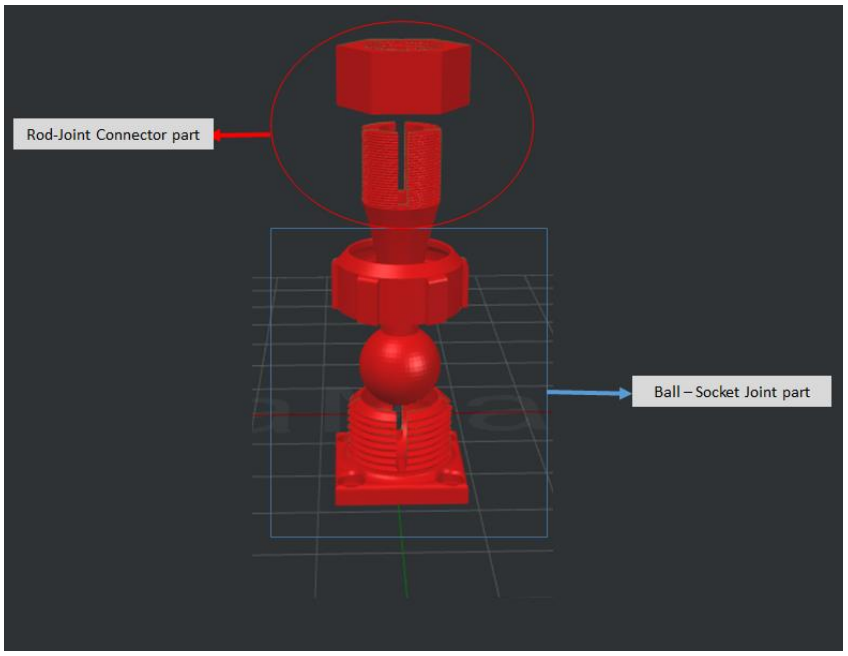
Figure 5: Proposed Solution for an Uneven Hips Height

To maintain parallel between the rod and the ground during double support phase, we propose a solution as following:

1) Both leg must be in configuration which height from the hip to the ground is the same.
2) The height between both tip of the leg (H), must always equal to distance of pole connector from the ground (h)

For pole-root connect design, we use ball-socket joint instead of 3 revolute joint as RABBIT. With constraint robot cannot move around the pole, but restricted to walk in some certain area due to constraint from ball-socket joint. We choose to use this design as it's considered cheaper and less complex to create a joint with 3 DOF rather than 3 joints each has 1 DOF. Moreover, ball-socket joint is proper to be used in our project as our robot cannot move fast due to restriction in power consumption in servos, which will be discussed in later section.

Design of the connector is adapted from the ball and joint by user “Coder- Tronics” from thingiverse.com Coder-Tronics use screw to force four side of the socket to go inward and lock the ball inside. The deeper screw go in the tighter the socket is. The same mechanism also applies to rod-joint connector part as well. The part of the robot-pole connector can be seen below:



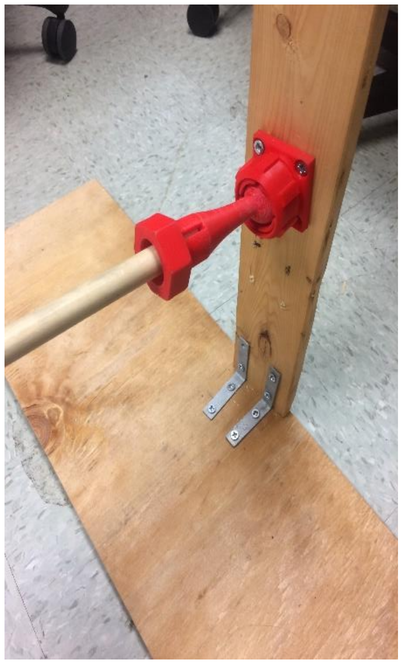


Figure 6: Pole-Robot Connector Design

Please note that distance between ground and the connector is the same as the height of the robot during support phase. Therefore, before mounting the connector to the pole one should generate a walkable gait first before mounting the connector

3.2 Design of Robot Mechanism

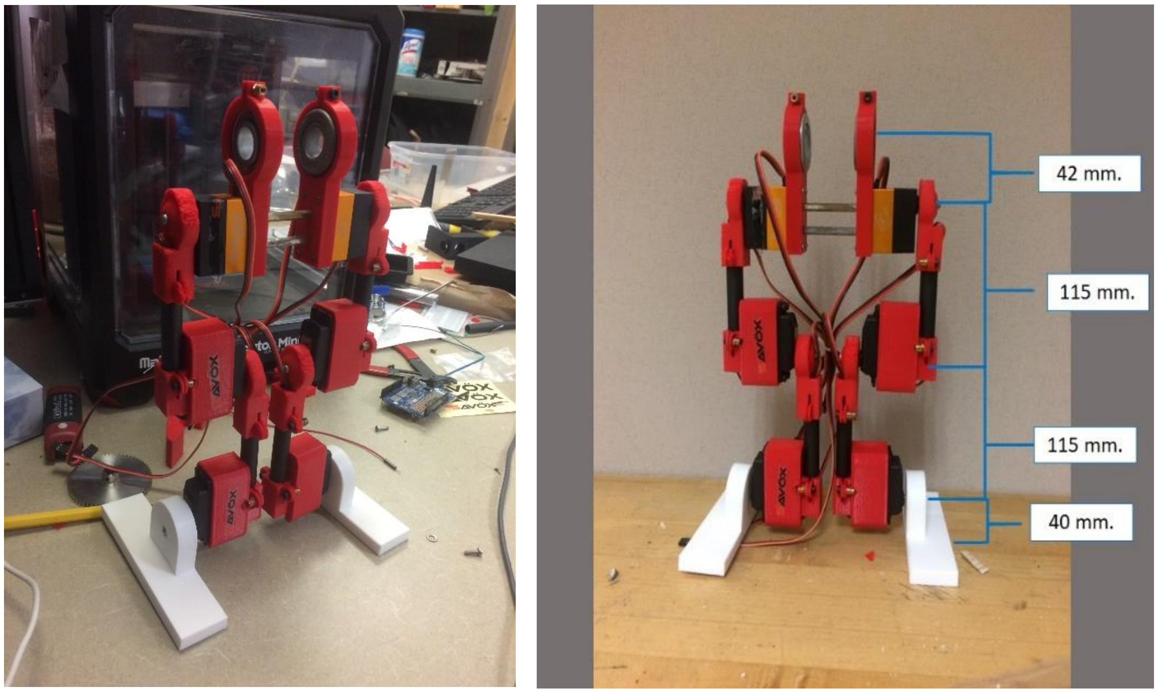


Figure 7: The Complete Robot Body and its overall measurement. The measurement of the robot is measure from each of center of revolute joint to one another except from the foot part which we measure from the center of the revolute joint to the ground

The robot consist of 6 revolute joint, and each joint assigned to hip, knee and ankle of each side of the robot. The height of robot is 312 mm. or 0.312 m. We add gear bearing to connect robot hip with the plywood rod from the pole. With gear bearing, the robot can rotate around the rod more freely.

There are two types of servos in our robot: SAVOX – SV1270TG and Roboard RS1270. We use SAVOX servos for knees and ankles, when Roboard is used only for hips.

For the design of the hips of the robot, we create a loop hole that is large enough to insert gear bearing into it. The hole has an open end which can be tighten using screw. The loop is connected to a box which connects to the servo Roboard. The design is shown below:

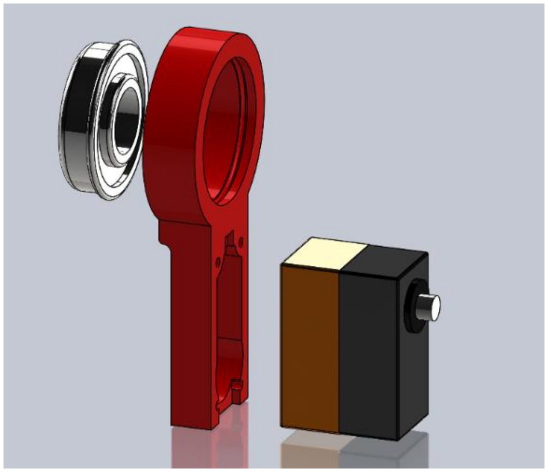


Figure 8: Robot Hip Design. Though do not appear in the picture, the loop has a part with hole which can be tighten with screw in order to lock gear bearing in place

Upper leg and lower leg have the same design: the top of robot leg is a short cylinder which directly connect to gear horn of the servo. The cylinder and servo gear horn is joined together by a screw that goes through the center of the cylinder to the screw hole of the gear horn. To connect a rod with the cylinder, we use a mechanism that require to tighten screw on the end of the mechanism. The rod is a hollow

