Team 2

BME – 673 Bio Robotics

3 – DOF Robotic leg: COM trajectory planning and control

Nikhil Lokesh Chitty

NJIT ID: 31484273

Objectives:

The objectives of the project are classified into the following sub-objectives:

a. Kinematic and Dynamic modelling of 3 DOF robotic leg.
b. Forward (pen and paper) and inverse kinematics for COM of the system.
c. Trajectory planning of COM using Simulink and petre corke robotics toolbox.
d. Implementation of planned motion using Arduino Uno, dynamixel shield and dynamixel motors on hardware.

Methods:

a. Kinematic and Dynamic modelling of 3 DOF robotic leg:

Software used: Matlab & Simulink, Simscape multibody toolbox, Simulink contact forces library.

Implementation: The robotic leg setup is simulated by measuring the approximate lengths and masses of each link and joint in Simulink. Using the Simulink multibody toolbox, individual link elements are created. These links are connected by revolute joints available in the toolbox. The Simulink model generated approximates the robotic leg setup shown in figure below.

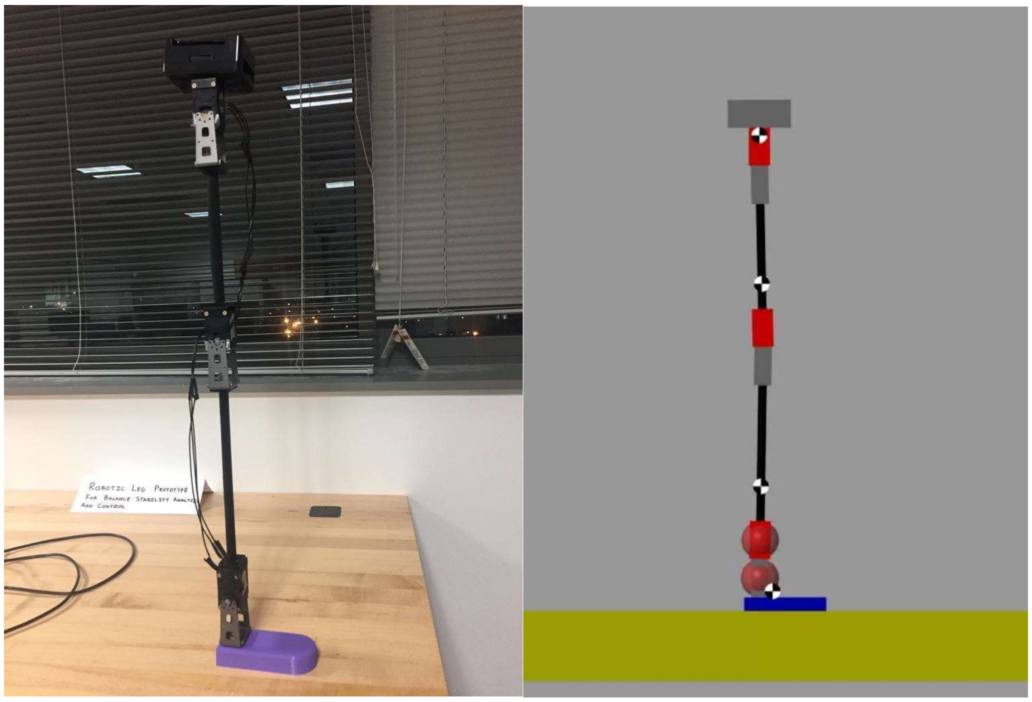


Figure: Robotic leg hardware setup (left) and Robotic leg simulated (right)

The robotic leg setup is placed on a floor (rigidly fixed to world frame) with a 6-DOF joint in between. This means the leg is free to move in any direction in a 3-dimensional space relative to the floor. To stop the robotic leg from penetrating into the floor, contact forces between robotic leg joints and the floor are modelled using Simulink contact forces library. The parameters are of the contact stiffness and contact damping are chosen to mimic the real surface and robotic leg interactions.

The toolbox automatically computes the kinematic and dynamic equations of the system and displays the motion of the setup in a mechanical explorer window.

b. Forward (pen and paper) and inverse kinematics for COM of the system.

The centre of mass of robotic leg is interesting because the leg stays balanced only when the centre of mass of the complete system lies inside a rectangular region in sagittal plane of the leg. The dimensions of the allowed rectangular region are

breadth = foot length and length = COM max height from the floor and the breadth edge of rectangle coincides with the robotic leg foot.

Therefore, tracking the COM of the system and restricting it from moving beyond the safe regions is of critical importance in order to plan and control the trajectory of the system.

To track COM of the system, let us fix a local frame {F} with origin at the centre of back edge of the foot touching the ground, the y-axis along the foot and z-axis perpendicular to floor and towards the ankle.

There are four rigid bodies in the system and each rigid body is made-up of 2 to 3 rigidly fixed bodies of different dimensions and masses.

Parameters and their notations: (lengths are along y-axis and heights are along z-axis)

foot_m = foot mass; foot_l = foot length; foot_h = foot height

support_m = support mass; support_l = support length; support_h = support height;

motor_m = motor mass; motor_l = motor length; motor_h = motor height;

link_m = link mass; link1_l = lower leg length; link2_l = upper leg length;

hip_m = hip mass; hip_l = hip length; hip_h = hip height;

(Foot and support) Rigid body 0:

m0 = foot_m+support_m;   where m0 is rigid body 0 mass

l0 = foot_h+support_h;   where l0 is rigid body 0 total length along z-axis

z0 = (foot_m*(foot_h/2)+support_m*(foot_h+support_h/2))/(support_m+foot_m);

y0 = (foot_m*(foot_l/2)+support_m*(support_l/2))/(foot_m+support_m);

where (y0,z0) are the co-ordinates of COM of rigid body 0 WRT {F}.

(motor, link1, support) Rigid body 1:

m1 = motor_m+link_m+support_m;   body1 mass

l1 = motor_h+link1_l+support_h;

where l1 is body 1 total length along z-axis when q1(joint angle) = 0;

a1 = (motor_h/2*motor_m + (motor_h+link1_l/2)*link_m + (motor_h+link1_l+support_h/2)*support_m)/(motor_m+link_m+support_m);

a1 is length from joint 1 to COM of Rigid body 1(W.R.T joint 1).

z1 = l0 + a1*cos(q1);

y1 = support_l/2 - a1*sin(q1);

where (y1,z1) are the co-ordinates of COM of rigid body 1 WRT {F}.

