LABS AND DATA ANALYSIS 2

UNIVERSITY OF BIRMINGHAM

DEPARTMENT OF CHEMICAL ENGINEERING

Particle Tracking and Modelling
MATLAB and Python

Author:
Abbas Moosajee

ID No:
2163680

Course Lead:
Dr Kit Windows-Yule

Instructor:
Leonard Nicusan

May 21, 2022

Contents

List of Figures

List of Tables

Table 1: Nomenclature Table

Symbol	**Names**	**Units**
COR	Coefficient of Restitution	-
ρ	Density	kg m^{-3}
Cd	Drag Coefficient	-
$\mathbf{F}_{\text{rho}}$	Fluid Density	kg m^{-3}
$\mathbf{fps}$	frames per second	Hz
$\mathbf{g}$	Gravitational Acceleration	m s^{-2}
GPE	Gravitational Potential Energy	kg m^2 s^{-2}
$\mathbf{h}$	Height	m
KE	Kinetic Energy	kg m^2 s^{-2}
$\mathbf{t}_{\max}$	Maximum Time	s
$\mathbf{P_a}$	Particle Area	m^2
$\mathbf{P_m}$	Particle Mass	kg
t	Time	s
dt	Time step	s
TE	Total Energy	kg m^2 s^{-2}
$\mathbf{v}$	Velocity	m s^{-1}
$\mathbf{V}$	Volume	m^3
DEM	Discrete Element Method	-

1 Overview (P3)

As some form of powder or granular mixture is involved in most manufacturing processes, being able to accurately predict their behaviour is extremely beneficial(Blais et al. 2019). However, unlike fluid flow, particle behaviour cannot be condensed down to a single set of equations as the particles are interacting and exerting various forces such as contact or van der Waals. The development of the Discrete Element Method(DEM), as part of the Lagrangian numerical methods, created a set of time driven mathematical equations that can model the behaviour of two distinct systems interacting.

Created by Cundall and Strack (1979), the soft sphere DEM uses the dynamics of the collision to calculate impact force and predict behaviour post collision. By treating each particle as separate entities with distinct properties, complex systems involving interactions between thousands of particles can be created, allowing engineers to examine the material behaviour in different conditions and equipment, saving time and money as fewer experiments are conducted. Furthermore, one of its greatest benefits lies in the fact that DEMs can not only provide both macroscopic and microscopic properties of flow, but it can handle particles of all geometries(Lu, Third, and Müller 2015).

As heat and mass transfer, chemical reactions and mixing of multiple fluids are being integrated into DEM(Blais et al. 2019), it provides engineers the opportunity to computationally experiment with a wider range of conditions and understand the limits of how their processes. This is especially useful when scaling up reactions from the lab phase to final production, cos doing this incorrectly is a process hazard, with serious consequences as seen in multiple previous incidents. For instance, at the T2 Laboratories in Jacksonville, Florida during the manufacturing of MCMT, a gasoline additive, was wrongly scaled up causing energy to increase unexpectedly creating a runaway reaction resulting in the deaths of 3 workers(Theis 2014).

This potential of the DEM can only be maximised, if the model is correctly calibrated which requires accurate values of any effective parameters. However, as discussed by Marigo's review for a DEM of cylindrical pellets(Marigo and Stitt 2015), correctly calibrating the model is the hardest part which is made more complicated as materials used have differing properties, especially on a global scale. While there is no standardized guideline to calibration a large cohort of researchers ten to fall in one of two categories. Where particle properties are painstakingly measured experimentally. On the other hand, the bulk flow materials can be studied, and then calibrate the DEM model to produce the same results(Coetzee 2017). While a successful method, the scientific complexity, computational power and time can sometimes make the hard sphere model a better starting point, at least for basic models. Stevens and Hrenya (2005), tested the validity of a soft-sphere model using COR as the only input, and replicated results from past experiments.

The impact of particles can be modelled by treating them as a hard sphere, where the change in energy upon impact is quantified as the Coefficient of Restitution(COR). By taking the impact to be instantaneous and binary, the COR can be calculated using different properties of an object. Commonly divided into the kinematic COR, which uses the velocity or kinetic energy of the ball. On the flip side, energetic COR is the most representative as it accounts for all forms of energy dissipation. Ismail and Stronge (2008) analysis of energy dissipation and COR in visco-plastic bodies shows that provided the impact is friction less and the particles do not slide all three CORs are practically the same. While the study only considers damping coefficients less than 1, the lack of any statistical difference for the initial bounces suggest this is a fair assumption to make at least for starting of the model.

