**INTRO TO ENGINEERING DESIGN PRT**

Spring 2025

# Trebuchet Analysis

Nicholas Quillin

E-mail: niq1894@mavs.uta.edu

In Partial Fulfillment of Course Requirements

Department of Mechanical and Aerospace Engineering
University of of Texas at Arlington

Arlington, Tx

Cover image generated by ChatGPT

## Table of contents

Table of contents ..... 2

Table of Figures ..... 3

Preface ..... 4

Trebuchet Analysis

1. KINEMATICS OF A TREBUCHET ..... 4

1.1. Kinematics of a driven pendulum ..... 5

1.2. Kinematics of the rotating arm ..... 9

2. ENERGY-BASED TREBUCHET SIMULATOR ..... 9

3. COMPUTER-VISION TRAJECTORY TRACER ..... 14

4. ESTIMATING ENERGY EFFICIENCY ..... 15

Appendix

A. MATLAB Code for Trebuchet Trajectory Solver ..... 17

B. OpenCV trajectory tracer ..... 19

C. MATLAB adjustment calculator ..... 21

## Table of Figures

Figure 1 Chaotic motion of a double pendulum ..... 5
Figure 2 Idealized Pendulum ..... 6
Figure 3 Pendulum in gravitational field ..... 6
Figure 4 Pendulum with normal force ..... 7
Figure 5 Additional force on the pendulum (assume previous equilibrium holds) ..... 8
Figure 6 Pendulum in polar coordinates ..... 8
Figure 7 Main arm of a trebuchet ..... 9
Figure 8 Coordinates of a trebuchet ..... 10
Figure 9 Sample output from Trebuchet solver ..... 12
Figure 10 Time variant sample output from trebuchet solver ..... 13
Figure 11 Custom Mask Tuner Program ..... 14
Figure 12 Ball Detection result ..... 15

## Preface

While the main goal of this course is to learn the fundamental principles of design, the final project challenges the students to try to accomplish a goal given a set of criteria. While this necessitates the employment of various principles within the class such as drawing, modeling and dimentioning, for such a project to be brought from start to finish requires additional work that is outside of the scope of this class.

The goal of this project is to lob a projectile some specified distance by means of a trebuchet. In this instance, the trebuchet had to strictly be gravity powered, as well as limited to traditional materials.

When we asked about the strict design criteria, we were told that the main challenge of this project was the simulation of the trebuchet system. So, with that in mind when the others in my group decided to remain working in a similar capacity as we had done before the transition, I spent most of my time tackling this challenge head-on, while delegating much of the 3d modelling work to my teammates as I wrote about in a previous correspondence. While this meant that most of the work that I completed was not strictly within the realm of physical design itself, in this report I will try my best to intertwine the design and first principles together with the design of a trebuchet. It also required me to learn how to use Computer Algebra Systems, Variational Calculus, and Lagrangian Mechanics in order to complete it. In other words, I have no SolidWorks models, simulations, or even drawings, which make up the majority of the PRT rubric; but, due to the uncharacteristically difficult task of analysis that I will present here, and its criticality to accomplishing the mission, I hope that I will be awarded full credit for my efforts, nevertheless.

## 1. KINEMATICS OF A TREBUCHET

For the sake of analysis, a trebuchet is a driven double pendulum. It consists of a arm, a suspended counterweight that drives the arm through a constant downward force, and a sling. A double pendulum is a notoriously chaotic system as small differences in initial condition can lead to large changes.

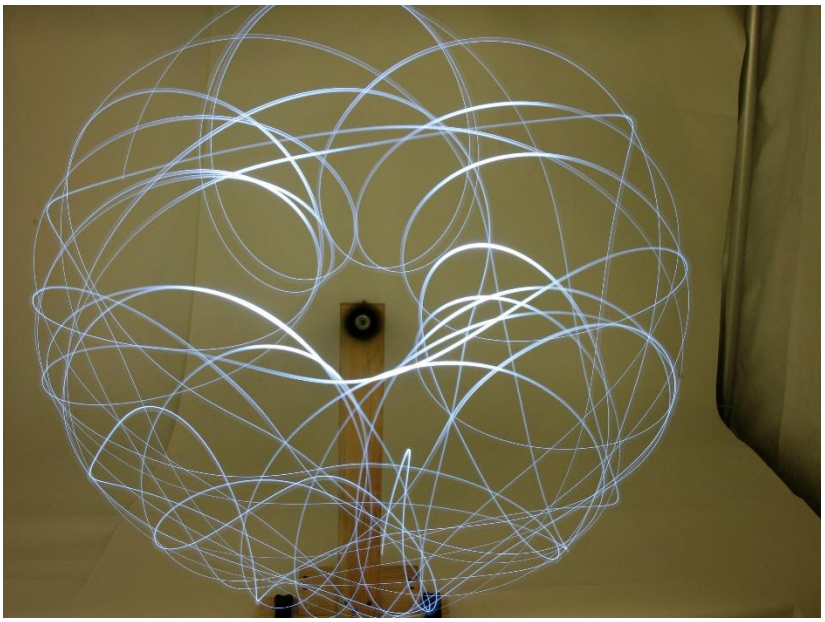

*Figure 1 Chaotic motion of a double pendulum (Make Magazine)*

Nevertheless, I was determined to take a procedural approach to design. The first thing I seeked to understand is how does such a mechanism moves; What makes trebuchets advantageous to the point that they became such prolific siege weapons in the middle ages. And the first thing that I thus tried to understand is why does the arm swing outward.

