

MSE 312: Mechatronics Design II

Summer 2021

Integration for a Throwing Robotic Arm:

Student Name	Student ID
Sepehr Rezvani	301291960

Prepared by: Long March

Submission Date: August 9th, 2021

Abstract

Long March completed mechanical, electrical and control system of a throwing robotic arm that lunches a Lacrosse ball. This report shows the process of completing integration portion of the project. In addition, the report includes total cost breakdown, innovation, analysis, and improvements our team made in the process. Our team learned that every subsection of project must be perfected, otherwise small errors add up causing inaccuracies. For instance, this report includes results of our demonstration, errors and what caused those errors. It was concluded that multiple factors contributed to demonstration results being inaccurate. By understanding our mistakes, readers can use suggestions made in this report to improve and optimize their simulation results.

Contents

Abstract	ii
Introduction	1
Total Costs.....	2
Innovative and Creative Design	3
Improvements.....	4
Motor Upgrade	4
Fixing Simulation Breaking Bugs and Optimizations.....	6
Updated Cup Model.....	6
Rotational Actuation Joint Limits.....	7
Optimizations of Simulation	7
Solving Input Parameters Based on Desired Landing Position	8
Results.....	12
Analysis and Evaluation	14
Landing Position Testing	14
Accuracy of Position.....	15
Simulation Time Improvements.....	16
Simulation Realism.....	17
Conclusions and Further Work	18
References	19
Appendices.....	20
Appendix A: Analysis and Evaluation	20
Landing Position Testing	20
Contributions	22

Introduction

This mechatronics design project was proposed to test and develop our mechanical, electrical, and controls skills. It also provided an opportunity to simulate an industrial project with randomly assigned groupmates while working remotely. It taught us valuable lessons on working in a group, delegating tasks, and organizing focused and productive meetings. Long March was ultimately tasked with designing a throwing mechanism that is capable of accurately and repeatedly launching a projectile with known mass and size, in this case, an NCAA rated lacrosse ball, to a randomly assigned distance within 0.2-1.5m. This report seeks to demonstrate the integration process of the project. Combining a mechanical, electrical, and control system into a functioning simulation that solved the assigned problem. The report will review the total cost breakdown, innovation, analysis, and improvements made throughout the project. These results will be discussed in-depth and suggestions for improvements will be introduced.

Total Costs

Provided a \$500 budget this project provided ample room to complete the design. The project ultimately came in under budget with a price of \$291.01. Our frame's primary material was standard 6061 30x30mm aluminum extrusion. An advantage of using extrusion is the vast number of available fasteners. This prevents us from needing to weld, drill, tap, or any other time-consuming and expensive connection process. The bucket was also constructed using thin aluminum sheets that are can be easily modified to the desired dimensions. Since we used simple materials, they can be constructed at one of the group member's houses for zero cost.

It should be noted however that our design only actually required 83cm of extrusion for the entire frame. However, the minimum size that can be purchased is the full 5.3m. This means there are approximately 4.5m that could be used for other projects, sold to minimize cost towards project, or the 83 cm amount could be retrieved from a local company using their scraps.

Table 1: Project Cost Breakdown

Item Description	Part Number	Manufacturer	Lead Time (Weeks)	Cost (CAD)
Servo Motor [1]	82890001	Crouzet	2-3	\$136
Motor Controller [2]	EV-VNH7040AY	STMicroelectronics	1-2	\$8
30x30mm Aluminum 5.3m Extrusion [3]	670085	TSLOTS	0-1	93.03
M6 T-Slotted Nuts x 12 [4]	664112	TSLOTS	0-1	24

M6 x 12mm Black Oxide Finish Steel Socket Cap Screw [5]	11103339	Fastenal	0-1	20
1566 Carbon Steel Shaft	1346K11	McMaster Carr	1-2	10
Total Cost				291.03

Innovative and Creative Design

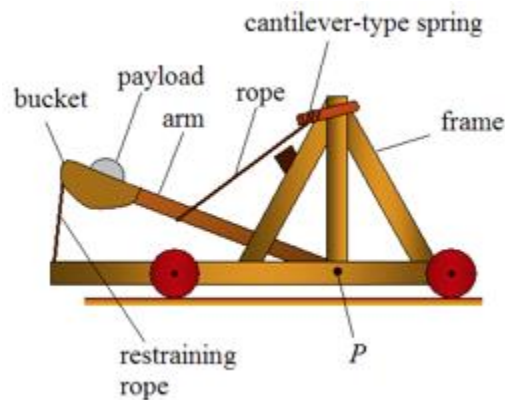


Figure 1: Mangonel Styled Catapult [7]

