MSE222 Project:

Design and Fabrication of a Dynamic System

Final Report

Submitted By:

Group 12

The Nam Anh Mai 301326816

Cao Thanh Nhat Tan 301269797

Ramy ElMallah 301328602

Sherwin Jay Pacamarra 301347128

SIMON FRASER UNIVERSITY

MECHATRONICS SYSTEM ENGINEERING

KINEMATICS AND DYNAMICS OF RIGID BODIES AND MECHANISMS

Table of Contents

Introduction

The objective of this project is to build a physically dynamical system that involves movement of a ball, solve combinations of kinetics problems and implement them on different software to help visualize the progress. Recycled wood, plastic, and spring are glued together to create the bars, curved path, and initial impact element. For each required criterion, a variation of equations is used, and the code is created to determine ball's velocity and position after each part.

Procedure

The project starts with the visualization of the parts' shape and location and a mindset of simplicity and minimal quantity of parts required. During this stage, we do not specify the dimension and material we will use, but just try to minimize the uncertainty elements that may come up during the analysis.

After some consideration, the team decided that we need more data in order to make a decision on the design of the latter half of the project. Therefore, we started on the analysis and MATLAB code for each part, this helped us gain a better understanding of the ball's movement and the effect of friction and Coefficient of Restitution on the velocity of the ball.

When the MATLAB code for each individual component was completed (in draft form), we proceeded to simultaneously working on both building (experiment with different materials) and coding. We used what we find in one section to improve the other, make the code and the prototype resemble each other as much as possible until we achieved an acceptable result.

Part 1: Analysis of Each Component in the System

I. Spring

The conservation of energy law was used in analyzing the velocity of the ball as it is being pushed by the spring while the spring is moving from the compressed position to its equilibrium position. An assumption was made that the balls roll without slipping to simplify the calculations. As the spring is pushing the ball on a horizontal plane, there is no change in the ball potential energy due to the weight of the ball. The following equation was used which is derived from the conservation of energy law:

$$V_{G2} = \sqrt{V_{G1}^2 + \frac{5k}{7m} * [(S_{eq} - S)^2 - (S_{eq} - S - ds)^2}$$

Where S_{eq} is the equilibrium position X-coordinate of the spring, V_{G2} is the velocity at the end of the small element under analysis and V_{G1} is the velocity at the beginning of the element, which is zero at the very first element as the ball starts from rest.

The analysis of the ball interaction with the spring was implemented in MATLAB as a function that takes four parameters:

1. S_1: which is the X-coordinate of the spring tip under compression.
2. S_e: which is the X-coordinate of the spring equilibrium position.
3. K: the spring constant.
4. M: the mass of the ball.

The function outputs a 3-column matrix with the position of the ball in X and Y coordinates through time, while time starts from zero as this is the very first element in the design. The plots for ball position and velocity will be shown in **Figure 1** and **Figure 2** in **Appendix I**.

II. Rotating Element

Upon reaching the end of the horizontal bar, the ball will hit a vertical rod at rest. Its momentum will transfer partly to the rod and conservation of momentum is used to calculate ball velocity after the collision. Take B (rod hinge) as a reference point:

$$(H_B)_1 = m_b v_1 (l - r_b)$$

$$(H_B)_2 = m_b v_2 (l - r_b) + I_b w_{r2} \qquad w_{r2} = \frac{v_{r2}}{l - r_b}$$

$$I_B = \frac{1}{3} m_r l^2 + \frac{2}{5} m_b {r_b}^2 + m_b (l - r_b)^2$$

Where v_1, v_2 are ball velocity magnitudes before and after impact, v_{r1}, v_{r2} are rod velocity likewise, w_{r2} is rod angular velocity after impact and I is the mass moment of inertia when the ball is in contact with the rod.

Also, the coefficient of restitution is given as:

$$e = \frac{v_2 - v_{r2}}{v_{r1} - v_1} = \frac{v_2 - v_{r2}}{-v_1} = 0.01$$

After solving for v_2 in above equations, conservation of energy is used to predict the time and position where the ball will drop to the ramp below.

$$T_1 = \frac{1}{2} m_b {v_2}^2 + \frac{1}{2} I_B {w_{r2}}^2$$

$$V_1 = -(m_b + m_r) \bar{y}_1 g$$

$$\overline{y_1} = \frac{m_r \frac{l}{2} + (l - r_b) m_b}{m_r + m_b}$$

$$T_2 = 0$$

$$V_2 = -(m_b + m_r) \bar{y}_2 g \qquad \bar{y}_2 = \bar{y}_1 \cos(\theta)$$

$$T_1 + V_1 = T_2 + V_2$$

Where T, V represent kinetic and potential energy, $\bar{y}_1$ is initial centroid position in vertical axis, $\bar{y}_2$ is final vertical centroid position when the ball is about to fall, and θ is the angle between the rod then and the vertical axis through the hinge.

The angle, θ, is then known, and the positions of the ball during this period can be plotted as follows:

