MEAM 2480-101 Lab 2 Final Report

Ashna Khemani, Katie Li, Zihao Zhou

DESIGN AND APPROACH

For this lab, we designed our launcher around the principle of energy conservation. We adopted springs as our energy source. We manually pull on a set of springs, then release them, converting the stored elastic potential energy in the springs to the kinetic energy of the ball. We chose springs over rubber bands because spring force can be accurately modelled as linear function of extension, so we can easily gauge our energy input by measuring the extension. Additionally, springs perform more consistently than rubber bands, which will deform and degrade over time, reducing our control over the energy input.

Our springs are integrated with a launch track and shooting board. The launch track is a rectangular piece of laser-cut acrylic with a 2-inch gap in the middle along which the ball travels. This gap functions like a rail way, constraining the ball on both sides, ensuring a clean straight shot. On the two side edges of the track, we mounted a pair of vertical slotted walls. The springs run from the front of the track through these two slots on the side walls and connect to the shooting board at the very back of the launcher. We propped the front end of the track on top of a simple screw and bolt inserted into an arced slot in the outer wall of the angle setter. The base of the track rested on a pivot at the center of the arc.

We purchased our own set of springs with spring constant k = 8N/cm. We marked the launch velocity along the track in increments of 2 m/s, calculated using energy conservation $\frac{1}{2}mv^2 = \frac{1}{2}kx^2$. The method we used to model the trajectory will be discussed in later sections.

PERFORMANCE

Our model predicted an initial condition set of **52.5 degrees and 6.25 m/s.**

Attempt 1: attempt1.MOV

- This attempt followed our model's calculation exactly and was a severe undershot.
- Noting that we lost energy, we agreed to pull harder for the next attempt. There was also a chance that the ball had accidentally hit a hand or surface at some point.

Attempt 2: attempt2.mov

- Similar to attempt 1, this was much closer but still an undershot despite an increase in power to a theoretical 10 m/s launch velocity. We agreed to pull even harder for the next attempt.

Attempt 3: attempt3.mov

- This was our **best attempt**. It was a slight overshot of 30cm.
- We set the launch velocity to about 14 m/s. We saw here that our angle was set too high.

Attempt 4:attempt4.mov

- This was a severe overshot. We reduced the angle to 47.5, a 5 degree reduction, but retained a similar launch power since our velocity adjustment system seemed unreliable.
- This severe overshot actually matched our model's prediction for the initial conditions we used, notably due to the more than doubling of the intended velocity.
- The trajectory looked reasonable. We decided to match the model again for the next attempt.

Attempt 5: attempt5.MOV

- This was an undershot. In a last ditch effort, we lowered the power to what our model predicted and set the angle to 50, a 2.5 degree increase.

Attempt 6: (Rerun on Monday)

Our model predicted an initial condition set of **55 degrees and 5 m/s.**

- Performed some recalibrating test shots; decided to do **54 degrees and 8 m/s.**
- Hit the wall target, slight undershot in horizontal distance.

MODELLING

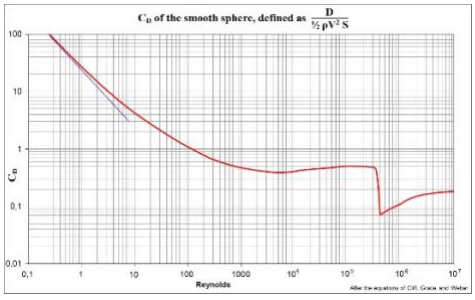
Trajectory Computation and Initial Condition Search Code

To find the precise launch velocity and launch angle that allow the test ball to pass through the designated hole and land on the designated target, we formulate the equation of motion for the ball with air drag; we then assign a random initial condition to this equation and solve for the trajectory of the ball numerically in MATLAB; finally we implement a MATLAB search code that loops over all possible trajectories generated by all possible initial conditions then identifies the trajectory and the accompanying initial condition that allows the ball to pass exactly through the designated hole and land on the designated target.