Simple and reliable. Our approach to our design is a modern interpretation of an ancient relic, the Mangonel catapult. The Mangonel consists of a long wood arm with a bucket (early models used a sling) with a rope attached to the end. The arm is then pulled back (from a natural 90° angle) then energy was stored in the tension of the rope and the arm. Then the bucket would be loaded. When released the Mangonel's arm would return to its equilibrium position, when it encountered the beam (or block) the arm would stop but the missiles stored in the bucket would continue to launch toward the enemy. The Mangonel fired projectiles in an overhead arc, the angle of the path of the projectile could be determined by a block placed on the beam that stopped the Mangonel's arm by using a block that stopped the arm earlier than 90° angle would result in a path angle (above the horizontal) equal to the angle between the arm and the 90° angle. [6]

Our design differs by using state-of-the-art DC motors and cascade controllers to precisely land the ball at the desired position. Each level of control was tuned to near perfection beginning with the current, then speed, and finally, position. It is constructed using aluminum extrusion that permits minuscule adjustments and mechanical tuning to be performed using a simple hex

key and without drilling and tapping new holes. Instead of tension in the arm, torque will be generated using a gearbox between the motor shaft and the shaft of the system.

Improvements

Motor Upgrade

During the testing phase of our control system, we encountered a recurring issue where the motor would reach saturation before moving the arm with the ball. This made tuning near impossible as the actual current, speed, and position would oscillate wildly around the desired value or even shoot off to infinity. This can be seen in Figure 2 below.

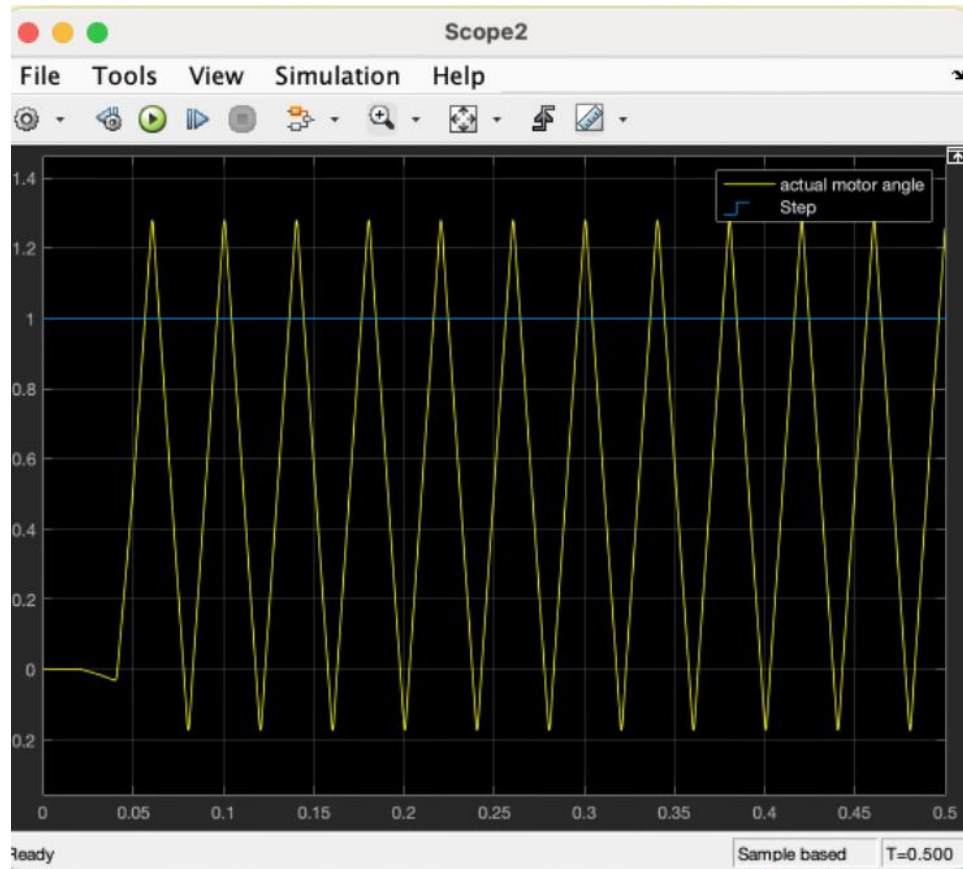


Figure 2: Position Vs. Time of Old Motor

Even with friction reduced to 10^{-12} the shaft would not move as intended. Ultimately, we decided to compromise the low-power version for a stronger motor and motor controller. This increase of rated current increased the torque to be within the nominal range required to move

the arm. We decided the increase in power and cost was worth the trade-off. See below for an overall specification comparison between the two motors.

Table 2: Motor Comparison

Specification	Initial Motor GM8724S009 From Lab	Final Motor 82890001[1]
Rated Torque	0.1Nm	0.270Nm
No Load Speed	720rpm	3200rpm
No Load Current	1.2A	3A
Rated Voltage	12V	24V
Power Consumption	4W	10.8W

After changing the motor for a more powerful one, even before tuning, it was rapidly apparent that our issue was mitigated. Even with friction on the order of magnitude 10^{-2} the arm would rotate to the desired value. Brief tuning yielded the result below.