As described in Asteriou, Saroglou, and Tsiambaos (2012), the kinematic COR is based on Newton's Theory of Particle Collision which can be represented using the ratio of velocities as in Eq.(1). Furthermore, for the initial free-fall motion of a particle this can be simplified to an equation using height.

$$COR_V = v_a/v_b \tag{1}$$

$$COR_H = \sqrt[2]{h_a/h_b} \tag{2}$$

While theoretically, the CORs calculated from Eq.(1) and Eq.(2), should give us similar values as they are essentially the same. One must recognise that most scientific studies measuring the COR are using professional high speed cameras, capable of capturing 40,000 frames per second(fps) also cost over $40,000 as discussed by Hastie (2013). This extreme cost is a barrier to entry for most home scientists that often need a rough estimate.

Alternative approaches, like the one pioneered by Bernstein (1977) used the audio to determine time between impact, and then calculating CORs. While quick and easy they cannot provide a lot of information about the projectile's motion. Thus the accuracy of values obtained from videos captured using smartphone cameras will provide more information on the reliability of this method.This is a quick and easy method to calculate the COR as it simply requires one to identify the max height before and after impact. However, as this is impacted by the drag of air on the ball, it should only be used as a rough estimate.

Creating and calibrating DEMs used at industrial scales often requires controlled scientific experiments and extreme computational power, making it inaccessible to a majority of the global scientific community. This, becomes especially problematic when having to teach engineering students about DEMs but unable to provide them access to one, let alone try to create their own.

However, as the motion of a simple bouncing ball shares many basic characteristics with multi-particle interactions,so trying to model this behaviour can provide an idea of the common pitfalls that may be faced when scaling-up to more complex simulations. Thus, the motion for a single bouncing ball can be modelled by implementing a basic numerical solvers like Euler's to calculate the motion of the particle with some iteration. Furthermore, as the ball is travelling through air one must account for the changing acceleration due to resistance forces like drag and buoyancy.

While it was shown to be relatively small for a standard tennis ball, if a larger sized particle were used, the increase in surface area can make air resistance a significant factor especially if thrown from large heights. By performing some basic home experiments, one can use video processing functions such as Hough Cirlces present in the OpenCV library Itseez (2015) to track the motion of a falling ball filmed using a basic smartphone camera. By running a series of repeats by with various parameters, on can get COR values of any spherical object, allowing the projectile model to be calibrated.

The results of a model created in MATLAB and Python using these guidelines can be seen below.

2 Modelling Projectile Motion (P1)

Numerical methods allow one to solve problems involving integration or differential systems relatively easily, compared to doing it analytically. Especially useful for non-linear equations where only an approximation may be possible, as shown by the simplicity to calculate a motion's projectile.

2.1 PROJECTILE MOTION WITH VARYING ANGLES (P1-Q2)

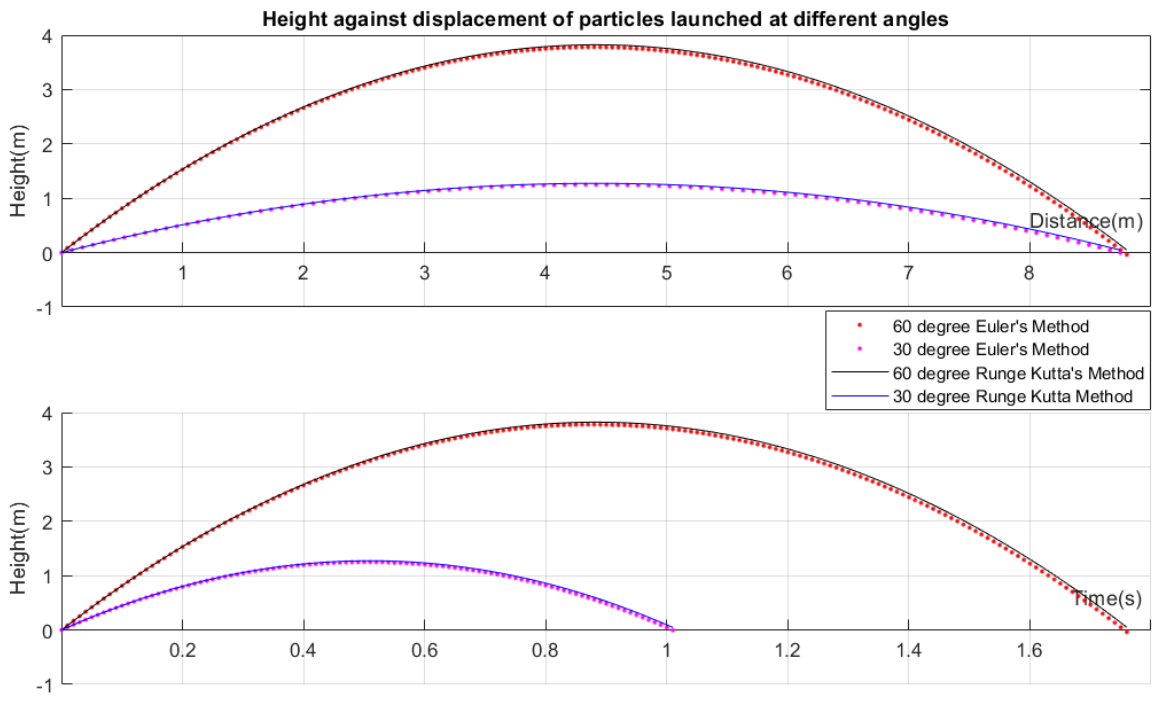

Figure 1: Projectile motion for particles launched at 30 and 60 degrees. (P1-Q2fun.m)