### 1.1. Kinematics of a driven pendulum

An important facet of the design of a system is to break down its functioning to its atomic pieces, while the mechanism itself is chaotic and complex the machinations of each individual element are not. So the first place I started with is characterizing the most important component of the system the sling.

The sling uses a whip-like motion to cause the projectile to rapidly accelerate. This allows the arm to reach out farther and hence transfer more energy to the projectile.

To get started, I defined a pendulum as a massless rod, with a certain length, with a mass on one end and forces acting on either end.

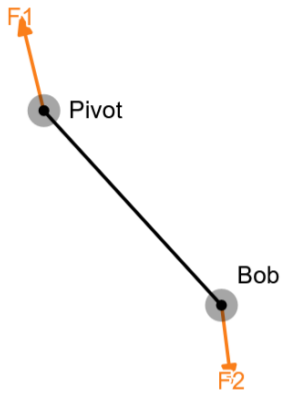


*Figure 2 Idealized Pendulum*

In a gravitational field the force on one end is always pointing downwards due to gravity

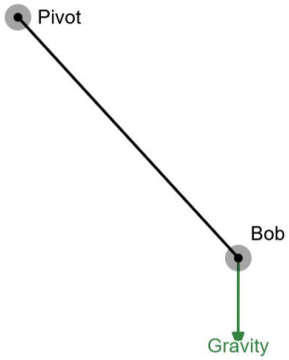


*Figure 3 Pendulum in gravitational field*

For a simple free pendulum if the pivot is said to be fixed, there must be an equal and opposite force in the y-direction which must resist the gravitational force.

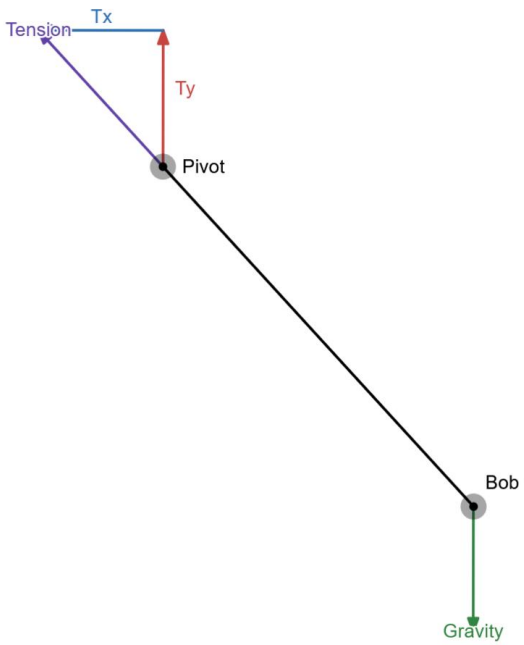


*Figure 4 Pendulum with normal force*

Where the resulting x-component of force causes the periodic oscillations typically described by the device.

To move the pivot would mean that an additional force must be added to the pivot beyond the normal force. The component of this force that acts in the direction of the rod leads to the tension force increasing. The component of this force that acts orthogonally to the rod leads to a rotational moment of the mass. The resulting motion of the pendulum can be described as a linear combination of these two forces.

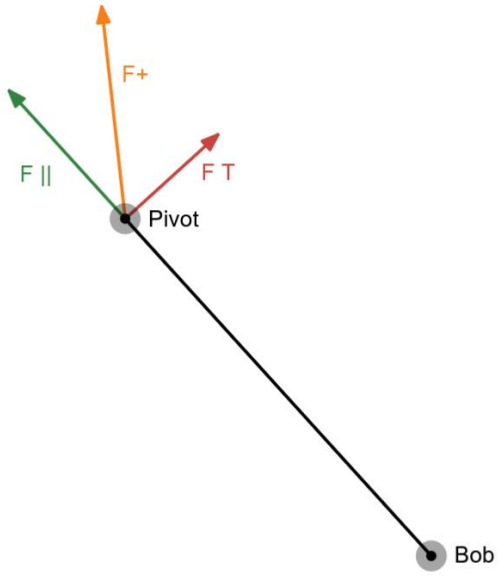


*Figure 5 Additional force on the pendulum (assume previous equilibrium holds)*

As such the orthogonal component leads to the extension of the arm, leading to the swinging motion of the sling. This leads to the projectile radius increasing resulting in higher string tensions. As the net force is a linear combination of the two it follows that the extension of the sling is driven by both the overall distance from the trebuchet pivot and the motion of the arm then the arm alone.

Since the length of the rod is constant, the orientation of the system can be reduced to a single angler parameter. So while the components discussed here are relative to x and y, in actual analysis everything is done in polar coordinates.