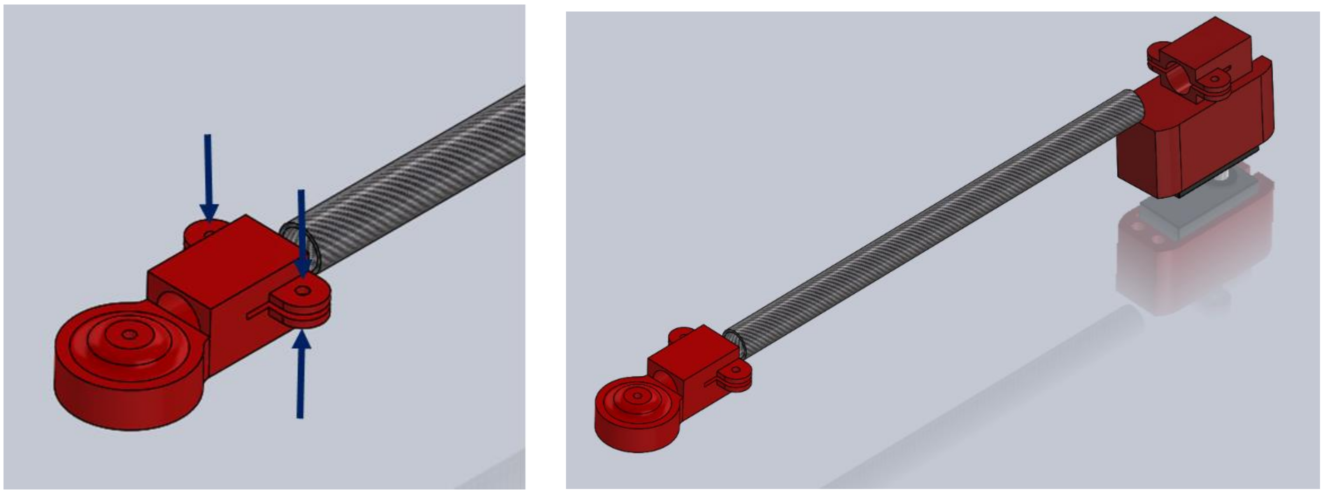


Figure 9: Lower and Upper Leg Design. From left to right, 1) mechanism used to join rod and cylinder shaped box, which connect to gear horn of a servo. Two screw must insert through holes at end of the connector (blue arrows) and tighten in order to lock the mechanism. 2) Overall Design of Lower and Upper Leg

cylinder with length 100 mm and made from carbon fiber. The rod also connect to a servo box at the other end.

For simplicity in generating walking gait, we decide to use flat feet for our robot the feet. The design of the feet is shown below:

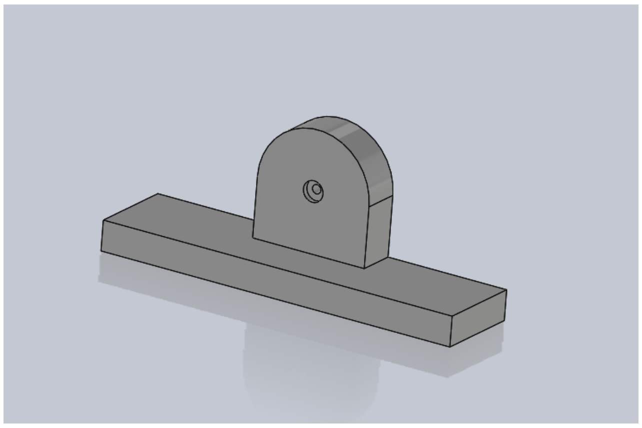
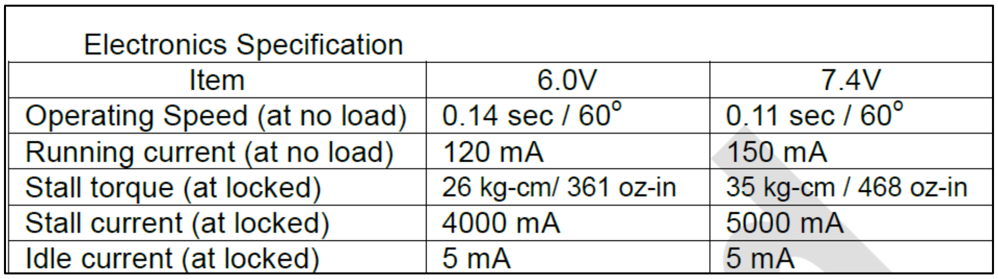


Figure 10: Design of Robot Foot

Chapter 4: Robot Electronics Design

4.1 Servo Specification

Servo we use is SAVOX – SV1270TG and Roboard RS1270. Both servo has the same electronics specification (same stall torque or no load speed at the same voltage). Their specification provided by the manufacturer can be seen as below:



Electronics Specification

Item	6.0V	7.4V
Operating Speed (at no load)	0.14 sec / 60°	0.11 sec / 60°
Running current (at no load)	120 mA	150 mA
Stall torque (at locked)	26 kg-cm/ 361 oz-in	35 kg-cm / 468 oz-in
Stall current (at locked)	4000 mA	5000 mA
Idle current (at locked)	5 mA	5 mA

Table 1: Specification of Servo

According to the specification, the minimum voltage for full efficiency of the servo is at 6V and its stall current is at 4A. Most of available Arduino servo shield cannot handle current greater than 3A. Therefore, we require to assign only one servo per a shield and the servo must be used at lower voltage than 6V. In order to estimate minimum voltage that will be compatible with general servo shield, we estimate stall current by assuming linear relationship between stall current and voltage. At stall current 3A, the estimated voltage is equal to about 4.6V and estimated stall torque is about 17 kg-cm. From this specification, we can acquire specification for our robot as following:

1) The power supply must be provide power at least 4.6V (or above depending on servo shield). And current supply must be at least more than 3x4 = 12 A.
2) Generated gait must not require torque more than 17 kg-cm or about 1.67 Nm

4.2 Power Supply and Servo Shield

Power supply that we use is PHEVOS DC universal regulated switching power supply. This power supply can convert 110V ac to 5V dc with maximum current at 40A which is enough for our usage. The voltage of the power supply can be slightly adjust in range between 4.1 – 5.5V; therefore, we can adjust the supply voltage to be 4.6 V as we intend. Moreover, price of the power supply is considered cheap comparing to general adjustable power supply

Figure 11: PHEVOS Power Supply.

For servo shield, we choose to use servo shield “Adafruit 16-Channel 12-bit PWM/Servo Shield - I2C interface”. The shield can handle current at maximum 10A and voltage at 5V, which more than our specification. One important feature of the shield is that it is stackable, which significantly reduce number of Arduino controller required for the project: Instead of assign one microcontroller per one shield, stackable shield allows us to use one microcontroller for multiple shield. Although maximum current the shield can handle is far greater than our required current, we decide to use one shield for one servo to prolong lifetime usage of the shield.

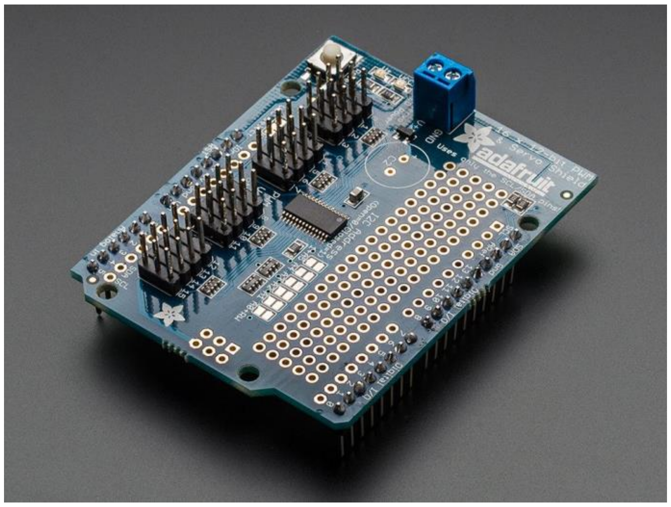


Figure 12: Adafruit 16-Channel 12-bit PWM/Servo Shield - I2C interface module

Chapter 5: MATLAB/Simulink Simulation

5.1 Robot Modeling

In order to estimate torque required to move robot along a gait, we require a simulation. We choose to use MATLAB/Simulink to create our simulation. In over all, we create a model and assemble them in Solidwork and import to Simulink using Simscape-Solidwork plugin. After we have imported the model, we require to simulate interaction between robot and its environment.

To simplify modeling process and optimize computational performance, we do not consider environment interaction with every part of the robot. Except for a point at the center of front and back edge of each foot.

Moreover, in our simulation, the only environment interaction that we simulate is only for ground force reaction which consists of normal force and friction.

