

Optimizing a Dynamically Tuned Projectile Launcher

Part I: Summary

Our goal was to design and optimize a 3-mass, 3 spring projectile launcher system by calculating the ideal masses and spring constants for the greatest launch velocity. Using MATLAB simulations we calculated the ideal combination of masses was 0.110 lb at the top, 0.806 lb in the middle, and 5.84 lbs at the bottom. Additionally, the ideal combination of spring constants was 15.0 lb/in for the top spring, 120 lb/in for the middle spring, and 912 lb/in for the bottom spring. Following these calculations, we assembled and tested a launcher using available materials that most closely matched the ideal values. Finally, we implemented image processing code in combination with a high-speed camera to measure the velocity of the projectile which we found to be 40.7 ft/s.

Part II: Predicted Masses and Spring Stiffnesses

Since our optimizer script assumed that any mass and spring stiffness was possible within the minimum and maximum given, we had to choose the masses and springs provided that were closest in mass and spring stiffness to those yielded by our script. Below are the tables with the optimal values on a continuous range and those provided on test day.

Table 1: Optimal Spring Stiffness and Masses Based on MATLAB Code		
	Spring Stiffness (lb/in)	Masses (lb)
Top	15.0	0.110
Middle	120	0.806
Bottom	912	5.84

Table 2: Optimal Spring Stiffness and Masses Provided on Test Day		
	Spring Stiffness (lb/in)	Masses (lb)
Top	15.0	0.110
Middle	134.4	0.829
Bottom	959	5.84

We ran both sets of masses and spring constants through our function “launch_velocity_function(design_vars)” to compute a launch velocity. It yielded these values:

	Any set of masses and springs	Provided masses and springs
Launch velocity	43.3 ft/s	43.2 ft/s

Table 3: Predicted Launch Velocities

Modifying the masses and spring constants to those provided on test day barely lowered our predicted launch velocity, so we had high expectations going into the BDW on test day.

Part III: Methods of Determining Optimal Masses and Spring Constants

III-A: Derivation of Equations of Motion

First, we drew out free body diagrams showing the forces acting on each mass. The compression of a spring is equal to the displacement of the top of the spring minus the displacement of the bottom of the spring. Applying Hooke’s Law, the force exerted by a spring is equal to the negative of the spring constant multiplied by this difference, which we called $x_a - x_b$.

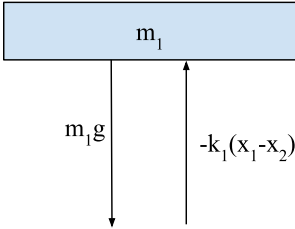
$$F_s = -k(x_a - x_b)$$

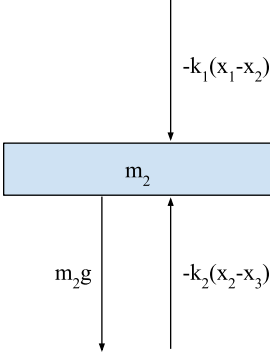
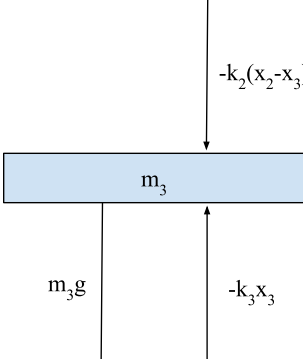
Formula (1): Hooke’s Law

Using our knowledge from ENGN40, we noted the velocity of a mass (v_a) is equal to the derivative of the displacement of that mass:

$$\frac{dx_1}{dt} = v_1 \quad \frac{dx_2}{dt} = v_2 \quad \frac{dx_3}{dt} = v_3$$

These derivations allowed us to split our second order differential equation for the force exerted by each spring a mass into two first order differential equations so that MATLAB’s first order ODE solver ode45 could solve them.