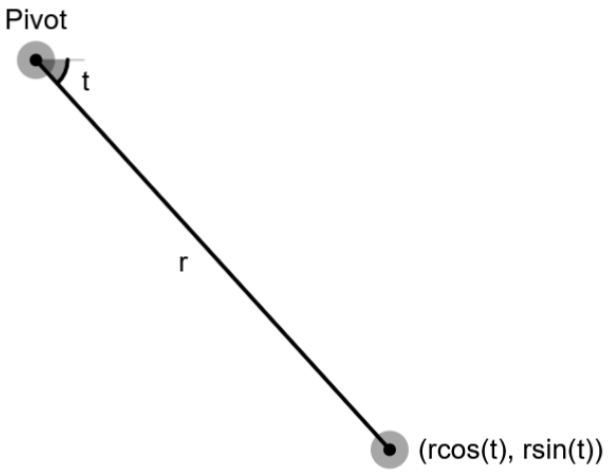


*Figure 6 Pendulum in polar coordinates*

### 1.2. Kinematics of the rotating arm

The main arm of a trebuchet has 4 forces working on it, the reaction force at the sling pivot, the force of gravity at its centroid, the normal force at the axel and the gravitational force of the counterweight.

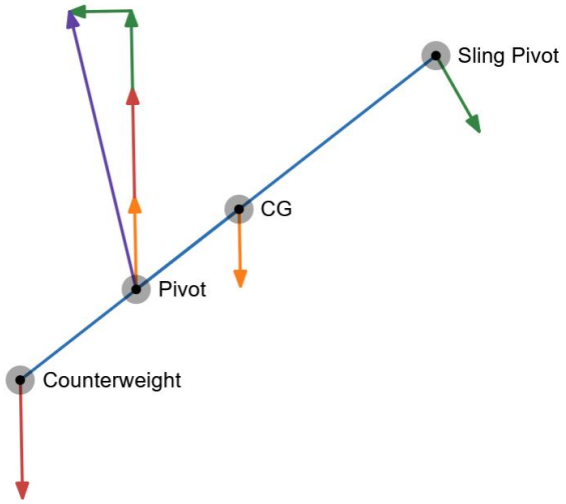


*Figure 7 Main arm of a trebuchet*

Starting from the last and moving to the first, the force of the counterweight always points downward with a magnitude of$m_c g$. The normal force of the axel is equal and opposite to the sum of the other forces to maintain its position. The force at the centroid like the counterweight, always acts downward and finally the force at the sling pivot is really unknown and the objective of a solution.

At this point it is impossible to ascertain how much force actually goes into the sling and this is about the limit of what we can do from analyzing just the forces. Notice that this system though has been reduced to single unknown vector which means that we require an additional equation to solve.

## 2. ENERGY-BASED TREBUCHET SIMULATOR

To understand the evolution of this uknown we turn to energy conservation. This is how we solve this complicated and chaotic system. Compared to forces, energy states are easy to directly calculate from the position of the elements. For now we will only consider rotational kinetic energy and potential energy.

$$KE_R = \frac{1}{2} I\omega^2$$

$$PE = mgh$$

While I could have added frictional losses to the model, I was uncertain of myself and just wanted to get the model to work on the simplest premise possible. Nevertheless, when I looked into the subject, friction turns out to be relatively negligible considering the axles are relatively small and that the projectile spends a short amount of time at high speed.

As such there are 4 sources of energy on the trebuchet, the Potential energy of the counterweight, the potential energy of the projectile, the rotational kinetic energy of the arm and the rotational kinetic energy of the sling.

$$KE_{R_{arm}} = \frac{1}{2}\sum I_{arm}\omega_a^2$$

$$KE_{R_{proj}} = \frac{1}{2}m_p(r_a^2\omega_a^2 + r_s\omega_s^2 + 2r_a r_s \omega_a \omega_s \cos(\theta_a - \theta_s))$$

$$PE_c = m_c g r_c \sin\theta_a$$

$$PE_{proj} = m_p g(r_a \sin\theta_a + r_s \sin\theta_s)$$

I have included this diagram to better illustrate the coordinates being used.

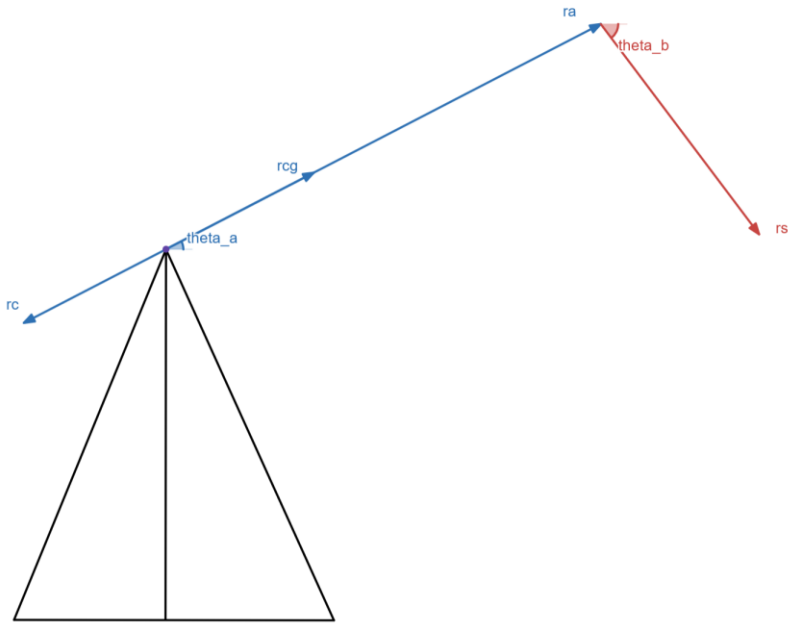


*Figure 8 Coordinates of a trebuchet*

Once I have all of the energies, I can use the principle of least action to determine the actual trajectory of the trebuchet. This begins by calculating the Lagrangian.

