PARABOLIC ENGINEERING

ENGR 1315-006 DESIGN PROJECT

Stone Preston, Josh Sowemimo, Kaytlyn Vavrecka, Kat Lubenow

Table of Contents

Introduction 3

Background 4

Methodology 5

- Design 5
  - Prototype 5
  - Final Design 10
  - Design of the MATLAB Program 17
  - Design References 21
- Materials 21
- Costs 22
- Preparation and Construction 23
- Testing 25
  - Live Experiments 25

Experimental Results 30

Discussion 33

Final Setup 35

Conclusions 36

Appendix 37

- Theoretical Data 39
- Experimental Data 40
- MATLAB Class Diagrams 43
- MATLAB Code 50
  - Launcher.m 50
  - LaunchSimulation.m 51
  - MissionControlModel.m 54
  - MissionControlView.m 54
  - MissionControlConroller.m 58
  - RocketLauncherSimulationGUI.m 62
  - ExportModel.m 66
  - ExportView.m 66
  - ExportController.m 67
  - ExportGUI.m 68

GuiHelpers.m 71
run.m 74
xlwrite.m 74
TestLauncher.m 79
TestLaunchSimulation.m 80
TestMissionControlModel.m 82
TestMissionControlView.m 83
TestMissionControlController.m 85
TestExportModel.m 86
TestExportView.m 87
TestExportController.m 88
TestGuiHelpers.m 89
TestRunner.m 92

Introduction

The purpose of this project was to utilize what we have learned throughout the semester to design and build a spring powered rocket launcher. The launcher needed to be able to hit a target between 8 and 15 meters away.

The main objectives for us were to write a MATLAB program that would enable us to accurately predict the necessary launch velocity and angle, design and build a launcher capable of hitting the target accurately and reliably, and learn as much as possible during the process.

Background

During the development of this project, we used many concepts we learned in class. Excel, MATLAB, kinematic equations, and unit conversions were the four most used concepts throughout this process. Excel was used to collect and organize data and compute experimental values. MATLAB was used to create a user interface that allowed the user to compute the required velocities for a desired distance, as well as the horizontal range, vertical range, and time of flight of a given launch velocity and launch angle. At the heart of the MATLAB code were the kinematic equations that were used to compute the projectile motion values. Because most of our data was recorded using imperial units, they had to be converted into metric units using the conversion skills we learned in class.

The desired outcome for this project was to be able to hit a target between 8 and 15 meters away. We also wanted to be able to apply the skills we learned in class to a practical application of engineering.

The military would benefit from a real world application of this project. Although our project is simple in nature, kinematics and projectile motion still apply to more complicated applications. The design could be modified to use a more powerful propulsion system instead of a spring. Robotics and other systems could be utilized to make a more precise launching system. The sky is the limit when it comes to rocketry.

Methodology

Our group designed a compression based launcher. We believed that the simplest solution would be the best solution, and this ideology is reflected in our design.

Design

Prototype

Some early sketches of our prototype launcher and projectile can be seen below. We wanted to be able to pivot the launcher arm around a screw to achieve different launch angles, and use holes drilled in the arm to change the position of the spring to achieve different velocities. The projectile design was a fairly basic rocket covered in foam.

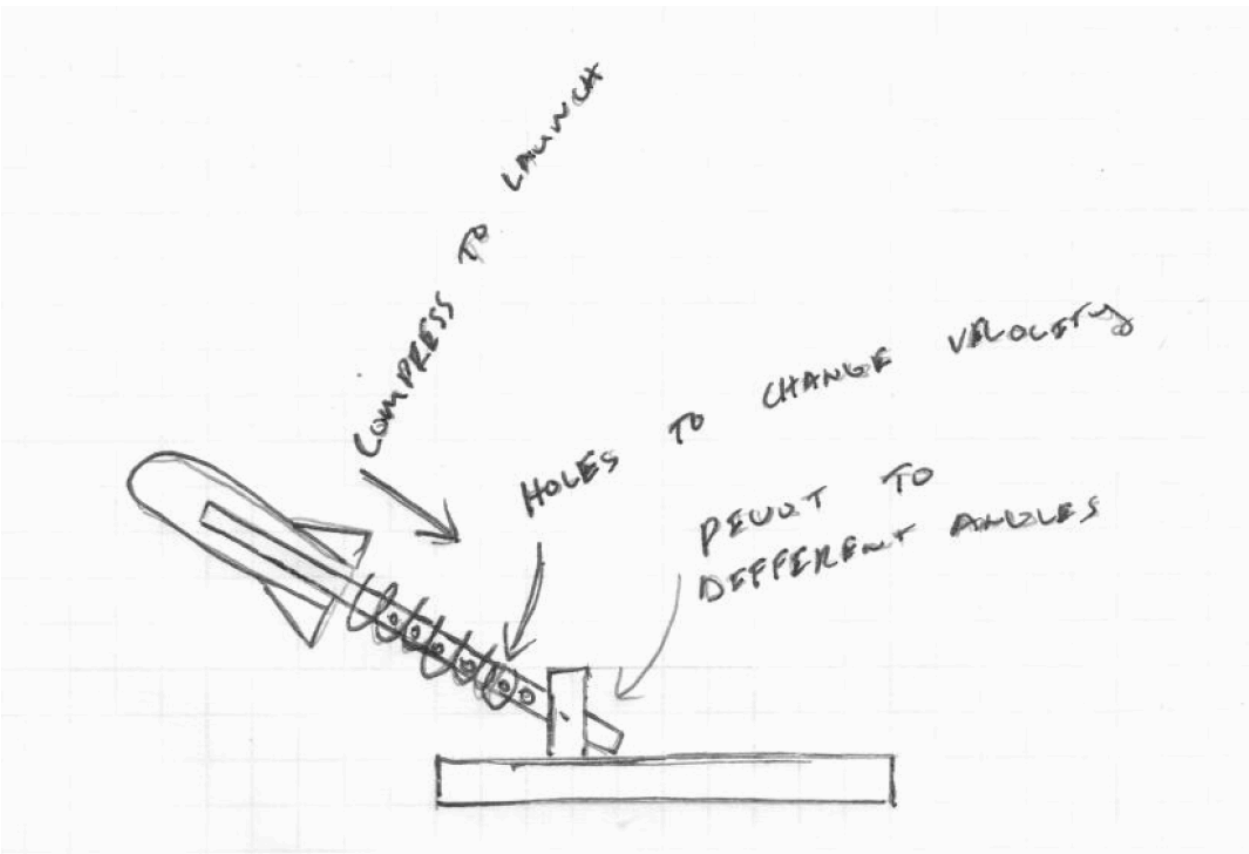


Figure 1: Prototype Launcher Design

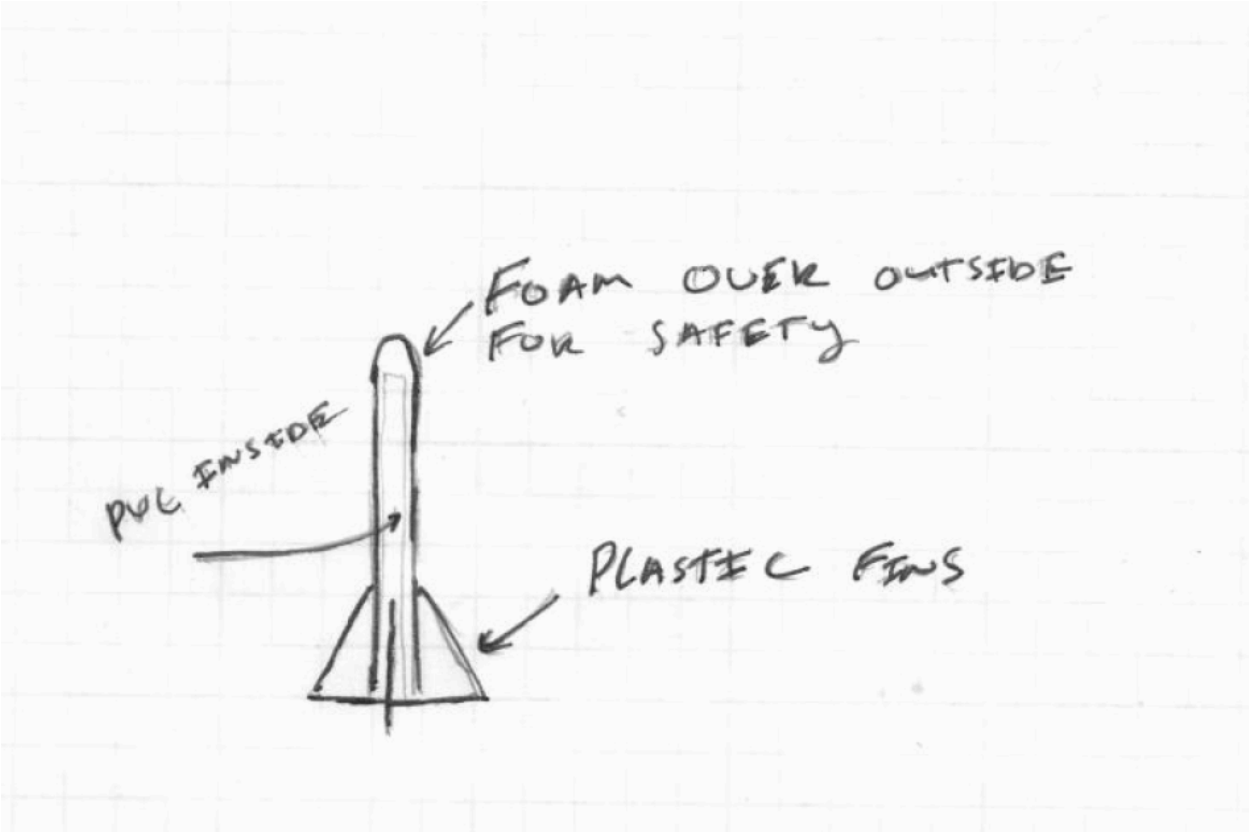


Figure 2: Prototype Projectile Design

After sketching a basic design, we discussed what basic properties the launcher should have. The main idea we all agreed on was that the launcher should have as few variables as possible. We decided the projectile should be launched from the same point on the arm regardless of the velocity. We also wanted to launch the rocket by pulling a pen to eliminate as much human error as we could.

With these tenets in place, we built a rudimentary prototype using materials that were on hand and a wooden dowel rod we purchased. We drilled a series of holes into the dowel rod to make it possible to change velocities. We also drilled a hole near the top of the rod that would serve as the launch point. A rough sketch of the arm is shown below.

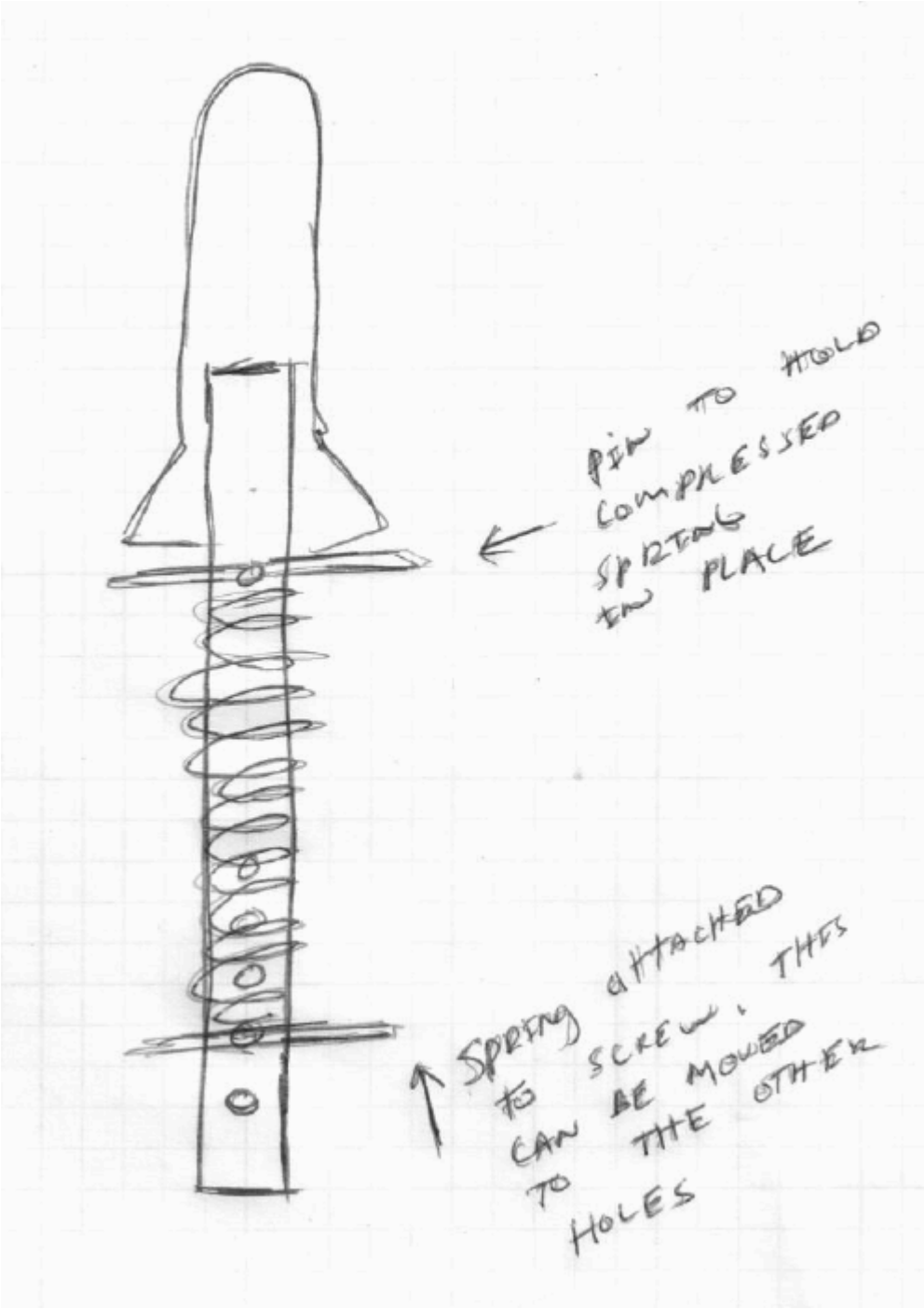


Figure 3: Prototype Launcher Arm Sketch

By placing the velocity holes at the bottom of the arm and the firing pin hole at the top, we ensured that the projectile would be launched from the same position along the arm every single time.

Figure 4: Prototype Velocity Settings

We cut a small mount for the arm from a 2x4 and fixed the launcher arm to the mount at around 30 degrees from the horizontal. The mount was then screwed to a wooden beam that served as the base.

Figure 5: Prototype Side View

We created a prototype rocket using a 10-inch piece of PVC pipe and pipe insulation foam. We cut plastic fins from an old bleach bottle and glued them to the rocket using an adhesive. The foam was

placed around the rocket and duct taped at the nose to create a nose cone. Since the PVC was hollow, the rocket slid over the launcher arm.

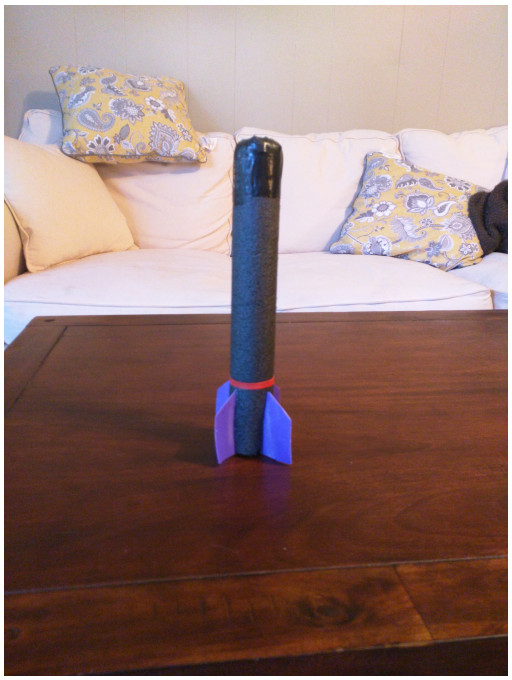


Figure 6: Rocket

We used two springs soldered together as the prototype spring. We quickly noticed a problem with the solder. It would catch and rub along the launcher arm which caused inconsistent launches.

The prototype functioned by placing a machine screw in one of the velocity holes. A washer was placed around the launcher arm and touching the screw. The washer was wired to the screw using thin wire. The 2 soldered springs were placed around the arm and the end of the spring series was wired to the washer. The user would then use a special PVC loader that we made to compress the spring to the firing pin hole and insert the pin. The rocket would then be launched by pulling the firing pin out of the whole in one swift motion. If the user wished to change the velocity, they simply moved the screw connected to the springs to a different hole.

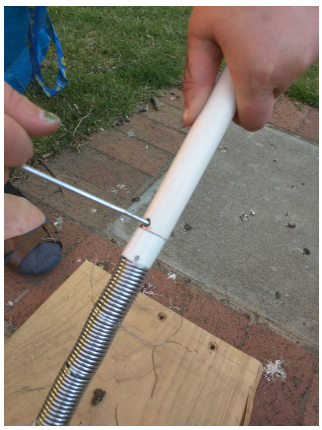


Figure 7: Loader being used to compress the spring

Final Design

Once we had the prototype launcher and projectile assembled, we performed several test launches to determine the effectiveness of our initial design. We were mainly interested in seeing how well the rocket flew, and how effective the launcher was at launching the rocket at different velocities. We tested the launcher and found it was relatively accurate, but lacked the power to reach the maximum distance required by the specifications. However, the prototype as a whole was functional and we decided to proceed with a final design that was similar to the prototype

The final design for the launcher did not change very much from our initial vision. We found that the prototype was not very stable due to its narrow base. Our final design used a larger base to stabilize the launcher during launch. We also decided to mount the launcher arm between two pieces of wood for added stability.

We decided it would be best to use a metal rod instead of a wooden one. The holes in the wooden rod wore down over time and caused the firing pin to release incorrectly. The metal rod would also decrease friction.

Instead of using multiple holes to change the velocity, we decided our design would be more flexible if we used PVC spacers of different sizes to increase or decrease the amount of spring compression. We made 6 spacers, starting with a quarter inch and increasing in quarter inch increments. In addition to these 6, we made a special eighth inch spacer that could be used to finely tune the velocity. Each spacer has a section taken out of it so that it is easy to snap onto the launcher arm. We numbered the spacers 1 through 6, starting with the quarter inch one. We used these numbers to refer to our velocities: v1, v2, v3, v4, v5, and v6. The velocity setting in which no spacers were utilized was referred to as v0. The larger the spacer, the greater the spring compression which leads to a greater initial velocity.

Figure 8: The 1.25 inch spacer

To increase the power of the launcher we utilized three springs in series instead of two. Instead of soldering them together, we wired them together using thin wire. We also wired small sections of PVC to the ends of our spring to make the ends flush.

The prototype rocket flew very well and we decided that we would not need to change any aspect of its design.

Once we finalized the changes that needed to be made to the prototype, we purchased and salvaged some materials to be used in the final design. After we knew what materials we would be working with, we created the parts using Inventor 2016. After modeling each part, they were brought together into an Inventor assembly and put together. This allowed us to see what the final launcher would look like before even beginning the final construction process.