	$F = ma$ $m_1 \left(\frac{dv_1}{dt} \right) = k_1(x_1 - x_2) - m_1 g$ $\frac{dv_1}{dt} = \frac{-k_1(x_1 - x_2) - m_1 g}{m_1}$ $\frac{dx_1}{dt} = v_1$	<p>Equation (1)</p> <p>Equation (2)</p>
---	--	---

	$F = ma$ $m_2 \left(\frac{dv_2}{dt} \right) = k_2(x_2 - x_3) - k_1(x_1 - x_2) - m_2 g$ $\frac{dv_2}{dt} = \frac{-k_2(x_2 - x_3) + k_1(x_1 - x_2) - m_2 g}{m_2}$ <p style="text-align: right;">Equation (3)</p> $\frac{dx_2}{dt} = v_2$ <p style="text-align: right;">Equation (4)</p>
	$F = ma$ $m_3 \left(\frac{dv_3}{dt} \right) = -k_3 x_3 + k_2(x_2 - x_3) - m_3 g$ $\frac{dv_3}{dt} = \frac{-k_3 x_3 + k_2(x_2 - x_3) - m_3 g}{m_3}$ <p style="text-align: right;">Equation (5)</p> $\frac{dx_3}{dt} = v_3$ <p style="text-align: right;">Equation (6)</p>

III-B: MATLAB Script Outline

Once we had a system of differential equations, we were now ready to solve them with MATLAB's ode45 function and find three masses and three spring constants that would yield the highest launch velocity with MATLAB's fmincon function. We had to provide some more information so that the script could solve our equations: the acceleration of gravity at Earth's surface in inches per second squared, the minimum mass, the maximum mass, the minimum spring constant, the maximum spring constant, the estimated lengths of three masses and springs, the maximum sum of masses, the maximum length our contraption could be, when to stop calculating the ODE, and six initial conditions: three masses and three springs.

In the end, our MATLAB script worked like this:

-
- State initial masses and spring constants
 - State constraints on masses and spring constants
 - Call fmincon function given initial conditions and constraints
 - Call launch_velocity_function given design variables
 - Pull masses and spring constants from design variables
 - State lengths of masses and spring constants
 - Calculate velocity just before contact with equation for velocity in freefall

- State six initial conditions for six differential equations
 - Call ode45 function given design variables, rough time span, initial conditions, and event function (separation of top mass)
 - Return launch velocity, which is the last element in the velocities of the top mass
 - Call event_function function given design variables
 - Determine when the difference between the displacements of mass 1 and mass 2 crosses 0 from the negative direction
 - Call our_ode function given design variables
 - State differential equations describing the motion of three masses
-

After running the code, we found that the top mass should weigh 0.110 lb, the middle mass should weigh 0.806 lb, and the bottom mass should weigh 5.84 lbs. The top spring should have a spring constant of 15.0 lb/in, the middle spring should have a spring constant of 120 lb/in, and the bottom spring should have a spring constant of 912 lb/in. Comparing this with the masses and springs we were given, we will choose the top mass of 0.110 lb, middle mass of 0.829 lb, and the bottom mass of 5.84 lbs. Additionally we will choose the springs with spring constants 15 lb/in, 134.4 lb/in, and 959 lb/in, respectively.

Based on the ideal masses and springs, we predicted that the launch velocity would be 519 in/s, which is equivalent to 43.3 ft/s. However, when we applied the values of the available masses and springs, we found that the launch velocity would be 518.79 in/s which we rounded to three significant figures to get 519 in/s. Thus, we predicted the launch velocity given the masses and springs closest to the mass and spring constants found in our optimization to be 43.2 ft/s.

Part VI: Testing

VI-A: Setup

After a restful night's sleep in preparation for test day, we were eager to see how our optimized projectile launcher would perform. Starting off test day we assembled the launcher using the following materials:

Materials
<ul style="list-style-type: none"> ● Masses: <ul style="list-style-type: none"> ○ 0.110 lb - Washer (not McMaster) ○ 0.829 lb - Part# 7786T14 ○ 5.84 lb - Part# 4470T15 ● Springs: <ul style="list-style-type: none"> ○ 15 lb/in - Part# 9657K215 - Color: none

- 134.4 lb/in - Part# 96485K161 - Color: Yellow/Blue
- 959 lb/in - Part# 96485K262 - Color: Silver
- Metal rod
- High speed camera
- Length reference rod
- White background for camera
- Ceiling

Prior to testing, we confirmed that our design met the constraints of the project such that the total mass used did not exceed 8 lb, the launched mass exceeded 0.1 lb, the height of the topmost point of the stack of springs and masses didn't exceed 38 inches at the time of the drop, and the springs and masses were selected from the provided lists.