$$\mathcal{L} = \sum KE - \sum PE$$

Where substituting the previous equations yields:

$$\mathcal{L} = \frac{1}{2}(I_{arm} + m_c r_c^2)\omega_a^2 + \frac{1}{2}m_p(r_a^2\omega_a^2 + r_s\omega_s^2 + 2r_a r_s \omega_a \omega_s \cos(\theta_a - \theta_s)) - m_c g r_c \sin\theta_a - m_p g(r_a \sin\theta_a + r_s \sin\theta_s)$$

As it can be seen this begins to get very verbose very quickly. Henceforth, I will omit the intermediary steps as they are very long. Not to mention I did not do this by hand as the focus of this project is on design and not on forcing me to suffer through tedious computations. Instead I took advantage of CAS systems to automate the process. Nevertheless I first use the Euler-Lagrange equation to find the equations of motion minimize the action for each refrence.

$$\frac{\partial \mathcal{L}}{\partial \theta_a} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \omega_a}\right) = 0$$

$$\frac{\partial \mathcal{L}}{\partial \theta_b} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \omega_b}\right) = 0$$

Then to solve the trajectory from this ODE, I rewrite the evolution of the system in terms of a nondimensional “Mass” matrix and a “Force” matrix.

$$M\left(\begin{bmatrix}\theta_a \\ \theta_b\end{bmatrix}\right)\left(\frac{d^2}{dt^2}\begin{bmatrix}\theta_a \\ \theta_b\end{bmatrix}\right) = F\left(\begin{bmatrix}\theta_a \\ \theta_b\end{bmatrix}, \frac{d}{dt}\begin{bmatrix}\theta_a \\ \theta_b\end{bmatrix}\right)$$

Which I can then plug right into ode45 which solves this system to compute the states of the trebuchet and stop when I reach maximum velocity.