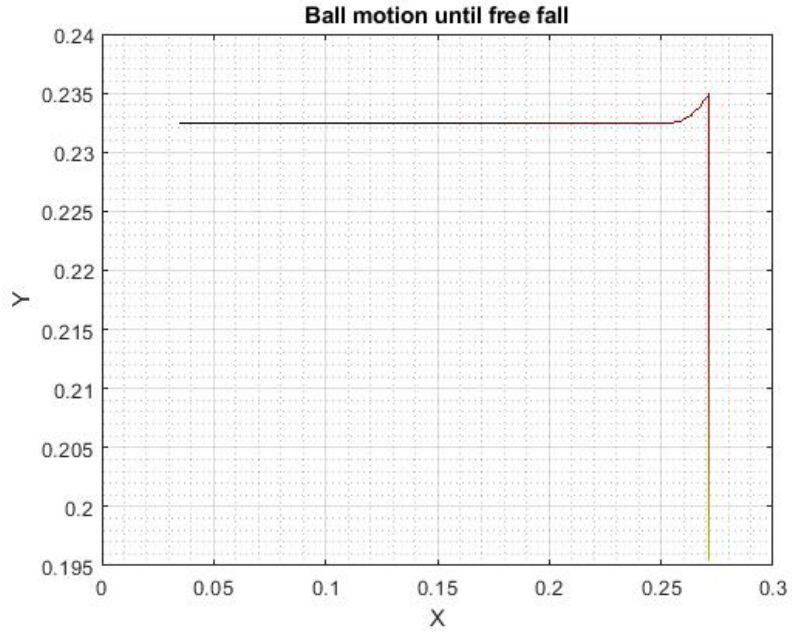


Figure *3.* *Ball during rotation element and freefall*

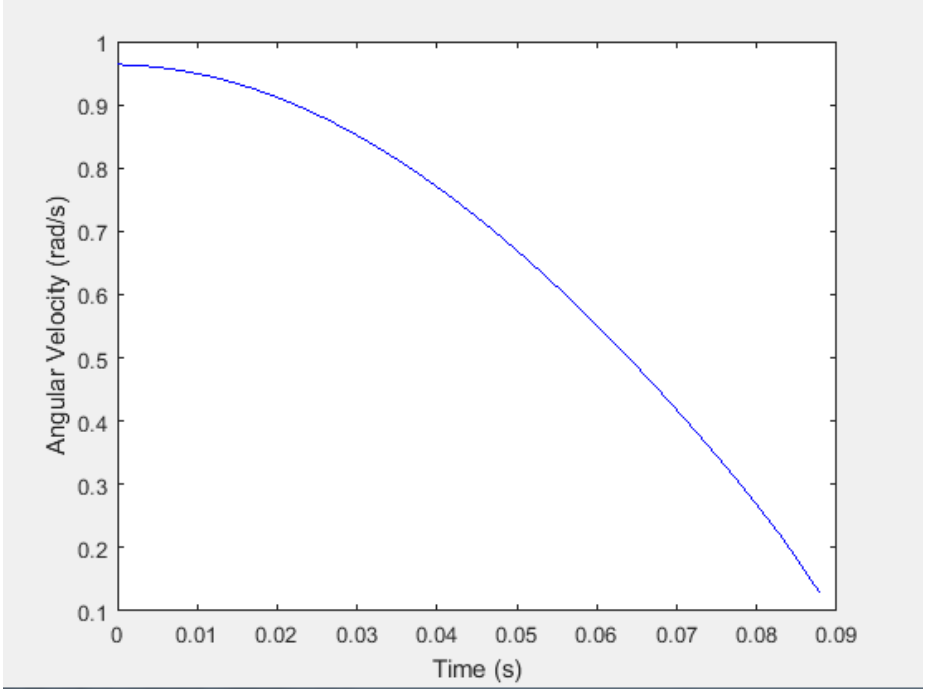


Figure 4. Angular Velocity in Rotation

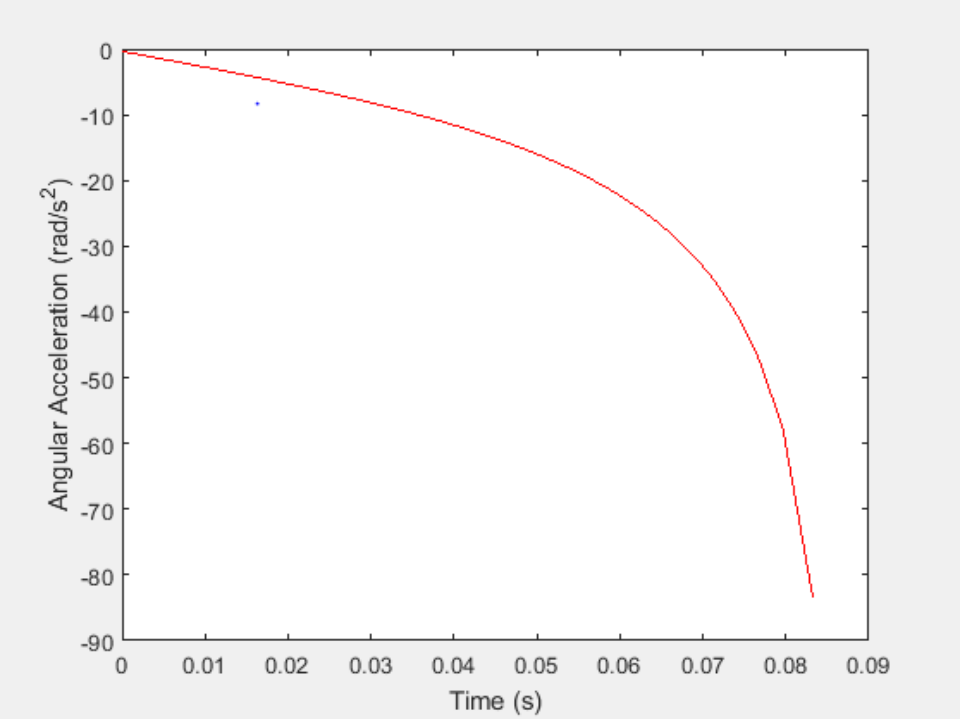


Figure 5. Angular Acceleration in Rotation

Above figures match with expectation for which the ball sticks with the rod and moves with it for a small amount of time. After losing most of the stored energy, the ball stops and starts falling freely to the ramp below. When the ball starts touching the rod, it transfers momentum and loses angular velocity and acceleration until momentarily stops midair before falling on the ramp.

III. Curvature

The conservation of energy law was used in analyzing the motion of the ball along the curvature. The ball was assumed to be rolling without slipping to simplify the problem. The ball encounters a change in potential energy between any two points along its curvature path due to the elevation change and that is equal to the change in the kinetic energy of the ball. The following equation was derived from the law of conservation of energy:

$$V_{G2} = \sqrt{V_{G1}^2 + \frac{10}{7}(r)(g)(\cos(\theta + d\theta) - cos\theta)}$$

where V_{G2} is the velocity at the end of the segment under analysis, V_{G1} is the velocity at the beginning of the segment under analysis, r is the radius of curvature, g is the free fall acceleration, θ is the angle at the beginning of the segment of interest measured from the reference of a vertical line the passes through the center of the path curve, and dθ is the angle of the small element. Ball movement plot is further illustrated in **Figure 6** in **Appendix I**.

The MATLAB function for this section takes an input of three parameters:

1. Θ1, which is the angle at which the curve begins with respect to the vertical reference.
2. Θ2, which is the angle at which the curve ends with respect to the vertical reference. Radius, which is the radius of the curve in meters.
3. The MATLAB code takes X, Y, Slope angle and Coefficient of Restitution as input and return an output matrix of x, y, and t.

IV. Slope Bounce

Since the air friction is insignificant and the fall distance is too short, the contact angle between the ball and the slope will be too small to bounce down. Therefore, the ball will always bounce up, then roll down. Slope bouncing graphs are represented **in Appendix I** as **Figure 7** and **8**.

- Projectile position equations:
  - $X = V_x * t + X_o$
  - $Y = V_y * t + \frac{1}{2} * g * t^2 + Y_0$
- Assumption: Coefficient of Restitution and gravity are the only factors affecting the velocity of the ball.
- The coefficient of Restitution: $e = \frac{V_{after\ impact}}{V_{before\ impact}}$ (as the floor does not move).
- Time of flight for each bounce: $T = \frac{2 * V_{after\ impact} * \sin(\theta - \alpha)}{g * \cos(\alpha)}$
- The ball stop bouncing when: $\theta \approx 0$

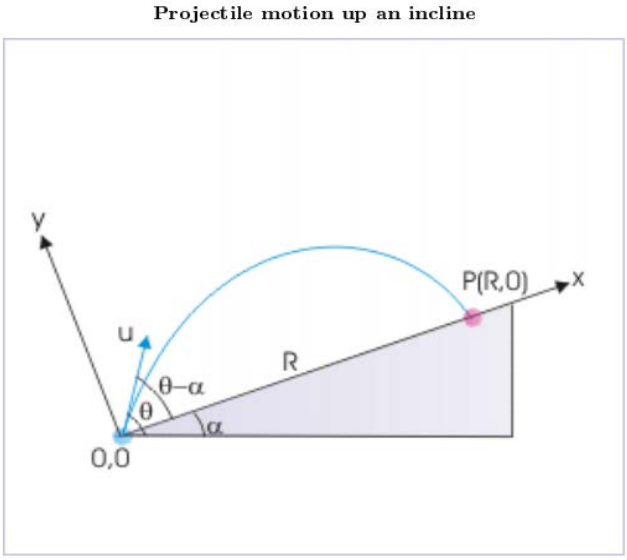


Figure 9. Projectile motion up an incline

V. Slope

In our system, there are three slopes to consider for analysis. The parameters are the angle and the length of the slope. However, calculating acceleration is key. From a free body diagram of a ball on a slope, we solve for acceleration where the positive x-direction is in the direction of the slope.:

$$ma = mgsin(\theta) - \mu mg$$

$$a = \ g\ (\ \sin(\theta) - \mu\ \cos(\theta)\ )$$

This acceleration that is found is constant and a function of angle, θ and the coefficient of friction, μ. With the same importance, time is calculated to construct a proper plot with the equation below.

$$Slope\ length = \frac{1}{2}at^2 + V_{mag}t$$

The sloped element is separated into exceedingly small sections where the final velocity of the last small element is the initial velocity for the next one. **Figure 10** in **Appendix I** will show the case when the ball travels on the ramp after free fall.

VI. Projectile

The projectile uses kinematics and the initial X and Y of the ball the instant it leaves the curvature. Only the force of gravity is acting on the ball. We assume that drag is negligible since the projectile time of flight is small.

- Time of 1st projectile: 0.1941 seconds
- Time of 2nd projectile: 0.0176 seconds

Solving for time is imperative since it governs how accurate the projectile plot is on MATLAB. We solve for time with the straight-line equation of the slope element that the projectile lands on and the following two equations from kinematics. With three equations and three unknowns, we let MATLAB solve for the exact X & Y coordinates that the ball lands as well as the time of flight

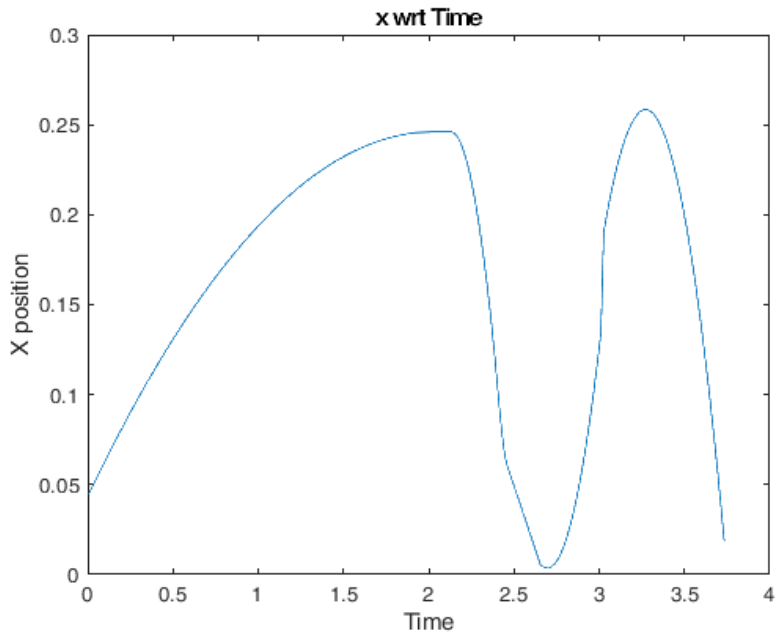
1. $x = x_0 + Vel_x t$
2. $y = y + Vel_y t + gt^2$, where g = 9.81 m/s^2

The projectile is shown in **Figure 11** as the ball leaps from the plastic curved path in actual model.