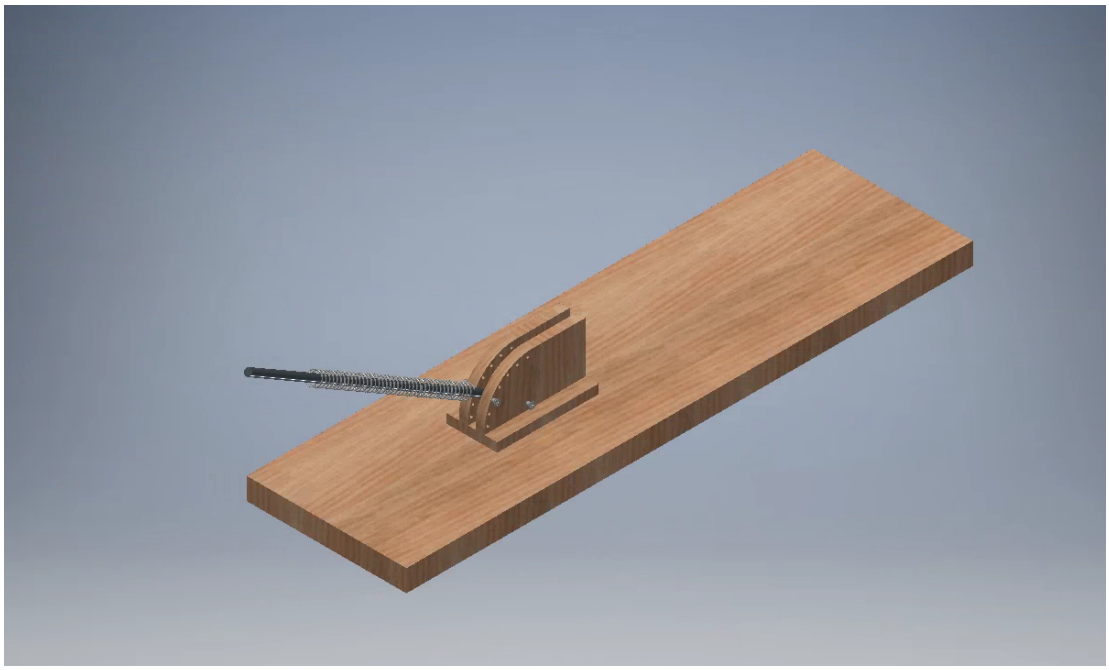


Figure 9: Launcher Assembly

After assembling the launcher in Inventor, drawings were created using the individual part files and assembly file.

Below is a drawing of the launcher base. We needed a large and heavy base for stability.

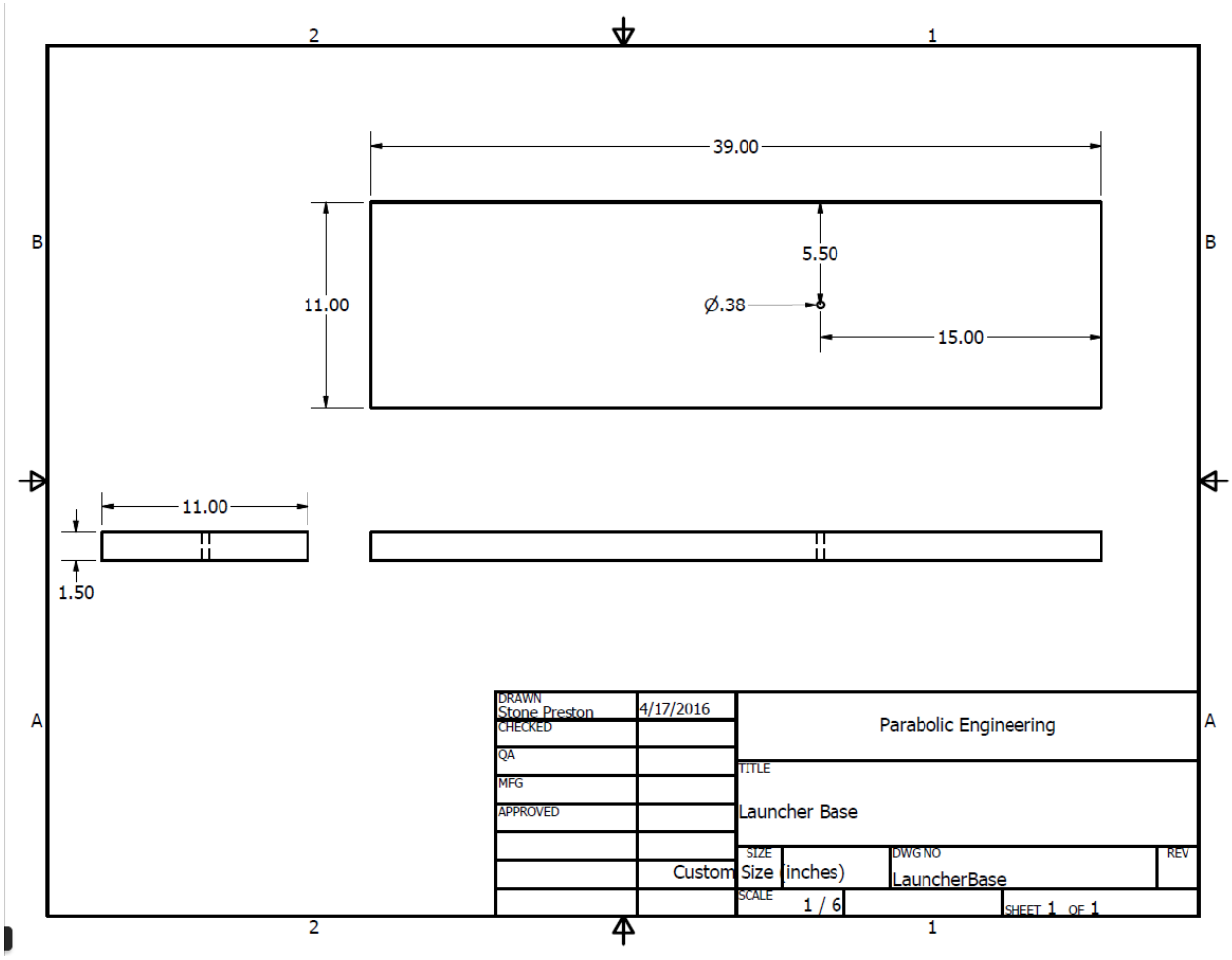


Figure 10: Launcher Base Drawing

The prototype arm was incredibly wobbly due to the fact that it was only attached to the mount on one side. To alleviate this issue, we decided that the arm would be mounted between two identical pieces of wood that would provide stability for the arm. These two side mounts would have holes drilled in them at designated angles. The launcher arm would be mounted to these pieces using two screws that could be removed in order to change the launch angle.

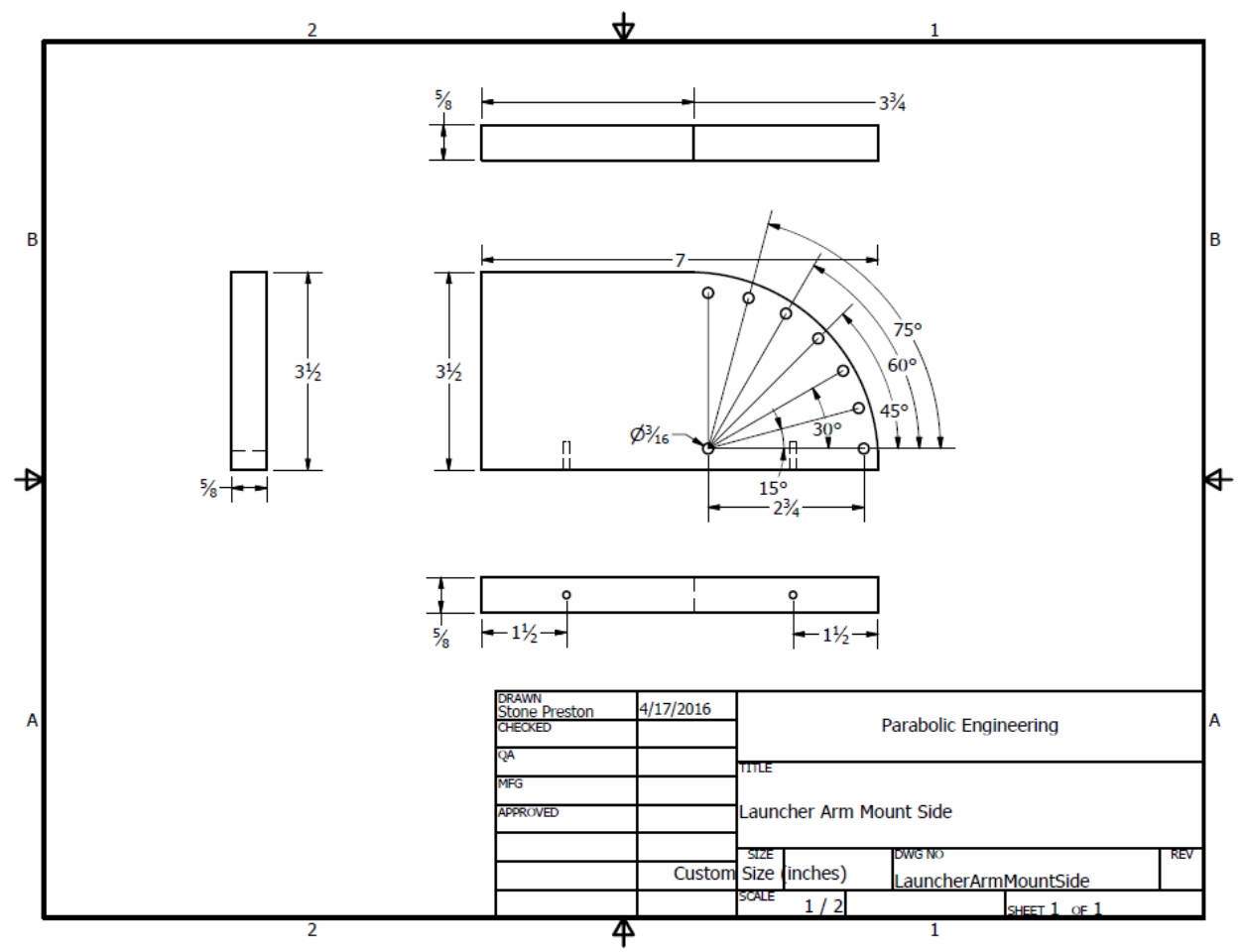


Figure 11: Arm Mount Side Drawing

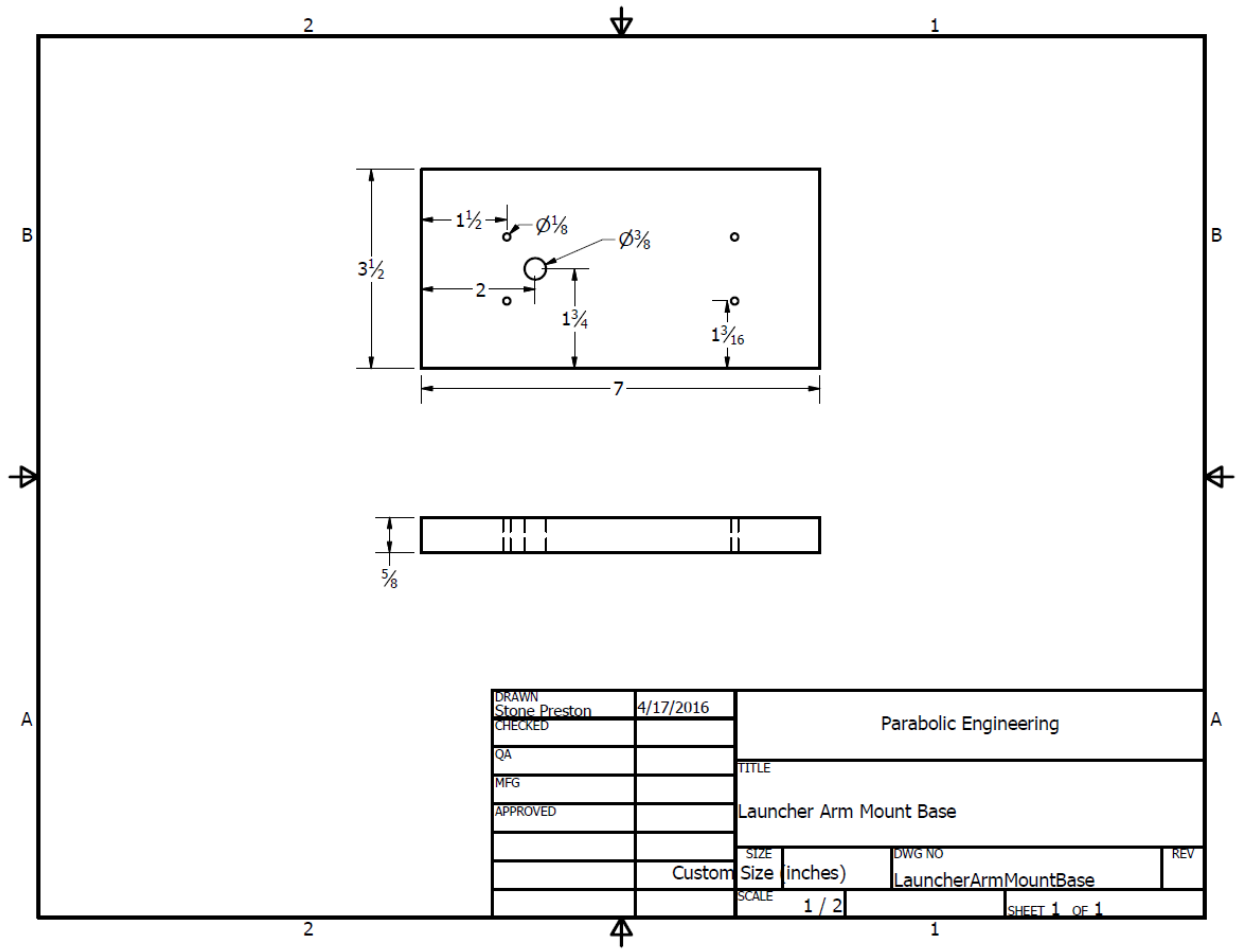


Figure 12: Arm Mount Base Drawing

The two arm mount side pieces would be attached to the arm mount base using wood screws. The arm mount base would then be bolted to the launcher base.

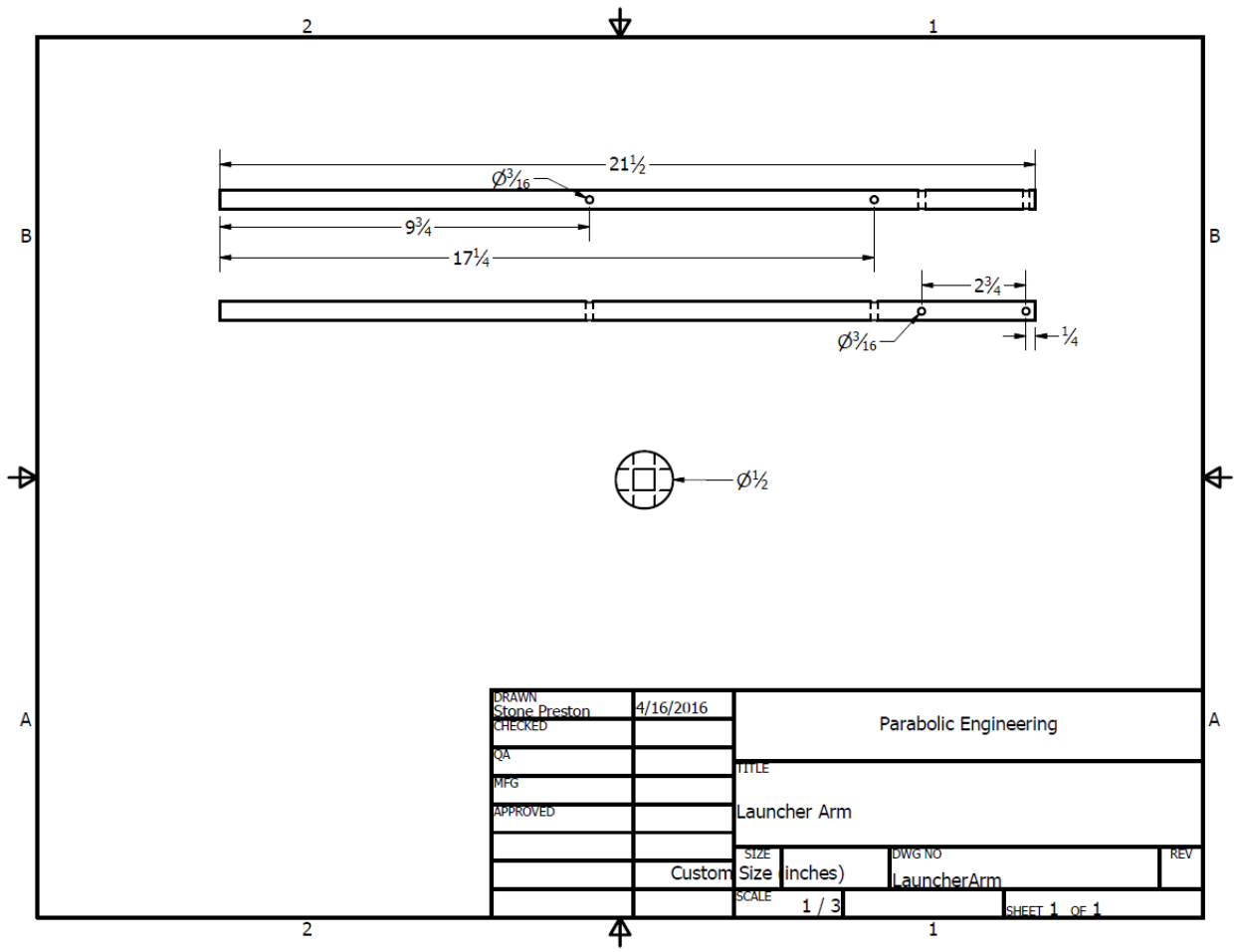


Figure 13: Launcher Arm

The figure above shows a drawing of the launcher arm. The leftmost hole is the firing pin hole. The hole directly to the right of that one is where the screw that keeps the spring attached goes. The two holes on the rightmost side of the arm are where the arm attaches to the mount.

The final launcher design functioned in a very similar way to the prototype. We wired three springs together and placed them around the arm. The end of the series of springs was wired directly to a small piece of PVC which was then wired to a screw at the end of the arm. The user would then use the same PVC loader to compress the spring to the firing pin hole and insert the pin. Once loaded, the rocket was slid onto the arm. The rocket would then be launched by pulling the firing pin out of the hole in one swift motion. The launch angle could be changed by removing the machine screw from the arm mount, pivoting the arm to the desired angle, and then placing the machine screw back through the hole to secure the arm.

The main change from the prototype to the final design was the way that the launch velocity was changed by utilizing different spacers. Instead of placing the screw that was attached to the springs in a different hole, the user simply loosened the wire assembly at the end of the arm, snapped the desired spacer on to the arm between the springs and the screw, and then tightened the wire. This

design gave us an easy way to change the velocity that did not rely on physical holes drilled into the arm. By making more spacers, we could achieve a wide range of velocities.

Figure 14: Wire assembly and spacer

Design of the MATLAB Program

The MATLAB program started out as a simple text based program. At the heart of the program were the Launcher and LaunchSimulation classes. The class diagrams for those classes can be seen below. Class diagrams for other classes used in the program are located in the appendix.

