Calculating the Coefficient of Friction of My Pool Table

Table of Contents

Problem Statement

This is part of a larger project in an attempt to prove something I postulated while practicing pool back in 2019. This will be further described in greater detail later, but for now, the goal is to calculate the effective coefficient of friction between my pool table and the balls when rolling at low speeds.

Assumptions

1. Since the balls are rolling slowly without any external forces acting on the system*, they are always considered to be rolling without slipping.
2. The balls always travel in a straight line.
3. Static/kinetic/rolling coefficient of friction is the same. This is what 'effective' is trying to capture.

*Note: The 'system' consists of the balls and the table.

Set Up

1. Placed tape measure along the length of the table on the edge.
2. Created a lego ramp with adjustable height to have consistent ball speeds.
3. Used the Slow-Mo feature on an iPhone 13 set to 195 frames per second.
4. Placed the phone on the table with the camera lined up with the end of the ramp, and start of the tape measure.

Procedure

1. Start recording.
2. Place Ball's center of mass as far back on the ramp as possible.
3. Relase ball and measure how far along the table it stopped using the measuring tape.
4. Record the stopping distance on the table.
5. Repeat process twice more for the given height of ramp.
6. After three trials, add one brick to the height of the ramp and repeat until the ramp is 5 bricks high.
7. Stop recording and upload the video to OneDrive.
8. Use FFmpeg to add the current video frame to the file stream.
9. Play back the video frame-by-frame to calculate the time the ball takes to travel the ramp.
10. Do least-squares regression to numerically calculate the effective coefficient of friction.

FFmpeg Command

- `ffmpeg -i data.MOV -vf "drawtext=fontfile=Arial.ttf: text=%{n}: x=(w-tw)/2: y=h-(2*lh): fontcolor=white: fontsize=69: box=1: boxcolor=0x00000099" -y output.MOV`

Calculating Velocities

```
clear all; clc;
g = 9.80665; %m/s^2
ramp = 0.0954; %m

stoppingDistances = [1 30.875 32.0 32.5;
                     2 41.5 41.5 41.75;
                     3 48.5 50.5 51;
                     4 60.25 60.25 60.125;
                     5 76.5 77.5 80.5]; %in

distanceData = zeros(1,15);
distanceData(1:3) = stoppingDistances(1,2:4);
distanceData(4:6) = stoppingDistances(2,2:4);
distanceData(7:9) = stoppingDistances(3,2:4);
distanceData(10:12) = stoppingDistances(4,2:4);
distanceData(13:15) = stoppingDistances(5,2:4);
distanceData = distanceData./39.3701 %m
```

```
distanceData = 1×15
    0.7842    0.8128    0.8255    1.0541    1.0541    1.0604    1.2319    1.2827 ···
```

```
times = [165/195 157/195 148/195 113/195 109/195 ...
         114/195 96/195 95/195 98/195 86/195 ...
         80/195 82/195 72/195 66/195 72/195]; %s
trialVelocities = ramp./times %m/s
```

```
trialVelocities = 1×15
    0.1127    0.1185    0.1257    0.1646    0.1707    0.1632    0.1938    0.1958 ···
```

Model

In order to calculate the effective coefficient of friction, we will use a modified stopping distance model similar to one derived in a 2.086 Problem Set.

This model, shown below, was originally made to describe the stopping distance of a vehicle, taking into account the reaction time of the driver, not present in our model, and a constant, which is present in our model.

$$SD = \frac{\nu^2}{2\mu_{eff}g} + b$$

```
designMatriX = [ones(15,1) trialVelocities'.^2]
```

```
designMatriX = 15×2
    1.0000    0.0127
```

```
    1.0000    0.0140
    1.0000    0.0158
    1.0000    0.0271
    1.0000    0.0291
    1.0000    0.0266
    1.0000    0.0376
    1.0000    0.0383
    1.0000    0.0360
    1.0000    0.0468
     ⋮
```