Formulating Equations of Motion

First, we select the best model for drag force on ball based on an estimated in-flight Reynolds number. Given the range of target and hole positions announced in lecture, we estimated an average flight speed of 4-6m/s. We then plug in the corresponding Reynolds number for this speed range into the following Drag Coefficient vs Reynolds Number graph, obtaining a drag coefficient of 0.45. The estimated Reynolds number also tells us that the drag force falls into the quadratic regime: $F_{Drag} \propto V^2$, therefore we can model the drag force as $\frac{1}{2}A\rho V^2 Cd$

For concision, we let $k = \frac{1}{2}A\rho Cd = 0.003459\ kg/m$ for our test ball.



Incorporating the expression for drag force, we formulate the governing differential equations of motion for the ball in flight:

$$\frac{dVx}{dt} = \frac{-k}{m}(Vx^2 + Vy^2)^{1/2} * Vx \qquad \text{EQ1}$$

$$\frac{dVy}{dt} = -g - \frac{k}{m}(Vx^2 + Vy^2)^{1/2} * Vy \quad \text{EQ2}$$

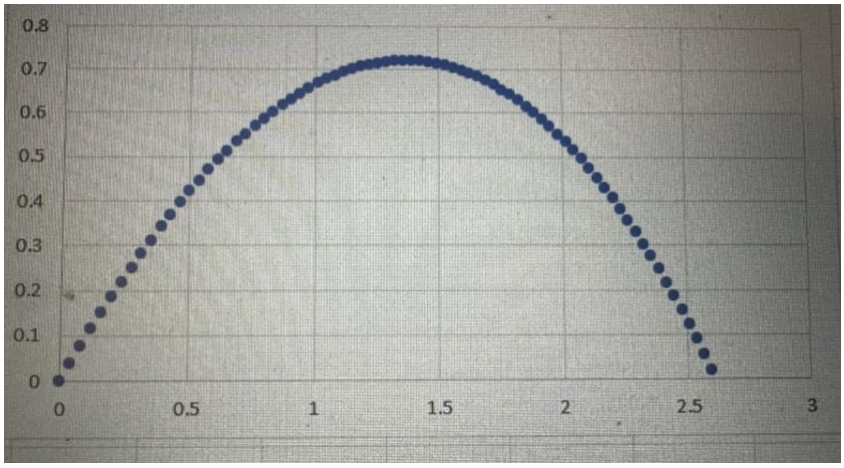
where Vx and Vy are respectively the x and y component of the velocity, and m=0.045kg

Solving Equations of Motion

We now solve this system of coupled non-linear ODEs numerically for initial condition Vx0 and Vy0. Accuracy and efficiency are our two main considerations when choosing the appropriate numerical ODE solver. We want to solution to be highly accurate but also not consume too much computing power because in our final step we need to loop over all possible initial conditions to find the optimal trajectory. Our ODE solver will run through once per each initial condition looped, amounting to a considerable calculation load.
We evaluate the trade-off between accuracy and efficiency. After some tests, we decide on 2nd order Runge Kutta solver, also known as the Predictor-Corrector scheme. This solver is much more accurate than Explicit Euler and at the same time much less time-consuming than higher order Runge Kutta schemes. We write our own 2nd order Runge Kutta solver in MATLAB:

```
% define grid spacing
h=0.001;
% define domain
T=5;
t =0:h:T; t=t';
% initialize solution
X = t*0; Y = t*0; Vx = t*0; Vy = t*0;
X(1) = X0; Y(1) = Y0; Vx(1) = Vx0; Vy(1) = Vy0;
for i = 1:length(t)-1
    %predictor values
    Xp = X(i) + 0.5*h*Vx(i);
    Yp = Y(i) + 0.5*h*Vy(i);
    Vxp = Vx(i) + 0.5*h*(-k/m)*(Vx(i)^2+Vy(i)^2)^0.5*Vx(i);
    Vyp = Vy(i) +0.5*h*(-k/m)*(Vx(i)^2+Vy(i)^2)^0.5*Vy(i)-0.5*h*9.81;
    %corrector values
    X(i+1) = X(i) + h*Vxp;
    Y(i+1) =Y(i) +h*Vyp;
    Vx(i+1)= Vx(i) + h*(-k/m)*(Vxp^2+Vyp^2)^0.5*Vxp;
    Vy(i+1)= Vy(i) + h*(-k/m)*(Vxp^2+Vyp^2)^0.5*Vyp-h*9.81;
end
% END TRAJ GENERATION
```