IV-B: Results

Utilizing the high-speed camera, we recorded each launch attempt. The “convertmovietoimages.m” script provided then converted this .mov file into a .jpg file for each frame. We then used MATLAB tracking software and the “track_masses.m” script provided to write a .csv file with time in the first column and the position of the respective masses in the next three columns.

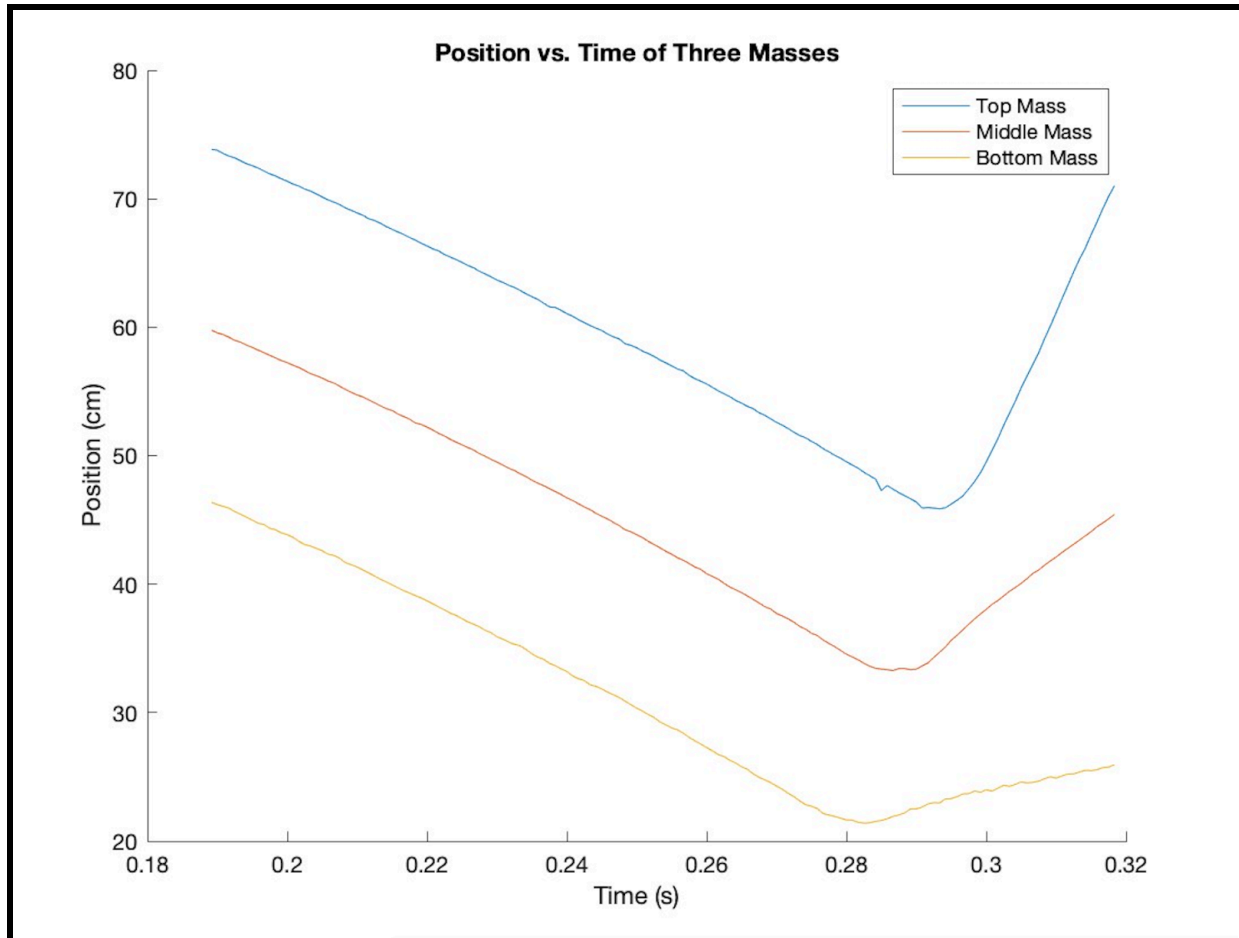


Figure 1: Position vs. Time of Three Masses

*See Appendix 2 for script for Figure (1)

Plotting position vs. time for each mass, we were able to visualize how our set of masses and springs is so efficient: the top mass accelerates to a speed (absolute slope of position) much greater than its speed just before contact.