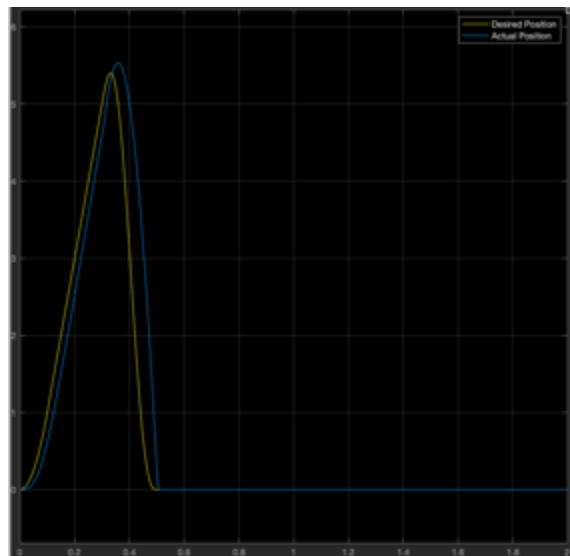


Figure 3: Desired and Actual Position vs Time - Final Motor

The decision to upgrade the motor yielded positive results. The motor was capable of launching the heavy projectile to the required distances and in a predictable manner.

Fixing Simulation Breaking Bugs and Optimizations

For simulations improvements, they can be split into two main groups. The first group is adapting the system so that it works and runs properly, and the second group is tweaking the system so that it is optimized.

Updated Cup Model

There are a few problems that needed to be fixed before the simulation would not act unreasonably. The initial problem that we ran into was our cup design was not the best for both holding the ball and importing it into MATLAB's Simulink. The initial design we used was a cone-shaped cup for the ball shown in the figure below:

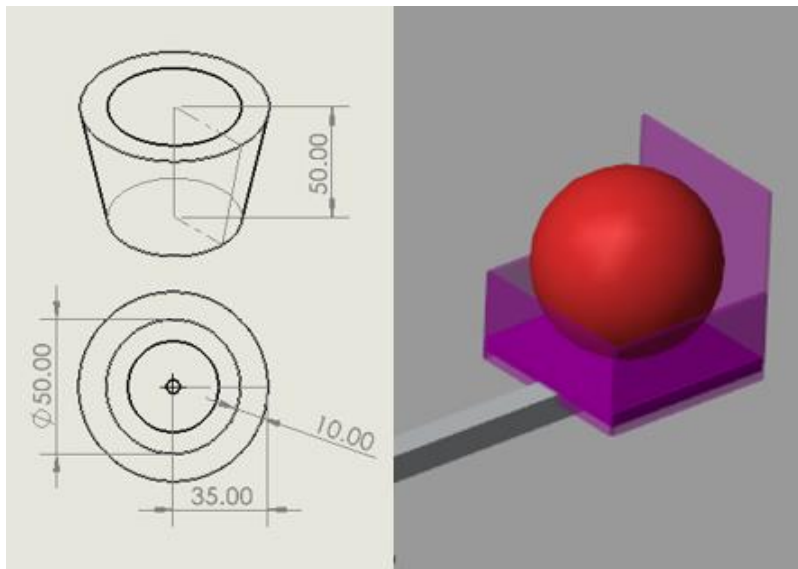


Figure 4: Left: Initial Cup Design, right: Updated Simulation Cup model

The two issues we realized with this design early on were during higher rotational speeds of the arm, it is not realistic to assume that the ball would remain stationary relative to the cup, and secondly, based on how Simulink imports SolidWorks models, it models concaved objects as a filled solid, which causes any object within it to act like it is on a trampoline. This would cause the ball to launch due to this spring force and not kinetic energy transferred from the arm. This was fixed by adjusting the model to be a box with an open top and four rectangular solids

Rotational Actuation Joint Limits

Using joint limits, we were able to create better simulations outputs and get values closer to the desired results. For the majority of the results, we limited the rotational joint to -45 to -140 degrees for the upper and lower bound. If this is not done, it would cause the ball to not launch out of the container.

Table 3: Comparison of Simulation with and without Rotational joint limits

Results 0.83m		
	With Joint Limit	Without Joint Limit
X Landing Position of Ball (m)	0.82894 Meters	0.035808 Meters
Error X Landing Position of Ball (m)	0.0010636 Meters	0.79419 Meters
Y Max Position of Ball (m)	0.37312 Meters	0.071648 Meters
Time taken for Simulation	0.56712 Seconds	0.43823 Seconds
Total Power Usage (W)	258.4577 Watts	204.6149 Watts
Elapsed time (s)	11.370518 seconds	11.674869 seconds

Something that should be noted is that depending on the actual real-life use case of the system, a hard stop to the joint limit might not be feasible. Typically, abrupt stops to a system, whether mechanical or electrical could cause harm to the integrity of the system. For now, a mechanical stop to the system will prove to be cheaper, but at the cost of a shorter life span of the system

Optimizations of Simulation

Table 4: Spatial contact Force vs Contact Force Library

Results 0.83m, Comparison with Spatial Contact Force vs Contact Force Library		
	Spatial Contact Force	Contact Force Library
X Landing Position of Ball (m)	0.82894 Meters	1.0053 Meters
Error X Landing Position of Ball (m)	0.0010636 Meters	-0.17527 Meters

Y Max Position of Ball (m)	0.37312 Meters	0.35918 Meters
Time taken for Simulation	0.56712 Seconds	0.56154 Seconds
Total Power Usage (W)	258.4577 Watts	268.8284 Watts
Elapsed time (s)	11.370518 seconds.	15.073665 seconds.