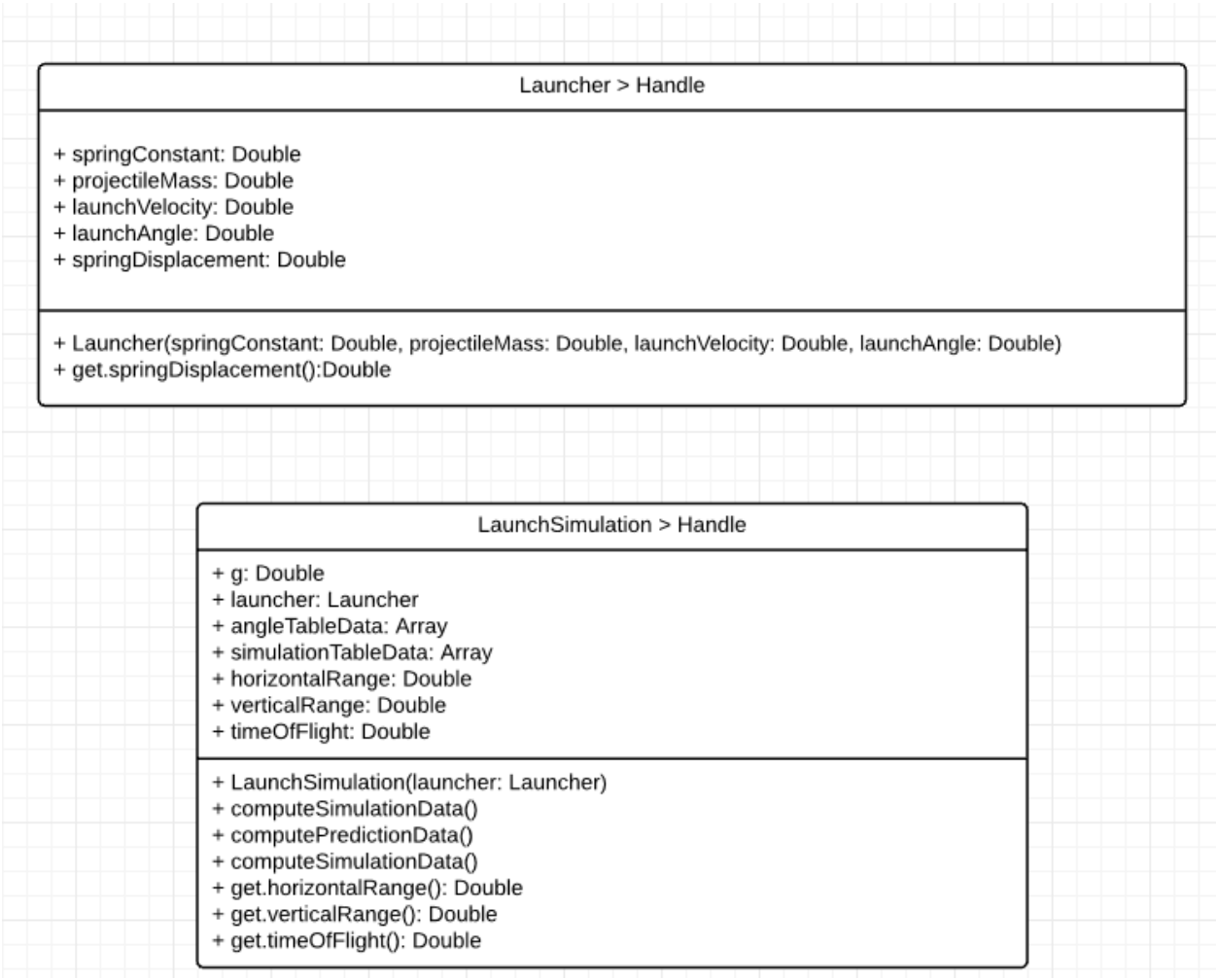


Figure 15: Class diagrams of the Launcher and LaunchSimulation classes

By using the object-oriented features of MATLAB, we were able to model the real world launcher object using code.

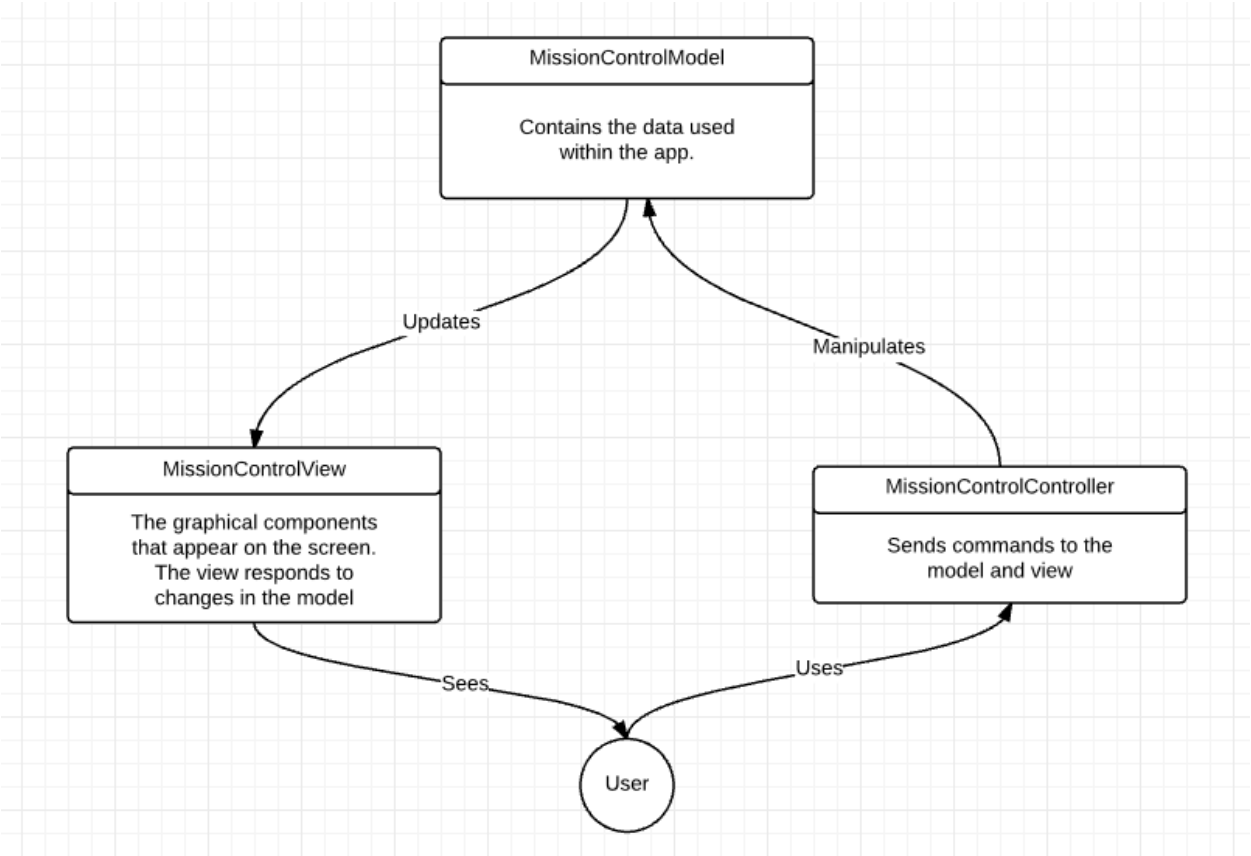
After creating a text based program, we decided to create a program that utilized a graphical user interface to manipulate the launcher object and display the simulation values. This was a long and extensive learning process.

We created a basic interface called Mission Control using GUIDE, the GUI editor that ships with MATLAB. GUIDE generates code as you build the interface. The code was a series of nested functions that closely resembled the spaghetti code of the past. It was extremely messy and hard to follow. Passing data around in the program required the use of structs such as the guidata struct. This quickly became cumbersome, especially when wanting to pass data between interfaces. We decided that we needed a way to alleviate some of the problems that using GUIDE-generated code caused.

In order to clean up the design of the program, we decided to implement the Model View Controller (MVC) design pattern. This is a tried and tested design pattern that is used in many different graphical applications. iOS, Android, and many other frameworks utilize this design pattern.

The MVC pattern is based on several simple but powerful ideas. The model object can communicate with the controller, the view object can communicate with the controller, and the controller can communicate with both the model and the view. This is the only communication allowed to happen between the objects.

We created several other classes to facilitate the pattern. When the user clicks a button, an event is fired in the MissionControlView. The controller responds to that event and manipulates the MissionControlModel. The MissionControlView listens for changes to the model object and updates itself accordingly. Note that it does not directly communicate with the model, it only listens for changes. This cycle can be seen in the diagram below.



The MissionControlView consists of labels, text fields, buttons, tables, and axes. The user can input data into the text fields and press buttons to fire events. When a button is pressed, the controller responds by manipulating the model using the data provided in the text fields. The view detects that the model has changed, and responds by displaying some data in a table, or graphing the trajectory of a projectile on an axes.

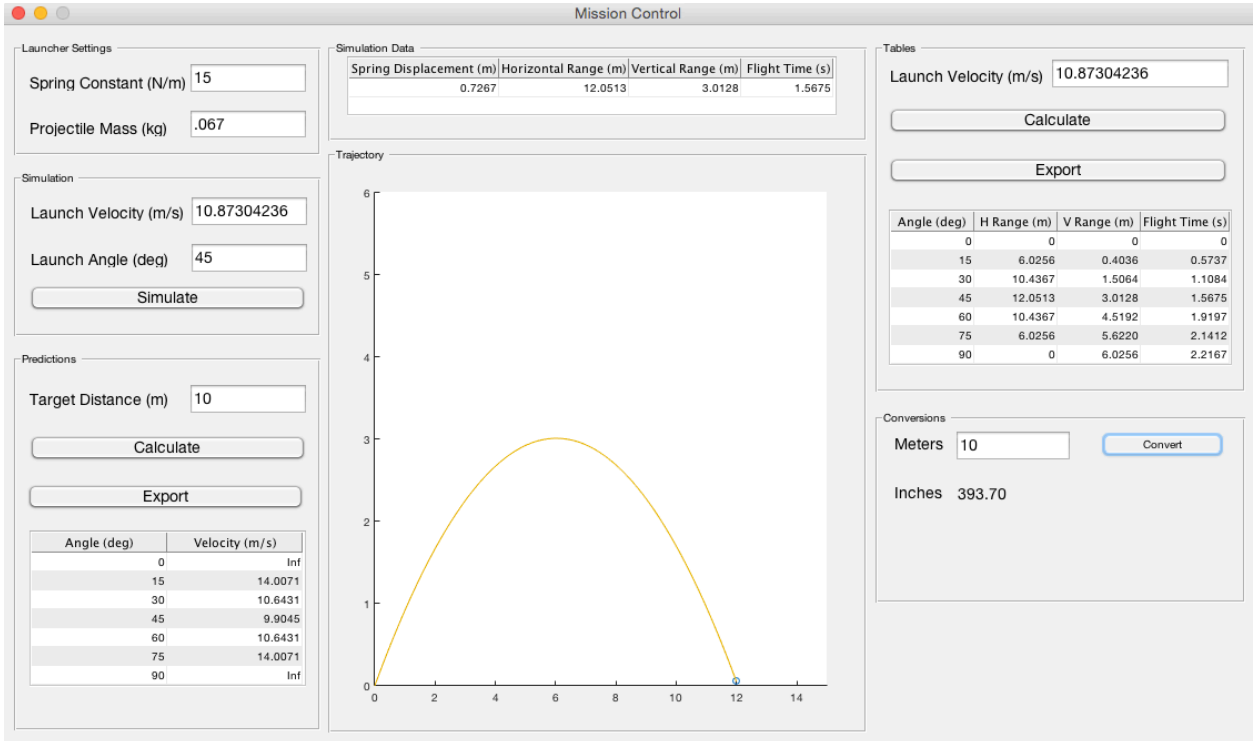


Figure 16: The Mission Control interface

In addition to creating a Mission Control interface, we also created an Export interface that allowed the user of the program to export the data contained in the tables directly to Excel.

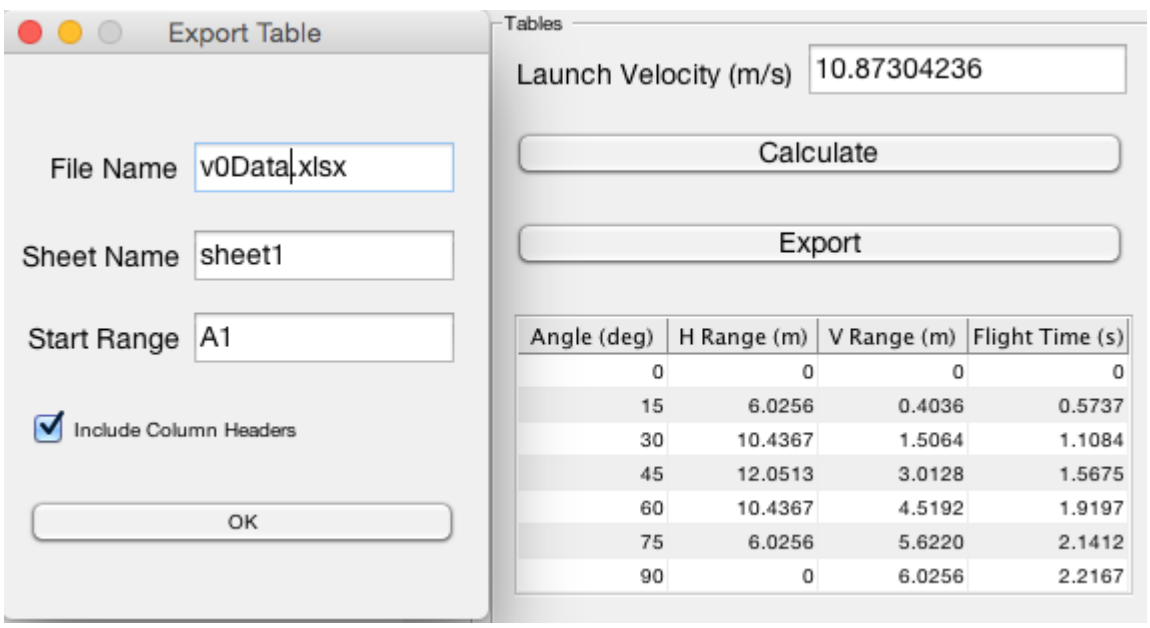


Figure 17: The export feature of the program

Utilizing the MVC pattern allowed us to pass data between the Mission Control and Export interfaces with ease. While implementing the export feature, we ran into a major problem. The xlswrite function that is native to MATLAB only works on Windows machines. This is because that function uses

the COM which only exists in the windows environment. To get around this, we found a program on the MATLAB file exchange called xlwrite, which uses open source Apache java libraries in order to write data to Excel, regardless of the operating system. The xlwrite program can be seen in the appendix.

We also decided that since the program was growing rather large and complex, it would be a good idea to write automated tests for the methods used in the application.

We wrote unit tests using the MATLAB unit test framework for almost all the methods in the app. These tests ensured that the methods function as expected. There are even a few tests that test functionality of the interface like input validation. These tests could then be run any time a change was made to the program, and they would catch any bugs introduced by adding new features. Our test suite allowed us to update and change code with confidence.

Design References

The following references were used extensively during the design process. We had very little MATLAB and Inventor experience, so these references were extremely helpful.

1. Lynda.com Inventor 2016 Essential Training (http://www.lynda.com/Inventor-tutorials/Exploring-initial-drawing-creation/373550/410924-4.html)
2. The MATLAB Central Forums (http://www.mathworks.com/matlabcentral)
3. Stack Overflow post on implementing MVC in MATLAB (http://stackoverflow.com/questions/20064610/whats-the-right-way-to-organize-gui-code)
4. MathWorks MATLAB videos (http://blogs.mathworks.com/videos/)
5. Xlwrite program (http://www.mathworks.com/matlabcentral/fileexchange/38591-xlwrite--generate-xls-x--files-without-excel-on-mac-linux-win)

Materials

The base of the launcher is a large and heavy piece of wood Stone's dad had in his garage that he let us use. The launcher arm was cut from a ½ inch metal rod. The base and sides of the launcher arm mount were cut from a small fence picket using a scroll saw.

Figure 18: Fence Picket

The rocket was made with a 10-inch piece of PVC pipe, pipe insulation, duct tape, and a bleach bottle.

Figure 19: Rocket Materials

Costs

Since we are college students, keeping the price of materials low while still creating an efficient launcher and projectile was very important to us. Because of this, we tried to salvage as many materials as we could. Luckily, Stone's family had just finished building a fence and had quite a lot of lumber and other materials on hand already. We were able to minimize costs by using supplies that we already had on hand.

Prototype Launcher				
Item	**Quantity**	**Unit Cost**	**Total Cost**	**Source**
Wood Beam	1	$ -	$ -	On Hand
2x4	1	$ -	$ -	On Hand
Dowel	1	$ 1.72	$ 1.72	Home Depot
Spring	2	$ 4.38	$ 8.76	Home Depot
Box of Wood Screws	1	$ 8.47	$ 8.47	Home Depot
Washer	1	$ -	$ -	On Hand
		Total	**$ 18.95**	

Table 1: Prototype Launcher Costs

Rocket				
Item	**Quantity**	**Unit Cost**	**Total Cost**	**Source**
10 ft PVC Pipe	1	$ 3.93	$ 3.93	Home Depot
6 ft PVC Pipe	1	$ 1.39	$	Home

Insulation			1.39	Depot
Roll of Duct Tape	1	$ -	$ -	On Hand
Bleach Bottle	1	$ -	$ -	On Hand
Adhesive	1	$ -	$ -	On Hand
		Total	**$ 5.32**	

Table 2: Rocket Costs

Final Launcher				
Item	Quantity	Unit Cost	Total Cost	Source
Wood Board	1	$ -	$ -	On Hand
Fence Picket	1	$ -	$ -	On Hand
Metal Rod	1	$ 4.73	$ 4.73	Home Depot
Spring	3	$ 4.38	$ 13.14	Home Depot
Box of Wood Screws	1	$ -	$ -	Same screws from prototype
10 ft PVC Pipe	1	$ -	$ -	Leftover from rocket
Box of Washers	1	$ 1.18	$ 1.18	Home Depot
Box of Nuts	1	$ 2.77	$ 2.77	Home Depot
Box of Machine Screws	1	$ 6.21	$ 6.21	Home Depot
Large Bolt and Nut	1	$ -	$ -	On Hand
		Total	**$ 28.03**	

Table 3: Final Launcher Costs

Preparation and Construction

Once we had our final design in place, it was time to create and assemble the launcher. We utilized Stone's grandfather's workshop in order to complete the launcher construction.

The large board would serve as the base of the launcher. A hole was drilled into the board in order to fasten the arm mount to the base. After the hole was drilled, nothing else needed to be done to the base of the launcher.

The arm mount sides and base were cut from the fence picket. The two sides of the mount were clamped together and holes were drilled into them using a drill press. The sides were then fastened to the mount using 2 screws in each side piece.

Figure 20: We utilized the drill press extensively during the construction phase

The base of the mount was then bolted to the launcher base. We also added screws to each of the corners of the base of the arm mount to offer further stability.

The arm of the launcher was cut from a metal rod using an angle grinder. The drill press was used to drill the holes in the arm. The arm was then secured to the mount using 2 machine screws. One of the screws is designed to be removed to change the launch angle.

Testing

Live Experiments

Figure 21: Testing setup

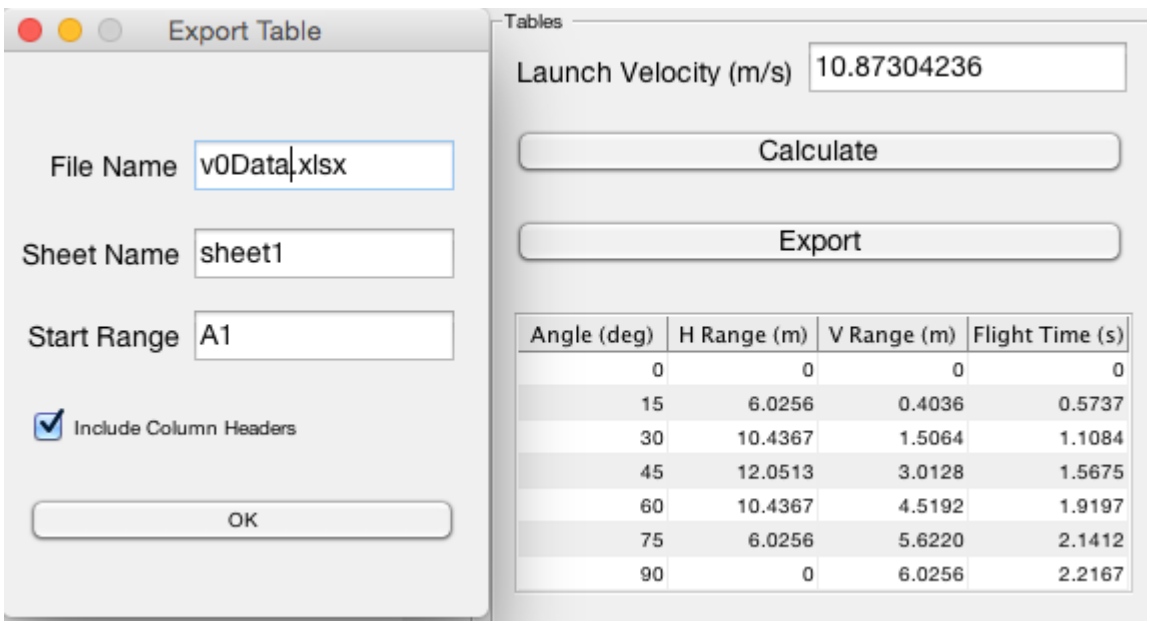
Testing the launcher was a lengthy process due to the fact that we had 7 different velocities. Once we got the launcher set up we weighed it down with concrete blocks. This prevented the launcher from moving during setup and launch. We laid a tape measure out along the path of the projectile. After setup, it was time to begin testing the different angles and velocities. We decided we would not be using angles above 45 since they would cause the projectile to hit the ceiling. Because of this, we only collected data for the angles we would be using for launches. The testing process relies on 4 people. The launch master fires the rocket and is in charge of the testing. The data recorder records the data in the spreadsheet. The spotters mark and call out where the rocket landed. The testing process is shown as a series of steps below.