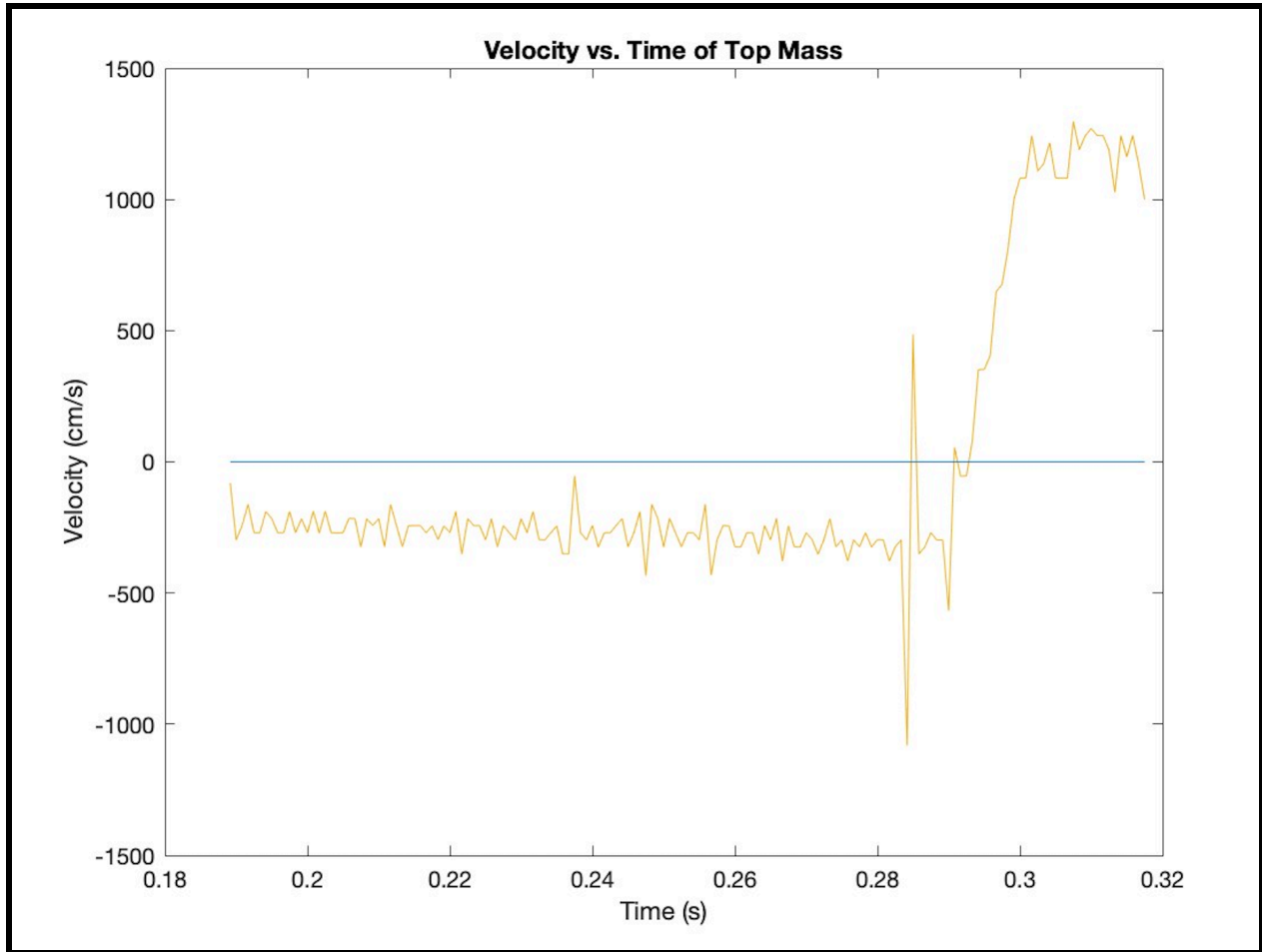


Figure 2: Velocity vs. Time of Top Mass

*See Appendix 2 for script for Figure (2)

While the velocity graph for the top mass is very rough, most likely due to the blurriness of the high speed camera not working well with the tracking software, it showed the launch velocity to be 1,240 cm/s, or 40.7 ft/s.