In the table above we have demonstrated the time difference of the simulation. What should be noted here is not so much the accuracy of the system between each of the block libraries used as the parameter tuning of each block is slightly different which is the main cause of the difference in position, but instead the elapsed time difference. The difference in time is almost equivalent to 50% slower using the contact Force library compared to the Spatial Contact force. This change in the system allows for faster compilation of the system with the integration of the system, as one block is just more optimized than the other.

Another way we have optimized the simulation was how we have measured the position of the ball at certain points in time. In the initial design of the Simulink block what we had in mind to obtain the value such as when the ball lands are obtaining the value of X-position when y is equal to the radius of the ball. We found with this logic of obtaining the value was that sometimes it would not work. We determined that due to how Simulink MATLAB works in discrete values, y distance will not necessarily be equal to a certain value we want. To fix this the system was changed to a latch system that works in conjunction with a hit crossing probe to detect if the ball passes a value ever so slightly above the radius of the ball.

Solving Input Parameters Based on Desired Landing Position

Long March had multiple ideas to design a system that solves for input parameters, W_{cruise} , T_{ret} , q_{ret} and a_{ret} . These variables are rotational speed arm from T_{ramp} (ramp time) to T_{ret} (time of retraction), time of retraction, the position of the arm and angular acceleration of arm at the time of retraction, respectively.

Initially, we wrote a Matlab function that had four inputs and four outputs. Inputs were Pos_{fin_x} , Pos_{fin_y} , T_{fin} and r . These are desired landing position in x and y direction, final time of simulation, and radius of arm (rotational radius), respectively. Outputs were W_{cruise} ,

T_{ret} , q_{ret} and a_{ret} . Then we wrote a function called 'practice.m' that solves for outputs using Matlab's 'Solve' function. Once the code was tested with the Simulink file, it was giving imaginary values for outputs. Therefore, we had to assume a_{ret} as a known parameter for Trial 2 instead of solving for it during the next stage of our design. This code will be included in 'Trial 1' folder, and will include 'main.m' (main function that calls solver function), and 'practice.m'. Table 5 shows function call, number of inputs and outputs in the main script and practice function.

Table 5: Inputs and Outputs in 'main.m' and 'practice.m'

Outputs	Inputs
'main' script	
[W_cruise, T_ret, q_ret, a_ret]	(2.5,0,T_fin,r)
'practice' function	
[W_cruise2, T_ret2, q_ret2, a_ret2]	(Pos_fin_x1, Pos_fin_y1, T_fin1, r1)

Next, we followed a similar process, but chose a_{ret} to be constant. The problem in this trial was that T_{ret} solved, but results were unreasonable. In some cases, negative, in others bigger than 1, which was incorrect since T_{fin} was 0.5. Table 6 shows how the program solved for T_{ret} , while changing a_{ret} . It can be observed that there is a mistake, either in the code or in the Simulink design. This code will be included in 'Trial 2' folder.

Table X – T_{ret} Relationship with a_{ret} ($T_{fin} = 0.5$)

Table 6: shows function call, number of inputs and outputs in the main script and practice function.

Return Acceleration: a_{ret}	Return Time: T_{ret}
-1	-1.2321
-3	-0.5000
-5	1.2746
-7	1.1547

Table 7: Inputs and Outputs in 'main.m' and 'practice.m'

Outputs	Inputs
'main' script	
[W_cruise, T_ret, q_ret]	(2.5,0,T_fin,r)
'practice' function	
[W_cruise2, T_ret2, q_ret2]	(Pos_fin_x1, Pos_fin_y1, T_fin1, r1)

Then it was observed that using a constant for q_ret as well seems to be the only solution, and always solves for T_ret within reasonable range. This code will be included in 'Trial 3' of folder. Note that Matlab files in trial folders are intended observation of coding progress. Simulink file is not included, hence Mechanics Explorer does not run. Details of this code will be explained in 'Analysis and Evaluation' section of this report. Table 8 shows function call, number of inputs and outputs in the main script and practice function.

Table 8: Inputs and Outputs in 'main.m' and 'practice.m'

Outputs	Inputs
'main' script	
[W_cruise, T_ret, Vel_fin_y, limit_value]	(des_pos,0.0315,T_fin,0.14,q_ret,a_ret)
'practice' function	
[W_cruise2, T_ret2, Vel_fin_y2, limit_value]	(Pos_fin_x1, Pos_fin_y1, T_fin1, r1, q_ret1,a_ret)

Note: q_ret and a_ret constants have been added to inputs. Vel_fin_y (ball velocity in y-direction, at landing) and limit_value (arm's starting angle) have been added to outputs.