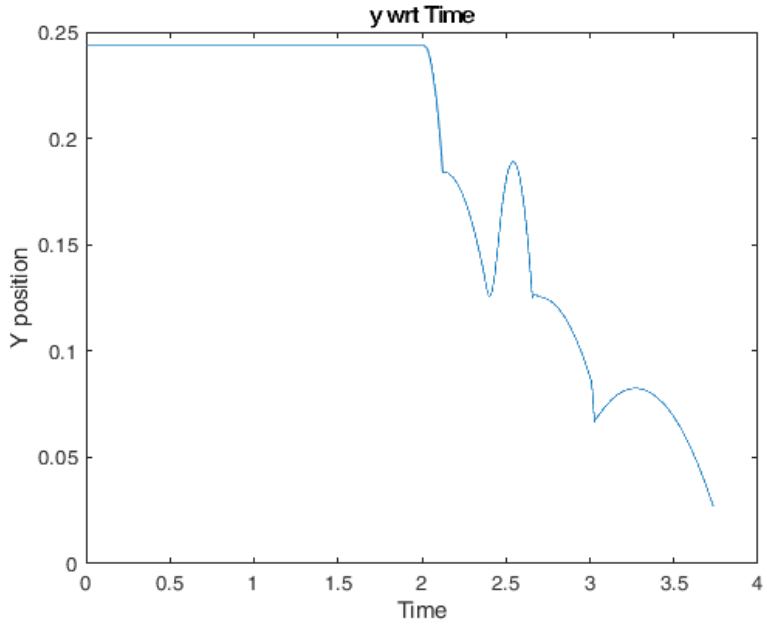
Part 2: Design, Simulate and Study the Dynamics System

I. Plots in the global axes

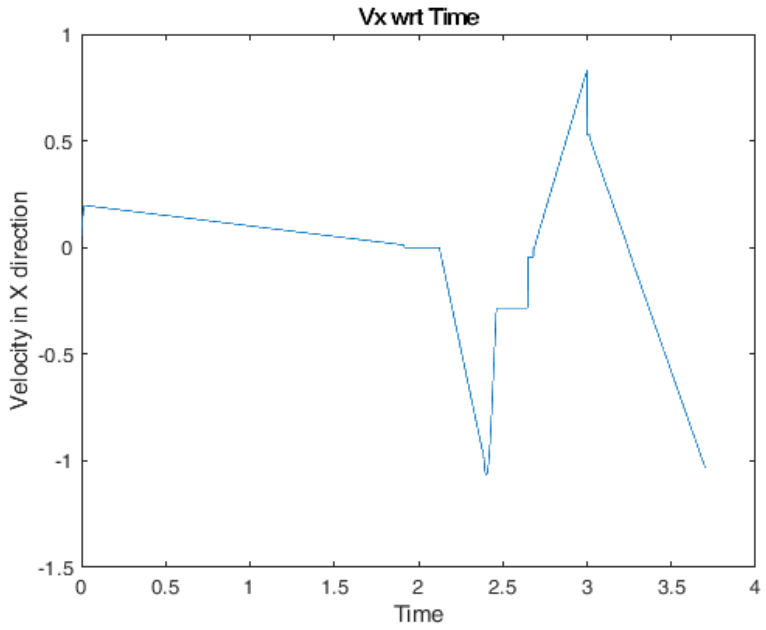
a. The position of the ball cg in x direction with respect to time:



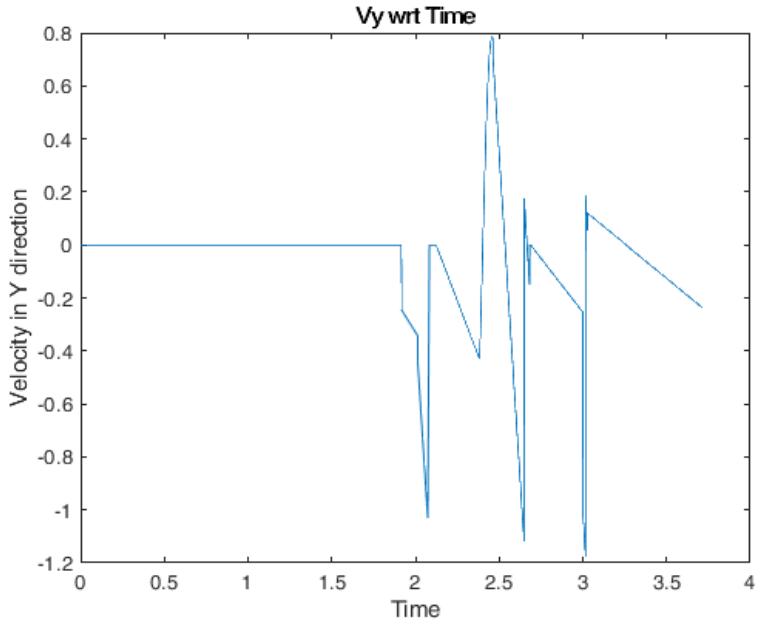
b. The position of the ball cg in y direction with respect to time:



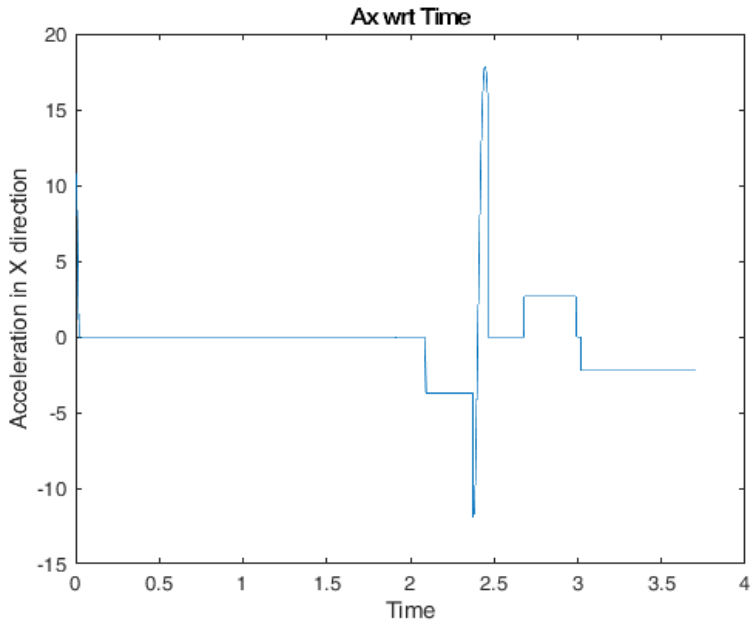
c. The velocity of the ball cg in x direction with respect to time:



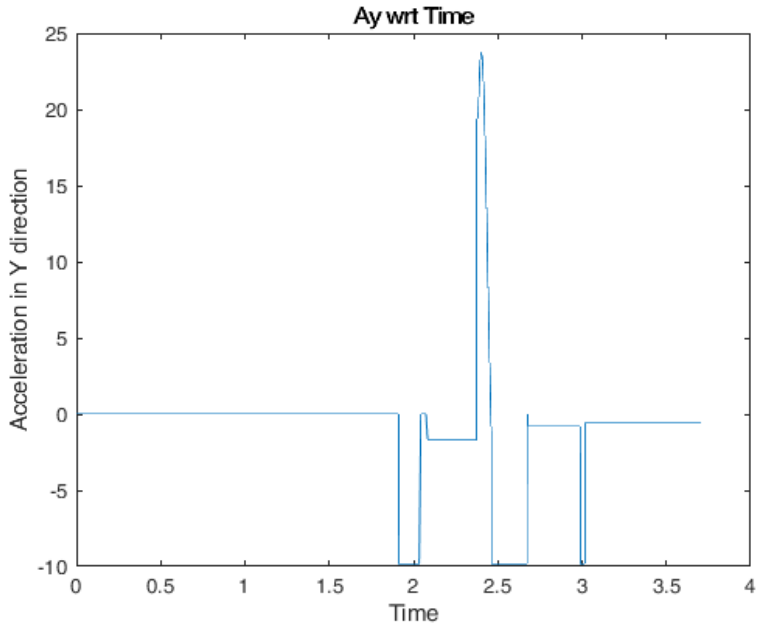
d. The velocity of the ball cg in y direction with respect to time:



e. Acceleration of the ball cg in x direction with respect to time:



f. Acceleration of the ball cg in y direction with respect to time:



II. AutoCAD and SolidWorks Drafting

Using AutoCAD to design the real model, the picture below shows exact dimensions and ratios, which are used for calculation and coding in MATLAB.