(motor, link2, support) Rigid body 2:

m2 = motor_m+link_m+support_m;   m2 is body2 mass.

l2 = motor_h+link2_l+support_h;

where l2 is body 2 total length along z-axis when q2(joint angle) = 0;

a2 = (motor_h/2*motor_m + (motor_h+link2_l/2)*link_m + (motor_h+link2_l+support_h/2)*support_m)/(motor_m+link_m+support_m);

Where a2 is length from joint 2 to COM of Rigid body 2(W.R.T joint 2).

z2 = l0 + l1*cos(q1) + a2*cos(q1+q2);

y2 = support_l/2 - l1*sin(q1) - a2*sin(q1+q2);

where (y2,z2) are the co-ordinates of COM of rigid body 2 WRT {F}.

(motor and hip) Rigid body 3:

m3 = motor_m+hip_m;   m3 is Rigid body 3 mass

l3 = motor_h+hip_h;

where l3 is body 3 total length along z-axis when q3(joint angle) = 0;

a3 = (motor_h/2*motor_m+(motor_h+hip_h/2)*hip_m)/(motor_m+hip_m);

Where a3 is length from joint 3 to COM of Rigid body 3(W.R.T joint 3).

z3 = l0 + l1*cos(q1) + l2*cos(q1+q2) + a3*cos(q1+q2+q3);

y3 = support_l/2 - l1*sin(q1) - l2*sin(q1+q2) - a3*sin(q1+q2+q3);

where (y3,z3) are the co-ordinates of COM of rigid body 3 WRT {F}.

COM of complete system:

Therefore, the position of centre of mass of complete system WRT {F} can be computed using the co-ordinates and masses of 4 rigid bodies WRT {F} present in the system as:

Zcom = (m0*z0+m1*z1+m2*z2+m3*z3)/(m0+m1+m2+m3);

Ycom = (m0*y0+m1*y1+m2*y2+m3*y3)/(m0+m1+m2+m3);

Where (Ycom, Zcom) are the co-ordinates of COM of robotic leg WRT {F}.

By computing time derivative of the above COM co-ordinates, we can obtain the COM velocity as follows:

Zdcom = (m1*zd1+m2*zd2+m3*zd3)/(m0+m1+m2+m3);

Ydcom = (m1*yd1+m2*yd2+m3*yd3)/(m0+m1+m2+m3);

Where zd1,zd2,zd3,yd1,yd2,yd3 are time derivatives of z1,z2,z3,y1,y2,y3 respectively.

And (Ydcom, Zdcom) are the COM velocities along y- and z- axis of {F}.

Since all the design parameters are constant, we can compute the position of COM if we know the joint angles and we can compute the COM velocity if we know the joint angles and joint angular velocities. To remove redundant computation of COM position and velocity, MATLAB functions COMPosition(q) and COMVelocity(q,qd) are written. The scripts for these functions are in appendix f.

```
[z,y] = COMPosition(q); % takes joint angles as inputs and returns COM position.
```

```
[zd,yd] = COMVelocity(q,qd); %takes joint angles and angular rates as inputs and return COM Velocity.
```

Trajectory planning of COM:

For the system to move in a stable path, the COM Y co-ordinate must satisfy two inequalities given by:

$$\frac{\dot{y}}{\omega} + y \geq 0$$

$$\frac{\dot{y}}{\omega} + y \leq foot\_l$$

$where$ ω = $\sqrt{g/COM\_Maxheigth}$ and $foot\_l$ = foot length

Using the Simulink model, COM position and Velocity functions, we can actuate the joints with a variety of inputs and plot the COM position and Velocity along with the boundary conditions to check for the stability of the system.

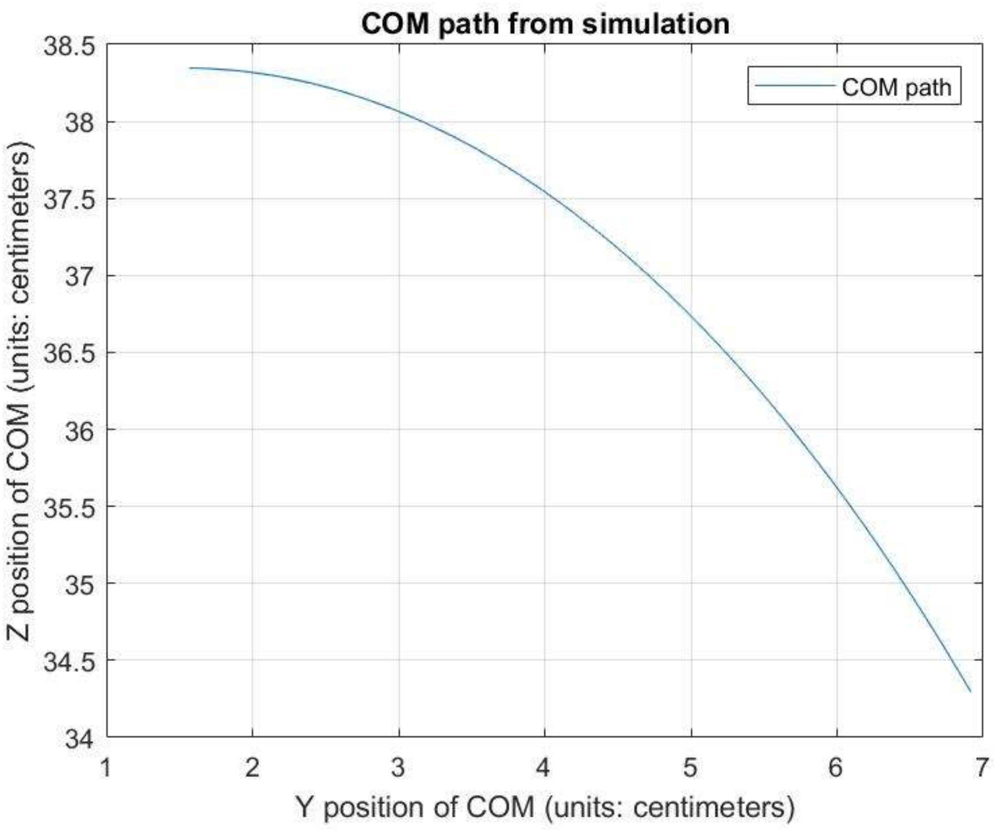
In this project, the input positions of the joints are given as