IV-C: Comparison of Predicted Velocity with Measured Velocity

We determined our experimental launch velocity is 40.7 ft/s. Compared to our predicted launch velocity of 43.2 ft/s, the experimental launch velocity was relatively close. As shown below, there is a 5.79% error between the two values:

$$\begin{aligned}
 \text{Percent Error} &= \left| \frac{E-T}{T} \right| * 100\% \\
 &= \left| \frac{40.7-43.2}{43.2} \right| * 100\% \\
 &= \left| \frac{-2.5}{43.2} \right| * 100\% \\
 &= |-0.0579| * 100\%
 \end{aligned}$$

$$= 0.0579 * 100\%$$

$$= 5.79\%$$

IV-D: Dynamic Efficiency of Device

% Energy Efficiency of Mass Launcher

```
clear all
close all
clc
```

% Step 1: Calculate the gravitational potential energy of the top mass based on its mass,
 % the gravity constant g, and the height that the group of masses and
 % springs falls before making contact with the bottom most spring.

```
weight = 0.110; % Weight of top mass in pounds
g = 386.0886; % Inches per second squared
mass = weight/g; % Mass of top mass in blobs
% Note that using the formula "U=mgh", this conversion becomes unnecessary,
% since one could just use U=weight*height, but it is nice to show.
```

```
hm1 = 0.125; % Height of top mass in inches
hm2 = 1; % Height of middle mass in inches
hm3 = 3; % Height of bottom mass in inches
hs1 = 5; % Height of top spring in inches
hs2 = 4; % Height of middle spring in inches
hs3 = 5; % Height of bottom spring in inches
hr = 38; % Height of rod in inches
fall_height = hr - (hm1+hm2+hm3+hs1+hs2+hs3);
```

```
U = mass*g*fall_height; % Gravitational potential energy in pound-inches.
% Step 2: Calculate the kinetic energy of the top mass based on its mass
% and launch velocity.
```

```
mass = mass;
launch_velocity = 488.4; % Launch velocity in inches per second
KE = 0.5*mass*launch_velocity^2; % Kinetic energy at launch in pound-inches
```

```
% Step 3: Find the ratio between them.
energy_ratio = KE/U; % Energy efficiency (dimensionless)
```

This script yields the value 15.5 for the dynamic efficiency of the device. Note that the mass launcher does not create energy. It transfers the gravitational potential energy of the entire assembly into kinetic energy, and optimizing the masses and springs means that the projectile

(the top mass) receives the largest possible portion of this energy given the constraints of the problem. Referring back to Figure (1), this is shown in how much the absolute slope of the position of the top mass changes compared to that of the other two masses.

IV-E: Note

During design testing, the projectile hit the ceiling in the BDW.

Going Further: Maximum BDW Ceiling Height Calculation

Using the launch velocity of the top mass and its mass, we can compute an estimate for the height of the BDW ceiling. Here is a MATLAB script that computes this:

```
% Calculate an estimate for the maximum possible height of the BDW ceiling
```

```
% Assume for simplicity that the projectile launches at the same height it  
% was at at the time of contact
```

```
hm1 = 0.125; % Height of top mass in inches  
hm2 = 1; % Height of middle mass in inches  
hm3 = 3; % Height of bottom mass in inches  
hs1 = 5; % Height of top spring in inches  
hs2 = 4; % Height of middle spring in inches  
hs3 = 5; % Height of bottom spring in inches  
height = hm1+hm2+hm3+hs1+hs2+hs3;
```

```
launch_velocity = 488.4;  
weight = 0.11;  
g = 386.0886;  
mass = weight/g;  
KE = 33.98;  
% KE at top = 0;  
% By work energy theorem, work done by gravity = 33.98-0 = 33.98  
gravity_work = KE - 0;  
U = gravity_work;  
% U = mg(h-height)  
% U = mgh - mg(height)  
% U + mg(height) = mgh  
% h = (U + mg(height))/(mg)  
h = (U + mass*g*height)/(mass*g);  
h_feet = h/12;
```

This script yields a ceiling height of 27.3 feet. Since the projectile hits the ceiling and bounces off, we know that the ceiling is actually lower than this, which is why this is an estimation for the maximum height of the ceiling.