5.2 Normal Force Modeling

To model Normal Force, we use damp-spring model. We estimate spring-stiffness to be about 2e5 N/m and damping coefficient to be 2e5 N*s/m. The algorithm of the ground reaction can be described as following:

Let	“ini_pos”	represent position of the ground
	“pos”	represents position of the object
	‘’v”	represent velocity of the object
	‘’f’	represent ground reaction force

If position of the object is more than position of the ground then there’s no reaction force. Otherwise, the normal force is equal to following equation:

$$f = -(2e4\,\frac{N}{m}) * (pos - ini\_pos) - (2e4\,\frac{N*s}{m}) * v$$

MATLAB code for normal force can be seen as below:

```
function f = ground_react(pos,v)

%In this case, we assume ini_pos to be at 0
ini_pos = 0;

if pos>=ini_pos
            f = 0;
else
            f = -2*10^4*(pos-ini_pos)-2*10^4*v;
end
```

Model of normal force can be modeling in Simulink as following block diagram:

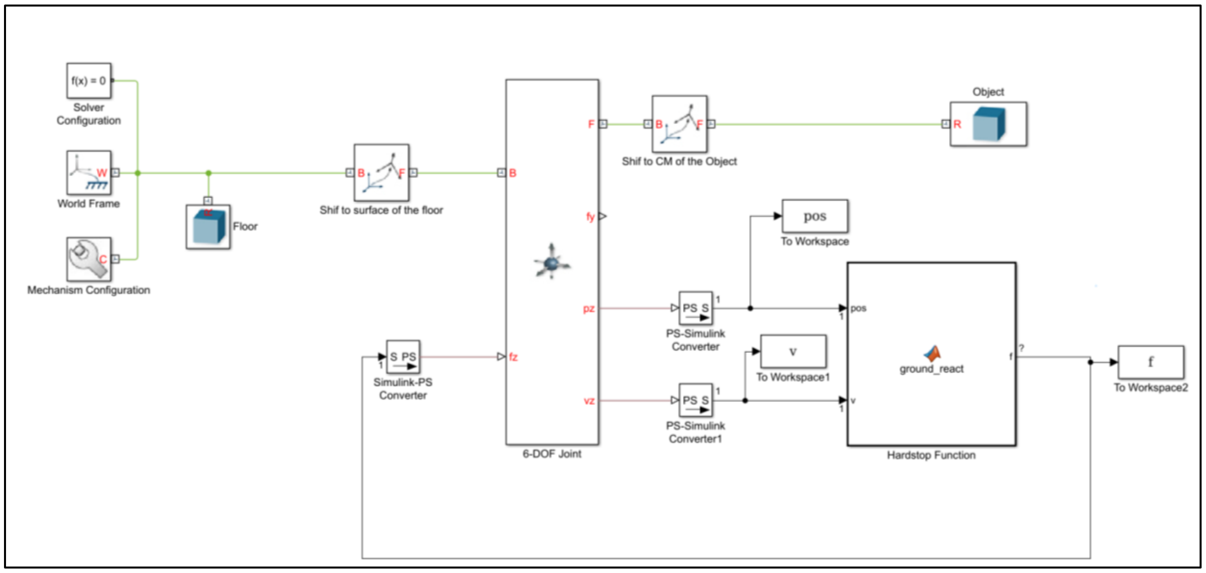


Figure 13: Simulink Block Diagram to Test Ground Reaction Force

The object is a cylinder with radius 1 m. and length 2 m. Mass of the object is 10 kg. To test our model, we have performed 3 experiments to test the simulation. Every experiments is performed from 0 to 10 s. The solver is auto-chosen solver with variable-step. The relative tolerance is 1e-6. Those experiment procedure and result can be seen as below:

a) We set initial position of the object to be at 0 m. We expect behavior of each variable as following:

   i) “f” is constant and close to 98 N.

   ii) “pos” is constant and close to -98/(2e5) = -4.9e-4 m.

   iii) “v” is constant and close to 0 m/s

The result after we run simulation can be shown as below:

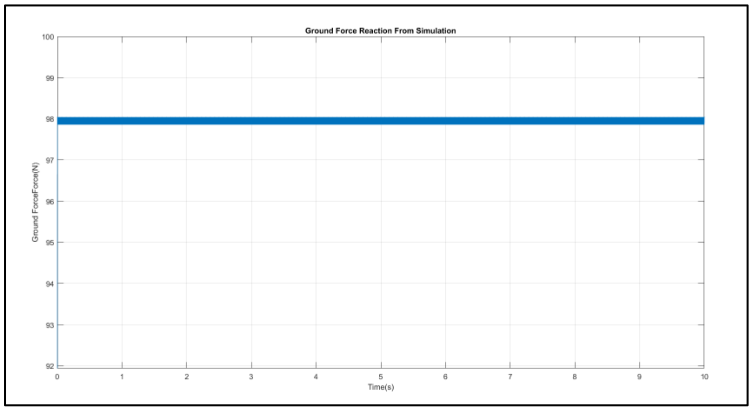


Figure 14: Ground Reaction Force from Simulation of Experiment 1

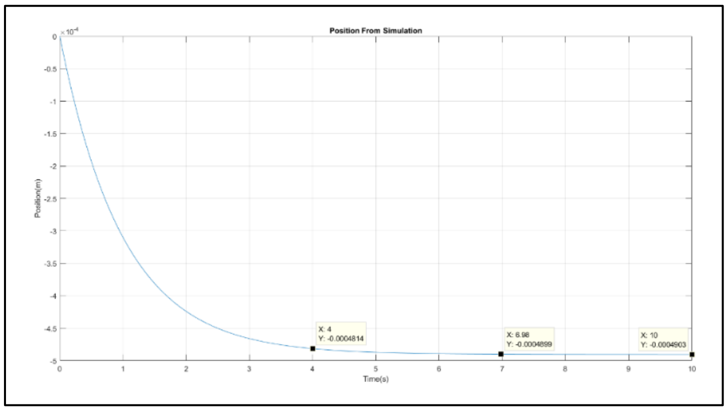


Figure 15: Position of the Object from Simulation of Experiment 1

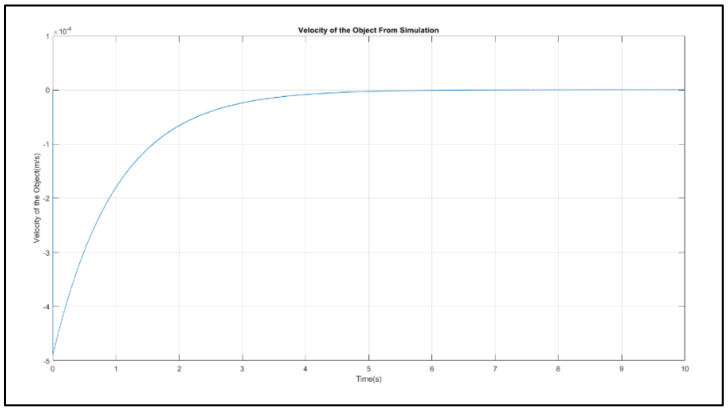


Figure 16: Velocity of the Object from Simulation of Experiment 1

The simulation provides good result which is close to what we have expected.

b) We set initial position of the object to be at 10 m. We expect behavior of each variable as following:

i) The object must reach the ground at $\sqrt{2 * \frac{10}{9.8}}$ = 1.43 s and after that, its position must be close to -98/ (2e5) = -4.9e-4 m.

ii) Velocity of the object must start with 0 m/s and reach its maximum at 1.43 s. The maximum velocity is 1.43*-9.8 = -14 m/s. After that, the velocity must be close to 0 m/s

iii) The Ground Reaction Force must be 0 N until the object reach the ground.

The result after we run simulation can be shown as below:

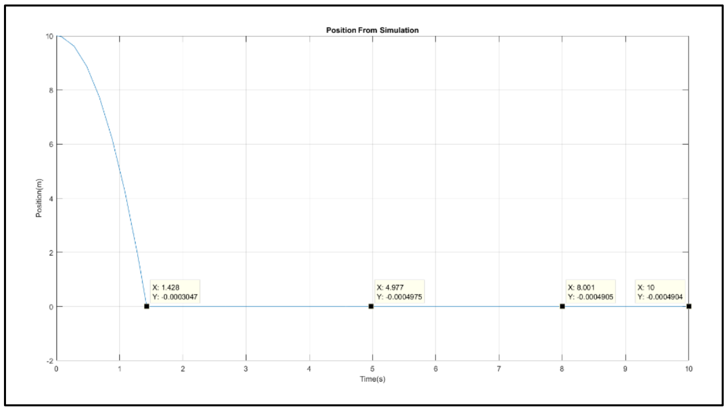


Figure 17: Position of the Object from Simulation of Experiment 2

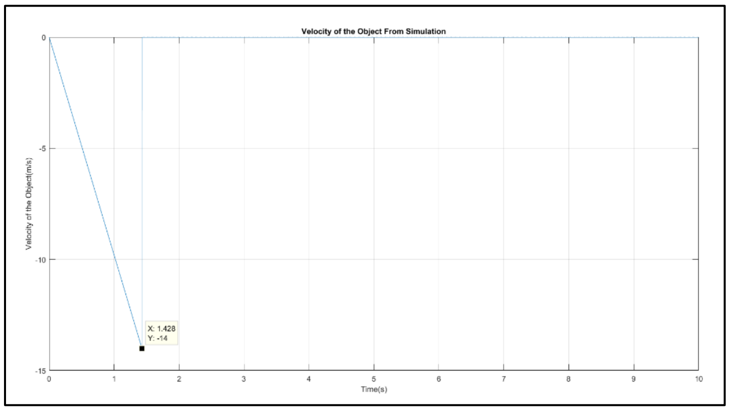


Figure 18: Velocity of the Object from Simulation of Experiment 2

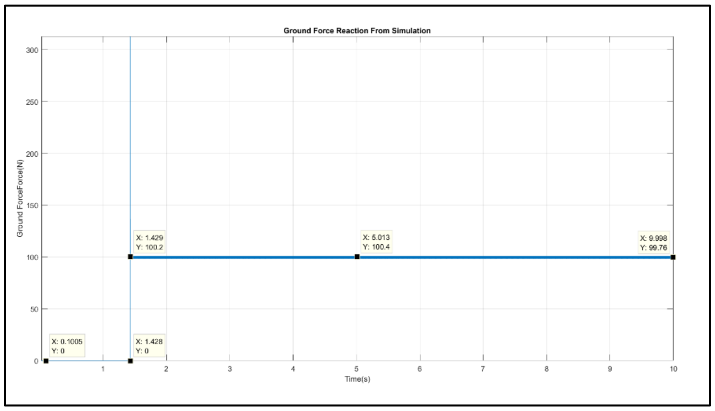


Figure 19: Ground Reaction Force from Simulation of Experiment 2

The result is according to expectation; therefore, the experiment 2 is success. Please note that with lower relative tolerance, the result can be vary.