Implement this ODE solver, we obtain the trajectory. The trajectory is asymmetric, the effect of air drag is evident.



Initial Condition Search Code

Our final step is writing a search code in MATLAB that loops over all possible initial conditions and identifies the specific initial condition(s) that allow the trajectory to pass through the hole and landing on a target. The search code incorporates a double for-loop for launch velocity Vs and launch angle θs with adjustable loop range and spacing, and two Booleans that decide whether “Target Hit” and “Hole Passed” are satisfied, the Boolean has an adjustable error tolerance range. The full code is given below.

```
% DOES TRAJ INCLUDE TARGET?
for ind=1:length(X)
    if Y(ind)>0 && Y(ind+1)<0
        if X(ind)>(Xt-err_margin) && X(ind)<(Xt+err_margin)
            targ_hit = true;
        end
    end
    if targ_hit == true
        break
    end
end

% DOES TRAJ INCLUDE HOLE?
for ind=1:length(X)
    if X(ind)>(Xh-err_margin) && X(ind)<(Xh+err_margin)
        if Y(ind)>(Yh-err_margin)&& Y(ind)<(Yh+err_margin)
            hole_passed = true;
        end
    end
    if hole_passed == true
        break
    end
end
```

Implementing the ODE solver and search code in sequence, we can find the initial launch speed and angle that allows the ball to hit within ± 0.5cm range of the target. Formal analysis of error is addressed in the next section.

Error and Analysis, Solution Convergence:

For any numerical work, it is critical to have a formal framework to quantify the error. It can be formally derived that our RK-2 ODE solver is 2^{nd} order accurate, meaning if the grid

spacing is reduced by half, the resultant error reduces by a factor of 4. On test day, we adopted a grid spacing of 0.01 seconds. We have tested finer grid spacing up to 0.0001 second but the result doesn't change by a measurable amount, demonstrating that 0.01 second grid spacing is fine enough to generate the "true" solution to the governing ODEs. This is the grid convergence test for our numerical solver. Confident with accuracy our ODE solver, we now evaluate the error of our search code. Because all numerical methods are discrete in time, it is highly unlikely if not totally impossible that there is a particular time point Tj where our computed trajectory will exactly pass through the hole position or target position. Therefore, we must examine a small range around the hole and target position, which we call margin of error. For example, suppose the hole is located at X=1, Y=1, we cannot define the Boolean Target Hit == true if and only if when X=1, Y=1, instead we must define the Boolean Target Hit == true if and only if when $X=1\pm h$, $Y=1\pm h$, where h is the margin of error. At h=0.5cm, and looping over Vs and θs at 0.01m/s and 0.01deg interval, our search code is able to consistently find valid solution for any given hole and target positions. For smaller h such as h=0.1cm, our code can find valid solution for some, but not all given target and hole positions. We can further reduce h by reducing the loop spacing, for example loop Vs at 0.001m/s, but the computing power of our laptop is not sufficient to comfortably calculate loop spacing finer than 0.01 Therefore, we conclude that our code has an error of 0.5cm, although in some cases it can do even better.

REFLECTIONS

Our theory is that our model was very accurate, given the results of Attempt 4, but our build was not robust and reliable enough to replicate the expected results. Once our MATLAB model was finally tuned, it produced a trajectory with a 0.5cm tolerance. The results it returned seemed reasonable to physical intuition.