1. Data recorder gets ready to record horizontal range data in spreadsheet
2. Launch master checks spring connections and makes sure wires are tight
3. Spotters position themselves down range
4. Launch master compresses spring and inserts firing pin

5. Launch master places rocket over the launcher arm
6. Launch master makes sure the testing team is ready
7. Launch master pulls the pin and fires the rocket
8. Spotters track the projectile in the air and watch closely where it lands. This distance is marked using a piece of pipe and called out so the data recorder can record the horizontal range. The pipe is used to gauge the accuracy of the launcher between trials. The projectile should land near the same spot each trial. Once marked, the spotters return the rocket to the launch master and the process begins again.

The process above was repeated 3 times for each possible angle and velocity combination.

The experimental data was used to compute the launch velocities of the different spacers. Details of these calculations are covered in the experimental results section. These velocities were then input into the MATLAB interface and exported to Excel using the export feature of the program.



Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	6.0256	0.4036	0.5737
30	10.4367	1.5064	1.1084
45	12.0513	3.0128	1.5675
60	10.4367	4.5192	1.9197
75	6.0256	5.6220	2.1412
90	0	6.0256	2.2167

The MATLAB program uses formulas based on kinematic equations in order to compute the theoretical values.

The principal kinematic equation governing projectile motion is

$$S_1 = S_0 + v_0 t + \frac{1}{2} a t^2 \#\#(1)\#$$

This equation was then manipulated into the equation for vertical displacement at ground level. Since we are on earth, -g was used in place of a.

$$S_y = v_0 \sin(\theta)\, t - \frac{1}{2} g t^2 \#(2)$$

Since the rocket is fired from ground level and then hits the ground at ground level, the vertical displacement is 0.

$$0 = v_0 \sin\theta\, t - \frac{1}{2} g t^2 \#(3)$$

We then used equation 3 to solve for the time of flight.

$$t = \frac{2v_0 sin\theta}{g} \#(4)$$

Equation 4 is used in the LaunchSimulation.m MATLAB class file to compute the time of flight.

```
% timeOfFlight
function value = get.timeOfFlight(this)

    value = ((2 * this.launcher.launchVelocity) * sind(this.launcher.launchAngle)) / LaunchSimulation.g;

end
```

Figure 22: Time of flight getter method

We derived the equation for horizontal displacement using equation 1.

$$S_x = v_0 \cos(\theta)\, t \#(5)$$

We then plugged the t value from equation 4 into equation 5 which gave us

$$S_x = v_0 \cos(\theta) \frac{2v_0 sin\theta}{g} \#(6)$$

Using the double angle trigonometric identity, we were left with

$$Range_x = \frac{v_0^2 sin2\theta}{g} \#(7)$$

This formula is used in the LaunchSimulation.m MATLAB class file to compute the horizontal range.

```
% horizontalRange
function value = get.horizontalRange(this)

    value = ((this.launcher.launchVelocity)^2 * sind(2 * this.launcher.launchAngle)) / LaunchSimulation.g;

end
```

Figure 23: Horizontal range getter method

When the projectile reaches its maximum height, the y component of the velocity is equal to zero. The y component of velocity is given by

$$v_y = v_0 sin\theta - gt \#(8)$$

Setting the value of the y component of velocity equal to 0 and solving for t gave us

$$t = \frac{v_0 sin\theta}{g} \#(9)$$

We then plugged the t value into equation 2. This left us with

$$Range_y = \frac{v_0^2 \sin^2 \theta}{2g} \#(10)$$

Equation 10 is used in the LaunchSimulation.m MATLAB class file to compute the vertical range.

```
% verticalRange
function value = get.verticalRange(this)

    value = ((this.launcher.launchVelocity^2) * sind(this.launcher.launchAngle)^2) / (2 * LaunchSimulation.g);

end
```

Figure 24: Vertical range getter method

Tables containing the theoretical horizontal ranges for each velocity and angle are shown below. The full exported theoretical tables that contain the horizontal ranges, vertical ranges, and time of flights are located in the appendix. The velocity values were calculated using the results of the experimental data.

v0	10.8730423607493 (m/s)		v1	11.1736436004412 (m/s)
Angle (deg)	H Range (m)		Angle (deg)	H Range (m)
0	0		0	0
15	6.025639662		15	6.363420556
30	10.43671404		30	11.02176771
45	12.05127932		45	12.72684111
60	10.43671404		60	11.02176771
75	6.025639662		75	6.363420556
90	0		90	0

Table 4: Theoretical horizontal range data for v0 and v1

v2	11.5575913062968 (m/s)		v3	11.8302806812403 (m/s)
Angle (deg)	H Range (m)		Angle (deg)	H Range (m)
0	0		0	0
15	6.808252645		15	7.133309937
30	11.79223949		30	12.35525524
45	13.61650529		45	14.26661987
60	11.79223949		60	12.35525524
75	6.808252645		75	7.133309937
90	0		90	0

Table 5: Theoretical horizontal range data for v2 and v3

v4	11.992997728834 (m/s)		v5	12.6003652215469 (m/s)

Angle (deg)	H Range (m)		Angle (deg)	H Range (m)
0	0		0	0
15	7.330886572		15	8.092212216
30	12.69746801		30	14.0161227
45	14.66177314		45	16.18442443
60	12.69746801		60	14.0161227
75	7.330886572		75	8.092212216
90	0		90	0

Table 6: Theoretical horizontal range data for v4 and v5

v6	12.9098034634847 (m/s)
Angle (deg)	H Range (m)
0	0
15	8.494547675
30	14.71298816
45	16.98909535
60	14.71298816
75	8.494547675
90	0

Table 7: Theoretical horizontal range data for v6

Experimental Results

The horizontal range experiments were conducted on two separate days. The first day of testing we collected data for all velocities at 15 and 30 degrees. This took approximately 3 hours to complete. The data for the 45-degree launch angle was collected on a separate day and was conducted outside in Stone's backyard due to the high vertical range of the projectile. This took approximately an hour and a half to complete. The testing was done using the same process that was listed previously.

Since we performed 3 trials of each angle and velocity, the horizontal ranges of each setting were averaged. The data was originally collected in inches, so these averages were then converted to meters by multiplying by .0254. The averages of the horizontal ranges in meters for each angle and velocity are shown in tables below. The original tables of experimental data in inches are listed in the appendix.

v0			v1	
Angle (deg)	H Range (m)		Angle (deg)	H Range (m)
15	6.544733333		15	7.078133333
30	9.906		30	11.24373333
45	11.6586		45	11.13366667

Table 8: Experimental horizontal range data for v0 and v1

v2			v3	
Angle (deg)	H Range (m)		Angle (deg)	H Range (m)
15	7.3914		15	7.501466667
30	11.5062		30	12.53913333
45	12.81853333		45	13.34346667

Table 9: Experimental horizontal range data for v2 and v3

v4			v5	
Angle (deg)	H Range (m)		Angle (deg)	H Range (m)
15	7.789333333		15	8.4836
30	13.57206667		30	15.07913333
45	12.827		45	14.26633333

Table 10: Experimental horizontal range data for v4 and v5

v6	
Angle (deg)	H Range (m)
15	9.414933333
30	15.52786667
45	14.3764

Table 11: Experimental horizontal range data for v6

Equation 7 was rearranged in order to solve for the initial velocity.

$$v_0 = \sqrt{\frac{(g)(Range_x)}{\sin(2\theta)}} \#(11)$$

The initial velocity was computed for each angle and velocity setting and compiled into tables.

v0			v1	
Angle (deg)	Velocity (m/s)		Angle (deg)	Velocity (m/s)
15	11.33171073		15	11.78443787
30	10.59298481		30	11.28559488
45	10.69443154		45	10.45089805

Table 12: Initial velocities for v0 and v1

v2			v3	
Angle (deg)	Velocity (m/s)		Angle (deg)	Velocity (m/s)
15	12.04239461		15	12.13172601
30	11.41655694		30	11.91798812
45	11.21382236		45	11.44112792

Table 13: Initial velocities for v2 and v3

v4			v5	
Angle (deg)	Velocity (m/s)		Angle (deg)	Velocity (m/s)
15	12.36231046		15	12.90148178
30	12.39915759		30	13.06945201
45	11.21752513		45	11.83016188

Table 14: Initial velocities for v4 and v5

v6	
Angle (deg)	Velocity (m/s)
15	13.5912101
30	13.26249046
45	11.87570983

Table 15: Initial velocities for v6

The values for each velocity were then averaged together.

v0	v1	v2
10.87304236	11.17364	11.557591

Table 16: Average velocities for v0, v1, and v2

v3	v4	v5	v6
11.83028068	11.993	12.600365	12.90980346

Table 17: Average velocities for v3, v4, v5, and v6

Once we had the averages for the initial velocities, we could input them into the MATLAB program and compare the experimental horizontal ranges to the theoretical horizontal ranges.

Discussion

The experimental and theoretical results for the horizontal range of v0 are shown below.

v0	10.8730423607493 (m/s)
Angle (deg)	H Range (m)
15	6.544733333
30	9.906
45	11.6586

Table 18: Experimental data for the range of v0

v0	10.8730423607493 (m/s)
Angle (deg)	H Range (m)
15	6.025639662
30	10.43671404
45	12.05127932

Table 19: Theoretical data for the range of v0

The initial velocity of the projectile is governed by the following equation.

$$v_0 = x\sqrt{\frac{k}{m}} \#(12)$$

This means that the velocity depends on the displacement of the spring, the spring constant of the spring, and the mass of the projectile.

The mass of the spring and spring constant were held constant during launches. Because we used spacers to increase or decrease the amount of spring compression, we were able to standardize the launch velocities. Using the v1 spacer should always cause a launch velocity that is very close to v1 each time. The design of the launcher lends itself to minimizing the number of variables present in and between launches. This property helped make our data consistent and close to that of the theoretical data.

The two data sets are fairly close, although they do differ by around .5 meters. This variance is likely due to a variety of factors.

One of the most obvious factors is human error. It was not possible for the launch master to compress the spring, load the rocket, and release the spring in the exact same manner each time the rocket was launched. This led to small differences between launches.

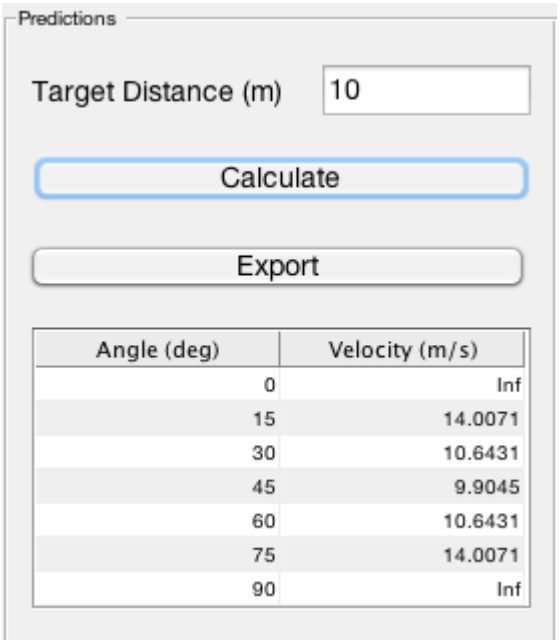
In addition to not being able to launch the rocket in the same manner each time, it was also hard to track the rocket in the air and mark exactly where it landed. We did our best to get as close as possible, but unfortunately error was still present.

Atmospheric conditions likely played a part as well. The theoretical calculations assumed no air resistance. However, air resistance was present during the experimental launches. The air resistance may have affected the horizontal ranges of the projectile significantly. We also tested the 45-degree angle on a relatively windy Lubbock afternoon. The gusts of wind most likely altered the path of the projectile in some way, which could explain the difference between the experimental and theoretical values.

Optimizing the launch would mean introducing as few variables as possible between launches. We could minimize human error by using a robot or machine to pull the pin with the same amount of force each time. We could also set up a high speed camera and go through frame by frame to find the point of contact of the rocket. This would allow us to obtain very accurate values for the horizontal range of the projectile. We could minimize environmental factors like wind by launching inside a large building with high ceilings like a gymnasium.

Final Setup

On the final launch day, we will measure the distance from the launcher to the center of the target. We will then input that distance into our MATLAB program. The program will return the velocities required to hit the target at various angles.



Angle (deg)	Velocity (m/s)
0	Inf
15	14.0071
30	10.6431
45	9.9045
60	10.6431
75	14.0071
90	Inf

Figure 25: Velocity prediction

We will then look at our experimental results and see which experimental velocity is closest to the theoretical velocity. We will limit ourselves to launch angles of either 15 or 30 degrees so we do not hit the ceiling on our launch.

Once we know what velocity we need, we will snap the corresponding spacer onto the launcher arm and get the launcher ready to launch. We will then fire the launcher. If we do not hit the target, we will have to apply some adjustments. If we need more distance, we can apply a larger spacer or use the current one in combination with the eighth inch spacer to dial in the velocity. We can also raise the launch angle. If we need less distance, we can use a smaller spacer or lower the launch angle.

Using the knowledge we have gained during development and testing, we should be able to hit the target using three launch attempts.

Conclusions

The project was a very enjoyable experience. We got a small taste of what it is like to be engineers and what it takes to design and build a program and device from scratch. Although the process went relatively smoothly, it was not immune to problems and issues.

We quickly learned that compression springs are not as easy to find as tension springs. Most of the springs we came across were either extremely small and weak, or large and impossible to compress. This was a huge issue for us because the spring is the most important part of the launcher. We were unable to find a single spring that would work, and were forced to use 3 springs in series which was less than ideal. Unfortunately, our problems did not stop at spring availability

After a first few launches we noticed that the projectile was coming off the launcher arm awkwardly, and we thought this might be due to the way the bare spring wire was interacting with the projectile during launch. We wired small pieces of PVC to the spring that the projectile could sit on. This PVC surface helped the rocket launch in a smoother fashion. It is a key component of the launcher design.

We learned so many things throughout the duration of the project. We learned how to model and draw parts using a parametric modeling program. We learned how to use some unfamiliar power tools like a drill press. We gained project management and communication skills. We also discovered what it is like to take mathematical theory into MATLAB and create a fully functioning app based on that theory. This project proved to be a valuable learning experience for every single one of us.

Time management and scheduling was something our team excelled at. We got started on the MATLAB portion of the project very early on in the semester. Since it was based purely on theory, we were able to complete it with ease even though we did not have a launcher design. We created a simple app with a graphical user interface that makes it easy to simulate a launch and create and export theoretical data.

Our team communicated well together. Everyone was very responsive in the group text and willing to meet. Files were shared using OneDrive so everyone had access to all aspects of the project. The MATLAB code was version controlled using Git and hosted on a Bitbucket team site so everyone could easily access the latest version of the code. Although we did many things well, we also made some bad decisions.

We waited until we had a month left to start designing and building the launcher. This did not leave much time for testing and problem solving. If we had encountered any large issues, it could have been devastating to the project timeline and put the entire project at risk. Fortunately, we did not encounter any large issues with the design.

If we had to do the project again, we would go with a tension based design since tension springs are much easier to find and there are many more varieties. We would also start designing and building the launcher around the same time we started the MATLAB code. Although we finished on time, it would have been nice to have more time to test and perfect the design.

Overall, we thoroughly enjoyed every aspect of the project. We all learned something new from this experience.

Appendix

Figure 26: The Parabolic Engineering Team – Kaytlyn Vavrecka, Josh Sowemimo, Kat Lubenow, and Stone Preston

Figure 27: The Launcher and Rocket

Theoretical Data

v0	10.8730423607493 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	6.025639662	0.40364132	0.573730977
30	10.43671404	1.506409915	1.108363136
45	12.05127932	3.012819831	1.567462178
60	10.43671404	4.519229746	1.919741264
75	6.025639662	5.621998341	2.141193155
90	0	6.025639662	2.216726271
v1	11.1736436004412 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	6.363420556	0.42626835	0.589592613
30	11.02176771	1.590855139	1.139005464
45	12.72684111	3.181710278	1.610796975
60	11.02176771	4.772565417	1.972815333
75	6.363420556	5.937152206	2.200389588
90	0	6.363420556	2.278010928
v2	11.5575913062968 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	6.808252645	0.45606645	0.60985214
30	11.79223949	1.702063161	1.178143864
45	13.61650529	3.404126322	1.666147031
60	11.79223949	5.106189484	2.040605032
75	6.808252645	6.352186195	2.275999171
90	0	6.808252645	2.356287729
v3	11.8302806812403 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	7.133309937	0.477841159	0.624240968
30	12.35525524	1.783327484	1.205940946
45	14.26661987	3.566654969	1.705458041
60	12.35525524	5.349982453	2.088750989
75	7.133309937	6.655468778	2.329699009

90	0	7.133309937	2.411881892
v4	11.992997728834 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	7.330886572	0.491076284	0.632826956
30	12.69746801	1.832721643	1.222527801
45	14.66177314	3.665443286	1.728915397
60	12.69746801	5.498164929	2.117480265
75	7.330886572	6.839810288	2.361742353
90	0	7.330886572	2.445055602
v5	12.6003652215469 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	8.092212216	0.542075432	0.664875534
30	14.0161227	2.023053054	1.284440899
45	16.18442443	4.046106108	1.81647374
60	14.0161227	6.069159162	2.224716897
75	8.092212216	7.550136784	2.481349274
90	0	8.092212216	2.568881798
v6	12.9098034634847 (m/s)		
Angle (deg)	H Range (m)	V Range (m)	Flight Time (s)
0	0	0	0
15	8.494547675	0.569026797	0.681203467
30	14.71298816	2.123636919	1.315984043
45	16.98909535	4.247273837	1.861082481
60	14.71298816	6.370910756	2.279351224
75	8.494547675	7.925520877	2.542285948
90	0	8.494547675	2.631968086

Table 20: Theoretical data exported from our MATLAB program

Experimental Data

	Trial 1			**Trial 1**
v0			v1	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	249		15	287
30	392		30	443

45	462		45	458
	Trial 2			**Trial 2**
v0			v1	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	262		15	262
30	392		30	443
45	458		45	440
	Trial 3			**Trial 3**
v0			v1	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	262		15	287
30	386		30	442
45	457		45	417

Table 21: Experimental data for v0 and v1

	Trial 1			**Trial 1**
v2			v3	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	292		15	298
30	446		30	496
45	505		45	518
	Trial 2			**Trial 2**
v2			v3	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	292		15	298
30	462		30	494
45	522		45	558
	Trial 3			**Trial 3**
v2			v3	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	289		15	290
30	451		30	491

45	487		45	500

Table 22: Experimental data for v2 and v3

	Trial 1			**Trial 1**
v4			v5	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	306		15	334
30	532		30	602
45	490		45	568
	Trial 2			**Trial 2**
v4			v5	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	306		15	334
30	534		30	594
45	485		45	566
	Trial 3			**Trial 3**
v4			v5	
Angle	Horizontal Distance (in)		Angle	Horizontal Distance (in)
15	308		15	334
30	537		30	585
45	540		45	551