c) The new experiment will be involved with various compress force on the object. The force will be according to following equation:

$$cf = \begin{cases} 0 & if \quad t > 5 \\ -2 * t & if \quad t \leq 5 \end{cases}$$

For	cf	represent	compressive force on the object
	t	represent	time

MATLAB Function that describes the compressive force can be shown as below:

```
function cf = compressive_force(t)

    if t>=5
                cf = 0;
    else
                cf = -2*t;
    end
end
```

The modified block diagram can be seen as shown below:

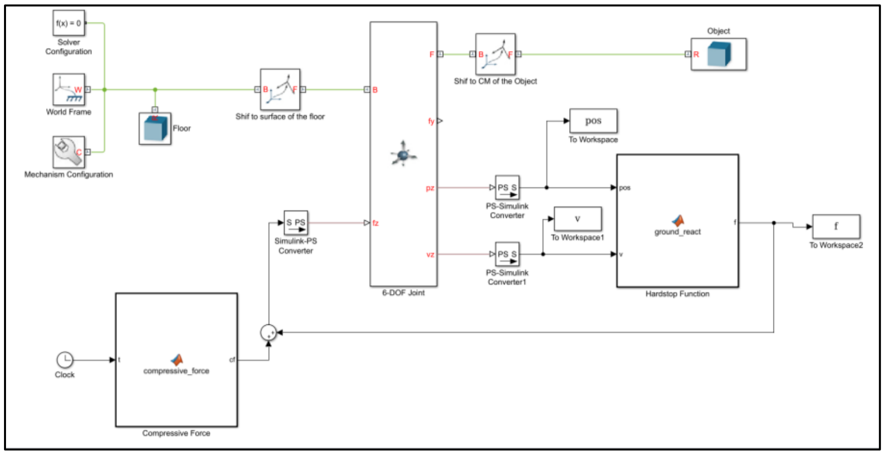


Figure 20: The Modified Simulink Block Diagram with Compressive Force

We expect behavior of each variable as following:

i) Position of the object must be close to 0 m. And velocity of the object is close to 0 m/s

ii) The ground reaction force behaves close to equation below:

$$cf = \begin{cases} 98 & if \quad t > 5 \\ 98 + 2 * t & if \quad t \leq 5 \end{cases}$$

The result after we run simulation can be shown as below:

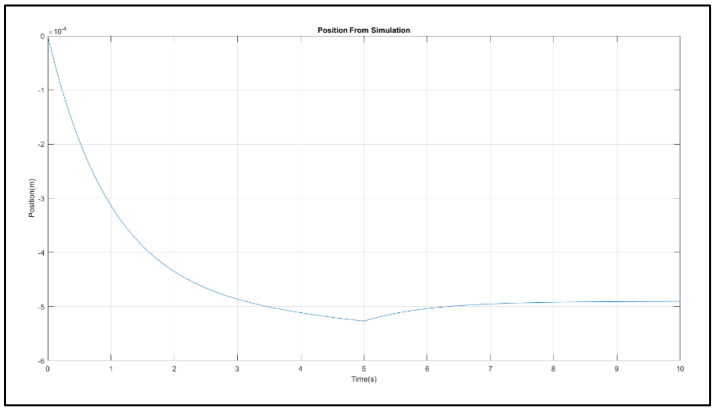


Figure 21: Position of the Object from Simulation of Experiment 3

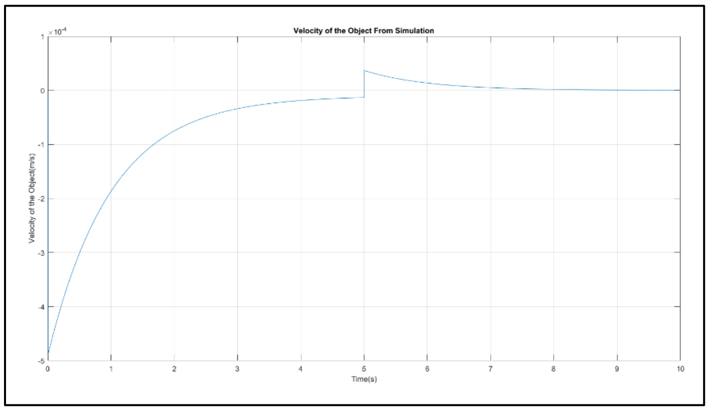


Figure 22: Velocity of the Object from Simulation of Experiment 3

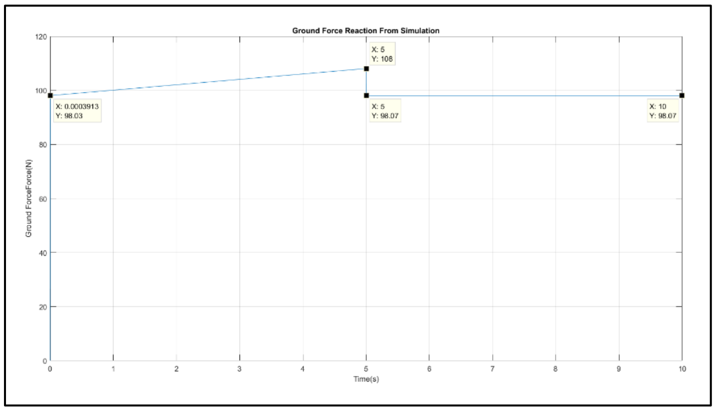


Figure 23: Ground Reaction Force of the Object from Simulation of Experiment 3

The result is according to expectation.

5.3 Friction Modeling

Friction is a force that resist object movement. Maximum magnitude of friction depend on ground reaction force. The ratio between maximum friction and ground reaction force is called coefficient of friction. Behavior of friction can be described as shown below:

a) In case that object is stationary, If there's any force f acting on the object and its magnitude is less than maximum friction force, the magnitude of friction always equal to f but act in opposite direction. Otherwise, magnitude of the friction is equal to maximum friction force.
b) If the object is moving, friction is acted on the opposite direction that the object move to, and its magnitude is equal to maximum friction

To start modeling friction, we consider the case such that **the object is stationary, and actuator force is less than the maximum friction**. One way we can model it is that we use force sensor to detect the actuator force directly feed forward the detected force into the system as friction force. The diagram and its result can be seen as below:

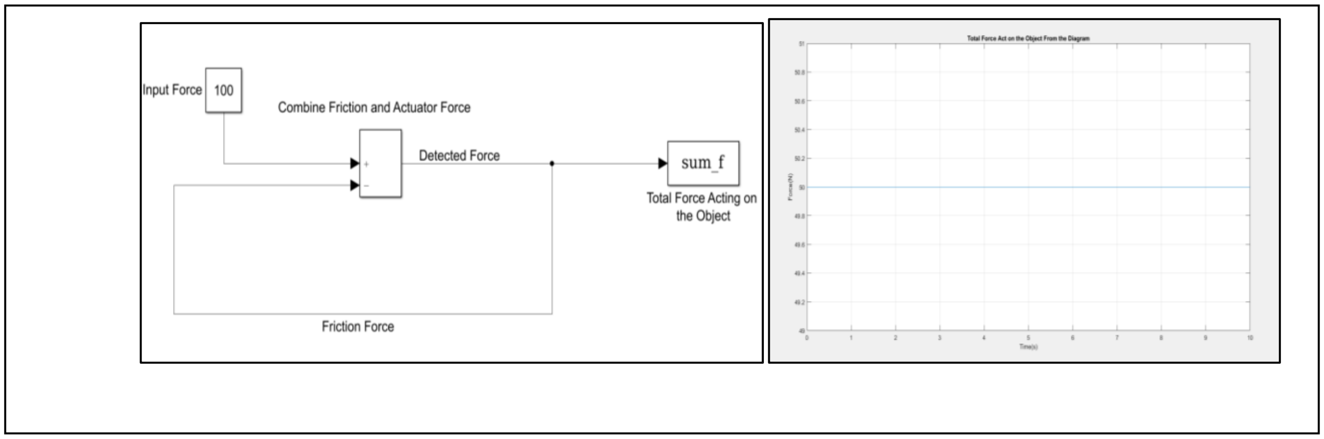


Figure 24: Simplify version of friction simulation. From left to right,1) Diagram of the Friction model which directly uses detected force to directly feedback as friction 2)Total Force as a result from the left diagram

Although algorithm seem to be reasonable, there's a mistake in our algorithm. Instead of staying at the same place, the object will move with the half of the actuator force.

To solve the problem, we introduce concept of quasi-locking. According to S. Lunzman etc, quasi-locking approximate locking with large viscosity and no actual locking. The quasi-locking method is a very robust method as it consume less time to operate a simulation, but its drawback is that it is not appropriate if the actuator force is large and constantly exert to the object. If using quasi-locking while object is exerted with large and constant force such as gravity, the object will drift away.