However, time was a constraint for us during this project. Though we believe our design was on the right track and had good potential, we did not allow ourselves enough time to test, validate, and iterate on more thoughtful solutions; we essentially went into the final performance test with the first prototype. This was largely due to a lack of availability right out of spring break, as well as the delays with the laser cutting machines, which meant we were waiting longer than anticipated in our work sessions and ended up with many unusable or flawed parts due to cutting errors.

As a result, we had a system that was difficult to operate and had many places where errors and inconsistencies were introduced. For example, our U-shaped track was very long and only connected across by a 1-inch band of acrylic at the back. This meant when the launcher was pulled back, the ends would twist, and one side of the track would go up. We believe this may have led to the undershooting scenarios, since a lot of the energy was lost to that elastic deformation on the pull, and snap-back to neutral on release.

Were we to do this project again, we may engage in better team and time management, which would allow for better engineering under less pressure and constraints. For example, to avoid the track twist described above, we could design a bracket to go on the underside of the track that would keep both sides parallel. We also could have ideally used MDF, which would have been sturdier and prevented as

much flexing, but were unable to this time as only acrylic would cut properly in the machines. We would also conduct more test runs with sample hole and target coordinates to validate a model and have a more consistent and quicker launching strategy.

We also would consider a different method of energy input. Though the springs are easier to model and we would have certainly needed to properly collect experimental data for rubber bands or bungee cords, their stretchiness makes it much easier to launch (These springs were strong for their size but very stiff.) and their flexibility would work better around various designs, like a trebuchet.