To view the varying paths taken by projectiles launched at two different angles, the Euler's function can be called upon for the different angles while being launched at a constant velocity of 10 m s-1. As the graph shows that the at thirty and sixty degrees, the particle travels the same horizontal distance but takes drastically different paths. The 30-degree projectile had a smaller Y component of velocity and so reached a smaller maximum height finishing its flight much faster. The 60-degree had a greater Y velocity and so reached higher and longer to fall back to the ground.

The step size, dt, determines the precision of the data to the actual value, as smaller the dt closer the approximation. Thus, to identify the best step size multiple variations of dt must be tried out. Starting out with 1 is impossible as the flight time itself is 1.1 and 1.7 seconds respectively for the two angles. While a dt of 0.1 will give a reasonable estimate of the flight data, the difference between the numerical methods is too great. Instead, a dt of 0.01 will provide a value extremely close to the actual without sacrificing any computational speed. However, if need be, the smallest dt recommended is 1E-7s, as anything smaller will not yield any more increase the precision of the values significantly but requires significantly larger computational power.

A popular alternative to Euler's numerical model is the Runge-Kutta method, specifically the fourth order variation(Zill 2018) as shown in Eq.(3). As this is a fourth order differential equation its accuracy will be greater as the truncation While the accuracy of the Euler's method depends significantly on the time step, the Runge-Kutta method is more forgiving as it seems to give the same values consistently for a wider range of time steps.

$$y_{n+1} = y_n + ak_1 + bk_2 + ck_3 + dk_4 \tag{3}$$

2.2 PROJECTILE MOTION WITH IMPACT (P1-Q3a)

As the Euler's Function till now has simply been accounting for the motion of the particle till it reaches ground, it does not give any information on the particles behaviour after the particle makes impact with the ground. By modelling the motion of a 150g steel ball dropped from a height. While certain assumptions regarding the ball was taken such as the ball is hard, and the collision is pin-point so that inertia is negated.

Just like the basic Euler's function, the step size used to model particle impact is extremely important as it determines the frequency with which calculations are performed, and thus the precision of the values to reality. At large time steps like 1s and 0.5s, the stability of the model is risked as the model predicts the particle behaves irrationally, uncontrolled by the If statement. Thus, the largest possible dt that could be feasibly used was 0.1s, and by testing various other dts, the most accurate was seen to be 0.0005s, as anything smaller would just be too computationally intensive.

However, as the surface is smooth, friction does not impact the x component of velocity and so if the loop is left to run on its own it will continue running for every iteration until the Nth term. Thus, an If condition can be used to determine when the total energy of the ball is less than its weight, the loop can be broken as the ball will be unable to overcome the downward gravitational force and leave the ground. Furthermore, this allows one to increase the number of iterations for the loop to an over one million, ensuring that the ball's bouncing motion is always completed. This allows the model to be used for relatively extreme heights.