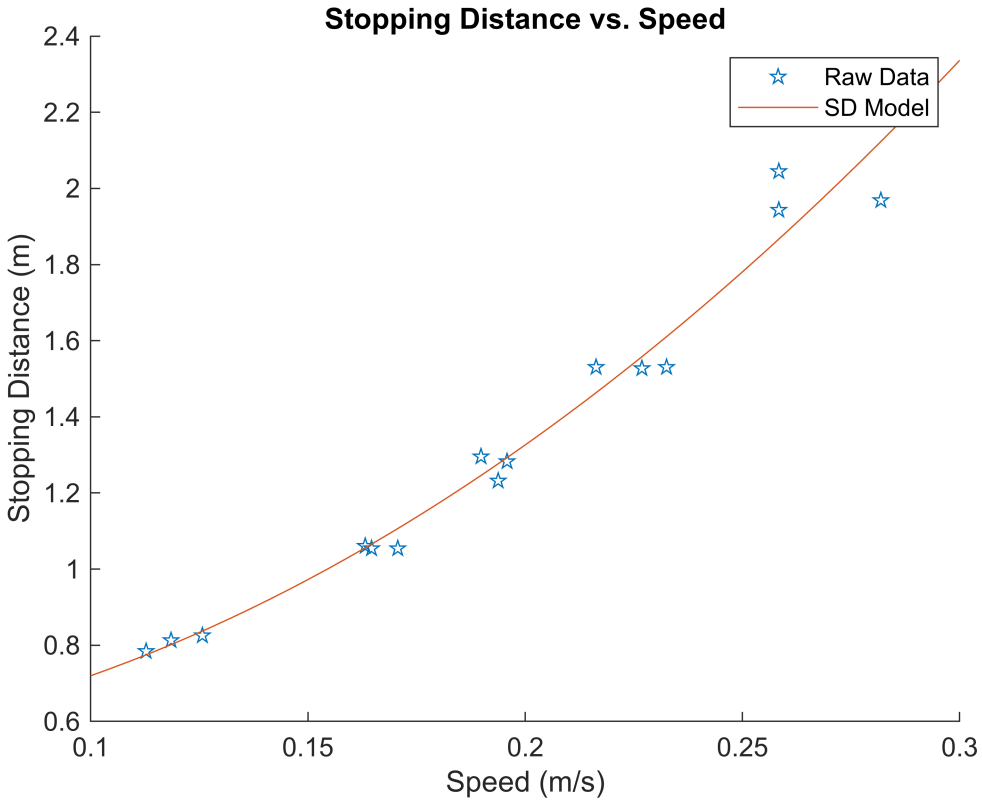
```
beta = designMatriX\distanceData';

mu_eff = 1/(2*g*beta(2)) %solving V^2 coefficient for mu_eff
```

```
mu_eff = 0.0025
```

```
SD_model = @(v) v.^2/(2*mu_eff*g) + beta(1);

scatter(trialVelocities,distanceData,'p')
xlabel('Speed (m/s)')
ylabel('Stopping Distance (m)')
title('Stopping Distance vs. Speed')
hold on
fplot(SD_model,[0.1,0.3])
legend('Raw Data','SD Model')
hold off
```



Justification for Neglecting Air Resistance

Heuristically, we know that not only are we dealing with a projectile with a small cross sectional area, but we are measuring it at very low velocities, so the impact of drag would not be that great.

Conclusively, when we run the calculation, the force of drag formula supports our results.

$$f_d = \frac{1}{2}\rho v^2 C_d A$$

```
ramp = 0.0954; %m
rho_air = 1.293; %kg/m^3
C_d_sphere = 0.5;
R_ball = 1/2*2.25*2.54/100; %m
A_ball = pi*R_ball^2; %m^2
%fastest speed measured during testing
v = 0.2819; %m/s

f_d = 1/2*C_d_sphere*A_ball*rho_air*v^2 %N
```

```
f_d = 6.5895e-05
```

It is very tricky, algebraically speaking, to include this force into our model, as the acceleration of the system would not allow us to factor out the v^2 coefficient. Running through the dynamics, the acceleration would look something like this:

$$\frac{2mv^2}{AC_d\rho v^2 + 2\mu mg}$$

Thus, we neglect the term, and allow our Least Squares fit to absorb the error.

Conclusion

Assuming the methodology is sound, the coefficient of friction between the felt of my pool table and the pool balls is 0.0025. Further testing could be conducted to verify this coefficient at higher speeds, but this would require the balls to bounce off of the side rails, losing energy in the collision, and potentially skewing the results of such an experiment. For now, I am content with these results, and will use them to continue with the overarching project.