Appendix 1: MATLAB Script to Optimize Masses and Spring Constants

% ENGN0040 Dynamics and Vibrations

% Brown University

% Project 1: Mass Launcher

% Group #28: Madison Goeke, Carter Smith, Henry Zamore, Baurice Kovatchev

clear all % Clear workspace

close all % Close all figures

clc % Clear command window

% This program calculates a set of three masses and three
% springs that will, after being dropped from a certain height, propel the
% top most mass into the air at a maximum velocity, given some constraints
% on the masses and springs, initial guesses for the masses of the
% masses and the spring constants of the springs, and estimated lengths of
% masses and springs.

% Constraints on masses and spring constants

init_vars = [.11, 5, 50, 15, 270, 830]; % Initial guesses for masses and spring constants

min_vars = [0.11, 0.11, 0.11, 15, 15, 15]; % Minimum masses and spring constants
% given in instructions

max_vars = [5.84, 5.84, 5.84, 1310, 1310, 1310]; % Maximum masses and spring constants
% given in instructions

% Constraint on total mass

% fmincon has the optional constraint $Ax \leq b$. To adhere to this format,
% "A" must be a row matrix that only adds the three masses in the design
% variables, and each only once, hence the first three ones. The spring constants
% must not be included in this inequality, hence the last three zeros

A = [1, 1, 1, 0, 0, 0]; % Coefficients of design variables

b = [8]; % Upper bound on matrix "A" times each design variable. This is a
% 1x1 matrix since "A", a 1x6 matrix, is to be multiplied by the the matrix
% of design variables, a 6x1 matrix, yielding a 1x1 matrix with the sum of
% component wise products

[opt_vars, opt_vel] = fmincon(@(design_vars) launch_velocity_function(design_vars), ...
init_vars, A, b, [], [], min_vars, max_vars); % Call fmincon given the above

% listed constraints and initial guesses. Output three masses and three
% spring constants that minimize negative velocity (and therefore maximize
% positive velocity), and output that negative velocity

opt_vel = -opt_vel; % Negative to cancel out the negative placed in the ...

% "Find Launch Velocity" function needed for fmincon

opt_vel_feetpersecond = opt_vel/12; % Convert to ft/s

%% Predicted Launch Velocity Given Optimized Masses and Springs Provided

```

% Masses and spring constants provided closest in absolute value to those
% yielded by the optimizer
predicted_vel = -1*launch_velocity_function([.11,.829,5.84,15,134.4,959]);
% Negative to cancel out the negative placed in the "Find Launch Velocity"
% function needed for fmincon
predicted_vel_feetpersecond = predicted_vel/12; % Convert to ft/s

%% Find Launch Velocity Given 3 Masses and 3 Spring Constants
function v1 = launch_velocity_function(design_vars) % Takes in 1x6 matrix "design_vars"
% with masses and spring constants and calculates launch velocity "v1"

g = 386.0886; % Acceleration of gravity on Earth's surface (inches per second squared)
m1 = design_vars(1); % Weight of top mass is stored in the first component of design_vars
m2 = design_vars(2); % Weight of middle mass is stored in the second component of design_vars
m3 = design_vars(3); % Weight of bottom mass is stored in the third component of design_vars
k1 = design_vars(4); % Spring constant of top spring is stored in the fourth component of design_vars
k2 = design_vars(5); % Spring constant of middle spring is stored in the fifth component of design_vars
k3 = design_vars(6); % Spring constant of bottom spring is stored in the sixth component of design_vars

m1=m1/g; % Converts weight of top mass in pounds to mass in blobs by dividng by g
m2=m2/g; % Converts weight of middle mass in pounds to mass in blobs by dividng by g
m3=m3/g; % Converts weight of bottom mass in pounds to mass in blobs by dividng by g

m1l = 0.15; % Estimated length of one mass (inches)
m2l = 0.5; % Estimated length of another mass (inches)
m3l = 1; % Estimated length of a third mass (inches)
s1l = 4; % Estimated length of one spring (inches)
s2l = 4; % Estimated length of another spring (inches)
s3l = 5; % Estimated length of a third spring (inches)
total_length = m1l+m2l+m3l+s1l+s2l+s3l; % Total length of contraption (inches)
rod_length = 38; % Given in instruction (inches)
v0 = -1*sqrt(2*g*(rod_length-total_length)); % Estimated velocity just before
% contact, using the equation for velocity in freefall

% Initial conditions for ODE's
init_x1 = 0; % Initial displacement of top mass
init_x2 = 0; % Initial displacement of middle mass
init_x3 = 0; % Initial displacement of bottom mass
init_v1 = v0; % Initial velocity of top mass
init_v2 = v0; % Initial velocity of middle mass
init_v3 = v0; % Initial velocity of bottom mass
init_w = [init_x1; init_x2; init_x3; init_v1; init_v2; init_v3]; % Column vector
% with six initial conditions needed to solve six differential equations

tspan = [0, 20]; % Time thought to be much longer than the time before the event
% function ends the calculation