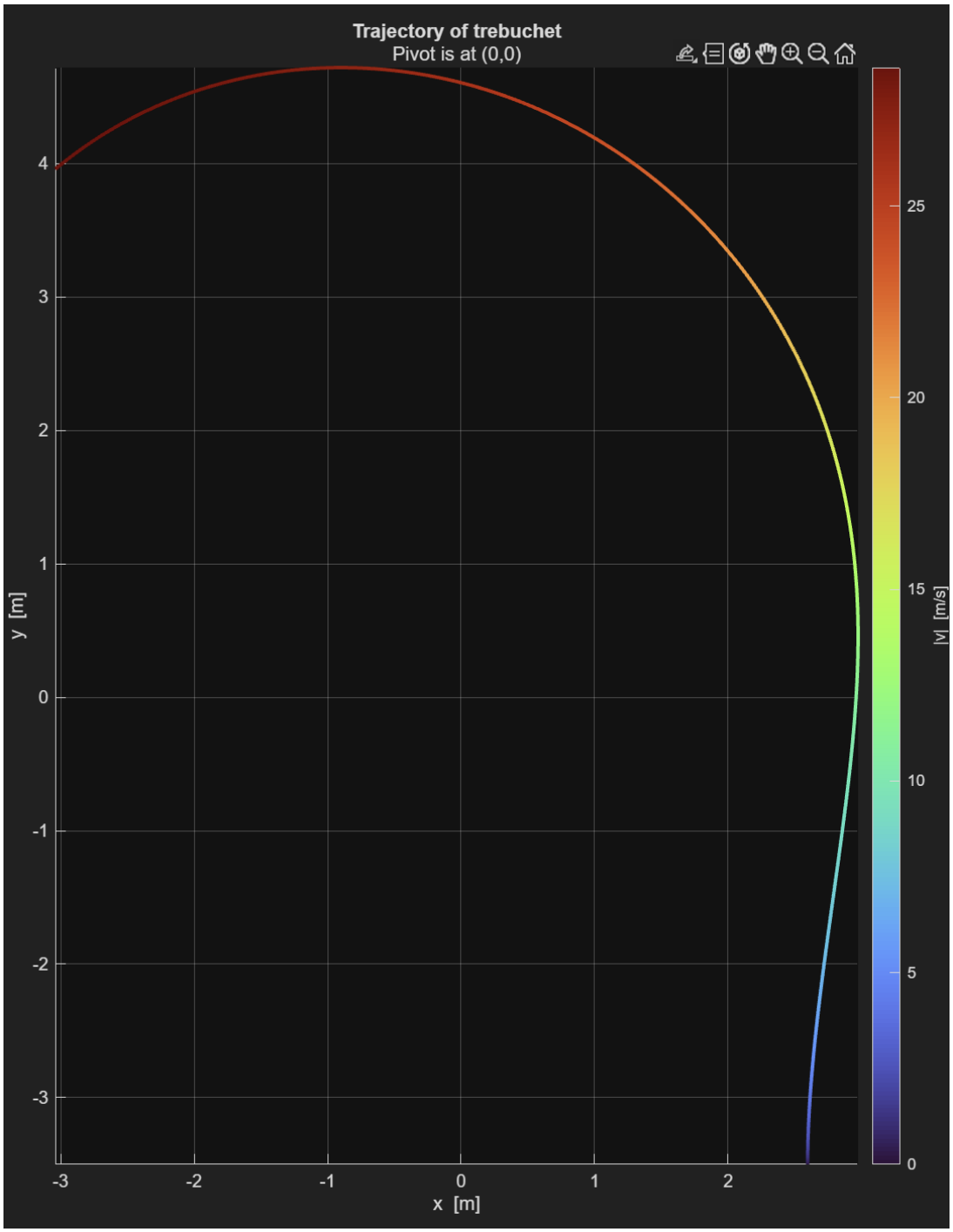


*Figure 9 Sample output from Trebuchet solver*

This model can also produce the time-dependent variables as well.

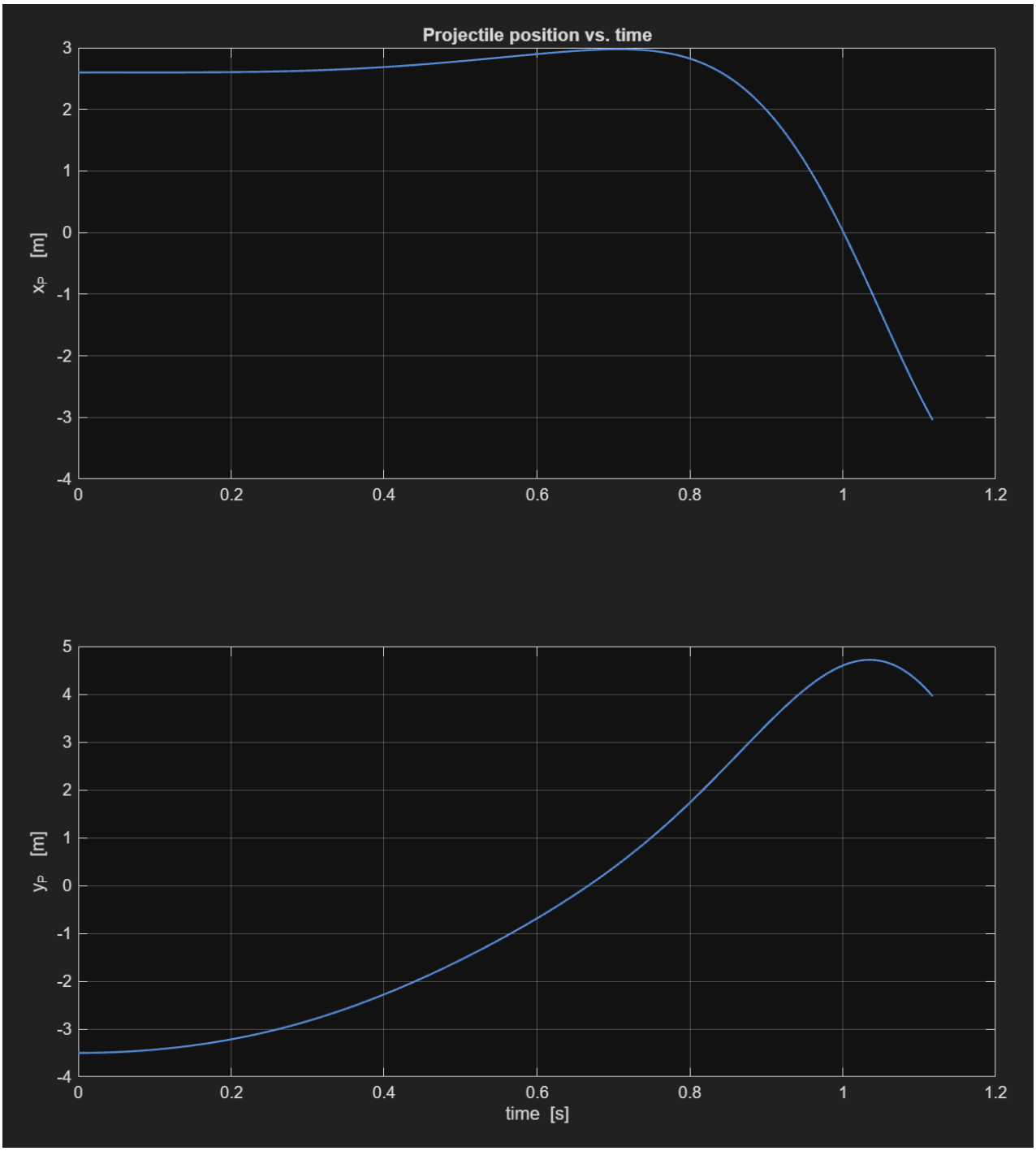


*Figure 10 Time variant sample output from trebuchet solver*

As such this program allows efficient search through the design space to find the optimal parameters for a trebuchet.