As we test our mode, the velocity of the object when it's drifting depend on viscosity coefficient: if the coefficient is large enough, the velocity can be estimated to be zero. An example of quasi-locking friction model in case that actuator force is less than maximum friction can be seen as below:

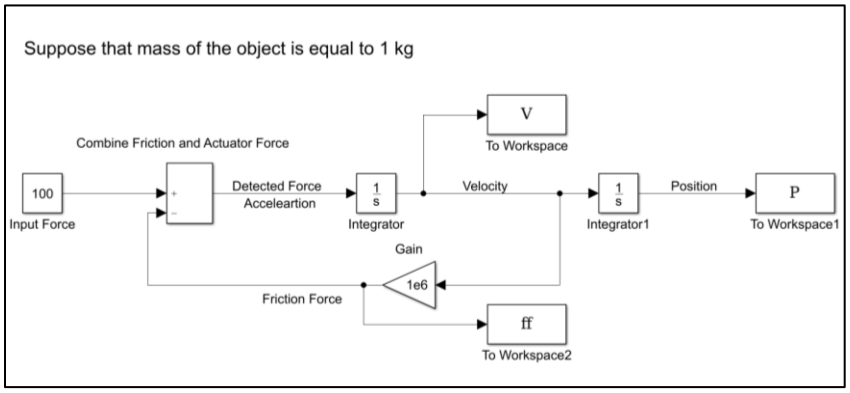


Figure 25: Simulink Diagram of quasi-locking friction in case that the maximum friction is more than exerted force. In the diagram, force 100 N is exerted to the object over time

The result of the simulation form diagram above can be seen as following:

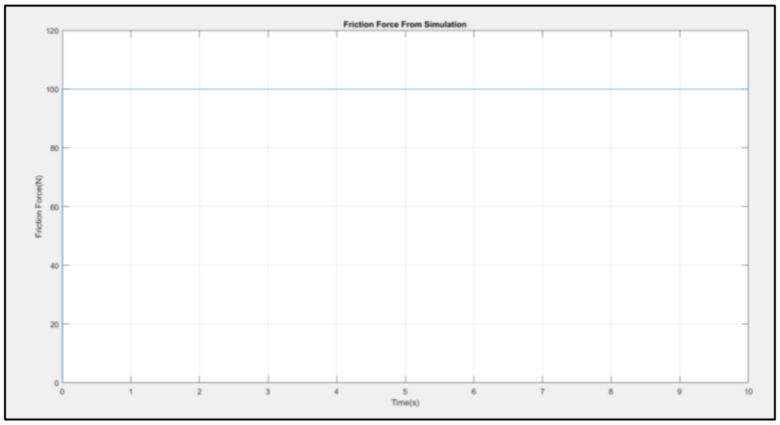


Figure 26: Graph of friction generated from the Simulation

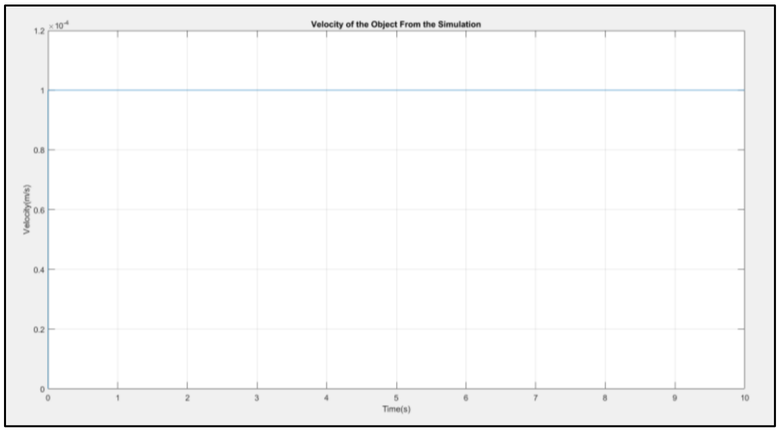


Figure 27: Graph of Velocity of the Object generated from the Simulation

From the graph above, friction generated by the simulation is equal to force exerted on the object over time. And velocity is not equal but can be approximated to be zero. Therefore, quasi-locking method can be used in our friction model.

We use quasi-locking in case of the exerted force is less than the maximum friction, there are two more case require to complete the friction model: when the exerted force is more than the maximum friction and when the object is moving.

We model the following simulation in a planar x-z. The gravity is acting along z-axis, while force exerted on the object have direction along x-axis. We write algorithm for friction model as shown below:

Let	“fz”	represent Normal force
	“fx”	represents Force acting on x-axis
	“vx”	represents Velocity of the object along x-axis
	‘’max_ff”	represent Maximum Friction Force
	‘’ff”	represent Friction Force acting on the object
	‘’mu”	represent Friction Coefficient

We suppose that mu equal to 1. If the ground force reaction is more than 0, Maximum friction force is always equal to following equation:

$$\max\_ff = mu * fz$$

Otherwise, the maximum friction is equal to 0 N

If the object is moving, then magnitude of friction is equal to maximum friction and its direction is opposite to what the object is moving to.

$$If\ |v_x| \geq 0, \qquad ff = -\frac{v_x}{|v_x|} * \max\_ff$$

In case that the object is stationary, If there's no force acting on x-axis (fx), then friction force acting on the object (ff) is equal to 0 N. If there's fx and maximum friction force is more than magnitude of fx, then

$$ff = 1e6 * -v_x$$

Otherwise, friction force is equal to maximum friction force but act in opposite direction with fx, which can be described by using quasi-locking, which can be shown as following:

$$ff = -\frac{f_x}{|f_x|} * \max\_ff$$

MATLAB code for the friction model can be seen as shown below:

```
function ff = friction_func(vx,fx,fz)
%We Suppose that friction coefficient is equal to 1
(1)  mu = 1;
 %Maximum Friction is equal to ground force reaction multiply by friction coefficient
(2)  if fz<=0
(3)        max_ff = 0;
(4)  else
(5)        max_ff = mu*fz;
(6)  end
                    %In case that the object is moving, the friction is equal to maximum friction
          (7)       If abs(vx)>0
          (8)             ff = -vx/abs(vx) * max_ff;
                    %If the object is stationary
          (9)        else
                          %If the exerted force is zero, then friction is equal to zero
          (10)            if fx ==0
          (11)                   ff = 0;
```

```
                                        % If the exerted force is less than the maximum friction, use quasi-
locking
                (12)                     elseif abs(fx)< max_ff
                (13)                                ff = -1e4 *vx;
                                        % If the exerted force is more than the maximum friction, friction is
                              equal to the %maximum friction
                (14)                    else
                (15)                                ff = -fx/abs(fx) * max_ff;
                (16)                    end
                (17)            end
        end
```

From the algorithm above, we found that the code can lead to fatal error in case that the object is stationary and the exerted force is less than the maximum friction. The friction that is calculated is equal to infinity due to vx = 0. Therefore, it's required for us to redefine **stationary of the object** in our simulation. We consider that velocity less than 1e-4 m/s can be approximated to be zero. Therefore, in our simulation, the object is considered to be stationary if its magnitude of velocity is less than 1e-4 m/s. Therefore, we rewrite the line (7) and (8) in our MATLAB code as following:

```
(7)     If abs(vx)>1e-4
(8)             ff = -vx/abs(vx) * max_ff;
```

Although we redefined the stationary pf the object, the problem still occur if the ground reaction force is not stable. If we let the ground reaction force to keep increasing over time, while velocity of the object is around 1e-4 m/s, the friction force will become unstable. It's because when the exerted force is less than maximum frication force, the friction force is not continuous during transition of state between non-stationary state and stationary state. An example of the error described above can be seen below:

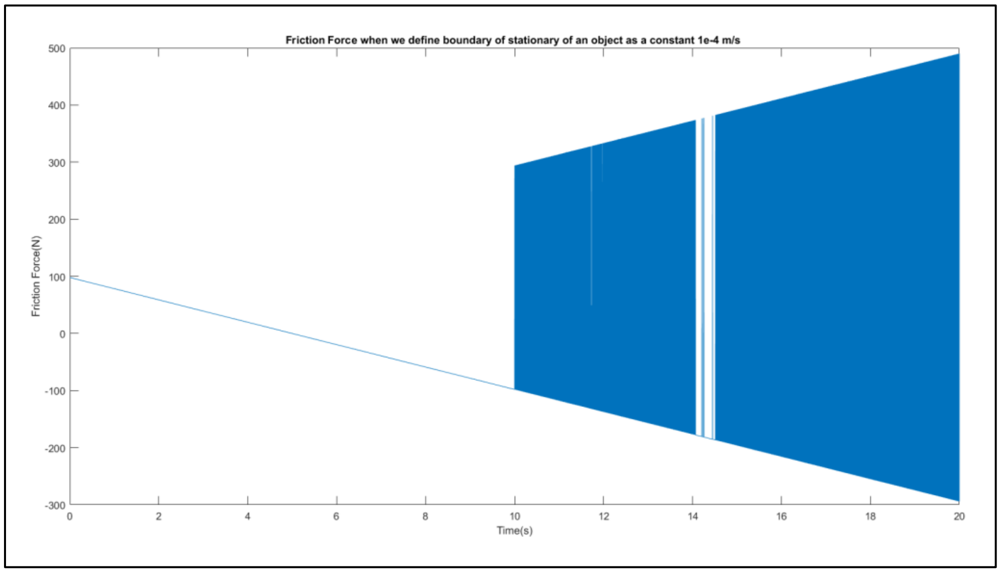


Figure 28: Failed Friction Code. The Friction Force when the boundary of stationary is set as a constant 1e-4. The Graph above is conducted by setting exerted force at 98N, while ground reaction force keep increasing by 19.6N/s (Reach 98 N at 5 S. and Velocity clo

To solve the problem, we set boundary of the stationary state to be

$$\frac{\max\_ff}{viscocity\ coefficient}$$

Other than problem with stationary definition, there's also exist an error in our algorithm: Simulink cannot solve algebraic loop when the magnitude of the exerted force between max_ff and 2*max_ff.