Another code change that improved our virtual prototype was introducing multiple if/else loops in 'practice.m' function. Long March proposed multiple solutions that can increase accuracy of landing position. Eventually, it was observed that q_ret must be manually changed based on desired landing position given by the instructor. In a perfect virtual design, q_ret must be solved automatically. However, as mentioned in the previous paragraph, q_ret was chosen to

be a constant. Therefore, q_{ret} was selected by trial and error for each landing range based on 'Pos_fin_x'. Refer to 'Trial 3' folder to see code changes.

Last change that contributed to design improvement was introducing `limit_value` in 'practice.m' function. When Long March was testing desired landing positions that were relatively further ($>1.2\text{m}$), it was observed the ball does not seem to reach that far. Since the electrical system was already tuned and could not choose a stronger DC motor, we changed initial arm position. In 'practice.m', it can be seemed values of `limit_value` is decreasing for further throws, indicating lower initial arm position. Refer to 'Trial 3' folder to see code changes.

Results

Given Desired Throwing Positions 0.22, 0.83, 1.27

For 0.22 a value of 0.245 was inputted during demonstrations as 0.22 was not working as intended due to unknown reasons. As the presentations were not recorded to reobtain the output values, the simulation was a rerun, which may cause simulation and elapsed time to vary slightly than on presentation date.

Table 9: Results of 0.22m Desired Position w/ Simulation

Results 0.22m (Input 0.245)	
X Landing Position of Ball (m)	0.21373 Meters
Error X Landing Position of Ball (m)	0.031267 Meters
Y Max Position of Ball (m)	0.26466 Meters
Time taken for Simulation	0.52295 Seconds
Total Power Usage (W)	212.1716 Watts
Elapsed time (s)	10.980934 seconds.

Table 10: Results of 0.83m Desired Position w/ Simulation

Results 0.83m	
X Landing Position of Ball (m)	0.82894 Meters
Error X Landing Position of Ball (m)	0.0010636 Meters
Y Max Position of Ball (m)	0.37312 Meters
Time taken for Simulation	0.56712 Seconds
Total Power Usage (W)	258.4577 Watts
Elapsed time (s)	11.370518 seconds.

Table 11: Results of 1.27m Desired Position w/ Simulation

Results 1.27m	
X Landing Position of Ball (m)	1.2388 Meters
Error X Landing Position of Ball (m)	0.031177 Meters
Y Max Position of Ball (m)	0.4365 Meters
Time taken for Simulation	0.60964 Seconds
Total Power Usage (W)	320.7294 Watts
Elapsed time (s)	10.782429 seconds.

Table 12: Results of Max Y Position w/ Simulation

Results Highest Throw (input of 1.5m, lower Limit=-140, upper Limit = -90)	
X Landing Position of Ball (m)	-0.013352 Meters
Y Max Position of Ball (m)	0.52308 Meters
Time taken for Simulation	0.66496 Seconds
Total Power Usage (W)	462.8029 Watts
Elapsed time (s)	11.488490 seconds

Table 13: Results of Furthest Position w/ Simulation

Results Furthest Throw (input of 1.5m, lower Limit=-140, upper Limit = -45)	
X Landing Position of Ball (m)	1.4983 Meters
Y Max Position of Ball (m)	0.47328 Meters
Time taken for Simulation	4.1461 Seconds
Total Power Usage (W)	454.5162 Watts
Elapsed time (s)	22.094963 seconds.

Analysis and Evaluation

Landing Position Testing

Once the Simulink program was improved, Long March's goal was to throw the ball with high accuracy by improving the Matlab code. To achieve that goal, our team focused on two things: first selecting kinematics formulas, then modifying the 'practice.m' code. In this section, we will explain how these two factors contributed to demonstration results, and how they can be improved in the future.

As mentioned in the Improvements section, the 'main' script solves for W_{cruise} and T_{ret} , uses those values and other parameters to run the Simulink program after. During our demonstration, three desired landing positions were given, and our simulation had 2.85%, 0.12%, and 2.46% error for each throw. One factor contributing to these errors is the kinematic formulas used. Here are two main formulas used for solving unknowns, W_{cruise} and T_{ret} :

$$\Delta x = \left(\frac{v + v_0}{2}\right)t$$

$$v^2 = v_0^2 + 2a\Delta x$$

For a more detailed explanation of how 'practice' function was created, refer to Appendix.

These equations were used for a perfect throw condition without considering forces acting against the arm or ball at retraction time. So, these factors could have affected final errors. For future work, friction (for e.g. air friction) can be implemented in formulas to increase accuracy of final results.