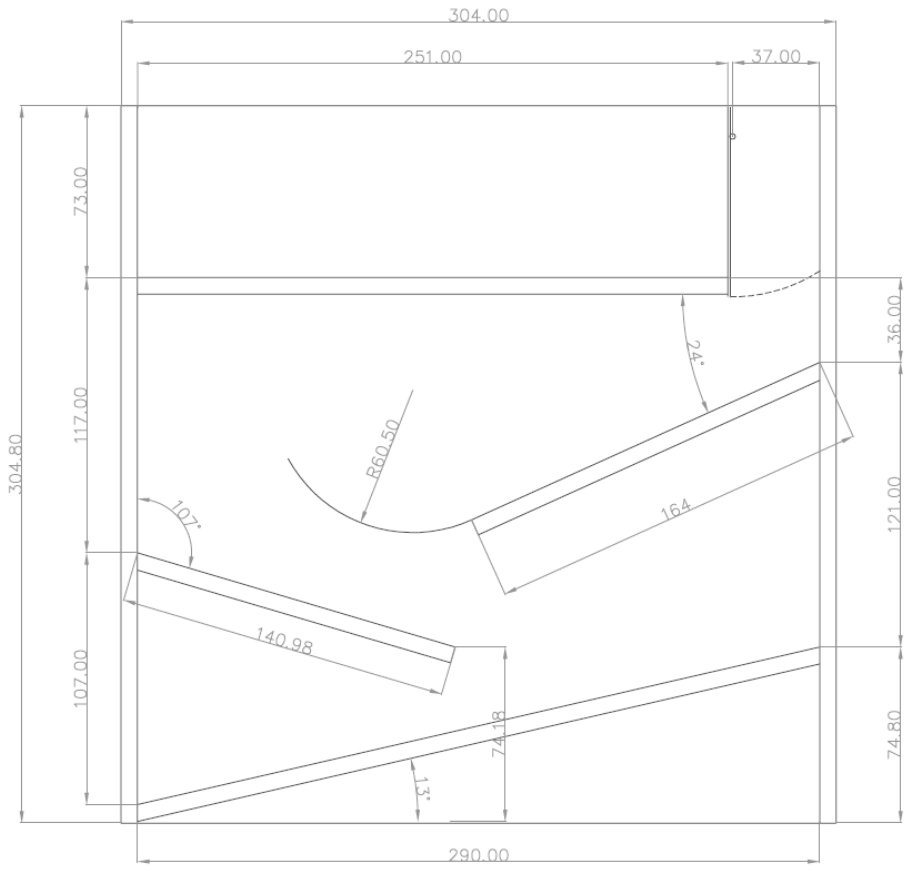


Figure 12. 2D Graph of the Dynamic System

In Solidworks, the assembly was created from all the parts modelled with a detailed simulation of reality and usage of materials in detail, which helps to provide a vivid prototype replicate for readers.

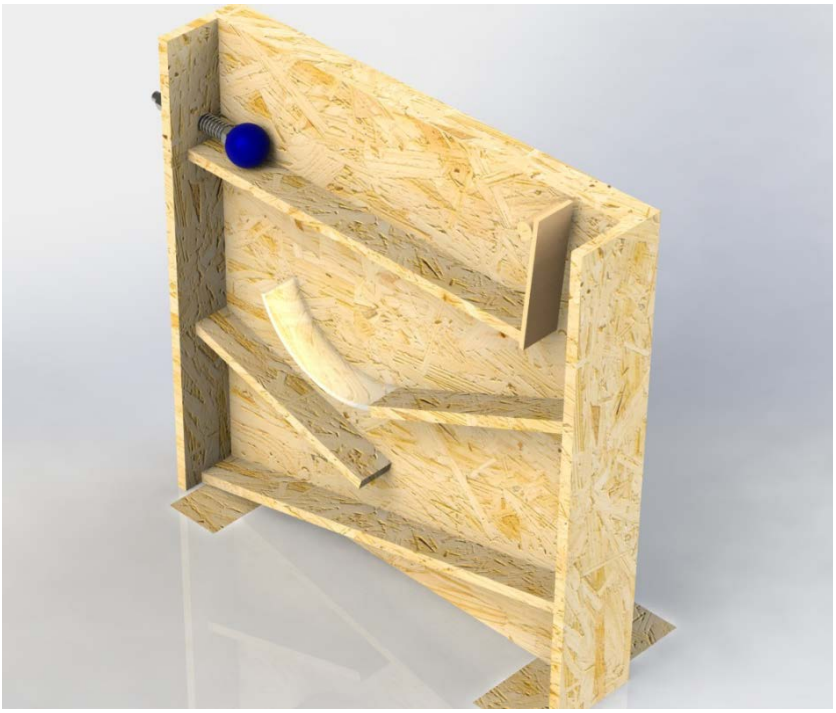


Figure 13. 3D Model of the Dynamic System in SolidWorks

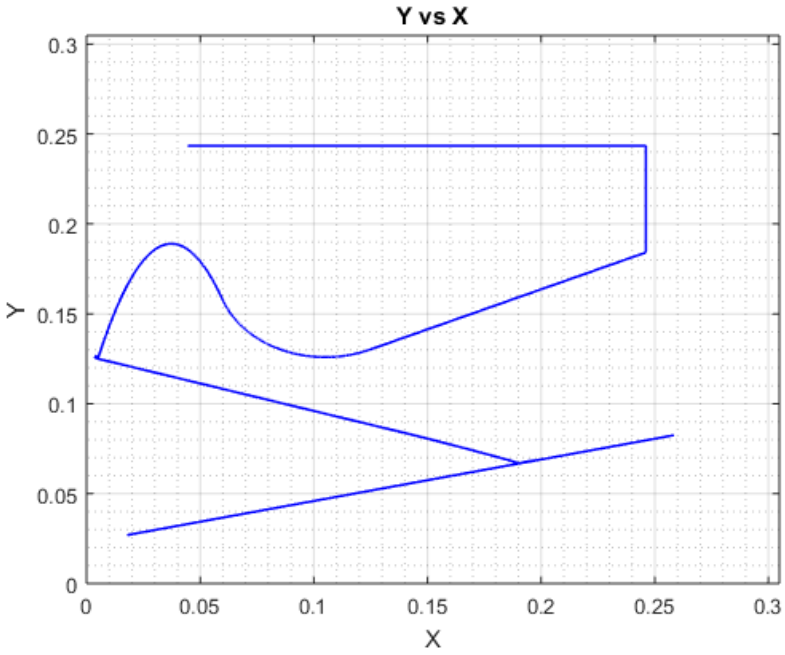


Figure 14. Position plot of the Marble Ball

The figure above describes the ball movement in the authentic system, for which the bottom left corner point is taken as the origin for global coordinates. After being released by the spring, it hits rotating element at the top right corner, however, this effect is not noticeable due to the rod's light weight. The

ball follows most of the path as in real case, and experiences two small bouncing after projectile motion and onto the last ramp.

III. Sensitivity Study

Constant Notation	-10%	Original Value	10%	Mean	Standard Deviation
Marble vs Wood (e)	3.6935	3.7365	3.7765	3.7355	0.0415
Spring Constant (k)	3.6795	3.7365	3.2878`	3.5679	0.2443
Friction (μ)	3.3452	3.7365	3.5922	3.558	0.1979
General Mean and Standard Deviation				3.649475	0.162673

Table 1. Mean and Standard Deviation of Total Ball Travel Time Sensitivity

- According to the test result, the total output time depends greatly on the Spring Constant as the spring provides the initial speed for the ball during the first horizontal part and the ball spent most of its movement on that part.
- The bounce and impact function only work within a certain range of contact angles of the ball (when the ball hits the slope), as most calculation and analysis is done with a bouncing up scenario in mind, therefore, it is not universal.

Part 3: Component Sourcing and Characterization

I. Component Sourcing

Element	Source	Duration of prior usage	Disposal after usage
Wooden 12x12” slab	Given by TA	Almost not used	With the project fixed on it, the whole slab will be saved and stored
Wooden bars x 6	Cut from scrap wood found in Home Depot	Unknown	Can be reused in another project
Popsicle sticks x 4	Jay’s house	About 7 months	Can be reused in another project
Marble Ball	Jay’s toy box	About 1 year	Reusable for another project
Spring & Bolt	SFU machine shop used parts	Unknown	Return to the machine shop.

Table 2. Materials Environmental Impact Assessment

- The first important element of our dynamic system is the 12x12 inches wooden slab which was given by the TA and did not seem that it has been used before.
- Secondly, we used wooden sheets that we managed to get from the scrap wood that Home Depot has. These sheets are assumed to not have been used before, but they were just cut to dimensions that are no longer applicable to being sold.
- Popsicle sticks and marbles were found at Jay's house.
- The spring bolt was taken from the machine shop used parts. So, we cannot determine the duration of its prior usage. However, we will return it back to the machine shop to be re-used by anyone in need of it again.