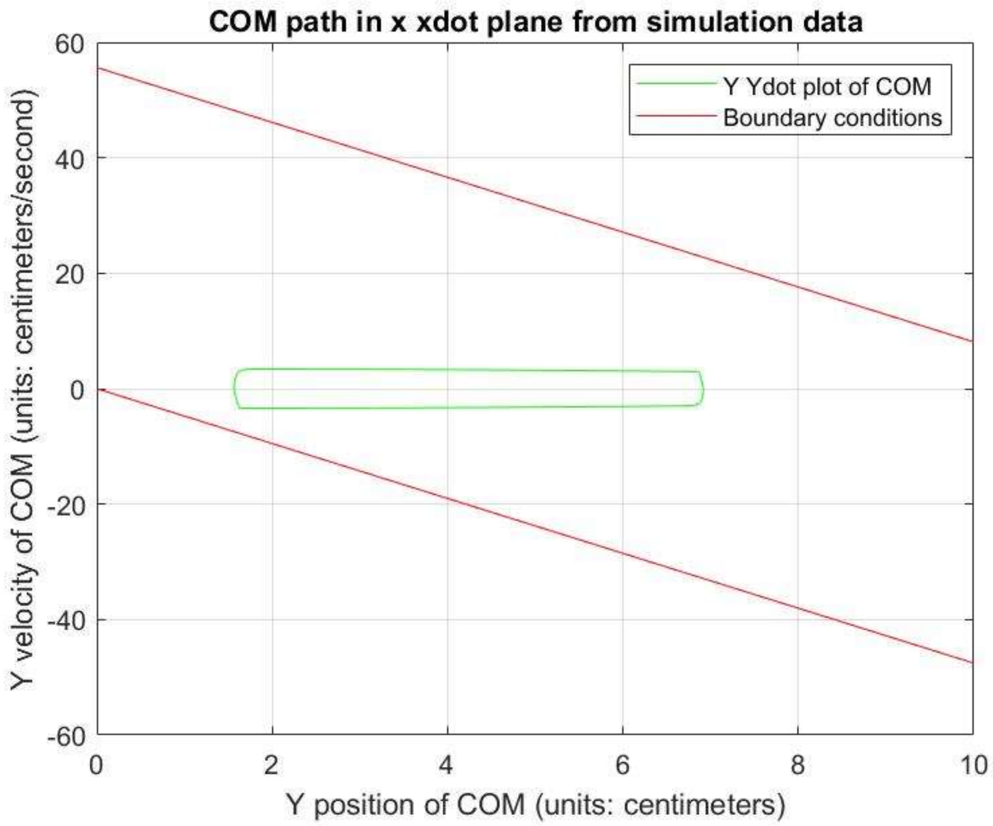
Ankle joint (Joint 1) = -30|sin(wt)| % where w = π/10

Knee joint (Joint 2) = 60|sin(wt)|

Hip joint (Joint 3) = -30|sin(wt)|

With the above input joint angles run the "Robotic_leg.slx" Simulink model, the Simulink model logs the data of joint angles and joint angular velocities into workspace. Run the script "Robotic_leg_simulation_plots.m: to plot the COM path and y $\dot{y}$ plot with boundary conditions as below. (look-up **Appendix a** for detailed steps)


COM path from simulation
COM path
Z position of COM (units: centimeters)
Y position of COM (units: centimeters)
38.5
38
37.5
37
36.5
36
35.5
35
34.5
34
1
2
3
4
5
6
7


COM path in x xdot plane from simulation data
Y Ydot plot of COM
Boundary conditions
Y velocity of COM (units: centimeters/second)
Y position of COM (units: centimeters)
60
40
20
0
-20
-40
-60
0
2
4
6
8
10

From the first figure we can observe that y-coordinate is within the [0,foot_l] and from the second figure, the y $\dot{y}$ plot of COM are within the boundary conditions. Therefore, the system is stable and the same can be seen in the animation from model explorer while running the Simulink model.

c. Forward and Inverse dynamics:

To move from joint space to actuator space, we need to compute the inverse dynamics of the system. However, in the Simulink multibody tool box, the forward and inverse dynamics are automatically computed. Even for the hardware implementation, since we are using dynamixel motors and shield, they are already equipped with an STM32 micro-controller to perform closed loop PID control for each motor. Therefore, we don't need to specifically compute or model the forward and inverse dynamics of the system.

d. Hardware implementation:

Hardware used: Arduino Uno, Arduino Nano, Dynamixel shield, Dynamixel XL430 motors, mechanical supports and 3D printed links.

Software used: Arduino IDE, PuTTY application.

Libraries required: SoftwareSerial, Dynamixel2Arduino.h and DynamixelShield.h

The dynamixel motors are daisy chained to one another and are connected to TTL pins of the dynamixel shield mounted on the Arduino Uno. An Arduino Nano (7,8) pins are connected to Arduino Uno (8,7) pins for serial data acquisition and debug purposes.

After connecting the circuit with necessary connections and connecting the 12V supply to dynamixel shield, turn on the dynamixel shield and set it to UART mode. Now dynamixel is ready to be programmed. Upload the "Sinusoidal_trajectory.ino" sketch to Arduino Uno and upload the "Nano_Debug_serial.ino" sketch to Arduino Nano. After successful upload, set the dynamixel shield to Dynamixel mode. The leg robot starts to move in the prescribed trajectory after opening the Uno serial monitor. To log the data, open PuTTY, set the comm port, baud rate same as that of Nano and a path to save the logged data. opening a PuTTY session starts logging the data in the prescribed path and file type.