Second factor that had a major effect on final errors is 'practice' function. As mentioned, in last part of 'Improvements' section, we had to choose q_{ret} by trial and error depending on desired landing position. Figure X is a sample of 'practice' function.

```
if Pos_fin_x1 == 0.2
    q_ret1 = 2.08;
elseif Pos_fin_x1 > 0.2 && Pos_fin_x1 < 0.3
    q_ret1 = 1.7;
```

Figure 5: Sample Selected from 'practice' function

It can be observed that q_{ret} is manually set depending on Pos_{fin_x} (desired landing position in x-direction). So, we can be certain that error will be minimal if Pos_{fin_x} is divisible by 0.05 m, since that is how 'practice' function was created. During our demonstration, neither of three values given were divisible by 0.05 m, hence contributing to error. For future work, we can increase number of if/else loops in the 'practice' function to include more possible landing position ranges.

Accuracy of Position

Overall, the results did not exactly meet the project requirements we set out to complete of being able to hit our target of $\pm 2\text{cm}$ of the target system. When we did our final testing, we were mainly concerned with values from 0.2 meters to 1.5 meters at intervals of 0.05 meters. Most of our simulation values were able to obtain these values relatively closely. We were able to get close to the desired position in two of the positions (0.22 and 1.27 meters) and get within 2 mm of the last position (0.83). With our testing method, we had assumed that the intermediate values between each input and output of the tested values would have a linear relationship. This was not the case. During our live presentation, the input value of 0.22 did not output a very desirable outcome. After receiving the approval of Dr. Helen Bailey, we reattempted the simulation run with an input value close to the desired but not the desired. In our case, we used the value of 0.245m. This input gave us an output of 0.21373 meters. This gives us a percent error of 2.85% and a value of 6mm off from the desired (Observed from the table below).

Table 14: Error Table for Each of the Demonstration Values

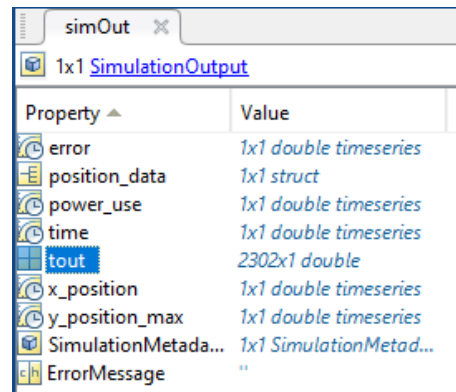
Desired Position (m)	Actual Position (m)	Error (m)	% Error
0.22	0.21373	0.00627	2.85%
0.83	0.82894	0.00106	0.13%
1.27	1.2388	0.0312	2.46%

This relationship can be observed somewhat in other values in the simulation. It is also possible that input values used can give pretty accurate output results. The biggest issue right now is the reliability of our results. A good portion of our values is "hardcoded" and does not change at a

high enough resolution for example qret. This error is compounded by the fact that it is required for us to create joint limits for the system to work properly. The most pertinent issue with our simulation right now is to reevaluate the logic of our code and double-check if the system is performing as we intended. This would require extensive bug checking and creating reasonable test cases for the system. As it stands, our simulation could be made more accurate and better through less hard coding of values like qret or hard limits on the joints instead of being set through the controls system, and the introduction to more equations and relationships between the parameters.

Simulation Time Improvements

Our ability to eliminate irrelevant portions of the robotic arm to the simulation has allowed us to help lower the simulation time to a total of only 33.13 seconds for all three simulation distances. Some of the techniques optimizations routes we have taken include modeling of the motor on the robotic arm system. It is undoubtedly computationally heavy to have to create and animate the motor attached to the arm shaft. It would have looked better in simulation but it would not change our outputs as the parameters of the motor are already considered in the Simulink program. Although our simulation is already somewhat fast it can be improved by looking at if certain values can be calculated faster in our function, or if the logic between blocks of the Simulink can be wired more efficiently. We believe that improving how values are computed and how the program run will provide the most meaningful change to the system is because Matlab's Simulink acts as a while loop that works with really small discrete values over time. If an operation or logic can be evaluated with one less time each loop, then throughout the run, it could save potentially thousands of instructions. From the simulation output, we can see in a simulation time of 1.2407 seconds long, the simulation has created over 2000 output values for one variable.



The screenshot shows a MATLAB/Simulink window titled 'simOut'. It displays a table of simulation output properties for a '1x1 SimulationOutput' block. The 'tout' property is highlighted in blue. The table lists various properties and their corresponding data types and sizes.

Property	Value
error	1x1 double timeseries
position_data	1x1 struct
power_use	1x1 double timeseries
time	1x1 double timeseries
tout	2302x1 double
x_position	1x1 double timeseries
y_position_max	1x1 double timeseries
SimulationMetada...	1x1 SimulationMetad...
ErrorMessage	"

Figure 6: Simulation Outputs

We can also look to see if it is possible to calculate each of the output values off of one variable as opposed to each variable individually. So certain high-cost calculations concerned with only the final values will only have to be done once as opposed to for each variable.