II. Component Characterization

- Coefficient of Restitution: Initially approximated to be 0.2 through prototype observation. Then after carefully measure the time, travel distance and experiment in MATLAB code, the final value is decided to be 0.15 as it gives the best simulation of the real model.
- Friction Coefficient: Is the last element to be characterized as its effect is thought to be minimal (assume the ball rolling without slipping).
- Spring Constant: We can also measure the spring constant by putting a weight under the spring vertically and measure the deformation. Using the equation: $Weight = K * \Delta x$, we can find the Spring Constant. This will give us better estimation.
- Mass of the ball: the ball was weighed to determine its mass. Some error could have been involved due to human reading, scale calibration, temperature and humidity, and other factors. To decrease the error, more readings could be taken in addition to using different scales.

The table below is the characterization of parameters for the spring and the ball:

Global Constant and Coefficient uses in the entire System				
Name	Value Used in the final model	Testing Values	Mean	Standard Deviation
Spring Constant (N/m) - k	126	120, 129, 134	127.25	5.85235
Mass of the Ball (kg) - m	0.0207	0.022, 0.026, 0.03	0.024675	0.004206

Table 3. Parameters Used in the Project

Conclusion

After more than a month of work, with knowledge learned from class lectures on kinetics and kinematics, assistance from professor and TA, and efforts from all members, the project was successfully created by hands and simulated in MATLAB. The ball was estimated to travel in about 3.5 to 4 seconds after timing, which reaches the project objective along with other physical requirements. All the parts were effectively analyzed and modeled by extensive coding, and the results obtained show close accuracy to real events. Throughout this process, more connections are linked between theory and practice, which is proved to be valuable in the understanding of dynamics concepts and solving a variety of real-life problems.

Appendices

I. Position Plot for each element

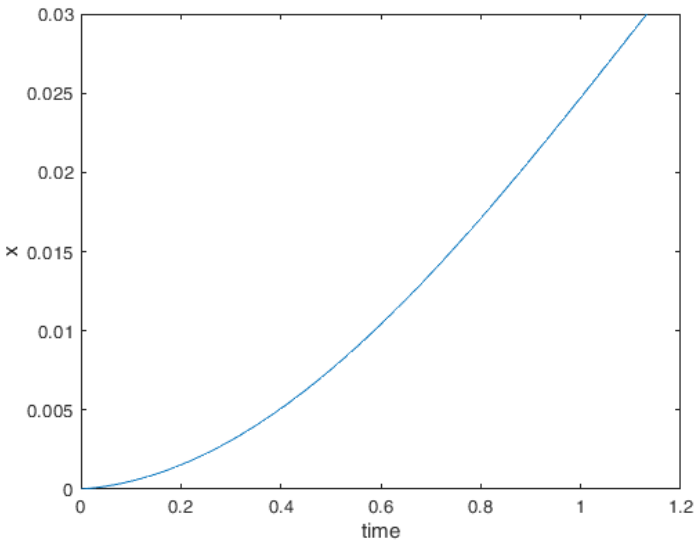


Figure 1. Graph of the ball position x versus time under the spring action

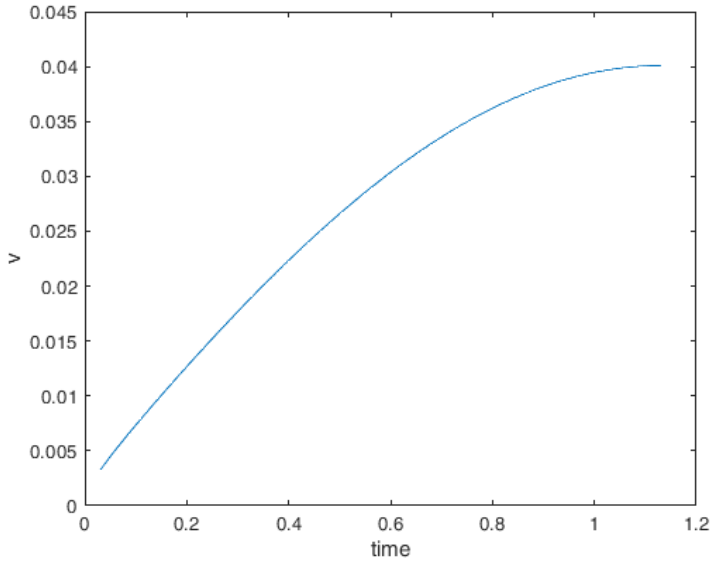


Figure 2. Graph of the ball velocity versus time under the spring action

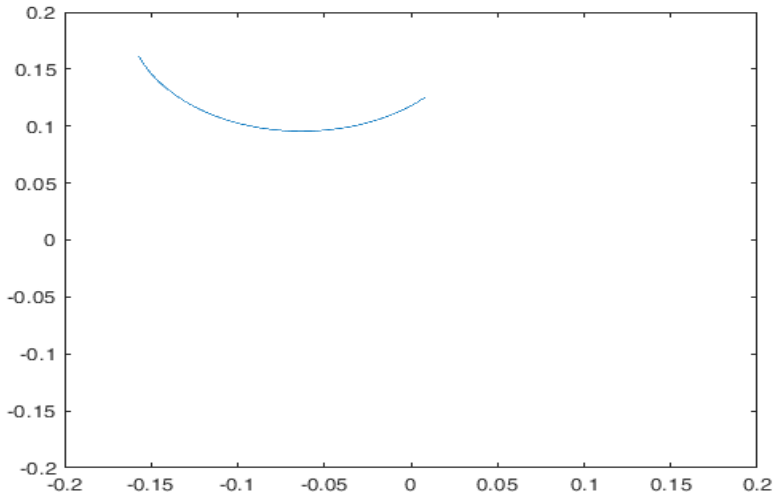


Figure 6. Position of the ball in x and y coordinates under sample parameters for the curvature function