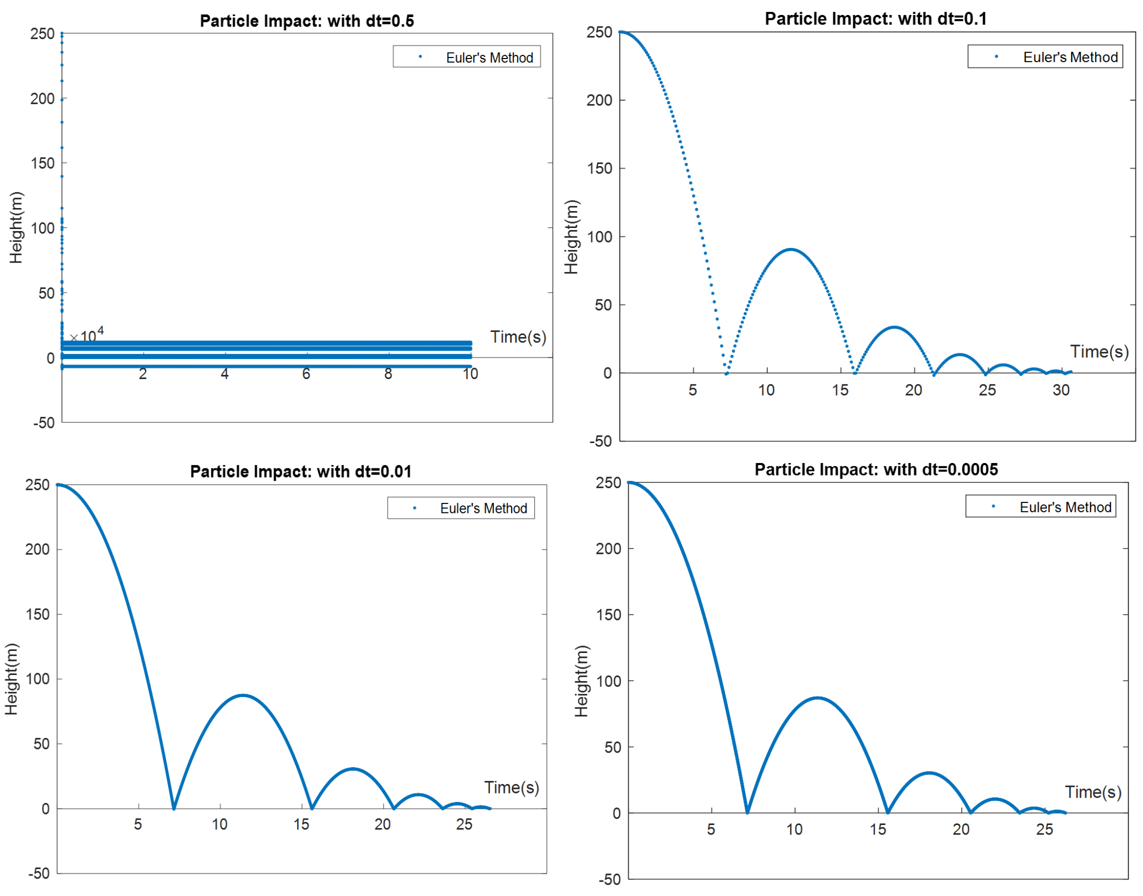

Figure 2: Motion of Particle dropped from 250m tested with different step sizes. (From $P1 - Q3a.m$)

2.3 PROJECTILE MOTION WITH DRAG (P1-Q3B)

Lastly, one must also account for the impact drag has on a projectile's flight path. Thus, by taking the drag coefficient of a sphere as 0.47, one can calculate the drag force and resultant forces on the particle. The resulting acceleration can then replace g in the original Euler's formula to see the impact of drag. An approximation of the Terminal Velocity can be obtained by calculating it using Eq.(4). However, using the particle and fluid densities a drag function can be implemented into the Euler's loop where the new acceleration is calculated by balancing the forces.

In addition, as the fluid is extremely dense, buoyancy must be accounted for in the total force in the direction of Y itself. This is shown to be a significant force especially for the denser fluids like water and glycerin where the terminal velocity decreases by over 1 m s-1. Especially, for the nylon ball in glycerin the terminal velocity is shown to have a positive terminal velocity, suggesting that the particle might actually be floating on the fluid

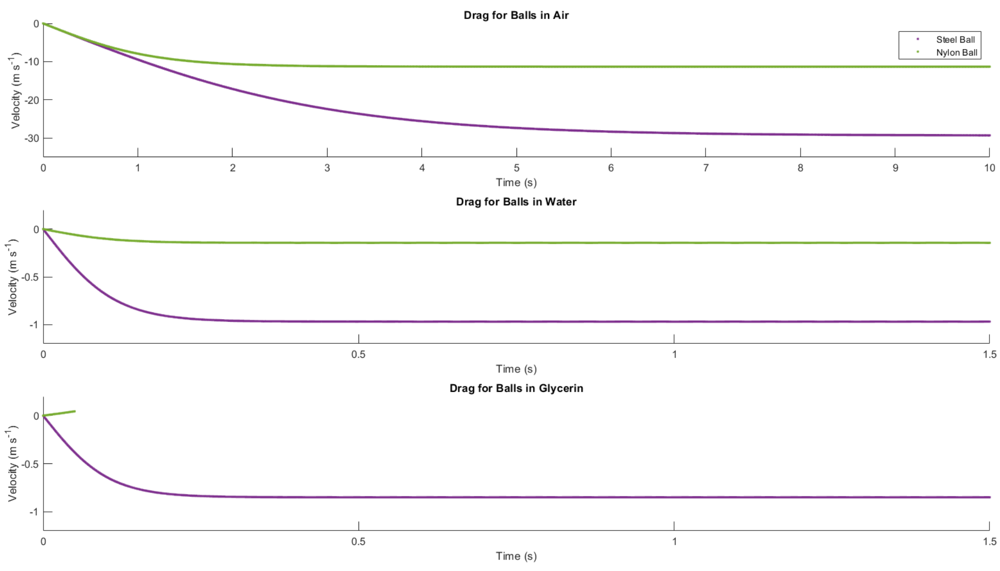

Figure 3: Terminal Velocity Graph for different balls in different fluids)

surface.

$$Theoretical\ Terminal\ Velocity = \sqrt{\frac{2 * P_m * g}{F_{rho.} * P_a * Cd}} \tag{4}$$

Table 2: Terminal Velocities calculated using Theoretical and Euler's Methods for different balls in various fluids

	Steel Ball	m=5.071E-4 kg	Nylon Ball	m=7.52E-5 kg
Fluid	Analytical TV	Euler's TV	Analytical TV	Euler's TV
Air	-29.37	-29.36	-11.31	-11.31
Water	-1.03	-0.96	-0.40	-0.14
Glycerin	-0.92	-0.85	-0.35	0.05

Overall, the Euler's method for computing projectile motion is a very quick and easy making simple calculations, and while it may make assumptions such as a perfectly symmetrical particle (Morales-Vega et al. 2021) to keep initial calculations correctly. However, as shown by the variations of the drag and impact particle code, it can be adjusted to suit specific needs provided the assumptions being made are very clear.