Simulation Realism

From our current system, we have elected to omit a pretty important factor that exists in real life, friction. This parameter would reduce the accuracy of our system in a real-life situation. The reason we have chosen to neglect this portion of the system is because of the high computational power needed in simulating aspects like aerodynamics of a system or air resistance, or complications it would add to our equations with values like friction between the ball and the cup or the shaft and the supporting beams. The next steps to improving the realism of the system would be through simulating better frictional values between the cup and the ball in the system and taking them into account in our functions when solving for parameters to input to the system.

Additional assumptions were subtle deflections in the mechanical system itself and the withdraw of bearings along the shaft. These assumptions were made due to the marginal, though non-zero, error it would cause in the simulation. If the design were to be created in real life these critical parameters would have to be accounted for.

Conclusions and Further Work

In terms of the initial deliverables set out for this project, we were able to satisfy most of the constraints. Here are the main constraints of the project in comparison to our simulation in numbers:

1. Total Cost of Under \$500, ours being 291.03
2. Total Simulation Time of under 3 minutes, ours being 33.1332 seconds of elapsed time
3. Operating Radius of under 30 cm, ours being 25 cm in operating radius
4. Throwing distance of 0.2-1.5 meters, ours being 0-1.5 meters
5. Resolution of throwing accuracy ± 2 cm, ours being ± 5 cm

Of the main constraints satisfied, we were able to satisfy constraints 1-4, with only the resolution of throwing accuracy being slightly off. Unfortunately, we were not able to properly create a model and simulate a Robotic Arm to an accuracy of ± 2 cm. Although the accuracy of the system could not be achieved, we were still able to get it in the range of ± 5 cm. For further improvements into the system, the main project goal would be to be able to control the robotic arm to throw the lacrosse ball to an accuracy of ± 2 cm. To do this we would have to revisit the parameters of the system, how they were created, and how the Simulink diagram handles the input parameters.

Aside from being able to model how the “perfect” model performs in the real world, this project does not take into manufacturing tolerances of the system. At the end of the day, our SOLIDWORK file imported into the simulation has been created with basically perfect fittings. With different manufacturing tolerances. For further works in the future, we should investigate how the manufacturing of the system would work, and to what degree of tolerances the system needs to adhere.

References

- [1] P. C. B. E. K. Y. S. Chain, "82890001 Gear Servo," *PCBEKY Supply Chain*, 26-Jul-2021. [Online]. Available: https://www.pcbekey.com/products/1_585431/82890001. [Accessed: 09-Aug-2021].
- [2] "Ev-vnh7040ay," *DigiKey*. [Online]. Available: <https://www.digikey.com/en/products/detail/stmicroelectronics/EV-VNH7040AY/7691037>. [Accessed: 09-Aug-2021].
- [3] "670085," *Proax*. [Online]. Available: <https://www.proax.ca/en/product/1338767/TSL670085>. [Accessed: 09-Aug-2021].
- [4] "664112," *Proax*. [Online]. Available: <https://www.proax.ca/en/product/1337048/TSL664112>. [Accessed: 09-Aug-2021].
- [5] "Carr," *McMaster*. [Online]. Available: <https://www.mcmaster.com/1346K11/>. [Accessed: 09-Aug-2021].
- [6] "History of catapults - physics of catapults," *Google Sites*. [Online]. Available: <https://sites.google.com/site/physicsofcatapults/home/history-of-catapults>. [Accessed: 09-Aug-2021].
- [7] "Catapult," *Wikipedia*, 24-Jun-2021. [Online]. Available: <https://en.wikipedia.org/wiki/Catapult>. [Accessed: 10-Aug-2021].

Appendices

Appendix A: Analysis and Evaluation

Landing Position Testing

This section included hand calculations and shows how the 'practice' function was created.

Table 15 gives a brief description of each variable. Note that 'initial' and 'final' position of the ball are based on Figure 7.