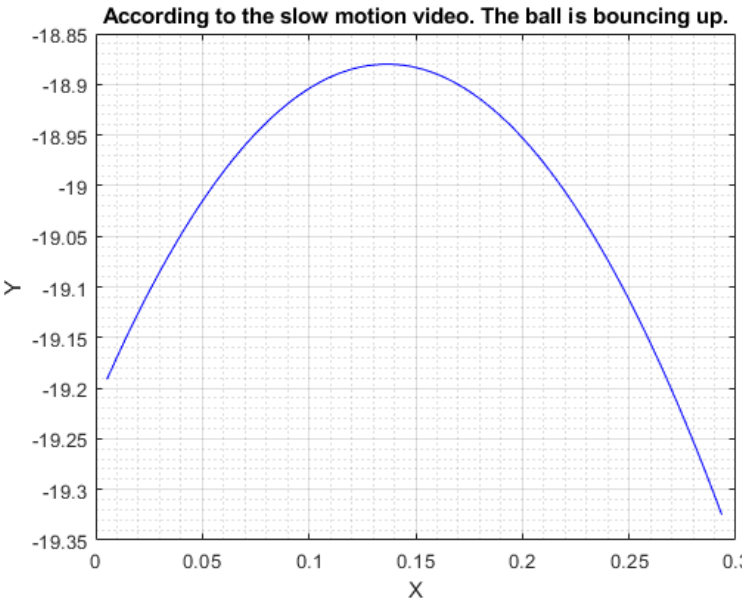


Figure 7. Position of the ball bouncing upward

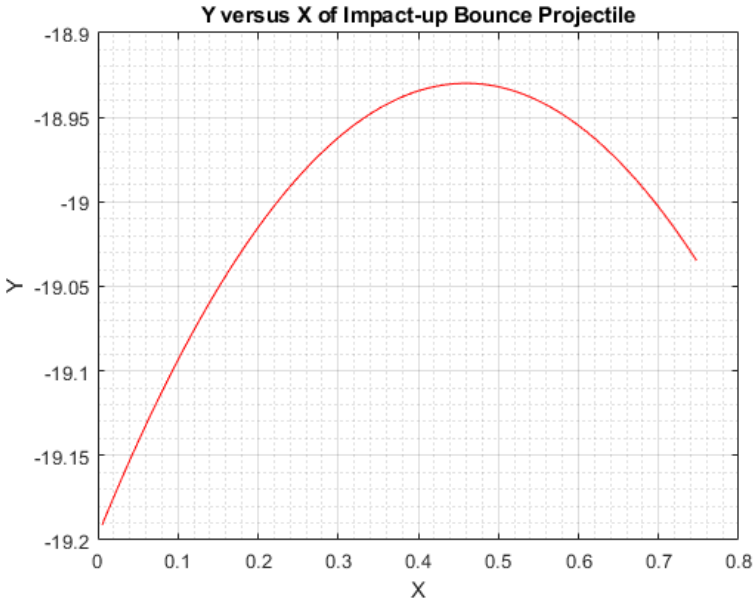


Figure 8. Relationship of v of the ball versus time

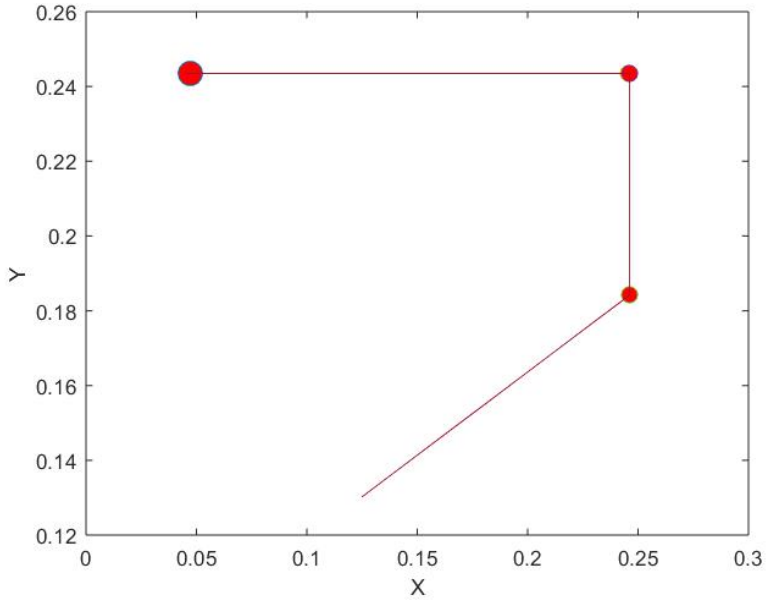


Figure 10. This graph represents the slope path after the horizontal and free fall element