Table 23: Experimental data for v4 and v5

	Trial 1
v6	
Angle	Horizontal Distance (in)
15	372
30	613
45	573
	Trial 2
v6	
Angle	Horizontal Distance (in)
15	370
30	613

45	525
	Trial 3
v6	
Angle	Horizontal Distance (in)
15	370
30	608
45	600

Table 24: Experimental data for v6

MATLAB Class Diagrams

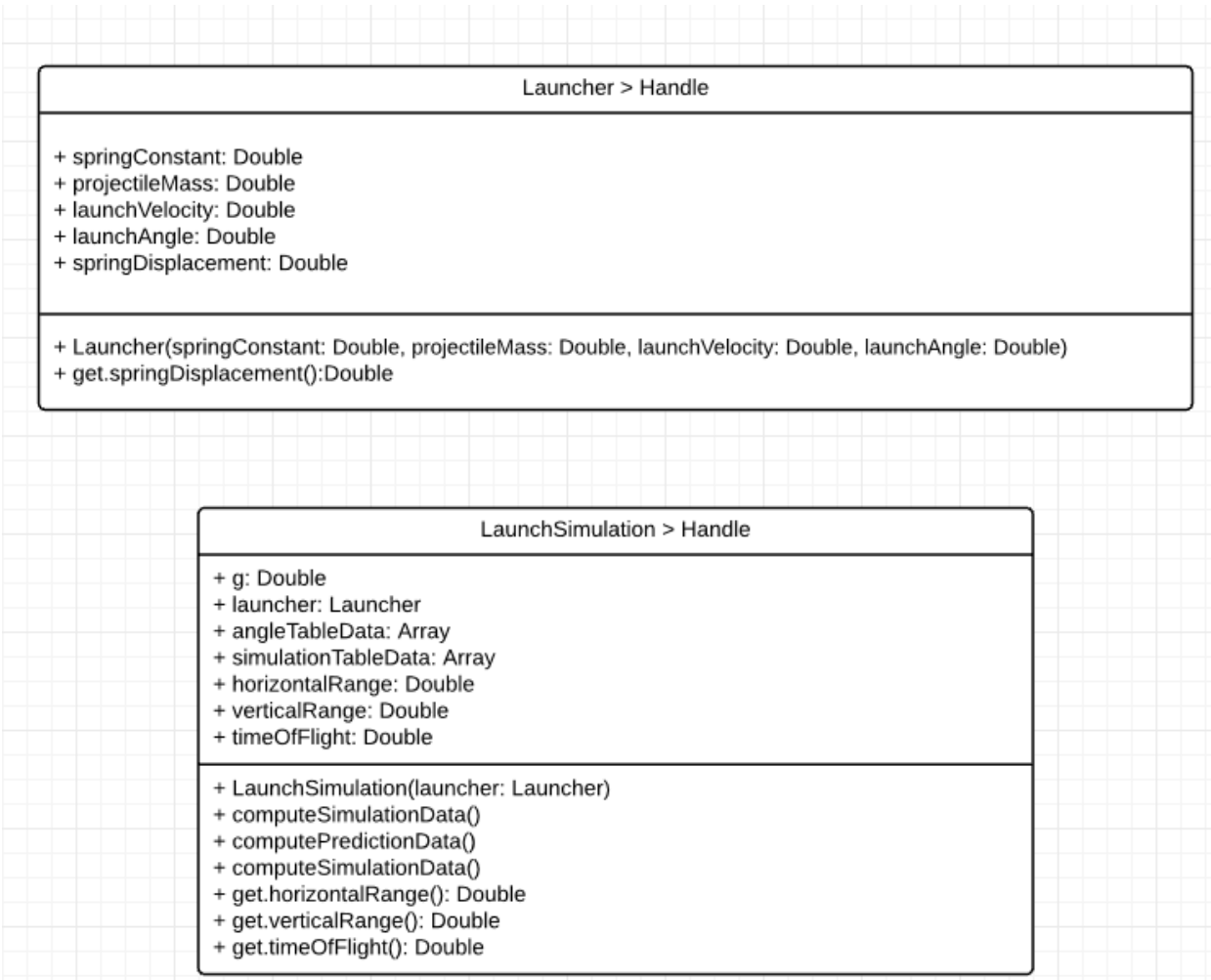


Figure 28: Launcher and LaunchSimulation class diagrams

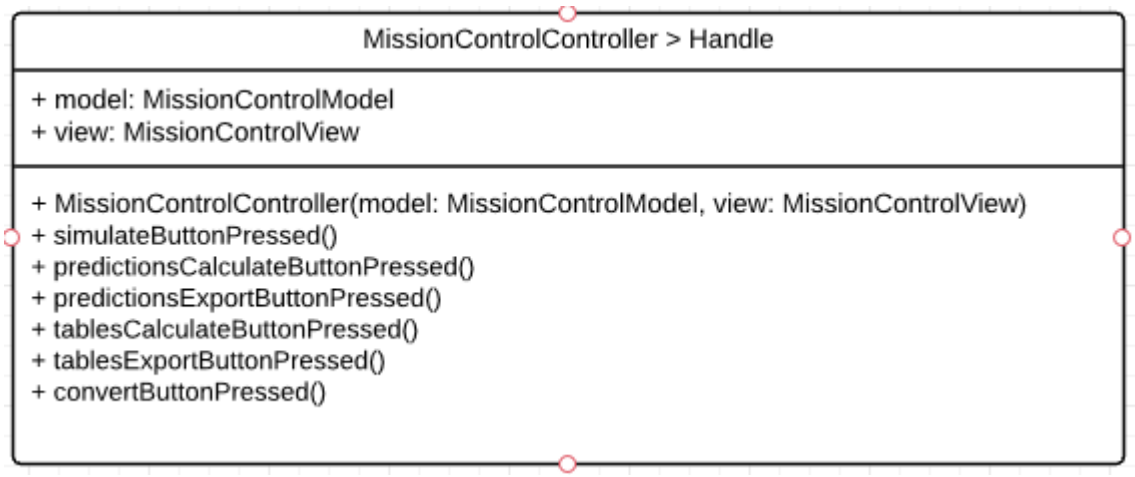


Figure 29: MissionControlController class diagram

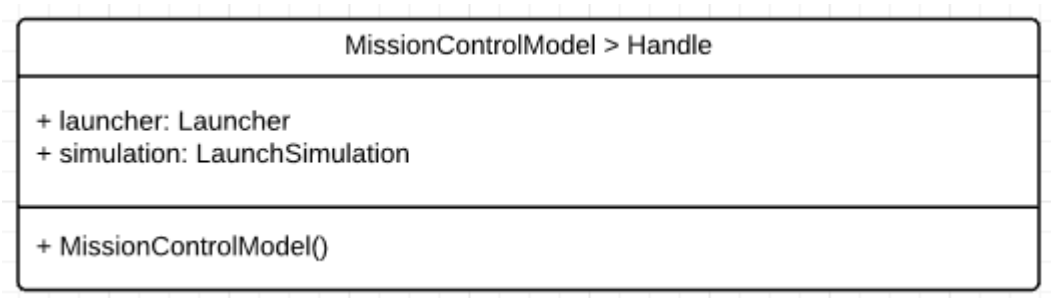


Figure 30: MissionControlModel class diagram

Figure 31: MissionControlView class diagram

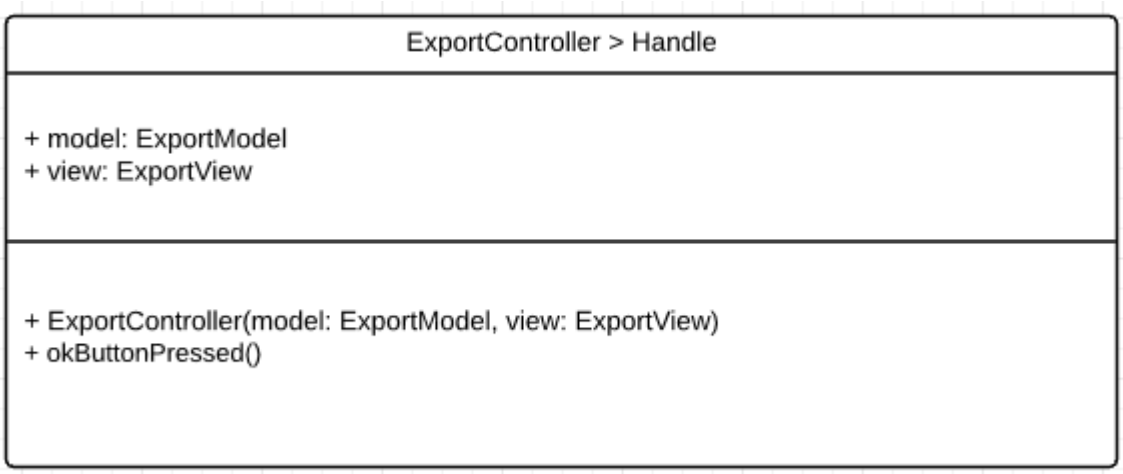


Figure 32: ExportController class diagram

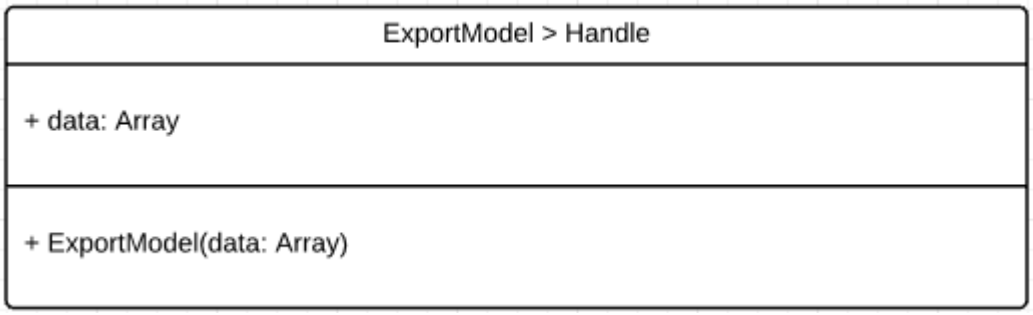


Figure 33: ExportModel class diagram

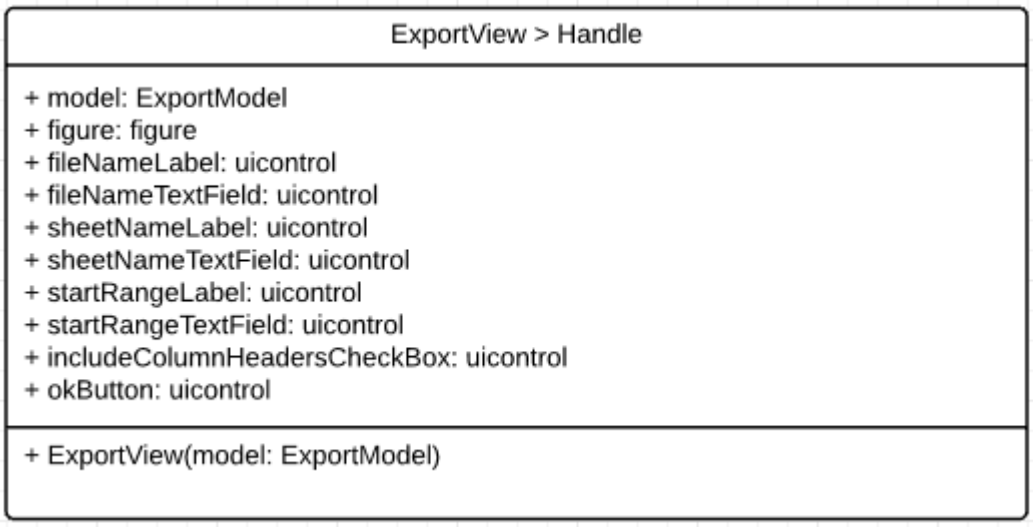


Figure 34: ExportView class diagram

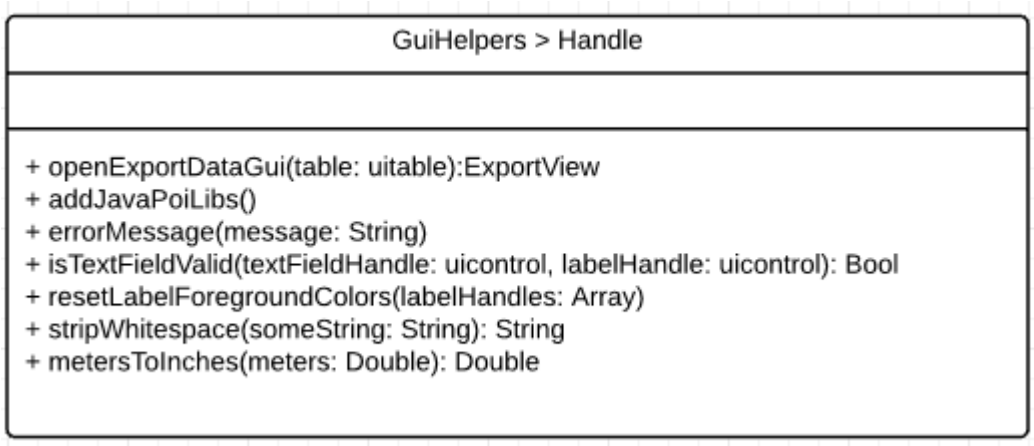


Figure 35: GuiHelpers class diagram

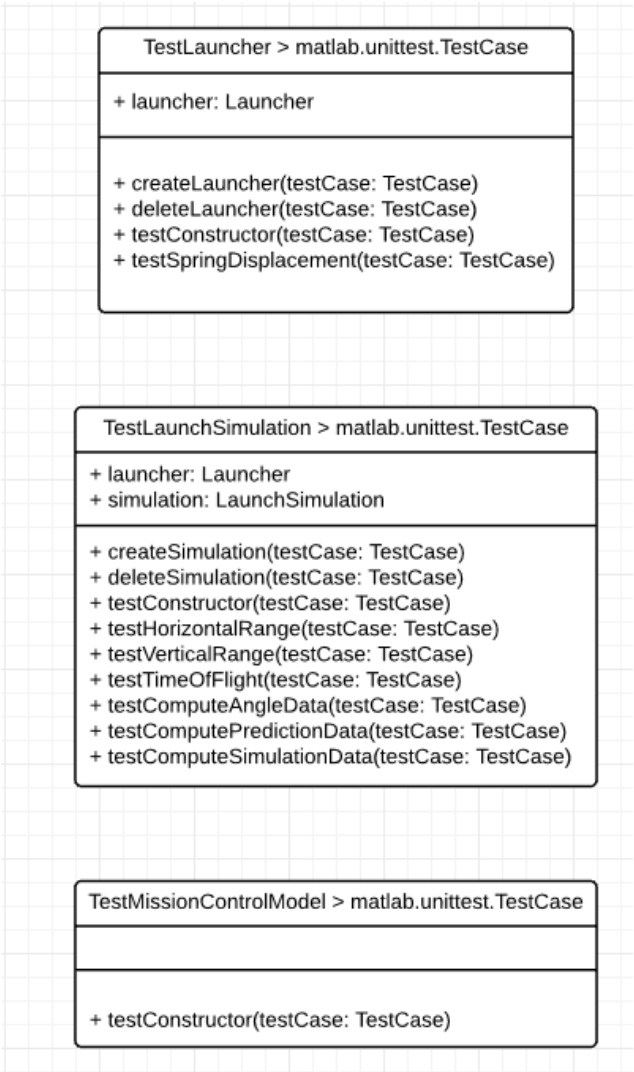


Figure 36:Test class diagrams

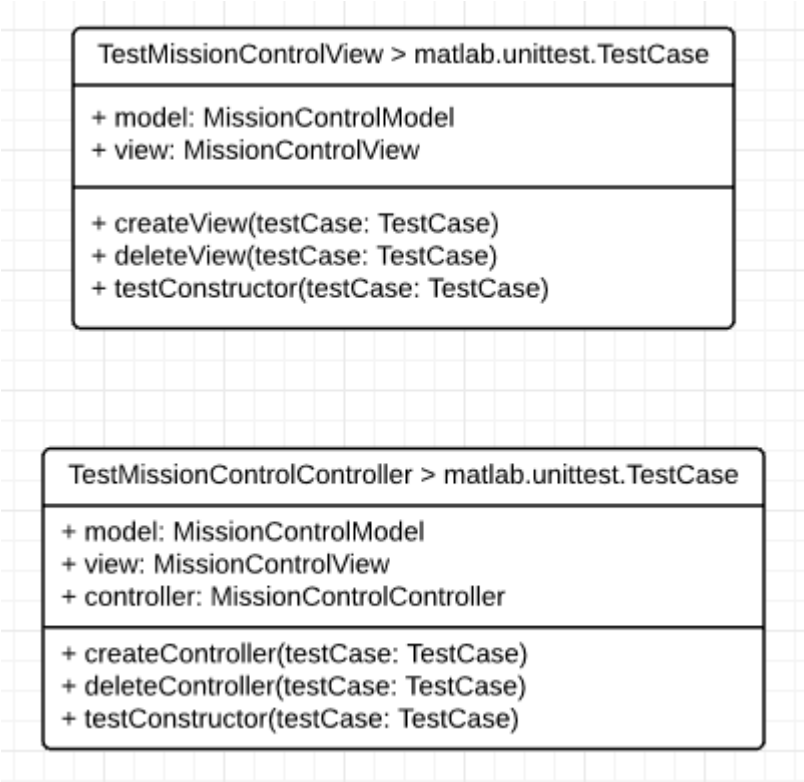


Figure 37: Test class diagrams

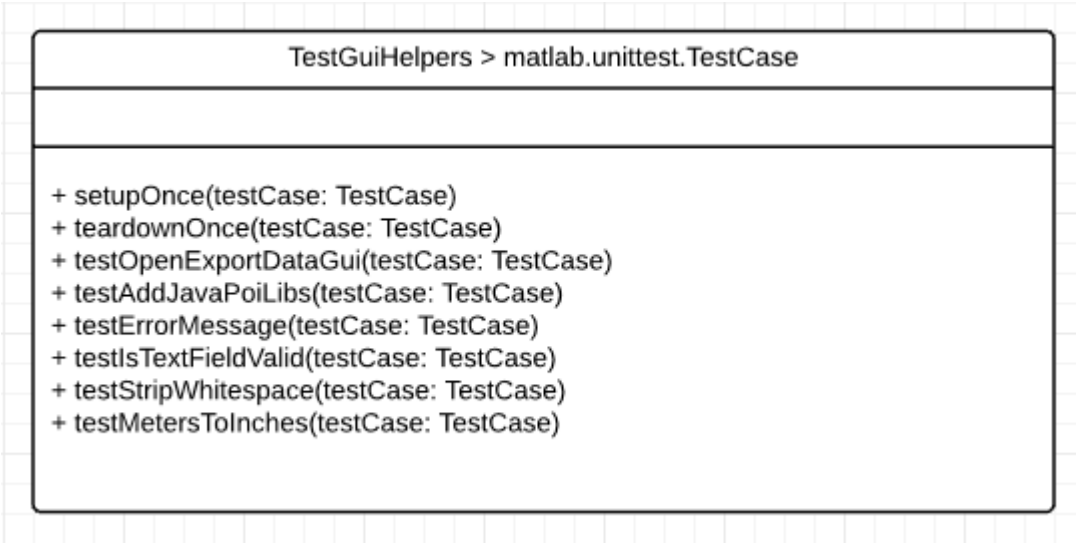


Figure 38: Test class diagrams

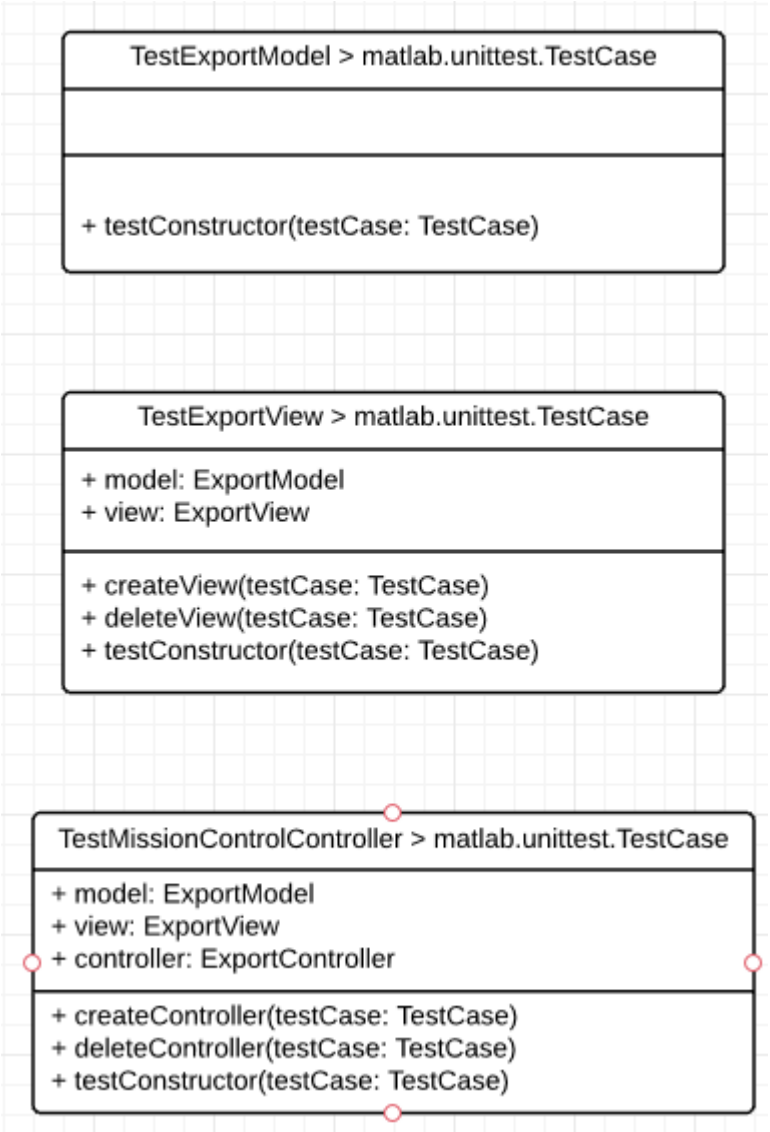


Figure 39: Test class diagrams

MATLAB Code

Launcher.m

```
classdef Launcher < handle

    %% Properties
    properties (Access = public, SetObservable)

        springConstant
        projectileMass
        launchVelocity
        launchAngle

    end

    %% Dependent Properties
    properties (Dependent)

        springDisplacement

    end


    %% Non-static methods
    methods

        % Constructor
        function this = Launcher(springConstant, projectileMass,
launchVelocity, launchAngle)

           this.springConstant = springConstant;
           this.projectileMass = projectileMass;
           this.launchVelocity = launchVelocity;
           this.launchAngle = launchAngle;

        end

        % Getter and Setters

        % springDisplacement
        function value = get.springDisplacement(this)

            %value = this.launchVelocity / sqrt(this.springConstant /
this.projectileMass);
            value = this.launchVelocity / sqrt(this.springConstant /
this.projectileMass);
        end

    end
end
```

LaunchSimulation.m

```
classdef LaunchSimulation < handle

    %% Constants
    properties (Constant)
        g = 9.81 % m/s^2
    end

    %% Properties
    properties (SetObservable)
        launcher
        angleTableData
        predictionsTableData
        simulationTableData
    end

    %% Dependent Properties
    properties (Dependent)

        horizontalRange
        verticalRange
        timeOfFlight

    end

    %% Methods
    methods

        % constructor
        function this = LaunchSimulation(launcher)
            this.launcher = launcher;
            this.angleTableData = [];
            this.predictionsTableData = [];
        end

        function tableData = computeSimulationData(this)

            % Construct data array for table using simulation object
            tableData = {this.launcher.springDisplacement ...
                         this.horizontalRange             ...
                         this.verticalRange               ...
                         this.timeOfFlight};

            this.simulationTableData = tableData;

            tableData = this.simulationTableData;

        end

        % Compute the table data for the angles for a given velocity
        function tableData = computeAngleData(this, velocity)
```

```
            % Create temporary simulation and launcher objects for
            % computing the table data
            tempLauncher = Launcher(this.launcher.springConstant,
this.launcher.projectileMass, velocity, this.launcher.launchAngle);
            tempSimulation = LaunchSimulation(tempLauncher);


            horizontalRanges = [];
            verticalRanges = [];
            angles = [];
            timeOfFlights = [];

            for angle = 0:15:90

                tempSimulation.launcher.launchAngle = angle;
                angles(end+1) = angle;
                horizontalRanges(end+1) = tempSimulation.horizontalRange;
                verticalRanges(end+1) = tempSimulation.verticalRange;
                timeOfFlights(end+1) = tempSimulation.timeOfFlight;
            end



            this.angleTableData = [angles' horizontalRanges' verticalRanges'
timeOfFlights'];

            % Delete the temporary objects
            delete(tempLauncher);
            delete(tempSimulation);

            tableData = this.angleTableData;

        end

        % compute the angle and velocity data for a given range
        function tableData = computePredictionData(this, horizontalRange)

            % Create temporary simulation and launcher objects for
            % computing the table data
            tempLauncher = Launcher(this.launcher.springConstant,
this.launcher.projectileMass, this.launcher.launchVelocity,
this.launcher.launchAngle);
            tempSimulation = LaunchSimulation(tempLauncher);


            velocities = [];
            angles = [];

            for angle = 0:15:90


                tempSimulation.launcher.launchAngle = angle;

                % solve for the velocity at the current angle
                % range = ((this.launcher.launchVelocity)^2 * sind(2 *
```

```
this.launcher.launchAngle)) / LaunchSimulation.g;
                % range * g = ((this.launcher.launchVelocity)^2 * sind(2 *
this.launcher.launchAngle))
                % (range * g) / sind(2 * this.launcher.launchAngle) =
(this.launcher.launchVelocity)^2
                % this.launcher.launchVelocity = sqrt((range * g) / sind(2 *
this.launcher.launchAngle))

                velocity = sqrt((horizontalRange * LaunchSimulation.g) /
sind(2 * tempSimulation.launcher.launchAngle));
                angles(end+1) = angle;
                velocities(end+1) = velocity;

            end

            this.predictionsTableData = [angles' velocities'];

            % Delete the temporary objects
            delete(tempLauncher);
            delete(tempSimulation);

            tableData = this.predictionsTableData;
        end


        %% Getters and Setters

        % horizontalRange
        function value = get.horizontalRange(this)

            value = ((this.launcher.launchVelocity)^2 * sind(2 *
this.launcher.launchAngle)) / LaunchSimulation.g;

        end

        % verticalRange
        function value = get.verticalRange(this)

            value = ((this.launcher.launchVelocity^2) *
sind(this.launcher.launchAngle)^2) / (2 * LaunchSimulation.g);

        end

        % timeOfFlight
        function value = get.timeOfFlight(this)

            value = ((2 * this.launcher.launchVelocity) *
sind(this.launcher.launchAngle)) / LaunchSimulation.g;

        end


    end
end
```

MissionControlModel.m

```
classdef MissionControlModel < handle

    %% Properties
    properties (Access = public, SetObservable = true)

        % might need to implement observable at the Launcher and
        % LaunchSimulation level instead of here, we will see
        launcher
        simulation

    end

    %% Non-static methods
    methods

        % Constructor
        function this = MissionControlModel()

            % initialize a launcher with zeros for its initial values
            this.launcher = Launcher(0,0,0,0);
            this.simulation = LaunchSimulation(this.launcher);

        end

    end
end
```

MissionControlView.m

```
classdef MissionControlView < handle

    %% Properties
    properties (Access = public)

        model

        figure

        % Launcher Settings
        launcherSettingsPanel
        springConstantLabel
        springConstantTextField
        projectileMassLabel
        projectileMassTextField

        % Simulation
        simulationPanel
        launchVelocityLabel
        launchVelocityTextField
```

```
        launchAngleLabel
        launchAngleTextField
        simulateButton
        simulationDataPanel
        simulationDataTable
        trajectoryPanel
        trajectoryAxes

        % Predictions
        predictionsPanel
        targetDistanceLabel
        targetDistanceTextField
        predictionsCalculateButton
        predictionsExportButton
        predictionsTable

        % Tables
        tablesPanel
        launchVelocityTablesLabel
        launchVelocityTablesTextField
        tablesCalculateButton
        tablesExportButton
        angleTable

        % Conversions
        conversionsPanel
        metersTextField
        metersLabel
        inchesLabel
        inchesValueLabel
        convertButton

    end

    %% Non-static methods
    methods

        % Constructor
        function this = MissionControlView(model)

            this.model = model;

            % Set up the property listeners
            addlistener(this.model.simulation.launcher, 'launchAngle',
'PostSet', @this.handlePropertyEvents);
            addlistener(this.model.simulation, 'angleTableData', 'PostSet',
@this.handlePropertyEvents);
            addlistener(this.model.simulation, 'predictionsTableData',
'PostSet', @this.handlePropertyEvents);
            addlistener(this.model.simulation, 'simulationTableData',
'PostSet', @this.handlePropertyEvents);
            % open the figure that was designed with GUIDE
            this.figure = openfig('RocketLauncherSimulationGUI.fig');

            %% Launcher Settings Panel
            this.launcherSettingsPanel = findobj(this.figure, 'tag',
```

```
'panelLauncherSettings');
            this.springConstantLabel = findobj(this.figure, 'tag',
'labelSpringConstant');
            this.springConstantTextField = findobj(this.figure,
'tag','editSpringConstant');
            this.projectileMassLabel = findobj(this.figure, 'tag',
'labelProjectileMass');
            this.projectileMassTextField = findobj(this.figure,
'tag','editProjectileMass');

            %% Simulation Panel
            this.simulationPanel = findobj(this.figure, 'tag',
'panelSimulation');
            this.launchVelocityLabel = findobj(this.figure, 'tag',
'labelLaunchVelocity');
            this.launchVelocityTextField = findobj(this.figure,
'tag','editLaunchVelocity');
            this.launchAngleLabel = findobj(this.figure, 'tag',
'labelLaunchAngle');
            this.launchAngleTextField = findobj(this.figure,
'tag','editLaunchAngle');
            this.simulateButton =  findobj(this.figure, 'tag',
'buttonSimulate');

            %% Simulation Data Panel
            this.simulationDataPanel = findobj(this.figure, 'tag',
'panelSimulationData');
            this.simulationDataTable = findobj(this.figure, 'tag',
'tableSimulationData');
            simulationTableData = {'' '' '' ''};
            set(this.simulationDataTable, 'Data', simulationTableData);

            %% Trajectory Panel
            this.trajectoryPanel = findobj(this.figure, 'tag',
'panelTrajectory');
            this.trajectoryAxes = findobj(this.figure, 'tag',
'axesTrajectory');