3 PARTICLE TRACKING VELOCIMETRY (P2)

Increasing presence of smartphones, allows people to use its various features in examining different phenomena observed in real life. For instance, using a smartphone's camera one can record a falling ball, and process the complete video file to obtain

Studies on particle collision have revealed that a particle's motion on the rebound is impacted by different properties of both the surface and particle itself. For instance, collision with hard surfaces like wood and concrete will result in less deformation, producing an elastic impact and allowing the ball to rebound to a greater height. Softer more elastic surfaces like cardboard and foam, however, deform and dissipate energy as the when placed under stress, reducing the energy transferred. Greater the drop height of the particle, higher the velocity of the ball when it makes contact, and so higher the rebound will be but the proportion of energy transferred should remain the same. All these different factors effecting the particle collision can be quantified to a single dimensionless ratio, known as the Coefficient of Restitution(COR).

As described in the literature overview, this ratio can be of any properties of the particle restitution is the ratio of post and pre-impact velocities. Also used as the parameter to describe the energy lost by the ball upon impact, it is given the boundaries of $0 < \varepsilon < 1$. By dropping a ball from different heights on various surfaces of different hardness, one can investigate the effect of these factors on the coefficient of restitution.

3.1 Code

Particle Tracking

The OpenCV library was used to process the video file by running each frame through a loop. While the particle can be tracked by identifying the contours of particle but as this is marred by presence of other curved lines, the accuracy is reduced. Thus, a further filter was implemented where the ball's color was isolated in the HSV color gamut. This method allows balls of any colors to be tracked as far as the upper and lower color boundaries are correctly identified,

The Coefficient of Restitution was calculated by manually defining a height above ground in the video, and then using an argwhere function to determine the index of all y positions below this, creating an array of positions where the ball has bounced. This can then be used to calculate the COR, where for height the max height before and after a bounce is called upon and then used in Eq.(2).

A similar method was tried for calculating the COR_V but indexing the velocity arrays and then calling the velocity immediately before and after bounce proved produced often inaccurate COR_V. However,for a ball that was being dropped rather than thrown, its initial velocity will be zero and so the velocity before impact is the most negative and after will be the most positive, and then used in Eq.(1). However, this shortcut is only valid for the first impact, and when the ball is dropped and not thrown.

Empirical and DEM bounce Model

The empirical model was created using the basic SUVAT equations rearranged to give velocity, height and time at max height and ground level. Using this model, allows one to find the properties of a projectile motion at maximum and minimum points, but does not give any information about the path it took in between.

Working off the projectile motion model created in , modelling a using a numerical solver like Euler's method, modelled to include drag, buoyancy and other possible factors affecting the motion of ball.

3.2 Experiments

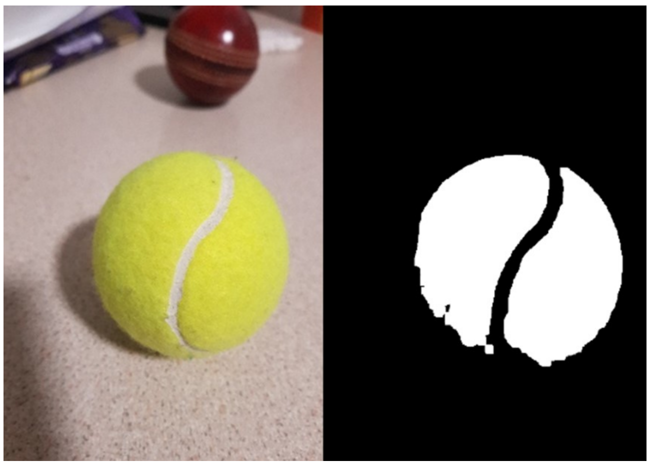
Figure 4: Tennis Ball Used, and colour isolated image

As the particle tracking code uses the OpenCV library to perform a series of Gaussian blurs, colour conversions to isolate the color of the tennis ball as shown in Figure 1. However, as this isolation requires the colour boundary to be precisely defined, pictures of the ball were taken and run through the ColorPicker.py, where the boundaries can be manually selected until satisfied

By taking the video in well-lit surroundings one environment perpendicular to the ground and capable of capturing the complete trajectory of the ball, from release till rest(ground). Similar colors cannot be present in the video as that will impact the quality of tracking, and multiple ball tracking is not possible as only the largest contour is tracked. Furthermore, motion in the frame should be kept to a minimum frame, especially if a red ball is being tracked as the color boundaries can detect human skin too.