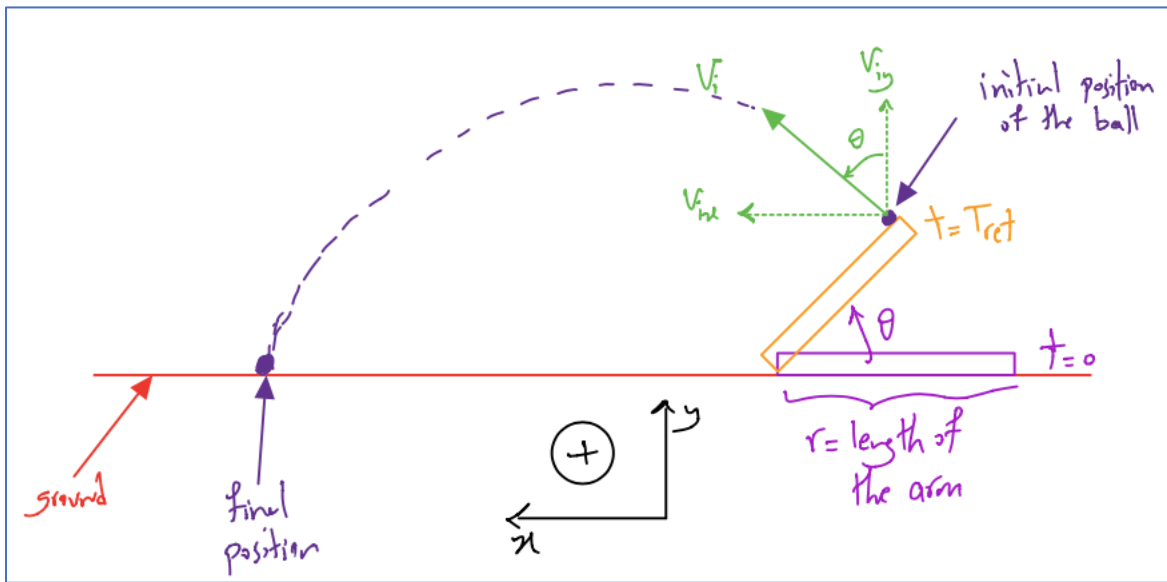


Figure 7: Free Body Diagram of the Ball at Retraction Time

Table 15: Variable Names, Description and Status

Variable Name	Description	Status
P_{fx}	Ball position when lands (x-axis)	Known
P_{ix}	Initial ball position (x-axis)	Known
P_{fy}	Ball position when lands (y-axis)	Known
P_{iy}	Initial ball position (y-axis)	Known
r	Radius of the arm	known
ω_{cruise}	Angular velocity of ball between ramp time and retraction time	Unknown, will be solved
q_{ret}	Angular position of arm at retraction time	known
V_{fx}	Ball velocity when lands (in the x-direction)	known
T_{fin}	Final time	Known

T_{ret}	Retraction time	Unknown, will be solved
V_{fy}	Ball velocity when lands (y-axis)	Unknown, will be solved

$$\Delta x = \frac{V + V_0}{2} t$$

$$\rightarrow \text{(in x-direction)} P_{fx} - P_{ix} = \left(\frac{r \times \omega_{cruise} \times \sin\left(\frac{q_{ret} \times 180}{\pi}\right) + V_{fx}}{2} \right) \times (T_{fin} - T_{ret})$$

$$\rightarrow P_{fx} - \left(-0.095 - \left(r \times \cos\left(\frac{q_{ret} \times 180}{\pi}\right) \right) \right) = \left(\frac{r \times \omega_{cruise} \times \sin\left(\frac{q_{ret} \times 180}{\pi}\right) + V_{fx}}{2} \right) \times (T_{fin} - T_{ret})$$

$$\rightarrow \text{(in y-direction)} P_{fy} - P_{iy} = \left(\frac{r \times \omega_{cruise} \times \cos\left(\frac{q_{ret} \times 180}{\pi}\right) + V_{fy}}{2} \right) \times (T_{fin} - T_{ret})$$

$$\rightarrow P_{fy} - \left(0.095 + \left(r \times \sin\left(\frac{q_{ret} \times 180}{\pi}\right) \right) \right) = \left(\frac{r \times \omega_{cruise} \times \cos\left(\frac{q_{ret} \times 180}{\pi}\right) + V_{fy}}{2} \right) \times (T_{fin} - T_{ret})$$

$$V^2 = V_0^2 + 2a\Delta d$$

$$\rightarrow \text{(in x-direction)} V_{fx} = V_{ix} \rightarrow \text{due to 0 air resistance assumption}$$

$$\rightarrow 0 = 2 \times \left((r \times a_{ret}) \times \sin\left(\frac{q_{ret} \times 180}{\pi}\right) \right) \times (P_{fx} - P_{ix})$$

$$\rightarrow \text{(in y-direction)} V_{fy}^2 - \left((r \times \omega_{cruise}) \times \cos\left(\frac{q_{ret} \times 180}{\pi}\right) \right)^2 =$$

$$2 \times \left((r \times a_{ret}) \times \cos\left(\frac{q_{ret} \times 180}{\pi}\right) - 9.81 \right) \times (P_{fy} - P_{iy})$$

Contributions

Member Name	Contribution
[REDACTED]	<ul style="list-style-type: none"> • Integrated Mechanical System with the Electrical System into the Simulink model. • Created Simulink Model • Obtained Simulink model parameter displays into Matlab function • Research of integration techniques of Robotic Throwing Arm • Report Writing, Editing
[REDACTED]	<ul style="list-style-type: none"> • New motor selection and integration • New motor controller selection and integration • Tuning of the Simulink cascade controllers (current, speed, and position) • Researched local suppliers for materials • Manufacturing methods • Innovative and creative design section
Sepehr Rezvani	<ul style="list-style-type: none"> • Hand calculations based on selected kinematic formulas • Coding practice.m function and main.m script • Report: <ul style="list-style-type: none"> - Abstract - Solving Input Parameters Based on Desired Landing Position - Landing Position Testing