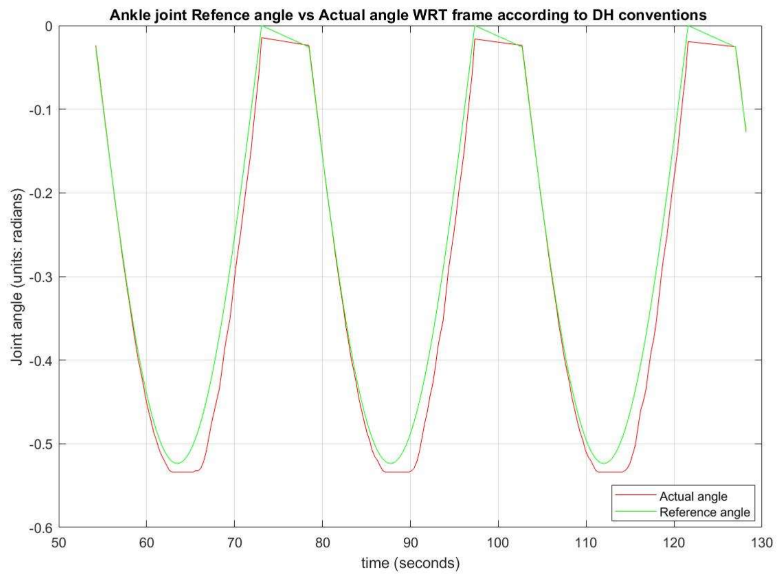
The sketches Sinusoidal_trajectory.ino and Nano_Debug_serial.ino are available in the **Appendix b**.

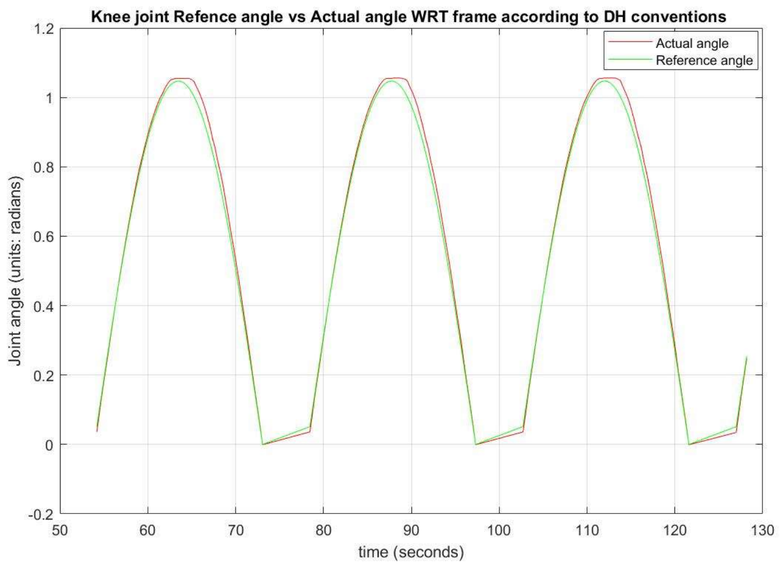
Results and conclusions:

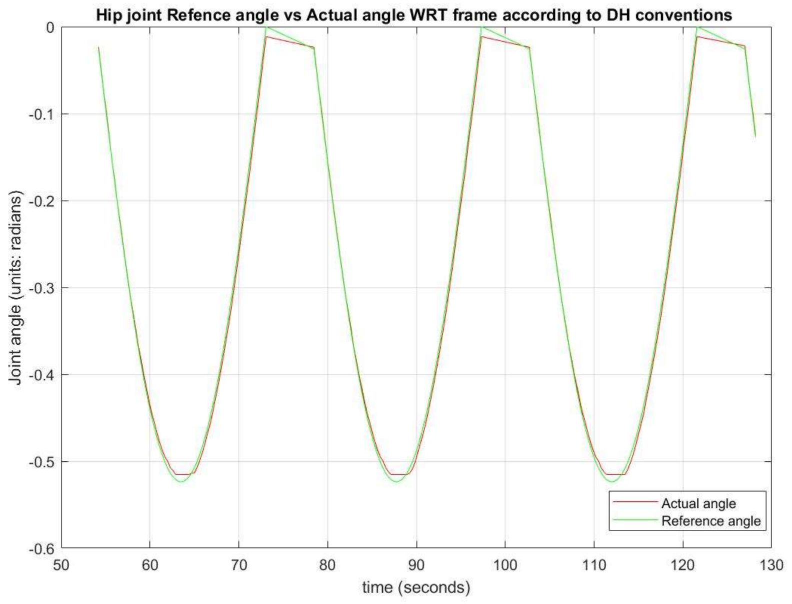
The data is logged into a text file named "Sin_position.txt"

The logged data is read and interpreted using matlab. To visualize the data into following plots, run the matlab script "file_read.m". The script is available in the (**Appendix c).**

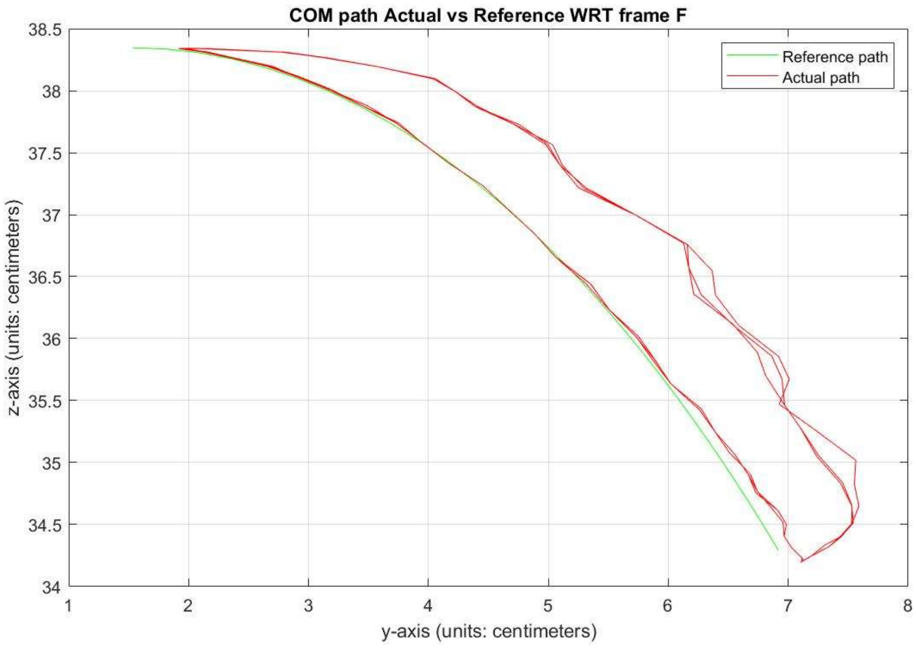
The reference joint angles vs the actual joint angles are as follows:





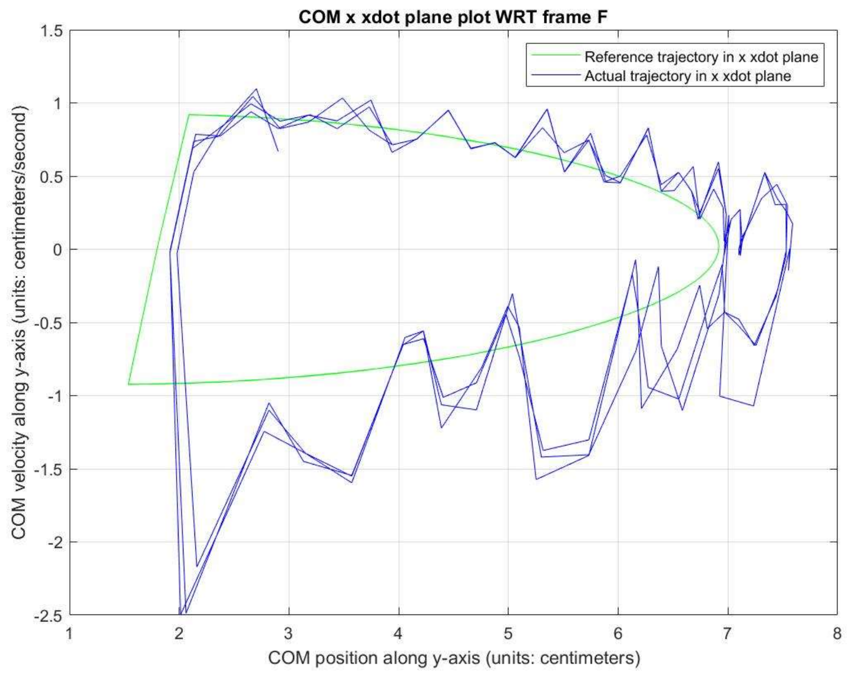


The COM path reference vs actual is as below:

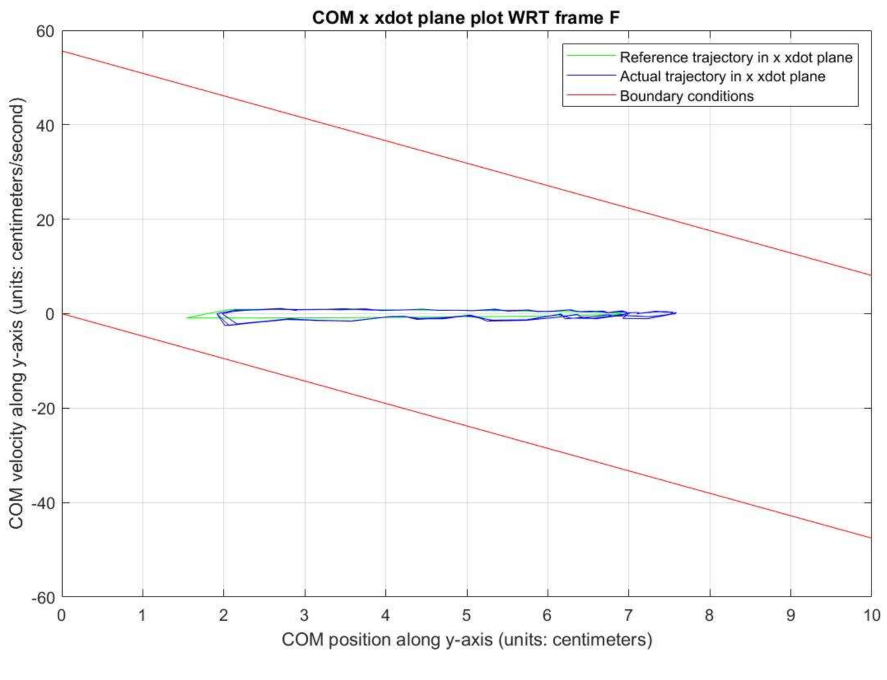


As we can see from the COM path plot, the COM deviates from the reference path when the leg is rising (or COM is moving in -ve y direction). And from the Ankle joint angle plot we can observe that the ankle motor is not able to track the reference position accurately when the leg is rising. And if the speed of leg movement is increased the ankle motor starts to blink its overload LED. Therefore, there is not enough torque available from the ankle motor. Replacing the ankle motor with a bigger rated torque motor would resolve this error. For a safe operation, the hardware implementation is done at a lower velocity.

The COM path in x xdot plane from the logged data is as follows:



The leg robot takes approximately 18 seconds to complete its movement. And the reference velocity of COM is always below 1 cm/s. The same plot plotted along with the stability boundaries is as follows:



Future works:

Since the system is stable for sinusoidal inputs at joint angles. We can test the performance of the system for different trajectories compare which one suits the best. I've attached a script for creating robotic leg object and to generate a trapezoidal velocity profile trajectory using Petre corke's robotics toolbox (in **Appendix d**). These joint angles can be fed into Arduino Uno as reference angles and the performance can be compared.

Appendices:

Appendix a:

Steps to run the Simulink Model

The Simulink model to simulate the above system is "Robotic_leg.slx". The system parameters are included in the Simulink model and can be accessed by

modelling -> model explorer -> Robotic_leg (in Simulink Root) -> model workspace.

To save the Simulink model as a new file or run it for the first time do the following configuration setup:

Modelling -> Model settings -> Simscape Multibody (from options on the left) -> Diagnostics -> Change 'Rigidly constrained block' under 'Topology' from error to warning.

In the Simulink model, go to "actuation and sensing Subsystem". Set the manual switch to Sinusoidal wave generator.

Run the Simulink model.

A mechanical explorer window pops up showing the animation of the robotic leg for the given actuations at joints. This Simulink model also logs the joint angles and angular velocity data to the workspace.

Run the "Robotic_leg_simulation_plots.m" to visualize the results of simulation. This script only runs after the Simulink model has ran.