            % add labels to axes (this isnt working for some reason. its
            % creating a new figure instead of adding it to the current one
            %axes(this.trajectoryAxes);
            %xlabel('Horizontal Position (m)');
            %ylabel('Vertical Position (m)');

            %% Predictions Panel
            this.predictionsPanel = findobj(this.figure, 'tag',
'panelPredictions');
            this.targetDistanceLabel = findobj(this.figure, 'tag',
'labelTargetDistance');
            this.targetDistanceTextField = findobj(this.figure, 'tag',
'editTargetDistance');
            this.predictionsCalculateButton =  findobj(this.figure, 'tag',
'buttonCalculate');
            this.predictionsExportButton = findobj(this.figure, 'tag',
'buttonExportPredictions');
            this.predictionsTable = findobj(this.figure, 'tag',
'tablePredictions');
            angles = [0;15;30;45;60;75;90];
```

```
            predictionTableData = [angles; [];];
            set(this.predictionsTable, 'Data', predictionTableData);

            %% Tables Panel
            this.tablesPanel = findobj(this.figure, 'tag', 'panelTables');
            this.launchVelocityTablesLabel = findobj(this.figure, 'tag',
'labelAngleTableVelocity');
            this.launchVelocityTablesTextField = findobj(this.figure, 'tag',
'editAngleTableVelocity');
            this.tablesCalculateButton =  findobj(this.figure, 'tag',
'buttonGenerateTable');
            this.tablesExportButton = findobj(this.figure, 'tag',
'buttonExportAngles');
            this.angleTable = findobj(this.figure, 'tag', 'tableAngles');
            angleTableData = [angles; []; []; []];
            set(this.angleTable, 'Data', angleTableData);

            %% Conversions Panel
            this.conversionsPanel = findobj(this.figure, 'tag',
'panelConversions');
            this.metersLabel = findobj(this.figure, 'tag', 'labelMeters');
            this.metersTextField = findobj(this.figure, 'tag', 'editMeters');
            this.inchesLabel = findobj(this.figure, 'tag', 'labelInches');
            this.inchesValueLabel = findobj(this.figure, 'tag',
'labelInchesValue');
            this.convertButton = findobj(this.figure, 'tag',
'buttonConvert');

        end

        %% Plotting Methods
       function plotVerticalVsHorizontalPosition(this)

           t = 0:.01:this.model.simulation.timeOfFlight;
           x = (this.model.simulation.launcher.launchVelocity *
cosd(this.model.simulation.launcher.launchAngle) .* t);
           y = (this.model.simulation.launcher.launchVelocity *
sind(this.model.simulation.launcher.launchAngle) .* t) + (-.5 *
LaunchSimulation.g .* (t.^2));

           xLim = [min(x), max(x) + 3];
           yLim = [min(y), max(y) + 3];
           set(this.trajectoryAxes, 'XLim', xLim);
           set(this.trajectoryAxes, 'YLim', yLim);
           hold(this.trajectoryAxes);
           comet(this.trajectoryAxes,x,y);
           hold(this.trajectoryAxes);

       end

       function handlePropertyEvents(this, property, ~)

          switch property.Name

             case 'simulationTableData'
```

```
                set(this.simulationDataTable, 'Data',
this.model.simulation.simulationTableData);

            case 'launchAngle'

                % The user changed the main simulation settings, so we need
to
                % plot the trajectory graph
                % if we let the view respond to this change in the model,
                % we have to wait for the plot to finish before the
                % simulation data table is updated. Because of this, this
                % method call has been moved to the controller so that we can
update the simulation data first, then graph.
                %this.plotVerticalVsHorizontalPosition()

            case 'angleTableData'

                set(this.angleTable, 'Data',
this.model.simulation.angleTableData);

            case 'predictionsTableData'

                set(this.predictionsTable, 'Data',
this.model.simulation.predictionsTableData);

          end

       end

    end

end
```

MissionControlConroller.m

```
classdef MissionControlController < handle

    %% Properties
    properties (Access = public)

        model
        view

    end

    %% Non-static methods
    methods

        % Constructor
        function this = MissionControlController(model, missionControlView)
```

```
            % initialize the model and view
            this.model = model;
            this.view = missionControlView;

            % Hook up the callbacks
            set(this.view.simulateButton, 'Callback',
@this.simulateButtonPressed);
            set(this.view.predictionsCalculateButton, 'Callback',
@this.predictionsCalculateButtonPressed);
            set(this.view.predictionsExportButton, 'Callback',
@this.predictionsExportButtonPressed);
            set(this.view.tablesCalculateButton, 'Callback',
@this.tablesCalculateButtonPressed);
            set(this.view.tablesExportButton, 'Callback',
@this.tablesExportButtonPressed);
            set(this.view.convertButton, 'Callback',
@this.convertButtonPressed);

            % Add the java POI's to the path
            GuiHelpers.addJavaPoiLibs();

        end

        % Simulate Button Callback
        function simulateButtonPressed(this, src, ~)

            % reset label colors to black for validation purposes
            labelHandles = [this.view.launchVelocityLabel
this.view.launchAngleLabel this.view.springConstantLabel
this.view.projectileMassLabel];
            GuiHelpers.resetLabelForegroundColors(labelHandles)

            % validate launch velocity and launch angle text fields
            if GuiHelpers.isTextFieldValid(this.view.launchVelocityTextField,
this.view.launchVelocityLabel) ...
                    &&
GuiHelpers.isTextFieldValid(this.view.launchAngleTextField,
this.view.launchAngleLabel)

                % validate the spring constant and projectile mass text
fields
                if
GuiHelpers.isTextFieldValid(this.view.springConstantTextField,
this.view.springConstantLabel) ...
                        &&
GuiHelpers.isTextFieldValid(this.view.projectileMassTextField,
this.view.projectileMassLabel)

                    % clear the axes
                    cla(this.view.trajectoryAxes);

                    % Set the launcher data
                    this.model.simulation.launcher.springConstant =
str2num(GuiHelpers.stripWhitespace(get(this.view.springConstantTextField,
'String')));
```

```
                    this.model.simulation.launcher.projectileMass =
str2num(GuiHelpers.stripWhitespace(get(this.view.projectileMassTextField,
'String')));
                    this.model.simulation.launcher.launchVelocity =
str2num(GuiHelpers.stripWhitespace(get(this.view.launchVelocityTextField,
'String')));
                    this.model.simulation.launcher.launchAngle =
str2num(GuiHelpers.stripWhitespace(get(this.view.launchAngleTextField,
'String')));

                    this.model.simulation.computeSimulationData();
                    this.view.plotVerticalVsHorizontalPosition()

                end

            end

        end

        % Predictions Calculate Button Callback
        function predictionsCalculateButtonPressed(this, src, ~)

            % reset label colors to black for validation purposes
            labelHandles = [this.view.targetDistanceLabel
this.view.springConstantLabel this.view.projectileMassLabel];
            GuiHelpers.resetLabelForegroundColors(labelHandles)

            % validate the target distance field
            if GuiHelpers.isTextFieldValid(this.view.targetDistanceTextField,
this.view.targetDistanceLabel)

                % validate the spring constant and projectile mass text
fields
                if
GuiHelpers.isTextFieldValid(this.view.springConstantTextField,
this.view.springConstantLabel) ...
                        &&
GuiHelpers.isTextFieldValid(this.view.projectileMassTextField,
this.view.projectileMassLabel)

                    range =
str2num(strtrim(get(this.view.targetDistanceTextField, 'String')));
                    simulation = this.model.simulation;

                    % Set the launcher data
                    this.model.simulation.launcher.springConstant =
str2num(GuiHelpers.stripWhitespace(get(this.view.springConstantTextField,
'String')));
                    this.model.simulation.launcher.projectileMass =
str2num(GuiHelpers.stripWhitespace(get(this.view.projectileMassTextField,
'String')));


                    this.model.simulation.computePredictionData(range);

                end
```

```
            end

        end

        % Predictions Export Button Callback
        function predictionsExportButtonPressed(this, src, ~)

            % call the helper function and pass in the table
            GuiHelpers.openExportDataGui(this.view.predictionsTable);

        end

        % Tables Calculate Button Callback
        function tablesCalculateButtonPressed(this, src, ~)

            % reset label colors to black for validation purposes
            labelHandles = [this.view.launchVelocityTablesLabel
this.view.springConstantLabel this.view.projectileMassLabel];

            % validate the angle table velocity text field
            if
GuiHelpers.isTextFieldValid(this.view.launchVelocityTablesTextField,
this.view.launchVelocityTablesLabel)

                % validate the spring constant and projectile mass text
fields
                if
GuiHelpers.isTextFieldValid(this.view.springConstantTextField,
this.view.springConstantLabel) ...
                        &&
GuiHelpers.isTextFieldValid(this.view.projectileMassTextField,
this.view.projectileMassLabel)

                    % Set the launcher data
                    this.model.simulation.launcher.springConstant =
str2num(GuiHelpers.stripWhitespace(get(this.view.springConstantTextField,
'String')));
                    this.model.simulation.launcher.projectileMass =
str2num(GuiHelpers.stripWhitespace(get(this.view.projectileMassTextField,
'String')));

                    velocity =
str2num(GuiHelpers.stripWhitespace(get(this.view.launchVelocityTablesTextFiel
d, 'String')));
                    simulation = this.model.simulation;
                    this.model.simulation.computeAngleData(velocity);

                end

            end

        end

        % Tables Export Button Callback
```

```
        function tablesExportButtonPressed(this, src, ~)

            % call the helper function and pass in the table
            GuiHelpers.openExportDataGui(this.view.angleTable);

        end

        % Convert Button Callback
        function convertButtonPressed(this, src, ~)

            if GuiHelpers.isTextFieldValid(this.view.metersTextField,
this.view.metersLabel)

                  meters = str2num(strtrim(get(this.view.metersTextField,
'String')));
                  inches = GuiHelpers.metersToInches(meters);
                  inchesString = num2str(inches, '%.2f');
                  this.view.inchesValueLabel.String = inchesString;
            end

        end

    end

end
```