A standard tennis ball, in the green HSV range was dropped against a solid color wall allowing maximum contrast, allowing it to be isolated more easily.

Aside from particle and surface properties, the impact velocity is the only significant factor affecting COR. Weir and Tallon (2005) said that particle size, densities, and Young's are just some of the factors affecting COR. However, as these properties are difficult to control, at least in a manner that would produce meaningful results, the decision was made to keep the particle constant and investigate the effect of other factors such as surface harness and approaching velocity. As the tennis ball's terminal velocity of 23.64 ms^{-1} will only be reached if dropped from a minimum height of 28.5m. However, as achieving such velocities is not possible, at least safely, one is restricted to perform the experiment using more achievable heights. By varying the heights between 2 levels at 75cm and 125cm, one is changing the impact velocity of the ball, while the surface hardness was changed by using materials at 3 levels of hardness; cardboard, carpet, and wood.

3.3 Results, Analysis and Discussion

Table 3: Average Coefficient of Restitution for different heights and surfaces

	Height(-): 0.75 m		Height(+): 1.25m	
	COR_V	COR_H	COR_V	COR_H
Cardboard (-)	0.50	0.51	0.17	0.34
Carpet (0)	0.26	0.75	0.18	0.77
Wood (+)	0.78	0.75	0.68	0.74

The coefficient of restitution is calculated using both COR_V and COR_H, to evaluate the accuracy of each method against the average RE of a tennis ball on a rigid surface, 0.72-0.76(Roux and Dickerson 2007; Colombo et al. 2016). As the COR_H for carpet and wood averages out to about 0.75, compared to the average COR_V of 0.47. These difference results in results suggests that height is a better measure of the restitution coefficient as it is not impacted by the quality of video and fps. This is important as the ratio for the restitution velocity is for velocity immediately before and after bounce which, becomes less accurate as the fps decreases.

While velocity is a better representative of the restitution coefficient, the reliability of the COR calculated using the ratio of heights makes it a more useful test to see the effects of the various factors.

The effect of the surface and heights on RE can be observed on the main effect graph, and where surface is shown to have a positive correlation 0.31 and the weaker effect of height with 0.099. The interaction effect of 0.076 suggests that there is little to know about their combined effect on RE, at least from the designed experiment.

However, the lack of a statistically significant difference in COR at increasing heights and impact velocity, may be because the range of heights was not spread enough.

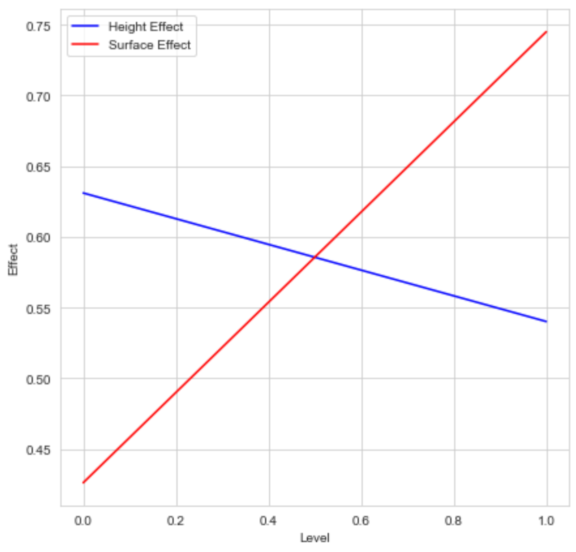

Figure 5: Terminal Velocity Graph for different balls in different fluids)