## 3. COMPUTER-VISION TRAJECTORY TRACER

In order to confirm my models with empirical tests I sought out a way to be able to use computer vision in order to track the ball. The first step is to figure out the relative position of the ball on the screen. This is done by using a mask to first isolate everything that has the same color as the ball. The mask is created manually for each video using a program that I made.

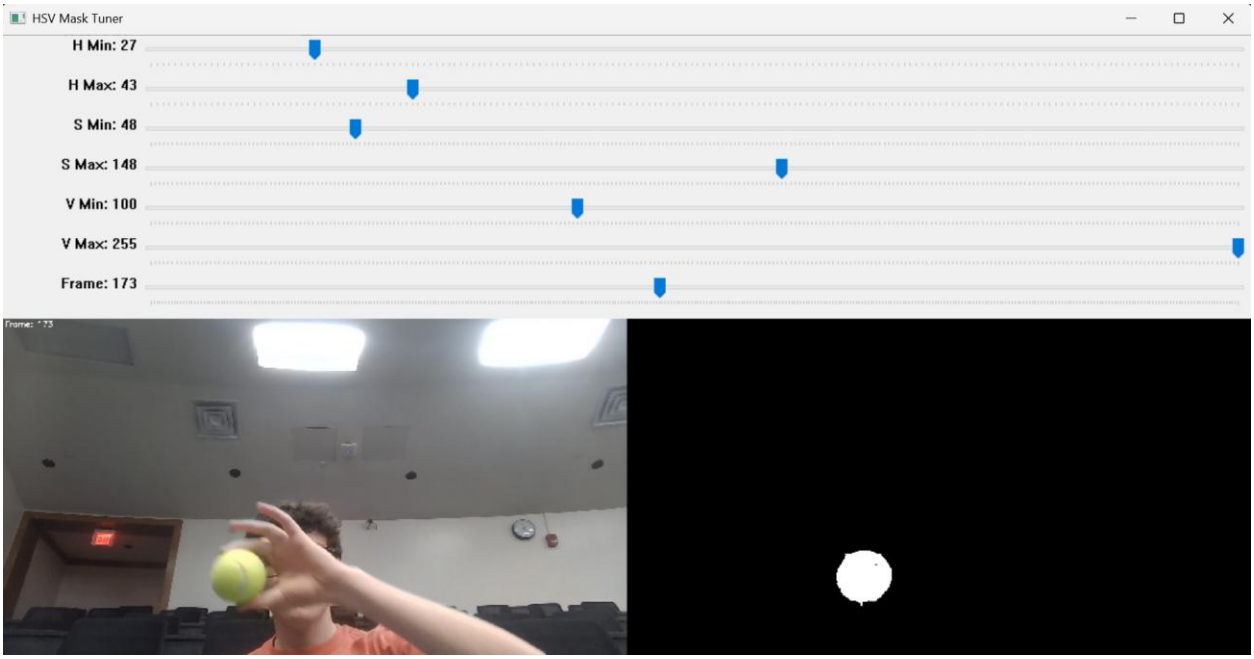


*Figure 11 Custom Mask Tuner Program*

Once the mask is created, I run the resulting video through a contour detection to detect the contours of the ball.


*Figure 12 Ball Detection result*

The center coordinates of the ball are then recorded into a CSV file to be processed in MATLAB. What I then do is normalize all the coordinates relative to the center of the screen that theoretically does not have any distortion by subtracting half of the resolution from the ball position.

$$(x', y') = (x - \frac{1}{2}res_x, y - \frac{1}{2}res_y)$$

Finally, to get the position of the ball I use the distance to the plane of the ball and the field of view to transform these coordinates into projected coordinates.

$$(x', y') = \left(\frac{xDtan(FOV)}{res_x}, \frac{yDtan(FOV)}{res_y}\right)$$

## 4. ESTIMATING ENERGY EFFICIENCY

According to the rules of the competition, the only real adjustment that can be made to the trebuchet is the counterweight mass. To facilitate this I first solved the ballistic formula in order to find the necessary launch velocity.

$$v_{req} = \sqrt{\frac{gr^2}{2\cos^2\theta\,(y_o + r\tan\theta)}}$$

Where $\theta$ now represents the launch angle and $y_o$ represents launch height. Then I remove the potential energy added to the projectile to get the theoretical kinetic energy of the projectile.

$$KE_{proj} = PE_{c_0} - PE_{proj_f} - KE_{arm_f}$$

Since $KE_{arm}$ changes as $PE_{c_o}$, I use a efficiency factor n of the previous simulation to be able to guesstimate a rough adjustment amount which can then be restimulated and iterated upon. Finally, when a solution is converged the mass of the counterweight is evaluated.

$$m_c = \frac{KE_{proj}}{ngh}$$

I admit this model is rudimentary compared to the previous analysis as it was put together in a few hours in preparation for field tests, but it was requested that I present this as well. Due to a lack of scale, the precision was not the best, but it was good enough to get a landing position within $\pm$1ft.