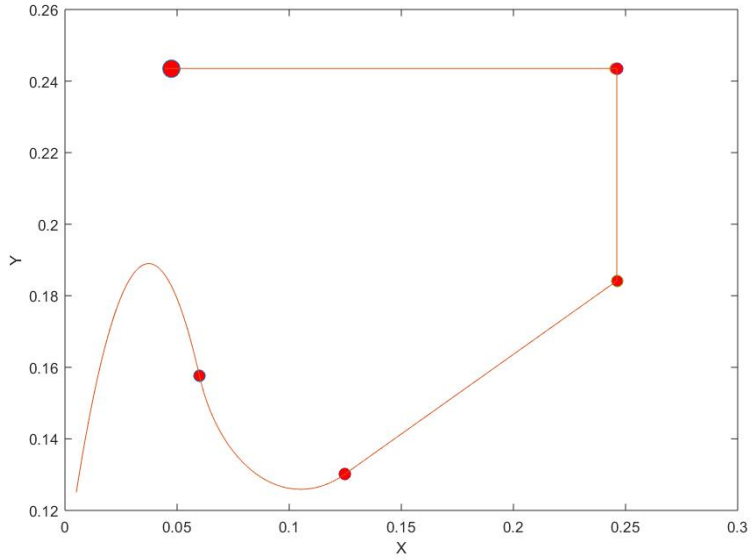


Figure 11. Projectile motion after curvature

II. X and Y plot

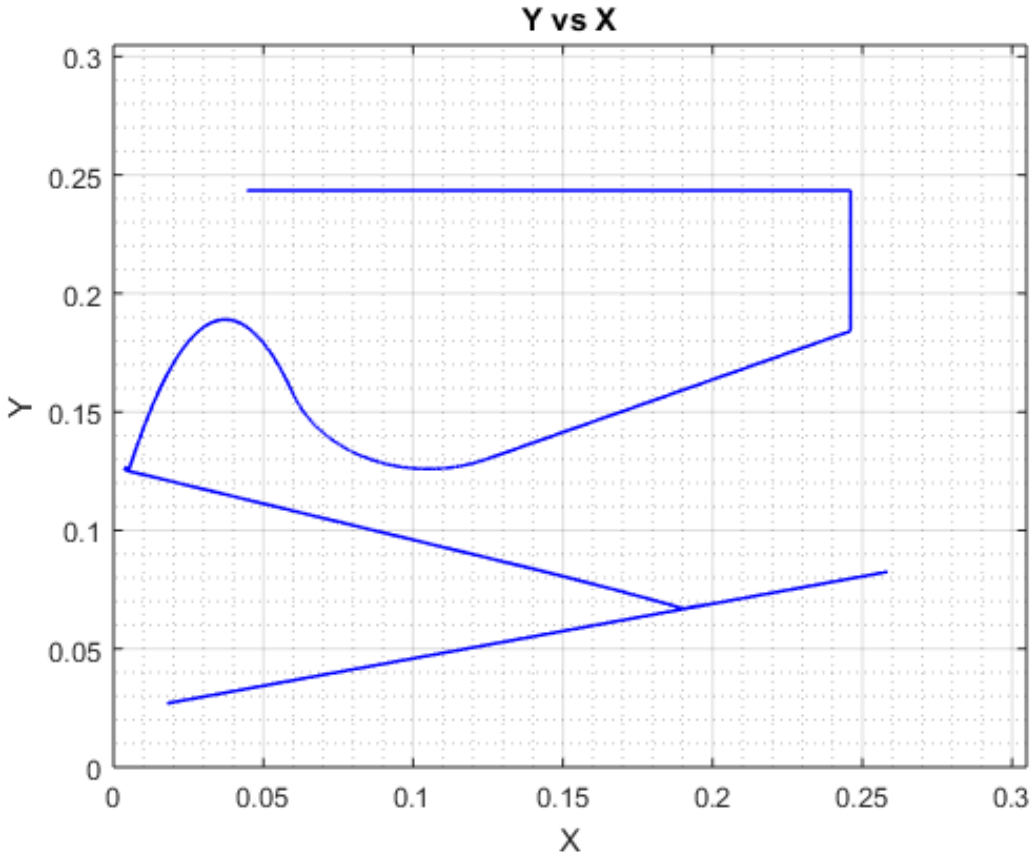


Figure 12. X versus Y position of the ball

III. Matlab Code

Main Program

```
clear global;
close all;
clc
clearvars;

%This is a list of constants that is dependent on the design
(START)
% I am assuming standard metric units (metre, kg, second, N,
etc...)


initialXafterSpring = 0.0475; %the actual value from left -
Value after the spring release position
SE = initialXafterSpring;
initialX = 0.0445; %assuming compression of 3.5 millimetres
initialY = 0.2435; %The actual value from the bottom in the
prototype
springK = 126;  %I don't what is its range or expected value
% springS = 0.035; %the difference between the spring
equilibrium and the compression (compression length is S)
mass = 0.0207; %mass of the ball
coeffE = 0.16;
global slopeF;
slopeF = 0.01;
% Constants List (END)

%global X,Y,time array declaration and initialization
global GlobalXYT;
GlobalXYT = [initialX, initialY, 0];

%EXAMPLE of how to caoncatenate to the global array a function
results
%         tempX = [3,6,7,0.3,0.5];
%         tempY = [45, 45,  67, 0.3, 0.5];
%         tempT = [0.001,0.002,0.003,0.1,2];
%         tempXYT = [tempX',tempY',tempT'];
%         GlobalXYT = [GlobalXYT; tempXYT];

%Spring Part
[x, y, time] = springDetail(initialX, SE, springK, mass);
%function calling
GlobalXYT = [GlobalXYT; [x, y, time]]; %adding the results to
the global array
```

```
%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 12,
'MarkerFaceColor', 'r');
hold on;

%Horizontal Part
horizontalResults=slope(0,0.198);
GlobalXYT = [GlobalXYT; horizontalResults];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

% Rotation Part
[result3] = ori_rotation(mass,0.005,0.0125,0.06);
GlobalXYT = [GlobalXYT; result3];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Freefall Part
[result4] = freefall();
GlobalXYT = [GlobalXYT; result4];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

loseEnergy(); %Assuming that the ball loses all its energy at
hitting the slope after the free fall

% subplot(1,2,2)
% v = diff(GlobalXYT(1:end,1))./ diff(GlobalXYT(1:end,3));
% vy = diff(GlobalXYT(1:end,2))./ diff(GlobalXYT(1:end,3));
% plot(v,vy)
% title('Vy wrt Vx')

%Slope before curvature Part
slopeResults=slope(156,0.13325);
GlobalXYT = [GlobalXYT; slopeResults];

%Draw Separator
```

```
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Curvature Part
%The following constants are only dependent on the geometry of
the design
theta1= -24.36;
theta2 = 70.26;
CurveRadius = 0.048;
[x, y, time] = curve(theta1, theta2, CurveRadius); %function
calling
GlobalXYT = [GlobalXYT; [x, y, time]]; %adding the results to
the global array

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%projectile part after curvature
projectileResults = projectile();
GlobalXYT = [GlobalXYT; projectileResults];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Impact Part
angle_impact_slope = 17; %degrees
%Coefficient of restitution is the last input. I assumed it to
be 0.5
[result] = Impact_up_left(angle_impact_slope,
GlobalXYT(end,1),GlobalXYT(end,2), GlobalXYT(end,3),coeffE);
%Output is in a 3 matrix-columns matrix (x,y,t). Can add into a
global array later.
GlobalXYT = [GlobalXYT; result];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Calculate slope length after first bouncing
calculatedLength = sqrt((GlobalXYT(end,1)-
0.13482)^2+(GlobalXYT(end,2)-0.08668)^2);
```

```
loseEnergy();

%Slope after first bouncing
slopeResults=slope(17,calculatedLength);
GlobalXYT = [GlobalXYT; slopeResults];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

% %projectile part after curvature
projectileResults = projectile2();
GlobalXYT = [GlobalXYT; projectileResults];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Impact2
angle_impact_slope = 13;
[result]=Impact_up_right(angle_impact_slope,
GlobalXYT(end,1),GlobalXYT(end,2), GlobalXYT(end,3),coeffE);
GlobalXYT = [GlobalXYT; result];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Slope after second impact - upwareds
slopeResults=slopeUpward(13);
GlobalXYT = [GlobalXYT; slopeResults];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 8,
'MarkerFaceColor', 'r');
hold on;

%Calculate downwards slope length after upwards slope
calculatedLength2 = sqrt((GlobalXYT(end,1)-
0.0195)^2+(GlobalXYT(end,2)-0.0195)^2);

%Slope after second impact - downwards
slopeResults=slope(167,calculatedLength2);
```

```
GlobalXYT = [GlobalXYT; slopeResults];

%Draw Separator
plot(GlobalXYT(end,1), GlobalXYT(end,2), 'O', 'MarkerSize', 12,
'MarkerFaceColor', 'r');
hold on;

elapsedTime=GlobalXYT(end,3)

drawWithTime()
%%
%figure;
% subplot(1,2,1)
x = GlobalXYT(:, 1);
y = GlobalXYT(:, 2);
plot(x, y ,'LineWidth', 1, 'color', 'b');
title('Y wrt X')
xlim([0,0.3048])
ylim([0,0.3048])

%%
elapsedTime=GlobalXYT(end,3)
figure;
subplot(1,2,1)
x = GlobalXYT(:, 1);
y = GlobalXYT(:, 2);
plot(x, y);
title('Y wrt X')
xlim([0,0.3048])
ylim([0,0.3048])
subplot(1,2,2)
v = diff(GlobalXYT(1:end,1))./ diff(GlobalXYT(1:end,3));
vy = diff(GlobalXYT(1:end,2))./ diff(GlobalXYT(1:end,3));
plot(v,vy)
title('Vy wrt Vx')



%Up slope impact. Debug only.
[result2] = Impact_up_right(12,
GlobalXYT(end,1),GlobalXYT(end,2), GlobalXYT(end,3),0.2);
GlobalXYT = [GlobalXYT; result2];
```

```
function [result]= Impact_up_right(angle, X, Y, global_time, c)
%Bounce functions are similiar to projtile function with
decreasing velocity after each bounce.
%Time: The total time the ball takes to bounce in seconds
result = [X,Y,global_time];
global GlobalXYT;
g=9.81;
localtime = 0;
timejump = 0.001; %Time step size is 1 ms
t= 0.000;
stop = false;
angle=angle*(pi./180); %slope angle
Vx = abs((GlobalXYT(end,1) - GlobalXYT(end-
1,1))/(GlobalXYT(end,3) - GlobalXYT(end-1,3)))*c;
Vy = abs((GlobalXYT(end,2) - GlobalXYT(end-
1,2))/(GlobalXYT(end,3) - GlobalXYT(end-1,3)))*c;
velocity = sqrt(Vx.^2 + Vy.^2); %calculating the initial
velocity from the last element data
x0=X;
y0=Y;
x=X;
y=Y;

%Real contact will be acot(vx/vy) + 2*angle, not the one below.
bounce_angle = acot(Vx/Vy); %Contact angle change after each
bounce.
landingtime = (2*velocity*sin(bounce_angle -
angle))/(g*cos(angle)); %Total time to complete 1 bounce
while (stop == false)

    if (localtime <= landingtime) %This check whether or not a
bounce is completed and move to a new bounce.
        localtime = localtime + timejump; %Local time of each
bounce. t is the temp time (only use inside this function).
        t= t + timejump;
        x = Vx*localtime + x0;
        y = Vy*localtime - 0.5*g*localtime^2 + y0;
        result = [result; x y (global_time+t)]; %Output.
    else
        localtime=0;
        x0=x;
        y0=y;
        Vy = -(Vy - g*landingtime)*c;
        Vx = Vx;
        velocity=(sqrt((Vx^2) + (Vy^2))); %New bounce velocity
```

```
        bounce_angle = acot(Vx/Vy); %Contact angle change after
each bounce.
        landingtime = (2*velocity*sin(bounce_angle -
angle))/(g*cos(angle)); %Total time to complete 1 bounce
    end
    if (bounce_angle - angle) <= 0
        stop = true;
        break; %Exit the function if the contact angle become
too close to slope angle.
    end
end
end
```

Spring

```
function [x,y,time] = springDetail(S1, Se, K, m)
  %S1 is the starting X position of the compressed spring
  %Se is the equilbrium position of the spring and the position
when the ball will be set free
  %K is the constant of the spring
  %m is the mass of the ball
  %units is kg, m, rad, etc...
 ds = 0.0001; %decrease for more accuracy

 global GlobalXYT;
 Vinitial = 0; %the ball starts from rest

 s = S1:ds:Se;

 time = 0 + GlobalXYT(end,3); %initialzing the the time
 x = GlobalXYT(end,1); %initialzing x in global coordinates
 for i = s(1:end-1)
  V2 = sqrt( (Vinitial.^2) + ((5*K)/(7*m))* ( (Se-i).^2 - (Se-i-
ds).^2 ) );
  [x] = [x; x(end)+ds]; %adding the increased dx
  [time] = [time; time(end)+((ds)/V2)]; %calcualting time and
add it to the previous time consumed
  %[time] = [time; (ds/V2)];
  Vinitial = V2;
 end
 y= ones(length(x), 1)* GlobalXYT(end,2);
```

Curve

```
function [x,y,time] = curve(theta1, theta2, radius)
  %theta1 is supposed to be negative
  %theta2 is supposed to be a positive value and the last angle
which is
  %the angle of the projectile
  %radius is for the curvature and not the ball
  %units is kg, m, rad, etc...
 dtheta = 0.001; %decrease for more accuracy

 global GlobalXYT; %letting the function know about the global
array
 Vx = (GlobalXYT(end,1) - GlobalXYT(end-1,1))/(GlobalXYT(end,3)
- GlobalXYT(end-1,3));
 Vy = (GlobalXYT(end,2) - GlobalXYT(end-1,2))/(GlobalXYT(end,3)
- GlobalXYT(end-1,3));
 Vinitial = sqrt(Vx.^2 + Vy.^2); %calculating the initial
velocity from the last element data

 theta1 = degtorad(theta1);
 theta2 = degtorad(theta2);
 theta = theta1:dtheta:theta2;


 chord = 2*radius*sin(dtheta/2); %length of the chord of the
circle corresponding to dtheta
 time = 0 + GlobalXYT(end,3); %initialzing the the time with the
last value of time from the last element
 x = GlobalXYT(end,1) + chord*-cos(theta1); %initialzing the x
in the global coordinates
 y = GlobalXYT(end,2) + chord*sin(theta1); %initialzing the y in
the global coordinates

 for i = theta(1:end-1)
  V2 = sqrt( (Vinitial.^2) +(10/7)*9.81*radius*(cos(i+dtheta)-
cos(i)));
  dx = chord*-cos(i+dtheta); %x distance moved through the
dtheta
  dy = chord*sin(i+dtheta); %y distance moved through the dtheta
  [x] = [x; x(end)+dx]; %adding the increased dx
  [y] = [y; y(end)+dy]; %adding the increased dy
  [time] = [time; time(end)+((dtheta*radius)/V2)]; %calcualting
time and add it to the previoud time consumed
  Vinitial = V2;
 end
```