DEM and Analytical Comparison

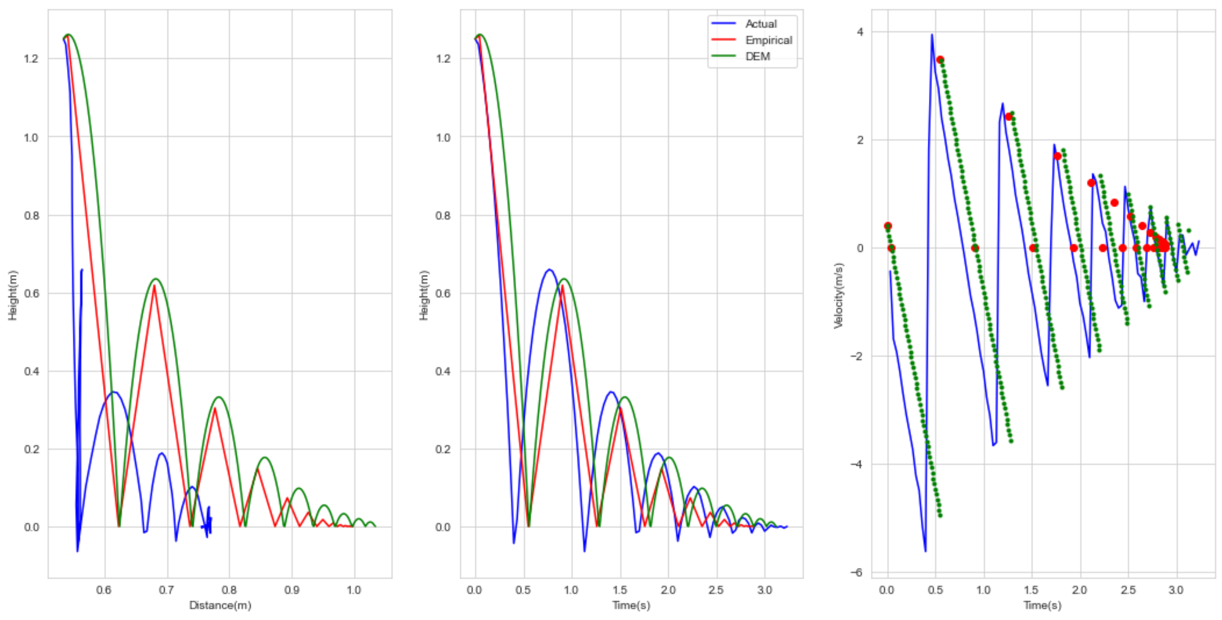

Figure 6: Empirical, DEM compared with actual motion of particle

A comparison of the DEM and empirical models for a ball dropped from 125cm on a wooden surface, can be seen in Figure 3- Empirical, DEM compared with actual motion of particle, where the predicted motions are plotted along with the actual one. The differences in the models can be attributed to factors such as the ignorance of spin on the ball and the lack of friction with air. Furthermore, by taking the ball as a hard sphere model where contact time is negligible.

In addition, as the model only calculates the x and y positions of the ball, it cannot account for the movement of the ball along z-axis in a 3D space. Furthermore, as it takes the surface to be smooth and no friction is exerted on the ball and so the x components of the ball is taken to be unaffected which is shown to be untrue.

4 References

Asteriou, P., H. Saroglou, and G. Tsiambaos (2012). “Geotechnical and kinematic parameters affecting the coefficients of restitution for rock fall analysis”. In: *International Journal of Rock Mechanics and Mining Sciences* 54, pp. 103–113. ISSN: 1365-1609. DOI: 10.1016/j.ijrmms.2012.05.029.

Bernstein, Alan D (1977). “Listening to the coefficient of restitution”. In: *American Journal of Physics* 45.1, pp. 41–44.

Blais, Bruno et al. (2019). “Experimental Methods in Chemical Engineering: Discrete Element Method—DEM”. In: *The Canadian Journal of Chemical Engineering* 97.7, pp. 1964–1973. ISSN: 0008-4034. DOI: 10.1002/cjce.23501. URL: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cjce.23501.

Coetzee, C. J. (2017). “Review: Calibration of the discrete element method”. In: *Powder technology* 310, pp. 104–142. ISSN: 0032-5910. DOI: 10.1016/j.powtec.2017.01.015.

Colombo, Federico et al. (2016). “Novel Methodology for Measuring the Coefficient of Restitution from Various Types of Balls and Surfaces”. In: *Procedia Engineering* 147, pp. 872–877. ISSN: 1877-7058. DOI: 10.1016/j.proeng.2016.06.287. URL: https://doi.org/10.1016/j.proeng.2016.06.287.

Cundall, P. A. and O. D. L. Strack (1979). “A discrete numerical model for granular assemblies”. In: *Géotechnique* 29.1, pp. 47–65. ISSN: 0016-8505. DOI: 10.1680/geot.1979.29.1.47.

Hastie, D.B. (2013). “Experimental measurement of the coefficient of restitution of irregular shaped particles impacting on horizontal surfaces”. In: *Chemical Engineering Science* 101, pp. 828–836. ISSN: 0009-2509. DOI: https://doi.org/10.1016/j.ces.2013.07.010. URL: https://www.sciencedirect.com/science/article/pii/S0009250913005022.

Ismail, K. A. and W. J. Stronge (2008). “Impact of Viscoplastic Bodies: Dissipation and Restitution”. In: *Journal of Applied Mechanics* 75.6, p. 061011. ISSN: 0021-8936. DOI: 10.1115/1.2965371.

Itseez (2015). *Open Source Computer Vision Library*. https://github.com/itseez/opencv.

Lu, G., J. R. Third, and C. R. Müller (2015). “Discrete element models for non-spherical particle systems: From theoretical developments to applications”. In: *Chemical engineering science* 127, pp. 425–465. ISSN: 0009-2509. DOI: 10.1016/j.ces.2014.11.050.

Marigo, Michele and Edmund Hugh Stitt (2015). “Discrete Element Method (DEM) for Industrial Applications: Comments on Calibration and Validation for the Modelling of Cylindrical Pellets”. In: *KONA Powder and Particle Journal* 32.0, pp. 236–252. ISSN: 0288-4534. DOI: 10.14356/kona.2015016.