## A. MATLAB Code for Trebuchet Trajectory Solver

```
function trebuchet
%-------------------------------------------------------------
%  2-DoF trebuchet model – stops at max projectile speed
%  Trajectory is color-coded by |v| (blue = slow, red = fast)
%-------------------------------------------------------------

%% physical parameters
p.g     =  9.81;           % gravity  [m/s^2]
p.m_c   = 50.0;            % counter-weight mass [kg]
p.m_p   =  1.0;            % projectile mass      [kg]
p.I_arm = 20.0;            % arm inertia about pivot [kg·m^2]
p.r_c   =  1.0;            % pivot → counter-weight [m]
p.r_a   =  3.0;            % pivot → arm tip        [m]
p.r_s   =  2.0;            % sling length           [m]

%% initial conditions
th_a0  = -30*pi/180;       % arm 30° below horizontal
th_s0  =  -pi/2;           % sling is hanging straight down
dth_a0 =  0.0;             % released from rest
dth_s0 =  0.0;

y0    = [th_a0; th_s0; dth_a0; dth_s0];
tspan = [0 5];             % calculation interval

%% integrate (event stops at max speed)
opts = odeset('RelTol',1e-9,'AbsTol',1e-9, ...
              'Events',@(t,y) maxSpeedEvent(t,y,p));
[t,y] = ode45(@(t,y) eom(t,y,p), tspan, y0, opts);

th_a  = y(:,1);                    % arm angle
th_s  = y(:,2);                    % sling angle
dth_a = y(:,3);                    % arm angular velocity
dth_s = y(:,4);                    % sling angular velocity

%% projectile position and speed
xP = p.r_a.*cos(th_a) + p.r_s.*cos(th_s);
yP = p.r_a.*sin(th_a) + p.r_s.*sin(th_s);

vx = -p.r_a.*dth_a.*sin(th_a) - p.r_s.*dth_s.*sin(th_s);
vy =  p.r_a.*dth_a.*cos(th_a) + p.r_s.*dth_s.*cos(th_s);
speed = sqrt(vx.^2 + vy.^2);

%% Figure 1 – color-coded trajectory
figure(1); clf
% use a surface object to get a continuous color gradient
surface([xP';xP'],[yP';yP'],zeros(2,length(xP)), ...
        [speed';speed'], ...
        'EdgeColor','interp','LineWidth',2);
view(2); axis equal tight; grid on
colormap(turbo);
cb = colorbar; cb.Label.String = '|v|  [m/s]';
xlabel('x  [m]'); ylabel('y  [m]')
title('Trajectory of trebuchet', 'Pivot is at (0,0)')

%% Figure 2 – x(t) and y(t)
figure(2); clf
subplot(2,1,1)
plot(t,xP,'LineWidth',1.2), grid on
title('Projectile position vs. time')
ylabel('x_P  [m]')
subplot(2,1,2)
```

```
plot(t,yP,'LineWidth',1.2), grid on
xlabel('time  [s]'); ylabel('y_P  [m]')
end
%=========================================================================
function dydt = eom(~, y, p)
%  y = [theta_a; theta_s; dtheta_a; dtheta_s]
th_a  = y(1); th_s  = y(2);
dth_a = y(3); dth_s = y(4);
Delta = th_a - th_s; cD = cos(Delta); sD = sin(Delta);

% mass matrix
M11 = p.I_arm + p.m_c*p.r_c^2 + p.m_p*p.r_a^2;
M12 = p.m_p*p.r_a*p.r_s*cD;
M22 = p.m_p*p.r_s^2;
M   = [M11 M12; M12 M22];

% forces
F1 =  p.m_c*p.g*p.r_c*cos(th_a) ...
    - p.m_p*p.g*p.r_a*cos(th_a) ...
    - p.m_p*p.r_a*p.r_s*sD*dth_s^2;
F2 =  p.m_p*p.r_s*( p.r_a*sD*dth_a^2 - p.g*cos(th_s) );

ddq = M \ [F1; F2];
dydt = [dth_a; dth_s; ddq];
end
%=========================================================================
function [value,isterm,dir] = maxSpeedEvent(~, y, p)
% Stop when projectile speed reaches its maximum

th_a  = y(1); th_s  = y(2);
dth_a = y(3); dth_s = y(4);
Delta = th_a - th_s; cD = cos(Delta); sD = sin(Delta);

% accelerations (same calc as in eom)
M11 = p.I_arm + p.m_c*p.r_c^2 + p.m_p*p.r_a^2;
M12 = p.m_p*p.r_a*p.r_s*cD;
M22 = p.m_p*p.r_s^2;
M   = [M11 M12; M12 M22];

F1 =  p.m_c*p.g*p.r_c*cos(th_a) ...
    - p.m_p*p.g*p.r_a*cos(th_a) ...
    - p.m_p*p.r_a*p.r_s*sD*dth_s^2;
F2 =  p.m_p*p.r_s*( p.r_a*sD*dth_a^2 - p.g*cos(th_s) );

ddq = M \ [F1; F2];
ddth_a = ddq(1); ddth_s = ddq(2);

% velocity and acceleration of projectile
vx = -p.r_a*dth_a*sin(th_a) - p.r_s*dth_s*sin(th_s);
vy =  p.r_a*dth_a*cos(th_a) + p.r_s*dth_s*cos(th_s);
ax = -p.r_a*(ddth_a*sin(th_a) + dth_a^2*cos(th_a)) ...
     -p.r_s*(ddth_s*sin(th_s) + dth_s^2*cos(th_s));
ay =  p.r_a*(ddth_a*cos(th_a) - dth_a^2*sin(th_a)) ...
     +p.r_s*(ddth_s*cos(th_s) - dth_s^2*sin(th_s));

dSpeed = vx*ax + vy*ay;   % (1/2) d(|v|^2)/dt

value  = dSpeed;   % zero at speed peak
isterm = 1;        % stop integration
dir    = -1;       % detect +→− crossing only
end