APPENDIX

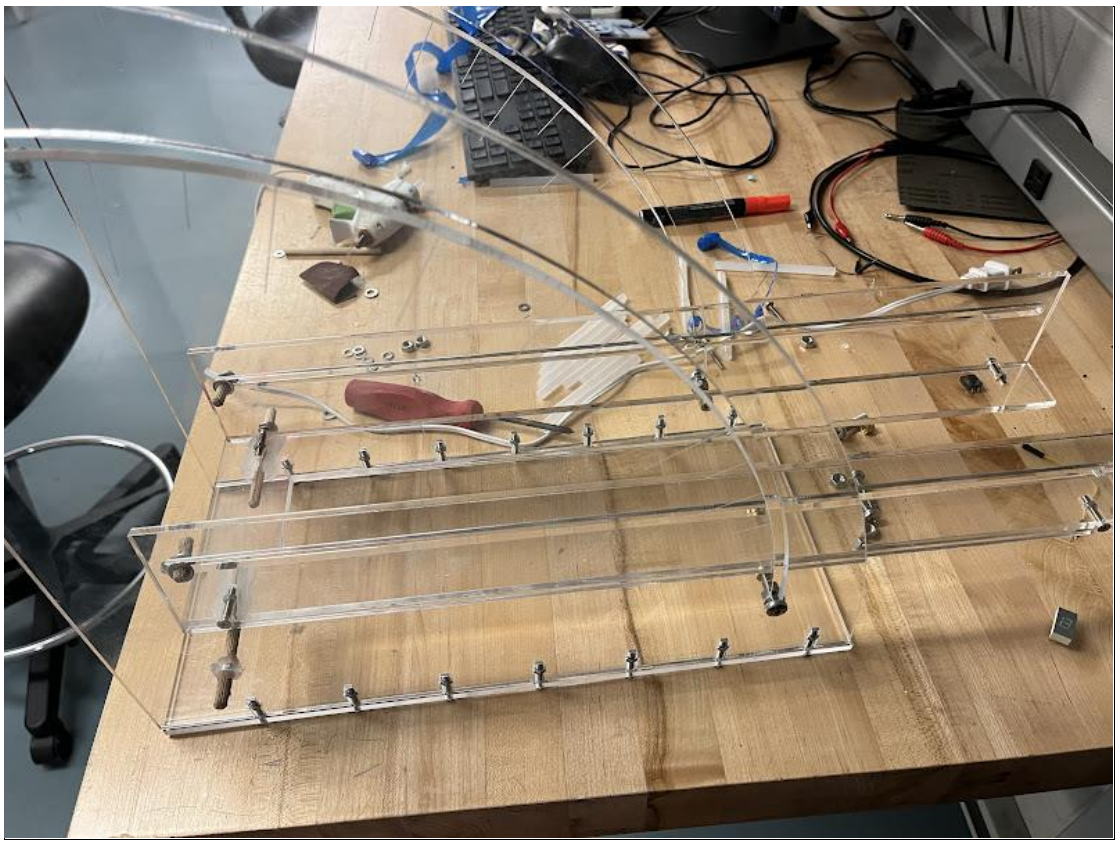


Figure 1. Final design of the launcher, though without the springs and launch holder attached.

COPY OF THE FULL CODE BELOW

```
Xh = 1;
Yh = 0.8;
Xt = 2;
err_margin = 0.05;
valid_vs = [];
valid_thetas = [];
```

```
for V = 0:0.01:7
    for theta = 0:0.01:85
        targ_hit = false;
        hole_hit = false;
        % GENERATE TRAJ
        % the equations to solve are:
        %dVx/dt = -k/m (Vx^2 +Vy^2)^0.5 *Vx;
        %dVy/dt = -g-k/m(Vx^2 +Vy^2)^0.5 *Vy;
        % implement RK2;
        % define initial condition;
        X0 =0;
        Y0 =0;
        Vx0 = V*cosd(theta);
        Vy0 =V*sind(theta);
        % parameter values
        k = 0.003459;
        m = 0.04;
        % define grid spacing
        h=0.001;
        % define domain
        T=5;
        t =0:h:T; t=t';
        % initialize solution
        X = t*0; Y = t*0; Vx = t*0; Vy = t*0;
        X(1) = X0; Y(1) = Y0; Vx(1) = Vx0; Vy(1) = Vy0;
        for i = 1:length(t)-1
            %predictor values
            Xp = X(i) + 0.5*h*Vx(i);
            Yp = Y(i) + 0.5*h*Vy(i);
            Vxp = Vx(i) + 0.5*h*(-k/m)*(Vx(i)^2+Vy(i)^2)^0.5*Vx(i);
            Vyp = Vy(i) +0.5*h*(-k/m)*(Vx(i)^2+Vy(i)^2)^0.5*Vy(i)-
0.5*h*9.81;
            %corrector values
            X(i+1) = X(i) + h*Vxp;
            Y(i+1) =Y(i) +h*Vyp;
            Vx(i+1)= Vx(i) + h*(-k/m)*(Vxp^2+Vyp^2)^0.5*Vxp;
            Vy(i+1)= Vy(i) + h*(-k/m)*(Vxp^2+Vyp^2)^0.5*Vyp-h*9.81;
        end
        % END TRAJ GENERATION

        % DOES TRAJ INCLUDE TARGET?
        for ind=1:length(X)
            if Y(ind)>0 && Y(ind+1)<0
                if X(ind)>(Xt-err_margin) && X(ind)<(Xt+err_margin)
```

```
                targ_hit = true;
            end
        end
        if targ_hit == true
            break
        end
    end

    % DOES TRAJ INCLUDE HOLE?
    for ind=1:length(X)
        if X(ind)>(Xh-err_margin) && X(ind)<(Xh+err_margin)
            if Y(ind)>(Yh-err_margin)&& Y(ind)<(Yh+err_margin)
                hole_hit = true;
            end
        end
        if hole_hit == true
            break
        end
    end

    % RECORD VALID SOLUTION
    if hole_hit && targ_hit
        valid_vs(end+1) = V;
        valid_thetas(end+1) = theta;
    end
  end
end
```