RocketLauncherSimulationGUI.m

```
%%%%%%%%%%%%%%%%%%%%%%
% THIS FILE IS NOT USED. IT IS JUST HERE BECAUSE IT HAS TO BE IN ORDER FOR
% THE VIEW TO GET BUILT
%%%%%%%%%%%%%%%%%%%%%%

function varargout = RocketLauncherSimulationGUI(varargin)


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RocketLauncherSimulationGUI_OpeningFcn,
...
                   'gui_OutputFcn',  @RocketLauncherSimulationGUI_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
```

```
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before RocketLauncherSimulationGUI is made visible.
function RocketLauncherSimulationGUI_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RocketLauncherSimulationGUI (see
VARARGIN)

% add the java poi libs for xlwrite
GuiHelpers.addJavaPoiLibs();

% Choose default command line output for RocketLauncherSimulationGUI
handles.output = hObject;

% Instantiate a launcher object with zeroed property values
launcher = Launcher(0,0,0,0);

% Instantiate a simulation object
simulation = LaunchSimulation(launcher);

% add simulation to handles structure
handles.simulation = simulation;

% add labels to axes
axes(handles.axesTrajectory);
xlabel('Horizontal Position (m)');
ylabel('Vertical Position (m)');

% set initial data for tables
simulationTableData = {'' '' '' ''};

set(handles.tableSimulationData, 'Data', simulationTableData);

angles = [0;15;30;45;60;75;90];

angleTableData = [angles; []; []; []];

set(handles.tableAngles, 'Data', angleTableData);

predictionTableData = [angles; [];];

set(handles.tablePredictions, 'Data', predictionTableData);

% Update handles structure
guidata(hObject, handles);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = RocketLauncherSimulationGUI_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function editSpringConstant_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function editSpringConstant_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editProjectileMass_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function editProjectileMass_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editLaunchVelocity_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function editLaunchVelocity_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editLaunchAngle_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function editLaunchAngle_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editAngleTableVelocity_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editAngleTableVelocity_Callback(hObject, eventdata, handles)


function editTargetDistance_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function editTargetDistance_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.

function axesLogo_CreateFcn(hObject, eventdata, handles)
    %axes(hObject);
    %imshow('placeholder1.jpg');



function editMeters_Callback(hObject, eventdata, handles)
% hObject    handle to editMeters (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editMeters as text
%        str2double(get(hObject,'String')) returns contents of editMeters as
a double


% --- Executes during object creation, after setting all properties.
function editMeters_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editMeters (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

ExportModel.m

```
classdef ExportModel < handle

    %% Properties
    properties (Access = public, SetObservable = true)

        data

    end

    %% Non-static methods
    methods

        % Constructor
        function this = ExportModel(data)

            this.data = data;

        end

    end
end
```

ExportView.m

```
classdef ExportView < handle

    %% Properties
    properties (Access = public)

        model

        figure

        fileNameLabel
        fileNameTextField
        sheetNameLabel
        sheetNameTextField
        startRangeLabel
```

```
        startRangeTextField
        includeColumnHeadersCheckBox
        okButton

    end

    %% Non-static methods
    methods

        % Constructor
        function this = ExportView(model)
            this.model = model;

            % open the figure that was designed with GUIDE
            this.figure = openfig('ExportGUI.fig');

            this.fileNameLabel = findobj(this.figure, 'tag',
'labelFileName');
            this.fileNameTextField = findobj(this.figure, 'tag',
'editFileName');

            this.sheetNameLabel = findobj(this.figure, 'tag',
'labelSheetName');
            this.sheetNameTextField = findobj(this.figure, 'tag',
'editSheetName');

            this.startRangeLabel = findobj(this.figure, 'tag',
'labelStartRange');
            this.startRangeTextField = findobj(this.figure, 'tag',
'editStartRange');

            this.includeColumnHeadersCheckBox = findobj(this.figure, 'tag',
'checkboxIncludeColumnHeaders');

            this.okButton = findobj(this.figure, 'tag', 'buttonOK');


        end

    end
end
```

ExportController.m

```
classdef ExportController < handle

    %% Properties
    properties (Access = public)
```

```
        model
        view

    end

    %% Non-static methods
    methods

        % Constructor
        function this = ExportController(model, view)

            % initialize the model and view
            this.model = model;
            this.view = view;

            % Hook up the callbacks
            set(this.view.okButton, 'Callback', @this.okButtonPressed);


        end

        % OK Button Callback
        function okButtonPressed(this, src, ~)

            % check if we need to include column headers

            if get(this.view.includeColumnHeadersCheckBox, 'Value') == 0

                % we need to remove the top row of the export data to get rid
of the
                % headers.
                this.model.data(1, :) = [];

            end

            filename = get(this.view.fileNameTextField, 'String');
            sheetName = get(this.view.sheetNameTextField, 'String');
            startRange = get(this.view.startRangeTextField, 'String');
            xlwrite(filename, this.model.data, sheetName, startRange);
            close(this.view.figure);

        end

    end
end
```

ExportGUI.m

```
%%%%%%%%%%%%%%%%%%%%%%
```

```
% THIS FILE IS NOT USED. IT IS JUST HERE BECAUSE IT HAS TO BE IN ORDER FOR
% THE VIEW TO GET BUILT
%%%%%%%%%%%%%%%%%%%%%%

function varargout = ExportGUI(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ExportGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @ExportGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ExportGUI is made visible.
function ExportGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ExportGUI (see VARARGIN)

% Choose default command line output for ExportGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ExportGUI wait for user response (see UIRESUME)
% uiwait(handles.guiExport);


% --- Outputs from this function are returned to the command line.
function varargout = ExportGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
function editFileName_Callback(hObject, eventdata, handles)
% hObject    handle to editFileName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editFileName as text
%        str2double(get(hObject,'String')) returns contents of editFileName
as a double


% --- Executes during object creation, after setting all properties.
function editFileName_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editFileName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in checkboxIncludeColumnHeaders.
function checkboxIncludeColumnHeaders_Callback(hObject, eventdata, handles)
% hObject    handle to checkboxIncludeColumnHeaders (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
checkboxIncludeColumnHeaders


% --- Executes on button press in buttonOK.
function buttonOK_Callback(hObject, eventdata, handles)

% check if we need to include column headers

if get(handles.checkboxIncludeColumnHeaders, 'Value') == 0

    % we need to remove the top row of the export data to get rid of the
    % headers.
    handles.exportData(1, :) = [];

end

filename = get(handles.editFileName, 'String');
sheetName = get(handles.editSheetName, 'String');
startRange = get(handles.editStartRange, 'String');
xlwrite(filename, handles.exportData, sheetName, startRange);
close(handles.guiExport);
```

```
function editSheetName_Callback(hObject, eventdata, handles)
% hObject    handle to editSheetName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editSheetName as text
%        str2double(get(hObject,'String')) returns contents of editSheetName
as a double


% --- Executes during object creation, after setting all properties.
function editSheetName_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editSheetName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editStartRange_Callback(hObject, eventdata, handles)
% hObject    handle to editStartRange (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editStartRange as text
%        str2double(get(hObject,'String')) returns contents of editStartRange
as a double


% --- Executes during object creation, after setting all properties.
function editStartRange_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editStartRange (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

GuiHelpers.m

```
classdef GuiHelpers < handle
```

```
    properties
    end

    methods (Static)

        function returnView = openExportDataGui(table)

            try
                exportData = [get(table, 'ColumnName')'; num2cell(get(table,
'Data'))];
                exportModel = ExportModel(exportData);
                exportView = ExportView(exportModel);
                exportController = ExportController(exportModel, exportView);
                returnView = exportView;

            catch e

                if strcmp(e.identifier, 'MATLAB:catenate:dimensionMismatch')

                    % display an error message
                    GuiHelpers.errorMessage('There is no data to export.');


                else

                    % display an error message
                    GuiHelpers.errorMessage('An error occurred.');


                end

                % return an empty cell matrix as a "null" value
                returnView = {};
            end

        end

        function addJavaPoiLibs()

            javaaddpath('poi_library/poi-3.8-20120326.jar');
            javaaddpath('poi_library/poi-ooxml-3.8-20120326.jar');
            javaaddpath('poi_library/poi-ooxml-schemas-3.8-20120326.jar');
            javaaddpath('poi_library/xmlbeans-2.3.0.jar');
            javaaddpath('poi_library/dom4j-1.6.1.jar');
            javaaddpath('poi_library/stax-api-1.0.1.jar');

        end

        function message = errorMessage(message)

            messageStruct.Interpreter = 'tex';
            messageStruct.WindowStyle = 'modal';
            message = msgbox(message, 'Error', 'error', messageStruct);
```

```
        end

        function value = isTextFieldValid(textFieldHandle, labelHandle)

            % is it blank?
            textFieldValue = strtrim(get(textFieldHandle, 'String'));
            if isempty(textFieldValue)

                set(labelHandle,'ForegroundColor','red');
                labelText = get(labelHandle, 'String');
                GuiHelpers.errorMessage(strcat(labelText, ' cannot be
blank.'));
                value = false;

            % does it contain more than just digt characters?
            elseif isempty(regexp(textFieldValue, '(?<=^| )\d+(\.\d+)?(?=$|
)|(?<=^| )\.\d+(?=$| )')) || ~all(~isspace(textFieldValue))

                set(labelHandle,'ForegroundColor','red');
                labelText = get(labelHandle, 'String');
                GuiHelpers.errorMessage(strcat(labelText, ' values must be
numeric.'));
                value = false;

            else

                set(labelHandle,'ForegroundColor','black');
                value = true;

            end
        end

        function resetLabelForegroundColors(labelHandles)
            arrayfun(@(labelHandle)
set(labelHandle,'ForegroundColor','black'), labelHandles);
        end

        function newString = stripWhitespace(someString)
            % trim the string
            trimmedString = strtrim(someString);
            % get rid of any space in the middle
            newString = trimmedString(~isspace(trimmedString));
        end

        function inches = metersToInches(meters)

            inches = meters * 39.3701;

        end

    end

end
```

run.m

```
clc
clear
close all

missionControlModel = MissionControlModel();
missionControlView = MissionControlView(missionControlModel);
missionControlController = MissionControlController(missionControlModel,
missionControlView);

% Spring Constant = 619.3697613
% Projectile Mass = .0617
```

xlwrite.m

```
function [status, message]=xlwrite(filename,A,sheet, range)
% XLWRITE Write to Microsoft Excel spreadsheet file using Java
%   XLWRITE(FILE,ARRAY) writes ARRAY to the first worksheet in the Excel
%   file named FILE, starting at cell A1. It aims to have exactly the same
%   behaviour as XLSWRITE. See also XLSWRITE.
%
%   XLWRITE(FILE,ARRAY,SHEET) writes to the specified worksheet.
%
%   XLWRITE(FILE,ARRAY,RANGE) writes to the rectangular region
%   specified by RANGE in the first worksheet of the file. Specify RANGE
%   using the syntax 'C1:C2', where C1 and C2 are opposing corners of the
%   region.
%
%   XLWRITE(FILE,ARRAY,SHEET,RANGE) writes to the specified SHEET and
%   RANGE.
%
%   STATUS = XLWRITE(FILE,ARRAY,SHEET,RANGE) returns the completion
%   status of the write operation: TRUE (logical 1) for success, FALSE
%   (logical 0) for failure.  Inputs SHEET and RANGE are optional.
%
%   Input Arguments:
%
%   FILE    String that specifies the file to write. If the file does not
%           exist, XLWRITE creates a file, determining the format based on
%           the specified extension. To create a file compatible with Excel
%           97-2003 software, specify an extension of '.xls'. If you do not
%           specify an extension, XLWRITE applies '.xls'.
%   ARRAY   Two-dimensional logical, numeric or character array or, if each
%           cell contains a single element, a cell array.
%   SHEET   Worksheet to write. One of the following:
%           * String that contains the worksheet name.
%           * Positive, integer-valued scalar indicating the worksheet
%             index.
%           If SHEET does not exist, XLWRITE adds a new sheet at the end
%           of the worksheet collection.
%   RANGE   String that specifies a rectangular portion of the worksheet to
```

```
%           read. Not case sensitive. Use Excel A1 reference style.
%           * If you specify a SHEET, RANGE can either fit the size of
%             ARRAY or specify only the first cell (such as 'D2').
%           * If you do not specify a SHEET, RANGE must include both
%             corners and a colon character (:), even for a single cell
%             (such as 'D2:D2').
%           * If RANGE is larger than the size of ARRAY, Excel fills the
%             remainder of the region with #N/A. If RANGE is smaller than
%             the size of ARRAY, XLWRITE writes only the subset that fits
%             into RANGE to the file.
%
%   Note
%   * This function requires the POI library to be in your javapath.
%     To add the Apache POI Library execute commands:
%     (This assumes the POI lib files are in folder 'poi_library')
%       javaaddpath('poi_library/poi-3.8-20120326.jar');
%       javaaddpath('poi_library/poi-ooxml-3.8-20120326.jar');
%       javaaddpath('poi_library/poi-ooxml-schemas-3.8-20120326.jar');
%       javaaddpath('poi_library/xmlbeans-2.3.0.jar');
%       javaaddpath('poi_library/dom4j-1.6.1.jar');
%   * Excel converts Inf values to 65535. XLWRITE converts NaN values to
%     empty cells.
%
%   EXAMPLES
%   % Write a 7-element vector to testdata.xls:
%   xlwrite('testdata.xls', [12.7, 5.02, -98, 63.9, 0, -.2, 56])
%
%   % Write mixed text and numeric data to testdata2.xls
%   % starting at cell E1 of Sheet1:
%   d = {'Time','Temperature'; 12,98; 13,99; 14,97};
%   xlwrite('testdata2.xls', d, 1, 'E1')
%
%
%   REVISIONS
%   20121004 - First version using JExcelApi
%   20121101 - Modified to use POI library instead of JExcelApi (allows to
%           generate XLSX)
%   20121127 - Fixed bug: use existing rows if present, instead of
%           overwrite rows by default. Thanks to Dan & Jason.
%   20121204 - Fixed bug: if a numeric sheet is given & didn't exist,
%           an error was returned instead of creating the sheet. Thanks to
Marianna
%   20130106 - Fixed bug: use existing cell if present, instead of
%           overwriting. This way original XLS formatting is kept & not
%           overwritten.
%   20130125 - Fixed bug & documentation. Incorrect working of NaN. Thanks
Klaus
%   20130227 - Fixed bug when no sheet number given & added Stax to java
%               load. Thanks to Thierry
%
%   Copyright 2012-2013, Alec de Zegher
%==============================================================================
==

% Check if POI lib is loaded
if exist('org.apache.poi.ss.usermodel.WorkbookFactory', 'class') ~= 8 ...
    || exist('org.apache.poi.hssf.usermodel.HSSFWorkbook', 'class') ~= 8 ...
    || exist('org.apache.poi.xssf.usermodel.XSSFWorkbook', 'class') ~= 8
```

```
    error('xlWrite:poiLibsNotLoaded',...
        'The POI library is not loaded in Matlab.\nCheck that POI jar files
are in Matlab Java path!');
end

% Import required POI Java Classes
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.hssf.usermodel.*;
import org.apache.poi.xssf.usermodel.*;
import org.apache.poi.ss.usermodel.*;

import org.apache.poi.ss.util.*;

status=0;

% If no sheet & xlrange is defined, attribute an empty value to it
if nargin < 3; sheet = []; end
if nargin < 4; range = []; end

% Check if sheetvariable contains range data
if nargin < 4 && ~isempty(strfind(sheet,':'))
    range = sheet;
    sheet = [];
end

% check if input data is given
if isempty(A)
    error('xlwrite:EmptyInput', 'Input array is empty!');
end
% Check that input data is not bigger than 2D
if ndims(A) > 2
    error('xlwrite:InputDimension', ...
        'Dimension of input array should not be higher than two.');
end

% Set java path to same path as Matlab path
java.lang.System.setProperty('user.dir', pwd);

% Open a file
xlsFile = java.io.File(filename);

% If file does not exist create a new workbook
if xlsFile.isFile()
    % create XSSF or HSSF workbook from existing workbook
    fileIn = java.io.FileInputStream(xlsFile);
    xlsWorkbook = WorkbookFactory.create(fileIn);
else
    % Create a new workbook based on the extension.
    [~,~,fileExt] = fileparts(filename);

    % Check based on extension which type to create. If no (valid)
    % extension is given, create XLSX file
    switch lower(fileExt)
        case '.xls'
            xlsWorkbook = HSSFWorkbook();
```

```
        case '.xlsx'
            xlsWorkbook = XSSFWorkbook();
        otherwise
            xlsWorkbook = XSSFWorkbook();

            % Also update filename with added extension
            filename = [filename '.xlsx'];
    end
end

% If sheetname given, enter data in this sheet
if ~isempty(sheet)
    if isnumeric(sheet)
        % Java uses 0-indexing, so take sheetnumer-1
        % Check if the sheet can exist
        if xlsWorkbook.getNumberOfSheets() >= sheet && sheet >= 1
            xlsSheet = xlsWorkbook.getSheetAt(sheet-1);
        else
            % There are less number of sheets, that the requested sheet, so
            % return an empty sheet
            xlsSheet = [];
        end
    else
        xlsSheet = xlsWorkbook.getSheet(sheet);
    end

    % Create a new sheet if it is empty
    if isempty(xlsSheet)
        warning('xlwrite:AddSheet', 'Added specified worksheet.');

        % Add the sheet
        if isnumeric(sheet)
            xlsSheet = xlsWorkbook.createSheet(['Sheet ' num2str(sheet)]);
        else
            % Create a safe sheet name
            sheet = WorkbookUtil.createSafeSheetName(sheet);
            xlsSheet = xlsWorkbook.createSheet(sheet);
        end
    end

else
    % check number of sheets
    nSheets = xlsWorkbook.getNumberOfSheets();

    % If no sheets, create one
    if nSheets < 1
        xlsSheet = xlsWorkbook.createSheet('Sheet 1');
    else
        % Select the first sheet
        xlsSheet = xlsWorkbook.getSheetAt(0);
    end
end

% if range is not specified take start row & col at A1
% locations are 0 indexed
if isempty(range)
    iRowStart = 0;
```

```
    iColStart = 0;
    iRowEnd = size(A, 1)-1;
    iColEnd = size(A, 2)-1;
else
    % Split range in start & end cell
    iSeperator = strfind(range, ':');
    if isempty(iSeperator)
        % Only start was defined as range
        % Create a helper to get the row and column
        cellStart = CellReference(range);
        iRowStart = cellStart.getRow();
        iColStart = cellStart.getCol();
        % End column calculated based on size of A
        iRowEnd = iRowStart + size(A, 1)-1;
        iColEnd = iColStart + size(A, 2)-1;
    else
        % Define start & end cell
        cellStart = range(1:iSeperator-1);
        cellEnd = range(iSeperator+1:end);

        % Create a helper to get the row and column
        cellStart = CellReference(cellStart);
        cellEnd = CellReference(cellEnd);

        % Get start & end locations
        iRowStart = cellStart.getRow();
        iColStart = cellStart.getCol();
        iRowEnd = cellEnd.getRow();
        iColEnd = cellEnd.getCol();
    end
end

% Get number of elements in A (0-indexed)
nRowA = size(A, 1)-1;
nColA = size(A, 2)-1;

% If data is a cell, convert it
if ~iscell(A)
    A = num2cell(A);
end

% Iterate over all data
for iRow = iRowStart:iRowEnd
    % Fetch the row (if it exists)
    currentRow = xlsSheet.getRow(iRow);
    if isempty(currentRow)
        % Create a new row, as it does not exist yet
        currentRow = xlsSheet.createRow(iRow);
    end

    % enter data for all cols
    for iCol = iColStart:iColEnd
        % Check if cell exists
        currentCell = currentRow.getCell(iCol);
        if isempty(currentCell)
            % Create a new cell, as it does not exist yet
            currentCell = currentRow.createCell(iCol);
```

```
        end

        % Check if we are still in array A
        if (iRow-iRowStart)<=nRowA && (iCol-iColStart)<=nColA
            % Fetch the data
            data = A{iRow-iRowStart+1, iCol-iColStart+1};

            if ~isempty(data)
                % if it is a NaN value, convert it to an empty string
                if isnumeric(data) && isnan(data)
                    data = '';
                end

                % Write data to cell
                currentCell.setCellValue(data);
            end

        else
            % Set field to NA
            currentCell.setCellErrorValue(FormulaError.NA.getCode());
        end
    end
end

% Write & close the workbook
fileOut = java.io.FileOutputStream(filename);
xlsWorkbook.write(fileOut);
fileOut.close();

status = 1;

end
```