Downward Slope and Horizontal Path

```
function [slopeResults] = slope(a,distance)
% Uses Newtons - Kinematics and Kinetic
%   Input Arguments are slope(ANGLE,LENGTH OF SLOPE)


global GlobalXYT;
global slopeF;

x0=GlobalXYT(end,1);
y0=GlobalXYT(end,2);
g = 9.81;
slopeLength = distance;
angle = a*(pi./180);
coefFriction = slopeF;   %Change this to match physical model
dtime=0.001;

acceleration = g*( sin(angle)-coefFriction*cos(angle) );
initialVelX=(GlobalXYT(end,1)-GlobalXYT(end-
4,1))/(GlobalXYT(end,3)-GlobalXYT(end-4,3));
initialVelY=(GlobalXYT(end,2)-GlobalXYT(end-
4,2))/(GlobalXYT(end,3)-GlobalXYT(end-4,3));
magnitudeOfVel = sqrt( initialVelX^2 + initialVelY^2 );

syms T
eqn = 0.5*acceleration*T^2+magnitudeOfVel*T-slopeLength==0;
%from kinematics
solT=double(solve(eqn,T));
calculatedTime=solT(solT>0);
calculatedTime=min(calculatedTime);
t=0:dtime:calculatedTime;

d= magnitudeOfVel*t+(1/2)*acceleration*t.^2;
x = x0 + d*cos(angle);
y = y0 - d*sin(angle);
time=GlobalXYT(end,3)+t;

slopeResults=[x',y',time'];
plot(x,y);

end
```

Freefall

```
%Free fall from the rod
function [result4] = freefall();
global GlobalXYT;
x0 = GlobalXYT(end,1);
y0 = GlobalXYT(end,2);
delta_x = GlobalXYT(end,1) - (0.251+0.037-0.164*cos(24/180*pi));
h = GlobalXYT(end,2) - 0.128184 - delta_x*tan(24/180*pi);
g = 9.81;
fall_time = sqrt(2*h/g);
t2 = 0:0.01:fall_time;
x = x0 + 0*t2;
y = y0 - 0.5*g*t2.^2;
t2=t2+GlobalXYT(end,3);
result4 = [x' y' t2'];
end
```

Upward Slope (with an initial velocity)

```
function [slopeResults] = slopeUpward(a)
% Uses Newtons - Kinematics and Kinetic
%   Input Arguments are slope(ANGLE,LENGTH OF SLOPE)
global GlobalXYT;
global slopeF;
x0=GlobalXYT(end,1);
y0=GlobalXYT(end,2);
g = 9.81;
angle = -a*(pi./180);
coefFriction = slopeF;   %Change this to match physical model
dtime=0.001;

acceleration = g*( sin(angle)-coefFriction*cos(angle) );
initialVelX=(GlobalXYT(end,1)-GlobalXYT(end-
4,1))/(GlobalXYT(end,3)-GlobalXYT(end-4,3));
initialVelY=(GlobalXYT(end,2)-GlobalXYT(end-
4,2))/(GlobalXYT(end,3)-GlobalXYT(end-4,3));
magnitudeOfVel = sqrt( initialVelX^2 + initialVelY^2 );

syms T
eqn = 0 == magnitudeOfVel + acceleration*T;
solT=double(solve(eqn,T));
calculatedTime=solT(solT>0);
calculatedTime=min(calculatedTime);
t=0:dtime:calculatedTime;


d= magnitudeOfVel*t+(1/2)*acceleration*t.^2;
```

```
x = x0 + d*cos(angle);
y = y0 - d*sin(angle);
time=GlobalXYT(end,3)+t;

slopeResults=[x',y',time'];

plot(x,y);
end
```

Projectile after a slope

```
function [results]= projectile()
%    NO INPUT ARGUMENTS NEEDED
%    I calculated time with 3 equations and used algabraic
substitutions

global GlobalXYT;

dtime = 0.001; %1 millisecond
initial_dtime = GlobalXYT(end,3)-GlobalXYT(end-4,3);
x0=GlobalXYT(end,1);
y0=GlobalXYT(end,2);
dposx=x0-GlobalXYT(end-4,1);
dposy=y0-GlobalXYT(end-4,2);
velx=dposx/initial_dtime;
vely=dposy/initial_dtime;
g=-9.81;

%Solving for time
syms X
syms Y
syms T
eq1 = X==x0+velx*T;
eq2 = Y==y0+vely*T+0.5*g*T^2;
eq3 = Y==-0.2967*X+0.1265;
[xCalc,yCalc,time]=solve([eq1,eq2,eq3],[X,Y,T]);
time=double(time);                    %Convert symbol to double
calculatedTime=time(time>0);          %Take positive time
calculatedTime=min(calculatedTime); %Take the smallest number in
the array

t=0:dtime:calculatedTime;

time=GlobalXYT(end,3)+t;
x=x0+velx*t;
y=y0+vely*t+(g*t.^2)/2;
results = [x',y',time'];
end
```

Projectile after the curve

```
function [results]= projectile2()
%    This function only works for our specific dimensions
%    because it calculates time based on our design
%    NO INPUT ARGUMENTS NEEDED
%    I calculated time with 3 equations and used algabraic
substitutions
%    This projectile function is for the positive slope element

global GlobalXYT;

dtime = 0.001; %1 millisecond
initial_dtime = GlobalXYT(end,3)-GlobalXYT(end-1,3);
x0=GlobalXYT(end,1);
y0=GlobalXYT(end,2);
dposx=x0-GlobalXYT(end-4,1);
dposy=y0-GlobalXYT(end-4,2);
velx=dposx/initial_dtime;
vely=dposy/initial_dtime;
g=9.81;

%Solving for time
%Getting Complex Time with this method
syms X
syms Y
syms T
eq1 = X==x0+velx*T;
eq2 = Y==y0+vely*T+0.5*g*T^2;
eq3 = Y==(74.8/290)*X+.0195;
[xCalc,yCalc,time]=solve([eq1,eq2,eq3],[X,Y,T]);
time=double(time);                    %Convert symbol to double
calculatedTime=time(time>0);          %Take positive time
calculatedTime=min(calculatedTime); %Take the smallest number in
the array

t=0:dtime:calculatedTime;

time=GlobalXYT(end,3)+t;
x=x0+velx*t;
y=y0+vely*t-(g*t.^2)/2;

results = [x',y',time'];

end
```

```
function [result3] = ori_rotation2(mb,mr,rb,l)
global GlobalXYT;
%mb: ball mass
%mr: rod mass
%rb: ball radius
%l: rod length
%d: horizontal bar length
%v1: ball initial velocity
g = 9.81;
e = 0.1; %coefficient of restitution
x0 = GlobalXYT(end,1);
y0 = GlobalXYT(end,2);
%Conservation of Momentum
v1 = (GlobalXYT(end,1) - GlobalXYT(end-1,1))/(GlobalXYT(end,3) -
GlobalXYT(end-1,3));
syms V2 VR2
I = (1/3)*mr*(l^2) + 0.4*mb*(rb^2) + mb*((l-rb)^2);
Ir = (1/3)*mr*(l^2);
eqns = [(l-rb)*mb*(V2-v1)+ I*VR2/(l-rb) == 0, VR2-V2-e*v1 == 0];
[v2,vr2] = solve(eqns, [V2 VR2]);
v2=double(v2);
vr2=double(vr2);
wr2 = vr2/(l-rb); %wr2: rod angular velocity after impact
%Conservation of Energy
y1 = (mr*(l/2) + 0*(l-rb))/(mr + 0);
syms anglee
eqanglee = 0.7*mb*((v2^2))+(0.5*Ir*(wr2^2))-mr*y1*g == -
mr*y1*g*cos(anglee)+ 0.7*mb*((l-rb)*0)+(0.5*Ir*0);
angle = solve(eqanglee, anglee);
angle = double(angle);
angle = degtorad(24);

theta = 0:0.001:angle;
omegaF =[];
tF = 0;
x = x0;
y = y0;
t = 0;
for i = 2:length(theta)
    syms OM;
    eqn = [0.7*mb*((v2^2))+(0.5*Ir*(wr2^2))- mr*y1*g == (-
mr*y1*g*cos(theta(i))+  0.7*mb*((v2^2))  +(0.5*Ir*(OM.^2)))];
    omega = solve(eqn, OM);
    omegad = double(omega);
    omegad = omegad(omegad>0);
    timee = (theta(i)-theta(i-1)) ./ omegad;
```

```
    omegaF = [omegaF; omegad];
    x = [x; x + l.*sin(theta(i)-theta(i-1))];
    y = [y; y + 2*l.*(sin((theta(i)-theta(i-1))/2)).^2];
    t = [t, t + timee];
    tF = [tF, tF(end)+timee];
end

figure;
plot(tF(1:end-1), omegaF, 'b');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

figure;
alphaF = diff(omegaF)./diff(tF(1:length(omega))');
plot(tF(1:length(alphaF)), alphaF, 'r')
xlabel('Time (s)');
ylabel('Angular Acceleration (rad/s^2)');

time = GlobalXYT(end,3)+ t;

result3 = [x y time];

end
```