```

```

options = odeset('RelTol',1e-10, 'Events',@(t,w) event_function(t,w,m1,m2,m3,k1,k2,k3));
% Adjustment to ode45 function: calls event function that detects the
% separation of the top mass

[t_plot, w_plot] = ode45(@(t,w) our_ode(t, w, m1, m2, m3, k1, k2, k3, g), tspan, init_w, options);

% Call ode45 function to solve the system of differential equations given
% the initial conditions, time span, and event function

v1 = -1*w_plot(end,4); % Velocity when event function is triggered is stored in
% the last element of fourth column of w_plot. Negative because fmincon is
% a minimizer, so in order to maximize this velocity, it has to minimize
% this velocity multiplied by -1.

end

%% Separation of Top Mass Event Function
function [ev,stop,dir] = event_function(t,w,m1,m2,m3,k1,k2,k3,g) % Detects when the top
% spring is no longer contracted and stops ODE calculation
x1 = w(1); % Displacement x1 is stored in the first element of w
x2 = w(2); % Displacement x2 is stored in the second element of w.
ev = x1-x2; % Event is detected when spring 1 is decompressed. This happens
% when x1-x2 = 0
stop = 1; % Stop if event occurs
dir = 1; % Detect only events when x1-x2 is increasing
end

%% ODE Definition
function dwdt = our_ode(t, w, m1, m2, m3, k1, k2, k3, g) % Takes in six design
% and the gravitational constant g in inches per second squared and outputs
% a matrix w as a function of time t

x1 = w(1); % Position of top mass is stored in the first component of w
x2 = w(2); % Position of middle mass is stored in the second component of w
x3 = w(3); % Position of bottom mass is stored in the third component of w
v1 = w(4); % Velocity of top mass is stored in the fourth component of w
v2 = w(5); % Velocity of middle mass is stored in the fifth component of w
v3 = w(6); % Velocity of bottom mass is stored in the sixth component of w

dv1dt = (-k1*(x1-x2) - m1*g)/m1; % Equation (1) in report
dx1dt = v1; % Equation (2) in report
dv2dt = (k1*(x1-x2) - k2*(x2-x3) - m2*g)/m2; % Equation (3) in report
dx2dt = v2; % Equation (4) in report
dv3dt = (k2*(x2-x3) - k3*x3 - m3*g)/m3; % Equation (5) in report
dx3dt = v3; % Equation (6) in report

```

```
dwdt = [dx1dt; dx2dt; dx3dt; dv1dt; dv2dt; dv3dt]; % Column vector containing  
% each equation in this system of first order differential equations
```

```
end
```

Appendix 2: MATLAB Script to Read .csv File and Plot Positions and Velocity vs. Time

% data_to_plots.m

data = readmatrix('mass_launcher_data.csv'); % Read .csv file

t = data(:,1); % Time is stored in first column of data

pos1 = data(:,2); % Position of top mass is stored in second column of data

pos2 = data(:,3); % Position of middle mass is stored in third column of data

pos3 = data(:,4); % Position of bottom mass is stored in fourth column of data

tv = t(1:end-1); % The velocity vector is one element shorter since it is calculated
% based on the difference of two values

vel1 = diff(pos1)/diff(t); % Calculating velocity of top mass

vel2 = diff(pos2)/diff(t); % Calculating velocity of middle mass

vel3 = diff(pos3)/diff(t); % Calculating velocity of bottom mass

figure(1) % Plot position of positions of the three masses vs. time

hold on

plot(t, pos1)

plot(t, pos2)

plot(t, pos3)

xlabel("Time (s)")

ylabel("Position (cm)")

title("Position vs. Time of Three Masses")

legend("Top Mass", "Middle Mass", "Bottom Mass")

hold off

figure(2) % Plot velocity of top mass vs. time

plot(tv, vel1)

xlabel("Time (s)")

ylabel("Velocity (cm/s)")

title("Velocity vs. Time of Top Mass")

hold off