TestLauncher.m

```
classdef TestLauncher < matlab.unittest.TestCase

    properties
        launcher
    end

    methods(TestClassSetup)

        function setup(testCase)

            % Create a launcher with spring constant: 15 N/m, projectile
mass:
            % .1 kg, launch velocity: 20 m/s, and launch angle: 45 deg
            testCase.launcher = Launcher(15, .1, 20, 45);

        end

    end
```

```
    methods(TestClassTeardown)

        function teardown(testCase)

            % Delete the launcher
            delete(testCase.launcher);

        end

    end

    methods (Test)

        function testConstructor(testCase)

            launcher = Launcher(15, .1, 20, 45);
            testCase.verifyEqual(launcher.springConstant, 15, ...
                'Constructor failed to correctly set springConstant');
            testCase.verifyEqual(launcher.projectileMass, .1, ...
                'Constructor failed to correctly set projectileMass');
            testCase.verifyEqual(launcher.launchVelocity, 20, ...
                'Constructor failed to correctly set launchVelocity');
            testCase.verifyEqual(launcher.launchAngle, 45, ...
                'Constructor failed to correctly set launchAngle');
        end

        function testSpringDisplacement(testCase)
            actSolution = testCase.launcher.springDisplacement;
            expSolution = 20 / sqrt(15 / .1);
            testCase.verifyEqual(actSolution,expSolution);
        end
    end

end
```

TestLaunchSimulation.m

```
classdef TestLaunchSimulation < matlab.unittest.TestCase

     properties
        launcher
        simulation
    end

    methods(TestClassSetup)

        function setup(testCase)

            % Create a launcher with spring constant: 15 N/m, projectile
mass:
            % .1 kg, launch velocity: 20 m/s, and launch angle: 45 deg
            testCase.launcher = Launcher(15, .1, 20, 45);
```

```
        testCase.simulation = LaunchSimulation(testCase.launcher);

    end

end

methods(TestClassTeardown)

    function teardown(testCase)

        % Delete the simulation
        delete(testCase.simulation);

    end

end

methods (Test)

    function testConstructor(testCase)

        launcher = Launcher(15, .1, 20, 45);
        simulation = LaunchSimulation(launcher);

        testCase.verifyEqual(simulation.launcher, launcher, ...
            'Constructor failed to correctly set launcher');

    end

    function testHorizontalRange(testCase)

        actSolution = testCase.simulation.horizontalRange;
        expSolution = ((20)^2 * sind(2 * 45)) / 9.81;
        testCase.verifyEqual(actSolution,expSolution);

    end

    function testVerticalRange(testCase)

        actSolution = testCase.simulation.verticalRange;
        expSolution = ((20^2) * sind(45)^2) / (2 * 9.81);
        testCase.verifyEqual(actSolution,expSolution);

    end

    function testTimeOfFlight(testCase)

        actSolution = testCase.simulation.timeOfFlight;
        expSolution = ((2 * 20) * sind(45)) / 9.81;
        testCase.verifyEqual(actSolution,expSolution);

    end

    function testComputeAngleData(testCase)
```

```
            actSolution = testCase.simulation.computeAngleData(20);
            angles = [0 15 30 45 60 75 90];
            hranges = [0 20.3874 35.3119 40.7747 35.3119 20.3874 0];
            vranges = [0 1.3657 5.0968 10.1937 15.2905 19.0217 20.3874];
            flightTimes = [0 1.0553 2.0387 2.8832 3.5312 3.9385 4.0775];
            expSolution = [angles' hranges' vranges' flightTimes'];
            testCase.verifyEqual(actSolution, expSolution, 'RelTol', .1);

        end

        function testComputePredictionData(testCase)

            actSolution = testCase.simulation.computePredictionData(20);
            angles = [0 15 30 45 60 75 90];
            velocities = [Inf 19.8091 15.0517 14.0071 15.0517 19.8091 Inf];
            expSolution = [angles' velocities'];
            testCase.verifyEqual(actSolution, expSolution, 'RelTol', .1);

        end

    end

end
```

TestMissionControlModel.m

```
classdef TestMissionControlModel < matlab.unittest.TestCase

    properties

    end

    methods (Test)

        function testConstructor(testCase)

            model = MissionControlModel();

            testCase.verifyNotEmpty(model.simulation, ...
                'Constructor failed to correctly set simulation');

            testCase.verifyNotEmpty(model.simulation.launcher, ...
                'Constructor failed to correctly set simulation.launcher');

        end


    end

end
```

TestMissionControlView.m

```
classdef TestMissionControlView < matlab.unittest.TestCase

    properties
       model
       view
    end

    methods(TestClassSetup)

        function setup(testCase)

            testCase.model = MissionControlModel();
            testCase.view = MissionControlView(testCase.model);

        end

    end

    methods(TestClassTeardown)

        function teardown(testCase)

            % Delete the simulation
            close(testCase.view.figure);
            delete(testCase.model);

        end

    end

    methods (Test)

        function testConstructor(testCase)

            model = MissionControlModel();
            view = MissionControlView(model);

            testCase.verifyEqual(view.model, model, ...
                'Constructor failed to correctly set model');

            testCase.verifyNotEmpty(view.figure, 'Constructor failed to
correctly set figure');

            % Launcher Settings
            testCase.verifyNotEmpty(view.launcherSettingsPanel, 'Constructor
failed to correctly set launcherSettingsPanel');
            testCase.verifyNotEmpty(view.springConstantLabel, 'Constructor
failed to correctly set springConstantLabel');
            testCase.verifyNotEmpty(view.springConstantTextField,
'Constructor failed to correctly set springConstantTextField');
            testCase.verifyNotEmpty(view.projectileMassLabel, 'Constructor
```

```
failed to correctly set projectileMassLabel');
            testCase.verifyNotEmpty(view.projectileMassTextField,
'Constructor failed to correctly set projectileMassTextField');

            % Simulation
            testCase.verifyNotEmpty(view.simulationPanel, 'Constructor failed
to correctly set simulationPanel');
            testCase.verifyNotEmpty(view.launchVelocityLabel, 'Constructor
failed to correctly set launchVelocityLabel');
            testCase.verifyNotEmpty(view.launchVelocityTextField,
'Constructor failed to correctly set launchVelocityTextField');
            testCase.verifyNotEmpty(view.launchAngleLabel, 'Constructor
failed to correctly set launchAngleLabel');
            testCase.verifyNotEmpty(view.launchAngleTextField, 'Constructor
failed to correctly set launchAngleTextField');
            testCase.verifyNotEmpty(view.simulateButton, 'Constructor failed
to correctly set simulateButton');
            testCase.verifyNotEmpty(view.simulationDataPanel, 'Constructor
failed to correctly set simulationDataPanel');
            testCase.verifyNotEmpty(view.simulationDataTable, 'Constructor
failed to correctly set simulationDataTable');
            testCase.verifyNotEmpty(view.trajectoryPanel, 'Constructor failed
to correctly set trajectoryPanel');
            testCase.verifyNotEmpty(view.trajectoryAxes, 'Constructor failed
to correctly set trajectoryAxes');

            % Predictions
            testCase.verifyNotEmpty(view.predictionsPanel, 'Constructor
failed to correctly set predictionsPanel');
            testCase.verifyNotEmpty(view.targetDistanceLabel, 'Constructor
failed to correctly set targetDistanceLabel');
            testCase.verifyNotEmpty(view.targetDistanceTextField,
'Constructor failed to correctly set targetDistanceTextField');
            testCase.verifyNotEmpty(view.predictionsCalculateButton,
'Constructor failed to correctly set predictionsCalculateButton');
            testCase.verifyNotEmpty(view.predictionsExportButton,
'Constructor failed to correctly set predictionsExportButton');
            testCase.verifyNotEmpty(view.predictionsTable, 'Constructor
failed to correctly set predictionsTable');

            % Tables
            testCase.verifyNotEmpty(view.tablesPanel, 'Constructor failed to
correctly set tablesPanel');
            testCase.verifyNotEmpty(view.launchVelocityTablesLabel,
'Constructor failed to correctly set launchVelocityTablesLabel');
            testCase.verifyNotEmpty(view.launchVelocityTablesTextField,
'Constructor failed to correctly set launchVelocityTablesTextField');
            testCase.verifyNotEmpty(view.tablesCalculateButton, 'Constructor
failed to correctly set tablesCalculateButton');
            testCase.verifyNotEmpty(view.tablesExportButton, 'Constructor
failed to correctly set tablesExportButton');
            testCase.verifyNotEmpty(view.angleTable, 'Constructor failed to
correctly set angleTable');

            % Conversions
            testCase.verifyNotEmpty(view.conversionsPanel, 'Constructor
failed to correctly set conversionPanel');
```

```
            testCase.verifyNotEmpty(view.metersLabel, 'Constructor failed to
correctly set metersLabel');
            testCase.verifyNotEmpty(view.metersTextField, 'Constructor failed
to correctly set metersTextField');
            testCase.verifyNotEmpty(view.inchesLabel, 'Constructor failed to
correctly set inchesLabel');
            testCase.verifyNotEmpty(view.inchesValueLabel, 'Constructor
failed to correctly set inchesValueLabel');
            testCase.verifyNotEmpty(view.convertButton, 'Constructor failed
to correctly set convertButton');

            close(view.figure);

        end

    end

end
```

TestMissionControlController.m

```
classdef TestMissionControlController < matlab.unittest.TestCase

    properties
       model
       view
       controller
    end

    methods(TestClassSetup)

        function setup(testCase)

            testCase.model = MissionControlModel();
            testCase.view = MissionControlView(testCase.model);
            testCase.controller = MissionControlController(testCase.model,
testCase.view);

        end

    end

    methods(TestClassTeardown)

        function teardown(testCase)

            close(testCase.view.figure);
            delete(testCase.view);
            delete(testCase.model);
            delete(testCase.controller);


        end
```

```
    end

    methods (Test)

        function testConstructor(testCase)

            model = MissionControlModel();
            view = MissionControlView(model);
            controller = MissionControlController(model, view);

            testCase.verifyEqual(controller.model, model, ...
                'Constructor failed to correctly set model');

            testCase.verifyEqual(controller.view, view, ...
                'Constructor failed to correctly set view');

            close(view.figure);

        end



    end

end
```

TestExportModel.m

```
classdef TestExportModel < matlab.unittest.TestCase

    properties

    end

    methods (Test)

        function testConstructor(testCase)

            % pass in 1 as the data argument
            model = ExportModel(1);

            testCase.verifyEqual(model.data, 1, ...
                'Constructor failed to correctly set data');



        end


    end
```

```
end
```

TestExportView.m

```
classdef TestExportView < matlab.unittest.TestCase

    properties
       model
       view
    end

    methods(TestClassSetup)

        function setup(testCase)

            testCase.model = ExportModel([1,2,3;1,2,3]);
            testCase.view = ExportView(testCase.model);

        end

    end

    methods(TestClassTeardown)

        function teardown(testCase)

            close(testCase.view.figure);
            delete(testCase.model);
            delete(testCase.view);

        end

    end

    methods (Test)

        function testConstructor(testCase)

            model = ExportModel([1,2,3;1,2,3]);
            view = ExportView(model);

            testCase.verifyEqual(view.model, model, ...
                'Constructor failed to correctly set model');

            testCase.verifyNotEmpty(view.figure, 'Constructor failed to
correctly set figure');
            testCase.verifyNotEmpty(view.fileNameLabel, 'Constructor failed
to correctly set fileNameLabel');
            testCase.verifyNotEmpty(view.fileNameTextField, 'Constructor
failed to correctly set fileNameTextField');
            testCase.verifyNotEmpty(view.sheetNameLabel, 'Constructor failed
```

```
to correctly set sheetNameLabel');
            testCase.verifyNotEmpty(view.sheetNameTextField, 'Constructor
failed to correctly set sheetNameTextField');
            testCase.verifyNotEmpty(view.startRangeLabel, 'Constructor failed
to correctly set startRangeLabel');
            testCase.verifyNotEmpty(view.startRangeTextField, 'Constructor
failed to correctly set startRangeTextField');
            testCase.verifyNotEmpty(view.includeColumnHeadersCheckBox,
'Constructor failed to correctly set includeColumnHeadersCheckBox');
            testCase.verifyNotEmpty(view.okButton, 'Constructor failed to
correctly set okButton');

            close(view.figure);

        end

    end

end
```

TestExportController.m

```
classdef TestExportController < matlab.unittest.TestCase

    properties
       model
       view
       controller
    end

    methods(TestClassSetup)

        function setup(testCase)

            testCase.model = ExportModel([1,2,3;1,2,3]);
            testCase.view = ExportView(testCase.model);
            testCase.controller = ExportController(testCase.model,
testCase.view);

        end

    end

    methods(TestClassTeardown)

        function teardown(testCase)

            close(testCase.view.figure);
            delete(testCase.view);
            delete(testCase.model);
            delete(testCase.controller);
```

```
        end

    end

    methods (Test)

        function testConstructor(testCase)

            model = ExportModel([1,2,3;1,2,3]);
            view = ExportView(model);
            controller = ExportController(model, view);

            testCase.verifyEqual(controller.model, model, ...
                'Constructor failed to correctly set model');

            testCase.verifyEqual(controller.view, view, ...
                'Constructor failed to correctly set view');

            close(view.figure);

        end



    end

end
```

TestGuiHelpers.m

```
classdef TestGuiHelpers < matlab.unittest.TestCase

    properties
        model
        view
        controller
    end

    methods(TestClassSetup)

        function setup(testCase)

            testCase.model = MissionControlModel();
            testCase.view = MissionControlView(testCase.model);
            testCase.controller = MissionControlController(testCase.model,
testCase.view);


        end

    end
```

```
    methods(TestClassTeardown)

        function teardown(testCase)

            close(testCase.view.figure);
            delete(testCase.view)
            delete(testCase.model);
            delete(testCase.controller);

        end
    end

    methods (Test)

        function testOpenExportDataGui(testCase)

            table = testCase.view.predictionsTable;

            % table is not populated with data yet, so openExportDataGui()
            % should not open the export window and open a message box
            % instead
            exportView = GuiHelpers.openExportDataGui(table);

            % Find the message box, verify it exists, then close it
            messageBoxHandle = findall(0,'Type','figure','Name','Error');
            testCase.verifyNotEmpty(messageBoxHandle, 'Error message failed
to display when trying to export empty data set');
            close(messageBoxHandle);

            % set the table data using random matrices for the angles and
            % velocities
            angles = rand(7,1);

            velocities = rand(7,1);
            tableData = [angles velocities];
            set(table, 'Data', tableData);

            % table is now populated, export window should open
            exportView = GuiHelpers.openExportDataGui(table);
            testCase.verifyNotEmpty(exportView, 'Export window failed to
open');
            % close the export figure
            close(exportView.figure);

        end

        function testAddJavaPoiLibs(testCase)

            GuiHelpers.addJavaPoiLibs();
            javaPath = javaclasspath('-dynamic');
            testCase.verifyNotEmpty(javaPath, 'POI libs were not added to
javapath');

        end
```

```
        function testErrorMessage(testCase)

            messageBoxHandle = GuiHelpers.errorMessage('test');
            testCase.verifyNotEmpty(messageBoxHandle, 'Message box failed to
open');
            errorString = messageBoxHandle.Children(3).Children.String(1);
            testCase.verifyEqual(errorString, {'test'}, 'Message box was not
set with correct message');
            % close the message box
            close(messageBoxHandle);
        end

        function testIsTextFieldValid(testCase)

            % pick one of the text fields and respective label to test with
            textFieldHandle = testCase.view.launchVelocityTextField;
            labelHandle = testCase.view.launchVelocityLabel;

            % set the string of the text field to be blank
            textFieldHandle.String = '';
            validity = GuiHelpers.isTextFieldValid(textFieldHandle,
labelHandle);
            testCase.verifyEqual(validity, false, 'Textfield validation
allowed blank value');
            % Find the error message box that was opened and close it
            messageBoxHandle = findall(0,'Type','figure','Name','Error');
            close(messageBoxHandle);

            % set the string of the text field to contain letters
            textFieldHandle.String = '123abc';
            validity = GuiHelpers.isTextFieldValid(textFieldHandle,
labelHandle);
            testCase.verifyEqual(validity, false, 'Textfield validation
allowed letters');
            % Find the error message box that was opened and close it
            messageBoxHandle = findall(0,'Type','figure','Name','Error');
            close(messageBoxHandle);

            % set the string of the text field to contain numbers with
            % spaces in between the digits
            textFieldHandle.String = '123 456 789';
            validity = GuiHelpers.isTextFieldValid(textFieldHandle,
labelHandle);
            testCase.verifyEqual(validity, false, 'Textfield validation
allowed numbers with spaces in between them');
            % Find the error message box that was opened and close it
            messageBoxHandle = findall(0,'Type','figure','Name','Error');
            close(messageBoxHandle);

            % set the string of the text field to contain numbers
            textFieldHandle.String = '123';
            validity = GuiHelpers.isTextFieldValid(textFieldHandle,
labelHandle);
            testCase.verifyEqual(validity, true, 'Textfield validation did
not allow valid value');
            % Find the error message box that was opened and close it
            messageBoxHandle = findall(0,'Type','figure','Name','Error');
```

```
            close(messageBoxHandle);

        end

        function testStripWhitespace(testCase)

            testString = ' test ';
            strippedString = GuiHelpers.stripWhitespace(testString);
            testCase.verifyEqual(strippedString, 'test', 'spaces were not
trimmed from the ends of the string');

            testString = ' t e s t ';
            strippedString = GuiHelpers.stripWhitespace(testString);
            testCase.verifyEqual(strippedString, 'test', 'spaces were not
removed from inside the string');

        end

        function testMetersToInches(testCase)

            meters = 5;
            actualInches = GuiHelpers.metersToInches(meters);
            expectedInches = meters * 39.3701;
            testCase.verifyEqual(actualInches, expectedInches, 'meters were
not converted to inches correctly');


        end

    end

end
```

TestRunner.m

```
clear
clc

testCase = TestLauncher;
res = run(testCase)

testCase = TestLaunchSimulation;
res = run(testCase)

testCase = TestMissionControlModel;
res = run(testCase)

testCase = TestMissionControlView;
res = run(testCase)

testCase = TestMissionControlController;
res = run(testCase)
```

```
testCase = TestExportModel;
res = run(testCase)

testCase = TestExportView;
res = run(testCase)

testCase = TestExportController;
res = run(testCase)

testCase = TestGuiHelpers;
res = run(testCase)
```