Script for "Robotic_leg_simulation_plots.m" is:

```
%% Run this script after running the Robotic_leg.slx simulink model.

%% save the data to local variables.

% q is joint angles

q = [out.q.Data(:,1), out.q.Data(:,2), out.q.Data(:,3)];

% qd are joint angular velocities

qd = out.qd.Data;

%% Compute the COM position and COM velocity using q and qd.

[Ycom, Zcom] = COMPosition(q);

[Ydcom, Zdcom] = COMVelocity(q, qd);



%% Plotting the results of simulation

% COM path from simulation
```

```
figure(1);

plot(Ycom, Zcom);

grid on;

title('COM path from simulation');

xlabel('Y position of COM (units: centimeters)');

ylabel('Z position of COM (units: centimeters)');

legend('COM path');


%% COM path in x xdot plane with stability boundaries

foot_l = 11.7;

w = (9.81/0.4338)^0.5;

x = 0:0.2:10;

xd = -w*x;


figure(2);

plot(Ycom, Ydcom, '-g', x,xd,'-r', x,xd+w*foot_l,'-r');

grid on;

title('COM path in x xdot plane from simulation data');

xlabel('Y position of COM (units: centimeters)');

ylabel('Y velocity of COM (units: centimeters/second)');

legend('Y Ydot plot of COM','Boundary conditions');
```

Appendix b:

Arduino Uno and Arduino Nano sketches

Arduino Uno sketch:

Sinusoidal_trajectory.ino

```
// Include necessary libraries.
#include <DynamixelShield.h>  // to communicate with dynamixel shied and motors
#include <SoftwareSerial.h>   // to communicate with Nano using software programmed serial communication.
// Debug serial to log data using Nano.
SoftwareSerial soft_serial(7, 8); // DYNAMIXELShield UART RX/TX
#define DEBUG_SERIAL soft_serial

// initializing motor id's
const uint8_t AnkleMotor_ID = 1;
const uint8_t KneeMotor_ID = 2;
const uint8_t HipMotor_ID = 3;
const float DXL_PROTOCOL_VERSION = 2.0;

// initializing global variables for future use.
double q[3] = {0,0,0}; // joint angles(degrees)
float qh[3] = {180, 180, 180};  // Joint angles home configuration(degrees)
float qr[3] = {0,0,0};          // Reference Joint angles
float w[3] = {0,0,0};           // Joint angular velocities
float theta = 0;                // variable used for ref trajectory generation

// dynamixel object initialisation
DynamixelShield dxl;
void setup()
{
  dxl.begin(9600);              //Initializing dynamixel comm with 9600 baud rate
  DEBUG_SERIAL.begin(57600);    //Initialize serial comm with Nano.
  dxl.setPortProtocolVersion(DXL_PROTOCOL_VERSION);
  // Get info of the dynamixel motors attached to the shield
  dxl.ping(AnkleMotor_ID);
  dxl.ping(KneeMotor_ID);
  dxl.ping(HipMotor_ID);
  // Setting the motors in position mode.
  // Torque has to be turned off to change the operating mode of
  // motors. As this is in EEPROM area of controller in motors

  dxl.torqueOff(AnkleMotor_ID);
  dxl.torqueOff(KneeMotor_ID);
  dxl.torqueOff(HipMotor_ID);

  dxl.setOperatingMode(AnkleMotor_ID, OP_POSITION);
  dxl.setOperatingMode(KneeMotor_ID, OP_POSITION);
  dxl.setOperatingMode(HipMotor_ID, OP_POSITION);

  dxl.torqueOn(AnkleMotor_ID);
  dxl.torqueOn(KneeMotor_ID);
  dxl.torqueOn(HipMotor_ID);
}
void loop()
{
  // Go to home configuration
  dxl.setGoalPosition(AnkleMotor_ID, qh[0], UNIT_DEGREE);
  dxl.setGoalPosition(KneeMotor_ID, qh[1], UNIT_DEGREE);
  dxl.setGoalPosition(HipMotor_ID, qh[2], UNIT_DEGREE);

  // wait for 5 seconds
  delay(5000);

  // Generate a sin wave from phase (0 - pi) in 64 steps.
  for(int i = 0; i<64; i++)
  {
    theta = M_PI/64.0*(float)i;

    q[0] = qh[0] - 30*sin(theta); // in Degrees
    q[1] = qh[1] + 60*sin(theta);
  q[2] = qh[2] - 30*sin(theta);



  // set the next reference position for all joints
  dxl.setGoalPosition(AnkleMotor_ID, q[0], UNIT_DEGREE);
  dxl.setGoalPosition(KneeMotor_ID, q[1], UNIT_DEGREE);
  dxl.setGoalPosition(HipMotor_ID, q[2], UNIT_DEGREE);

  // Read the present postion and velocity of joints
  qr[0] = dxl.getPresentPosition(AnkleMotor_ID, UNIT_DEGREE);
  w[0] = dxl.getPresentVelocity(AnkleMotor_ID, UNIT_RPM);
  qr[1] = dxl.getPresentPosition(KneeMotor_ID, UNIT_DEGREE);
  w[1] = dxl.getPresentVelocity(KneeMotor_ID, UNIT_RPM);
  qr[2] = dxl.getPresentPosition(HipMotor_ID, UNIT_DEGREE);
  w[2] = dxl.getPresentVelocity(HipMotor_ID, UNIT_RPM);

  // wait for 10 milliseconds.
  // Changing this wait time changes the frequency of input ref sin angles
  delay(10);

  // Print all the results to Nano.
  DEBUG_SERIAL.println();
  DEBUG_SERIAL.print(qr[0]);
  DEBUG_SERIAL.print(" , ");
  DEBUG_SERIAL.print(qr[1]);
  DEBUG_SERIAL.print(" , ");
  DEBUG_SERIAL.print(qr[2]);
  DEBUG_SERIAL.print(" , ");
  DEBUG_SERIAL.print(w[0]);
  DEBUG_SERIAL.print(" , ");
  DEBUG_SERIAL.print(w[1]);
  DEBUG_SERIAL.print(" , ");
  DEBUG_SERIAL.print(w[2]);
  DEBUG_SERIAL.print(" , ");
  DEBUG_SERIAL.println(millis());
}
```

```
     //while(1){};
    }
```

Arduino Nano sketch:

Nano_Debug_serial.ino

```
//Include necessary libraries.
#include <SoftwareSerial.h>   // to communicate with Arduino Uno over UART
SoftwareSerial serial(7,8);    // Initialize the rx/tx pins for serial communication

void setup()
{
  Serial.begin(115200);        // Initialize to enable communicaiton from Nano to Personal computer.
  while (!Serial) {
    ; // waits till the Serial monitor is open.
  }
  Serial.println("Nano ready to recieve! :)");
  serial.begin(57600);          // This baud rate has to match the UNO's Debug_serial baud rate
}
void loop()
{
  // if any data is available on programmed Rx pin. Print to Serial monitor.
  if (serial.available()) {
    Serial.write(serial.read());
  }
}
```