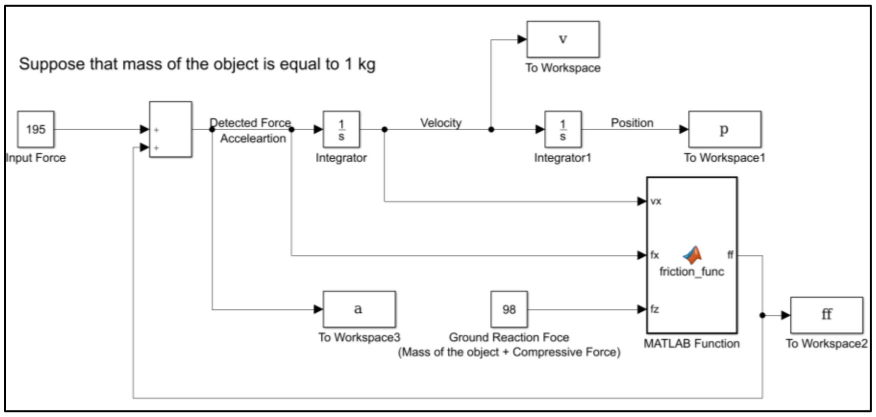
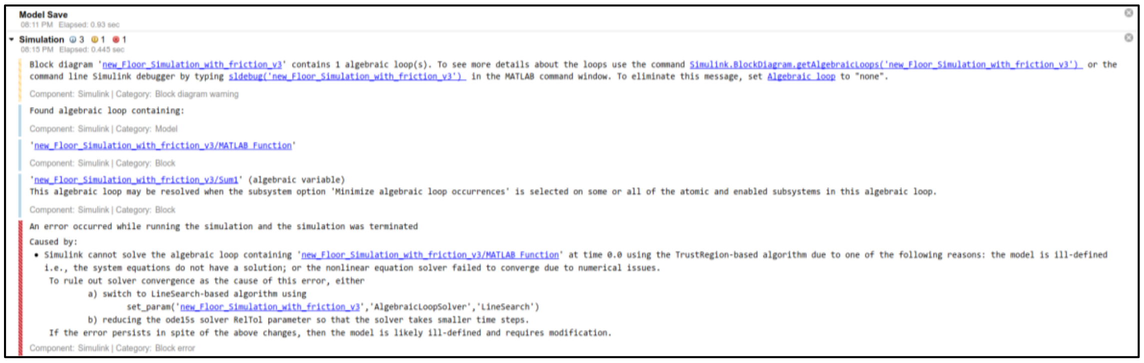


*Figure 29: The Simulation which have algebraic loop error due to using exerted force between max_ff and 2*max_ff*



```
Model Save
08:11 PM  Elapsed: 0.93 sec
Simulation  3  1  1
08:15 PM  Elapsed: 0.445 sec
Block diagram 'new_Floor_Simulation_with_friction_v3' contains 1 algebraic loop(s). To see more details about the loops use the command Simulink.BlockDiagram.getAlgebraicLoops('new_Floor_Simulation_with_friction_v3') or the
command line Simulink debugger by typing sldebug('new_Floor_Simulation_with_friction_v3') in the MATLAB command window. To eliminate this message, set Algebraic loop to "none".
Component: Simulink | Category: Block diagram warning
Found algebraic loop containing:
Component: Simulink | Category: Model
'new_Floor_Simulation_with_friction_v3/MATLAB Function'
Component: Simulink | Category: Block
'new_Floor_Simulation_with_friction_v3/Sum1' (algebraic variable)
This algebraic loop may be resolved when the subsystem option 'Minimize algebraic loop occurrences' is selected on some or all of the atomic and enabled subsystems in this algebraic loop.
Component: Simulink | Category: Block
An error occurred while running the simulation and the simulation was terminated
Caused by:
• Simulink cannot solve the algebraic loop containing 'new_Floor_Simulation_with_friction_v3/MATLAB Function' at time 0.0 using the TrustRegion-based algorithm due to one of the following reasons: the model is ill-defined
  i.e., the system equations do not have a solution; or the nonlinear equation solver failed to converge due to numerical issues.
   To rule out solver convergence as the cause of this error, either
          a) switch to LineSearch-based algorithm using
                  set_param('new_Floor_Simulation_with_friction_v3','AlgebraicLoopSolver','LineSearch')
          b) reducing the ode15s solver RelTol parameter so that the solver takes smaller time steps.
   If the error persists in spite of the above changes, then the model is likely ill-defined and requires modification.
Component: Simulink | Category: Block error
```

Figure 30: Error Message from the Simulation Above

The problem occur because there's no convergence of solution to our model between 2 cases:

a) While stationary, the exerted force more than the maximum friction.

b) While stationary, the exerted force less than the maximum friction.

The problem can be solved by eliminating a case where an object is stationary and exerted force is less than the maximum force. Eliminating the case will not change any in our algorithm because friction exerted from quasi-locking cannot be more than the maximum friction due to velocity at boundary of stationary of the object (v = 1e-4*max_ff)

With all change of algorithm, we could write a new MATLAB code as shown below:

```
function ff = friction_func(vx,fx,fz)
%We Suppose that friction coefficient is equal to 1
(1)  mu = 1;
 %Maximum Friction is equal to ground force reaction multiply by friction coefficient
(2)  if fz<=0
(3)        max_ff = 0;
(4)  else
(5)        max_ff = mu*fz;
(6)  end
                    %In case that the object is moving, the friction is equal to maximum friction
          (7)       If abs(vx)>1e-4*max_ff
          (8)                 ff = -vx/abs(vx) * max_ff;
                    %If the object is stationary
          (9)        else
                              %If the exerted force is zero, then friction is equal to zero
          (10)                if fx ==0
          (11)                          ff = 0;
                              % If the exerted force is not zero, then
          (12)                else
          (13)                           ff = -fx/abs(fx) * max_ff;
```

```
        (14)            end
        (15)        end
end
```

Next, we test our friction model by conduct an experiment on Simulink

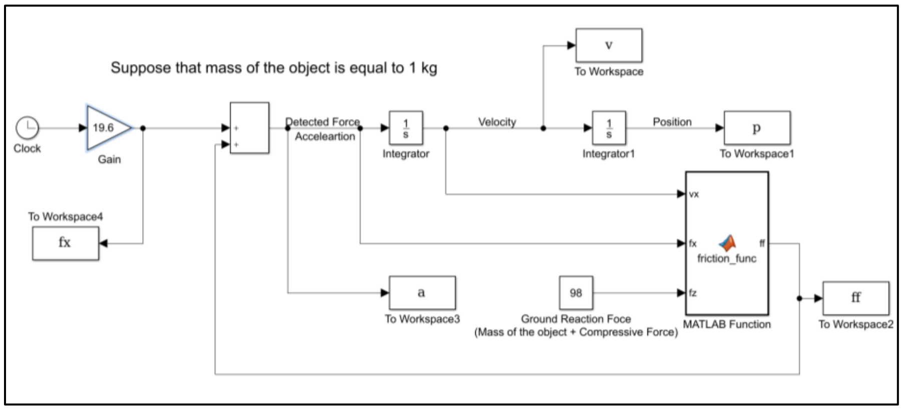


Figure 31: Diagram of Simulation that is used to test friction model

From the diagram above, mass of the object is 1 kg, but there's compressed force on the object, which made the ground force reaction to become 98 N. The exerted force behaves according to equation below

$$fx = 19.6 * t$$

As t represents the time since simulation starts.