```

## B. OpenCV trajectory tracer

```python
import cv2
import numpy as np
import csv
from tqdm import tqdm

# HSV color range for tennis ball (initial guess – tune via slider script if needed)
LOWER_COLOR = np.array([27, 48, 100])
UPPER_COLOR = np.array([43, 148, 255])

# Store first successful detection radius for auto-calibration
calibrated_radius = None

def locate_ball(frame):
    """Detect ball using HSV mask and return (x, y, radius) of the best contour."""
    global calibrated_radius

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, LOWER_COLOR, UPPER_COLOR)
    blurred = cv2.GaussianBlur(mask, (5, 5), 0)

    contours, _ = cv2.findContours(blurred, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        return None

    largest = max(contours, key=cv2.contourArea)
    (x, y), radius = cv2.minEnclosingCircle(largest)

    if radius < 3:
        return None

    # Auto-calibrate the first time
    if calibrated_radius is None:
        calibrated_radius = radius

    # Allow variation of ±40%
    if not (0.6 * calibrated_radius < radius < 1.4 * calibrated_radius):
        return None

    return int(x), int(y), int(radius)

def log_coordinate(csv_writer, frame_index, x, y, time_sec):
    csv_writer.writerow([frame_index, x, y, time_sec])

def draw_overlay(frame, x, y, radius):
    overlay_frame = frame.copy()
    cv2.circle(overlay_frame, (x, y), radius, (0, 0, 255), 2)
    cv2.putText(overlay_frame, f'({x},{y})', (x + 15, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.putText(overlay_frame, f'Radius: {radius}', (x + 15, y + 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 1)
    return overlay_frame

def generate_mask_frame(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, LOWER_COLOR, UPPER_COLOR)
    return cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

def export_video(frames, output_path, frame_rate, frame_size):
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, frame_rate, frame_size)
    for frame in frames:
        out.write(frame)
```

```
    out.release()

if __name__ == "__main__":
    VIDEO_PATH = "WIN_20250422_11_50_11_Pro.mp4"
    OUTPUT_CSV = "trajectory_output.csv"
    OUTPUT_OVERLAY = "trajectory_overlay.mp4"
    OUTPUT_MASK = "mask_video.mp4"

    cap = cv2.VideoCapture(VIDEO_PATH)
    if not cap.isOpened():
        print("Error: Cannot open video.")
        exit()

    frame_rate = cap.get(cv2.CAP_PROP_FPS)
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    overlay_frames = []
    mask_frames = []
    frame_index = 0

    with open(OUTPUT_CSV, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["frame", "x", "y", "time_sec"])

        with tqdm(total=total_frames, desc="Processing video") as pbar:
            while True:
                ret, frame = cap.read()
                if not ret:
                    break

                original_frame = frame.copy()
                result = locate_ball(frame)
                if result:
                    x, y, radius = result
                    time_sec = frame_index / frame_rate
                    log_coordinate(writer, frame_index, x, y, time_sec)
                    frame = draw_overlay(frame, x, y, radius)

                mask_frame = generate_mask_frame(original_frame)
                overlay_frames.append(frame)
                mask_frames.append(mask_frame)
                frame_index += 1
                pbar.update(1)

    cap.release()
    export_video(overlay_frames, OUTPUT_OVERLAY, frame_rate, (frame_width, frame_height))
    export_video(mask_frames, OUTPUT_MASK, frame_rate, (frame_width, frame_height))

    print(f"Trajectory data saved to {OUTPUT_CSV}")
    print(f"Overlay video saved to {OUTPUT_OVERLAY}")
    print(f"Mask video saved to {OUTPUT_MASK}")

```

## C. MATLAB adjustment calculator

```
function adjustment
%-------------------------------------------------------------
%  Rudementary adjustment model and range calculations
%-------------------------------------------------------------

g = 9.81;               % m/s^2
th = 45;                % degrees
y0 = 27 / 12 * 3.28;    % in -> m release height
R = 15 * 3.28;          % ft -> m Desired range
h= 15 / 12 * 3.28;      %in -> m the drop height of the weight
mp = 56e-3;             %kg mass of the projectile

%% Calculate The velocity necessary to achieve the required range
v0 = sqrt( g * R^2 / (2 * (cosd(th))^2 * (y0 + R * tand(th))) );
KE = 1/2*mp*v0^2
PEp = mp * g * h;
E = KE-PEp;

%% Use the energy required to calculate a counterweight mass based off numerical simulation
n=.3;
mc = E / n / g / h
end

```