Appendix c:

Reading and Interpreting the logged data:

The data logged is stored in text file named “Sin_position.txt”. To run the “file_read.m” script below, replace the absolute path in function readtable(“Your Path”) to the location of text file in your PC. Otherwise matlab cannot read the file.

file_read.m:

```
clear all;

% change the path of text file as needed.

%% Setup the Import Options and import the data

opts = delimitedTextImportOptions("NumVariables", 7);


% Specify range and delimiter

opts.DataLines = [4, Inf];

opts.Delimiter = ",";


% Specify column names and types

opts.VariableNames = ["q1", "q2", "q3", "w1", "w2", "w3", "t"];

opts.VariableTypes = ["double", "double", "double", "double", "double", "double", "double"];


% Specify file level properties

opts.ExtraColumnsRule = "ignore";

opts.EmptyLineRule = "skip";


%%  Import the data

% Specify the absolute path of the text file on your PC to successfully run

% the script.

JointAngles = readtable("D:\Arduino\BioRobotics\Sin_position.txt", opts);

j = table2array(JointAngles);


%% Clear temporary variables

clear opts JointAngles

%% Interpret the data

% Read the joint angles into q and time into t. initialize the home

% configuration
```

```
qh = [180,180,180];

q = [j(2:end,1)-qh(1), j(2:end,2)-qh(2), j(2:end,3)-qh(3)] .* pi/180;

t = j(1:end-1,7)./1000;


% Generate the reference sinusoidal input.

for i=1:length(q)

    qr(i,:) = [-30*abs(sin(pi*i/64)), 60*abs(sin(pi*i/64)), -30*abs(sin(pi*i/64))] .* pi/180;

end


%% Plot Joint angles reference vs actual

figure(1);

plot(t, q(:,1),'-r',t,qr(:,1),'-g');

grid on;

title('Ankle joint Refence angle vs Actual angle WRT frame according to DH conventions ');

xlabel('time (seconds)');

ylabel('Joint angle (units: radians)');

legend('Actual angle','Reference angle');


figure(2);

plot(t, q(:,2),'-r',t,qr(:,2),'-g');

grid on;

title('Knee joint Refence angle vs Actual angle WRT frame according to DH conventions ');

xlabel('time (seconds)');

ylabel('Joint angle (units: radians)');

legend('Actual angle','Reference angle');


figure(3);

plot(t, q(:,3),'-r',t,qr(:,3),'-g');

grid on;

title('Hip joint Refence angle vs Actual angle WRT frame according to DH conventions ');

xlabel('time (seconds)');

ylabel('Joint angle (units: radians)');

legend('Actual angle','Reference angle');


%% Compute COM position

% compute COM posirion for actual and reference joint angles given

[Yref, Zref] = COMPosition(qr);

[Yact, Zact] = COMPosition(q);
```

```
% plot the COM path of both actual and reference angles



figure(4);
plot(Yref, Zref,'-g',Yact, Zact, '-r');
grid on;
title('COM path Actual vs Reference WRT frame {F}');
legend('Reference path','Actual path');
xlabel('y-axis (units: centimeters)');
ylabel('z-axis (units: centimeters)');

%% Compute the veecity of joints (both reference and actual)
for i=1:length(t)-1
    qd(i,:) = [(q(i+1,1)-q(i,1))/(t(i+1)-t(i)), (q(i+1,2)-q(i,2))/(t(i+1)-t(i)), (q(i+1,3)-q(i,3))/(t(i+1)-t(i))];
    qdr(i,:) = [(qr(i+1,1)-qr(i,1))/(t(i+1)-t(i)), (qr(i+1,2)-qr(i,2))/(t(i+1)-t(i)), (qr(i+1,3)-qr(i,3))/(t(i+1)-t(i))];
end

% compute the COM velocity of both actual and reference joint velocities
% and positions
[Ydact, Zdact] = COMVelocity(q(2:end,:),qd);
[Ydref, Zdref] = COMVelocity(qr(2:end,:),qdr);
%% plot COM velocity vs position with the stability boundaries
foot_l = 11.7;
ww = (9.81/0.4338)^0.5;
xx = 0:0.2:10;
xxd = -ww*xx;

figure(5);
plot(Yref(2:end), Ydref, '-g',Yact(2:end), Ydact, '-b');
grid on;
title('COM x xdot plane plot WRT frame {F}');
xlabel('COM position along y-axis (units: centimeters)');
ylabel('COM velocity along y-axis (units: centimeters/second)');
legend('Reference trajectory in x xdot plane', 'Actual trajectory in x xdot plane');

figure(6);
plot(Yref(2:end), Ydref, '-g', Yact(2:end), Ydact, '-b', xx,xxd,'-r', xx,xxd+ww*foot_l,'-r');
grid on;
title('COM x xdot plane plot WRT frame {F}');
```

```
xlabel('COM position along y-axis (units: centimeters)');

ylabel('COM velocity along y-axis (units: centimeters/second)');

legend('Reference trajectory in x xdot plane', 'Actual trajectory in x xdot plane', 'Boundary conditions');
```

Appendix d:

Robotic leg object

Script for robotic leg trapezoidal trajectory generation using peter corke's robotics toolbox. Run "startup_rvc.m" before this script.

Robotic_leg_object.m:

```
%dimensions in cm.
foot_l = 11.7;
foot_w = 5;
foot_h = 2;
support_w = 4;
support_l = 2.5;
support_h = 5.5;
motor_w = 3.3;
motor_l = 3;
motor_h = 5.4;
link_r = 0.6;
link1_l = 19.4;
link2_l = 15;
hip_l = 9;
hip_w = 6;
hip_h = 4;

%%
L1 = Link('d',0,'a',0,'alpha',0);
L2 = Link('d',0,'a',0,'alpha',pi/2);
L3 = Link('d',0,'a',0,'alpha',-pi/2);
L4 = Link('d',0,'a',foot_h+support_h,'alpha',0);

L5 = Link('d',0,'a',support_h+motor_h+link1_l,'alpha',0);
L6 = Link('d',0,'a',support_h+motor_h+link2_l,'alpha',0);
L7 = Link('d',0,'a',motor_h+hip_h,'alpha',0);

Leg = SerialLink([L1 L2 L3 L4 L5 L6 L7],'name', 'Robotic leg');
qh = [0 0 pi/2 0 0 0 0];

%%
qt = qh + [0 0 0 0 pi/6 -pi/3 pi/6];
[q1,q1d,q1dd] = mtraj(@lspb, qh, qt, 6);
[q2,q2d,q2dd] = mtraj(@lspb, qt, qh, 6);

p = [q1;q2]; pd = [q1d;q2d]; pdd = [q1dd;q2dd];

%%
q = -[p(:,5), p(:,6), p(:,7)];
qd = -[pd(:,5), pd(:,6), pd(:,7)];


%%
figure(1);
subplot(311);
plot(q(:,1));
subplot(312);
plot(qd(:,2));
subplot(313);
plot(qdd(:,3));
grid on;

%%
```

```
T = Leg.fkine(q);
figure(2);
Leg.plot(q);
```