We expected result from the Simulation as following:

i) Object should be stationary (Position, velocity and acceleration should always be zero) until at t = 5s, where the exerted force overwhelm the friction. After that Position, velocity and acceleration will behave as following equation:

$$p = \frac{9.8}{3} * (t-5)^2 \text{ m.}$$

$$v = 9.8 * (t-5)^2 \text{ m/s.}$$

$$a = 19.6 * (t-5) \text{ m/s^2}$$

ii) Friction force keeps increasing and equal to the exerted force until t = 5s. After t=5s, friction force is constant at -98N (Inverse direction with the Exerted Force)

iii) By Plotting the graph between exerted force and acceleration, we should get a graph which acceleration is at 0 for any exerted force until the exerted force is more than 98N

The result after we run the simulation can be seen as below:

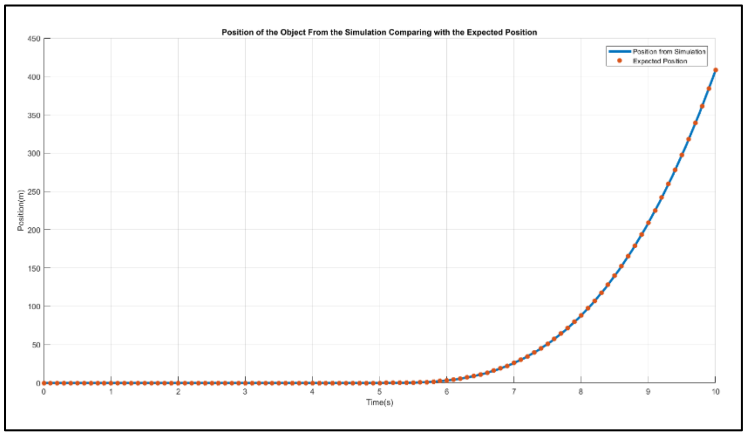


Figure 32: Position of the object from the friction simulation comparing with the expected one

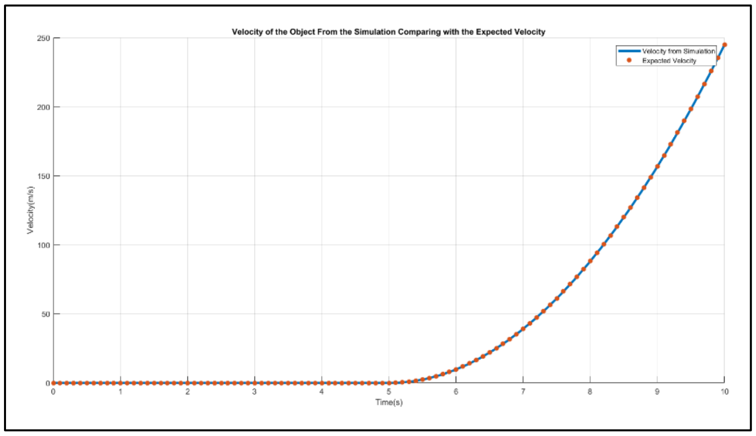


Figure 33: Velocity of the object from the friction simulation comparing with the expected one

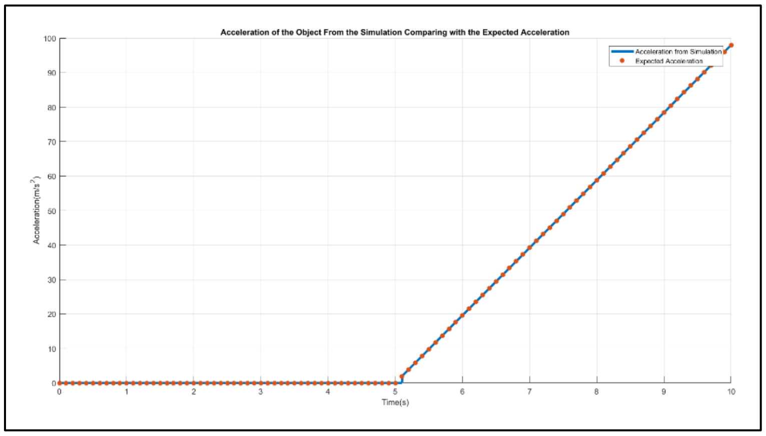


Figure 34: Acceleration of the object from the friction simulation comparing with the expected one

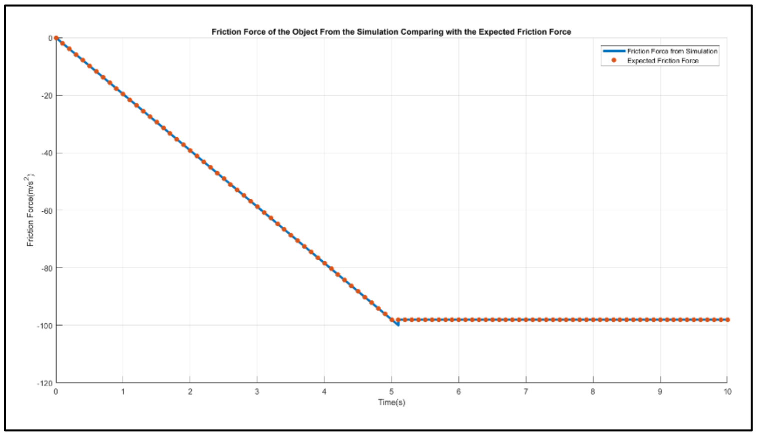


Figure 35: Friction of the object from the friction simulation comparing with the expected one

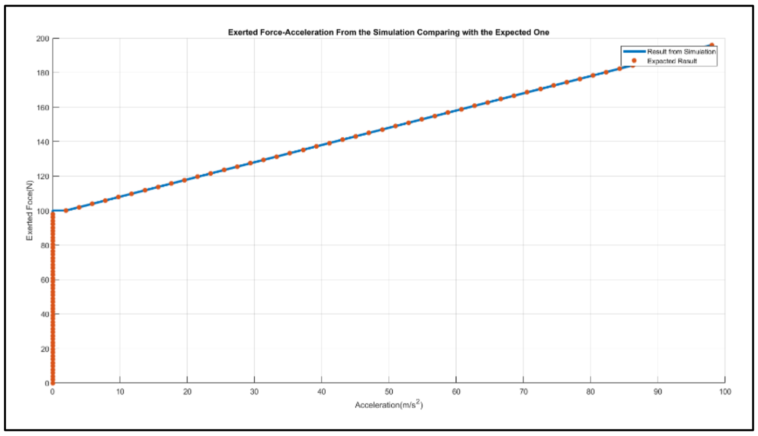


Figure 36: Graph of Exerted Force and Acceleration of the Object

The result from the simulation is close to what we have expected. Although there's some error in acceleration and friction around 5s where the exerted force overwhelm friction, the amount of error is acceptable.

Note: Viscosity Coefficient of quasi-locking can affect time required for computation. For faster computation, we must lower the viscosity coefficient. And vice versa, if more accuracy of simulation is needed.

Chapter 6: Gait design

Throughout previous section, we could summarize gait specification as following:

1) Generated gait must not require torque more than 17 kg-cm or about 1.67 Nm
2) During support phase, both leg must be in configuration which height from the hip to the ground is the same.
3) During double support phase, the distance between hip and ground must always equal to distance of pole connector from the ground.
4) In order to be able to estimate ZMP point using equation (2), the speed of each joint must be constant. Also, projection of center of mass of the robot must always be located inside convex hull.

Other than specification above, in order to further simplify gait design process, we decide to make gait which both feet always parallel to ground when body is perpendicular to the ground.

As weight of the robot is mostly locate at robot hip and knee. Therefore, our gait design depends on how we manage to get each of knees and hips to be in or close to convex hull as much as possible. The result is from our objective is shown on the next page:

Phase	Time Usage (s)	Angle Start (Degrees)			Angle End (Degrees)		
		Hip (ϴ_1)	Knee (ϴ_2)	Ankle (ϴ_3)	Hip (ϴ_1)	Knee (ϴ_2)	Ankle (ϴ_3)
1	1	70	50	70	85	80	85
	1	85	80	85	85	80	85
	1	85	80	85	70	50	70
	4	70	50	70	90	60	60
2	1	90	60	60	70	30	45
	1	70	30	45	60	60	90
	1	60	60	90	60	60	90
	4	60	60	90	70	50	70

Table 2: Walking gait of the robot. Gait period is 14 seconds. Each leg starts with different phase.

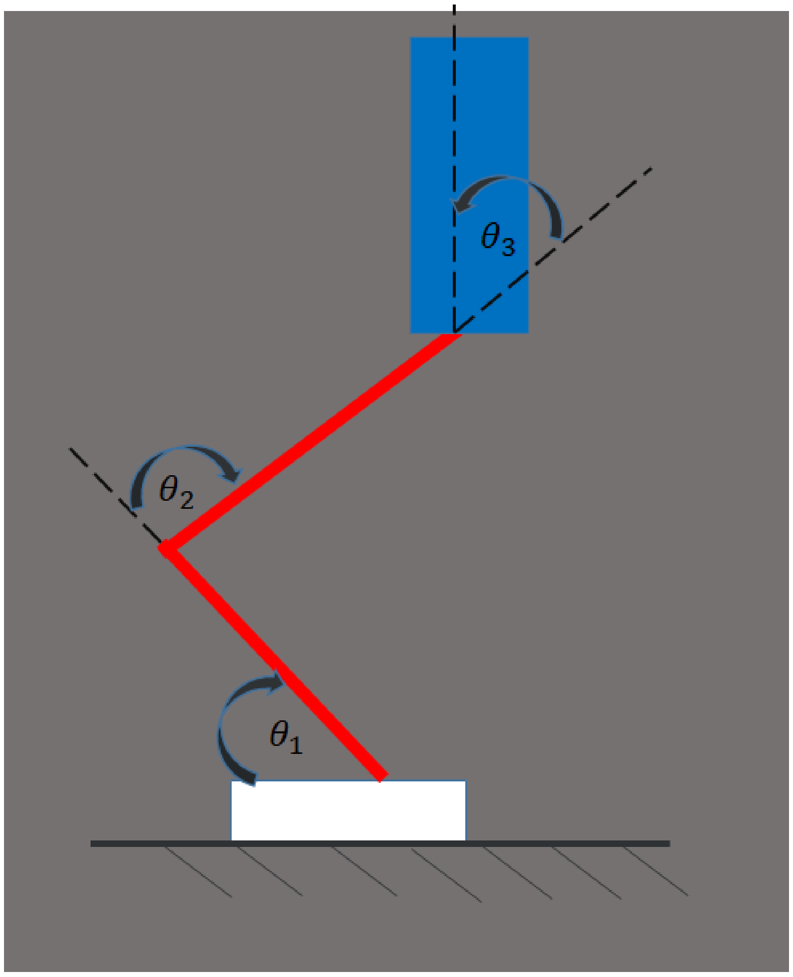


Figure 37: Robot Angle Configuration Walking Gait table

Chapter 7: Experiment and Result

First, we try our gait with simulation first. We run simulation for a period of gait or 14 seconds. During simulation, the robot behaves as we intended: the feet is parallel to ground and body perpendicular to the ground. In addition, the robot is able to perform stable walk for whole simulation. For torque required for moving robot, the maximum torque required for moving along the gait is 1.292 Nm which is lower than the gait specification (1.67 Nm). Graph of torque used when robot walking can be seen below:

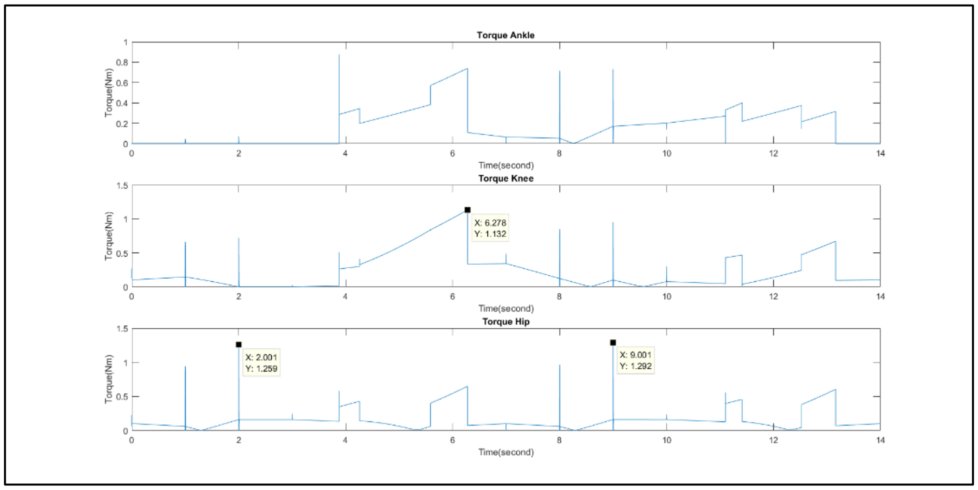


Figure 38: Graph of Torque required for the Robot to Walk

The result of the gait in simulation is satisfied all of the requirement.

Although the gait performance in simulation meets our specification, the performance in the real robot do not. With our experiment gait with the real robot, the robot performs barely stable walk. During single support phase, in some stance, both of robot leg touches the ground due to both hips is uneven. Moreover, the angle of the each joint is not exactly follow the gait. The error in every servo is add up, which result in feet is not parallel to the ground and body is not perpendicular to the ground. The robot is able to walk only because its feet is big enough for ZMP to be inside.

Conclusion

Although we succeed in building a walkable planar robot, the robot performance is not met all of specification: The robot feet are not parallel and the body is not perpendicular to the ground. In addition, during double support phase, both of the feet touches the ground.

To fix the problem, error in servo must be minimized as much as possible: either by trial and error by adding small angle to compensate the error or change for high accuracy servo.

References

[1] Boyd, S., & Vandenberghe, L. (2015). *Convex optimization*. Cambridge: Cambridge University Press.

[2] C. (n.d.). Ball and Socket by Coder-Tronics. Retrieved from https://www.thingiverse.com/thing:2069128

[3] Kajita, S., Hirukawa, H., Harada, K., & Yokoi, K. (2016). *Introduction to Humanoid Robotics*. Berlin: Springer Berlin.

[4] Lunzman, S., Kennedy, C., Kennedy, D., Miller, S., & The MathWorks. (2017, 07 13). *Physical Modeling of Mechanical Friction*. Retrieved from https://www.mathworks.com/tagteam/48847_91574V00_Friction%20Article_MD_HiRes_final.pdf

[5] Popovic, Marko. (2005). Ground Reference Points in Legged Locomotion: Definitions, Biological Trajectories and Control Implications. The International Journal of Robotics Research. 24. 1013-1032. 10.1177/0278364905058363.

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

Appendix A: Arduino Code

```
#include <Wire.h>

#include <Adafruit_PWMServoDriver.h>


//-----------------------------------------------------------------------------------

// Setting Servo

Adafruit_PWMServoDriver lankle = Adafruit_PWMServoDriver(0x40);

Adafruit_PWMServoDriver lknee = Adafruit_PWMServoDriver(0x42);

Adafruit_PWMServoDriver ankle = Adafruit_PWMServoDriver(0x41);

Adafruit_PWMServoDriver knee = Adafruit_PWMServoDriver(0x43);

Adafruit_PWMServoDriver hip = Adafruit_PWMServoDriver(0x44);

#define SERVOMIN  500 // this is the 'minimum' pulse length count (out of 4096) 100Hz -> 250 200Hz-
> 500 300Hz -> 730

// The real servomax is at 610 but I use 600 instead because I want the number to be divisble by 90. Also I
will never go beyond 180

#define SERVOMAX  1960 // this is the 'maximum' pulse length count (out of 4096) 100Hz -> 900
200Hz-> 1960 300Hz-> 2900


//-----------------------------------------------------------------------------------

// Setting error in each joint

// Tuning Result

// Note: leftt leg is the one with sticker


// Angle that make the part pararell to each other

// Right Ankle: 90

// Right Knee: 86

// Right Hip: 90


// Left Ankle: 99

// Left Knee: 90

// Left Hip: 100
```

```
//--------------------------------------------------------------------------------

// Start
int pulselen = 0;
int i = 0;

float rhip0, rknee0, rankle0, lhip0, lknee0, lankle0;
float rhip1, rknee1, rankle1, lhip1, lknee1, lankle1;
int t;



// Setup Servo

void setup() {
  Serial.begin(9600);

  hip.begin();
  knee.begin();
  ankle.begin();



  hip.setPWMFreq(200);
  knee.setPWMFreq(200);
  ankle.setPWMFreq(200);

  lknee.setPWMFreq(200);
  lankle.setPWMFreq(200);

// Set Up Initial Position
drive_servo(70,50,70,90,60,60,70,50,70,90,60,60,1000);
}
//------------------------------------------------------------------------------
void loop()
{
// Phase1 (1s): Right[70,50,70]->[80,70,80] Left[90,120,120]->[105,140,125]
drive_servo(70,50,70,90,60,60,85,80,85,75,30,45,50);
//--------------------------------------------------------------------------------
// Phase2 (1s): Right[80,70,80]->[70,50,70] Left[105,140,125]->[120,120,90]
drive_servo(85,80,85,75,30,45,85,80,85,60,60,90,50);
//------------------------------------------------------------------------------------
//Phase3 (1s)
drive_servo(85,80,85,60,60,90,70,50,70,60,60,90,50);
//---------------------------------------------------------------------------------
 //Phase4 (4s)
drive_servo(70,50,70,60,60,90,90,60,60,70,50,70,200);
//--------------------------------------------------------------------------------------
//Phase5 (1s)
drive_servo(90,60,60,70,50,70,75,30,45,85,80,85,50);
//---------------------------------------------------------------------------------
// Phase 6 (1s)
drive_servo(75,30,45,85,80,85,60,60,90,85,80,85,50);
//---------------------------------------------------------------------------------
// Phase 7 (1s)
drive_servo(60,60,90,85,80,85,60,60,90,70,50,70,50);
//---------------------------------------------------------------------------------
// Phase 8 (4s)
drive_servo(60,60,90,70,50,70,70,50,70,90,60,60,200);
//---------------------------------------------------------------------------------
}
```

```
long int convert(float x, float in_min, float in_max, float out_min, float out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}


void drive_servo(float rhip0,float rknee0,float rankle0,float lhip0,float lknee0,float lankle0,float
rhip1,float rknee1,float rankle1,float lhip1,float lknee1,float lankle1,int t)
{
   for(int i=0;i<=t;i++)
 {
    pulselen = convert(rhip0+(rhip1-rhip0)*i/t,0,220,SERVOMIN,SERVOMAX);
    hip.setPWM(14, 0, pulselen);
    pulselen = convert(rknee0+(rknee1-rknee0)*i/t-5,0,220,SERVOMIN,SERVOMAX);
    knee.setPWM(0, 0, pulselen);
    pulselen = convert(rankle0+(rankle1-rankle0)*i/t,0,220,SERVOMIN,SERVOMAX);
    ankle.setPWM(14,0, pulselen);


    pulselen = convert(180 -(lhip0+(lhip1-lhip0)*i/t),0,220,SERVOMIN,SERVOMAX);
    hip.setPWM(15, 0, pulselen);
     pulselen = convert(180 -(lknee0+(lknee1-lknee0)*i/t),0,220,SERVOMIN,SERVOMAX);
    knee.setPWM(1, 0, pulselen);
   pulselen = convert(180 -(lankle0+(lankle1-lankle0)*i/t),0,220,SERVOMIN,SERVOMAX);
    ankle.setPWM(15, 0, pulselen);
    delay(20);
}
             }
```