Appendix e:

COMPosition function:

```
function [x,y] = COMPosition(q)

%dimensions in cm.

foot_l = 11.7;

foot_w = 5;

foot_h = 2;

support_w = 4;

support_l = 2.5;

support_h = 5.5;

motor_w = 3.3;

motor_l = 3;

motor_h = 5.4;

link_r = 0.6;

link1_l = 19.4;

link2_l = 15;

hip_l = 9;

hip_w = 6;

hip_h = 4;


%mass and inertia

foot_m = 20;

support_m = 20;

motor_m = 57.2;

link_m = 20;

hip_m = 30;


%% Lengths of rigid bodies

l0 = foot_h+support_h;

l1 = motor_h+link1_l+support_h;

l2 = motor_h+link2_l+support_h;

l3 = motor_h+hip_h;


% COM lengths of rigid bodies

a1 = (motor_h/2*motor_m + (motor_h+link1_l/2)*link_m +
(motor_h+link1_l+support_h/2)*support_m)/(motor_m+link_m+support_m);
```

```
a2 = (motor_h/2*motor_m + (motor_h+link2_l/2)*link_m +
(motor_h+link2_l+support_h/2)*support_m)/(motor_m+link_m+support_m);

a3 = (motor_h/2*motor_m+(motor_h+hip_h/2)*hip_m)/(motor_m+hip_m);


% masses of rigid bodies

m0 = foot_m+support_m;

m1 = motor_m+link_m+support_m;

m2 = m1;

m3 = motor_m+hip_m;


% COM positions of rigid bodies

x0 = (foot_m*(foot_h/2)+support_m*(foot_h+support_h/2))/(support_m+foot_m);

y0 = (foot_m*(-foot_l/2)+support_m*(-support_l/2))/(foot_m+support_m);

x1 = a1*cos(q(:,1))+l0;

y1 = a1*sin(q(:,1))-support_l/2;

x2 = l0+l1*cos(q(:,1))+a2*cos(q(:,1)+q(:,2));

y2 = -support_l/2+l1*sin(q(:,1))+a2*sin(q(:,1)+q(:,2));

x3 = l0+l1*cos(q(:,1))+l2*cos(q(:,1)+q(:,2))+a3*cos(q(:,1)+q(:,2)+q(:,3));

y3 = -support_l/2+l1*sin(q(:,1))+l2*sin(q(:,1)+q(:,2))+a3*sin(q(:,1)+q(:,2)+q(:,3));


% COM of rigid body

Xcom = (m0*x0+m1*x1+m2*x2+m3*x3)/(m0+m1+m2+m3);

Ycom = (m0*y0+m1*y1+m2*y2+m3*y3)/(m0+m1+m2+m3);


x = -Ycom;

y = Xcom;

end
```

COMVelocity function:

```
function [xd, yd] = COMVelocity(q,qd)

%dimensions in cm.

foot_l = 11.7;

foot_w = 5;

foot_h = 2;

support_w = 4;

support_l = 2.5;

support_h = 5.5;

motor_w = 3.3;

motor_l = 3;

motor_h = 5.4;
```

```
link_r = 0.6;

link1_l = 19.4;

link2_l = 15;

hip_l = 9;

hip_w = 6;

hip_h = 4;

%mass and inertia

foot_m = 20;

support_m = 20;

motor_m = 57.2;

link_m = 20;

hip_m = 30;

%% lengths of rigid bodies

l0 = foot_h+support_h;

l1 = motor_h+link1_l+support_h;

l2 = motor_h+link2_l+support_h;

l3 = motor_h+hip_h;

% COM lengths of rigit bodies

a1 = (motor_h/2*motor_m + (motor_h+link1_l/2)*link_m +
(motor_h+link1_l+support_h/2)*support_m)/(motor_m+link_m+support_m);

a2 = (motor_h/2*motor_m + (motor_h+link2_l/2)*link_m +
(motor_h+link2_l+support_h/2)*support_m)/(motor_m+link_m+support_m);

a3 = (motor_h/2*motor_m+(motor_h+hip_h/2)*hip_m)/(motor_m+hip_m);


% masses of rigid bodies

m0 = foot_m+support_m;

m1 = motor_m+link_m+support_m;

m2 = m1;

m3 = motor_m+hip_m;

% velocity components of COM of rigid bodies

xd1 = -a1*sin(q(:,1)).*qd(:,1);

yd1 = a1*cos(q(:,1)).*qd(:,1);

xd2 = -l1*sin(q(:,1)).*qd(:,1)-a2*sin(q(:,1)+q(:,2)).*(qd(:,1)+qd(:,2));

yd2 = l1*cos(q(:,1)).*(qd(:,1))+a2*cos(q(:,1)+q(:,2)).*(qd(:,1)+qd(:,2));

xd3 = -l1*sin(q(:,1)).*(qd(:,1))-l2*sin(q(:,1)+q(:,2)).*(qd(:,1)+qd(:,2))-a3*sin(q(:,1)+q(:,2)+q(:,3)).*(qd(:,1)+qd(:,2)+qd(:,3));

yd3 = l1*cos(q(:,1)).*(qd(:,1))+l2*cos(q(:,1)+q(:,2)).*(qd(:,1)+qd(:,2))+a3*sin(q(:,1)+q(:,2)+q(:,3)).*(qd(:,1)+qd(:,2)+qd(:,3));

% Velocity of COM

Xdcom = (m1*xd1+m2*xd2+m3*xd3)/(m0+m1+m2+m3);

Ydcom = (m1*yd1+m2*yd2+m3*yd3)/(m0+m1+m2+m3);
```

```
xd = -Ydcom;

yd = Xdcom;

end
```