Morales-Vega, Patricia et al. (2021). “Experimental model for the description of the behaviour of a 9-mm projectile at a target”. In: *Forensic Sci Res* 6.1, pp. 67–74. ISSN: 2096-1790. DOI: 10.1080/20961790.2019.1697078.

Roux, Andre and Jennifer Dickerson (2007). “Coefficient of restitution of a tennis ball”. In: *Int. School Bangkok (ISB). J. Phys.* 1.1, p. 7.

Stevens, A.B. and C.M. Hrenya (2005). “Comparison of soft-sphere models to measurements of collision properties during normal impacts”. In: *Powder Technology* 154.2-3, pp. 99–109. ISSN: 0032-5910. DOI: 10.1016/j.powtec.2005.04.033.

Theis, Amy E (2014). “Case study: T2 Laboratories explosion”. In: *Journal of Loss Prevention in the Process Industries* 30, pp. 296–300.

Weir, Graham and Stephen Tallon (2005). “The coefficient of restitution for normal incident, low velocity particle impacts”. In: *Chemical Engineering Science* 60.13, pp. 3637–3647. ISSN: 0009-2509. DOI: 10.1016/j.ces.2005.01.040.

Zill, Dennis G. (2018). *A first course in differential equations with modeling applications / Dennis G. Zill.* eng. 11E.; Metric version / prepared by Aly El-Iraki. Australia : Cengage Learning. ISBN: 9781337669139.

5 Appendix

5.1 APPENDIX A- SOFTWARE VERSIONS

Coding Languages and Libraries used:

- MATLAB: 9.12.0.1884302 (R2022a)
- Python: 3.9.7 (MSC v.1916 64 bit (AMD64))
- OpenCV2: 4.5.5
- NumPy: 1.20.3
- Matplotlib: 3.5.1

5.2 Euler's Code(MATLAB)

```
% Euler's Function for projectile motion
%
% Written by:   Abbas Moosajee
% Written on:   11/03/2022
% Contact:      AHM080@student.bham.ac.uk


function [r_x,r_y,v_x,v_y,r_xa,r_ya,tt] = Eulersfunction(v,theta,r_xy,dt)
r_x=r_xy(1,:); r_y=r_xy(2,:); % Assigning the specific X and
% Y coordinates from initial vector
g=9.81;                        % Acceleration due to Gravity in m s^-2
v_x=v.*cosd(theta);            % X component of Velocity
v_y=v.*sind(theta);            % Y component of Velocity
t_max=(v_y+sqrt((v_y^2)+(2*g*r_y)))/g; % Maximum time of flight
% accounting for intial particle height
N=abs(t_max/dt);               % Calculating the Number of iterations loop

%% Analytic Solution
t_i=0;T_list=(t_i:dt:t_max);              % Complete array of times
r_ya = r_y+(v_y.*T_list-(g.*T_list.^2/2));% Analytical Y Position (m)
r_xa = r_x+(v_x.*T_list);                 % Analytical X Position (m)

%% Euler's Numerical method


for ts = 1:N %
    ts=ts+1;
%   v_x(ts) = v_x(ts-1);               % X Velocity is constant
    r_x(ts) = r_x(ts-1) + dt*v_x;      % New X Position (m)
    v_y(ts) = v_y(ts-1) + dt*(-g);     % New Y Velocity (m/s)
    r_y(ts) = r_y(ts-1) + dt*v_y(ts);  % New Y Position (m)
    tt(ts)  = (ts-1+dt)*dt;            % Total time of flight (s)
end                                    % End for loop
end                                    % End Function
```

5.3 Particle Tracking Code(Python)

```
"""
==================================================================================
    # University of Birmingham
    # Labs and Data Analysis 2
    # Portfolio 2:  Particle Tracking
    # Written by:   Abbas Moosajee
    # Written on:   10/04/2022
    # Contact:      AHM080@student.bham.ac.uk
    # Function:     Track a falling ball with OpenCV
==================================================================================
"""

import numpy as np
from cv2 import cv2


def Particle_Tracker(video, track, Particle, itr, height, ymin):
    count = 0
    vs = cv2.VideoCapture(video)
```

```
color = Particle["Color"]
m = Particle["Mass"]  # Mass in kg
Lower = color[0]      # Lower Color Boundary
Upper = color[1]      # Upper Color Boundary
g = 9.81              # Gravitational Acceleration
buffer = int(vs.get(cv2.CAP_PROP_FRAME_COUNT))
x, y, center = (None for _ in range(3))  # Defines position of particle
pts, yt, xt, ct = ([] for _ in range(4))   # Creates growable lists

# %% keep looping
while True:
    oframe = vs.read()  # Grabs current frame
    oframe = oframe[1]
    fps = vs.get(cv2.CAP_PROP_FPS)  # Frame rate of video
    dt = 1/fps                      # Time step
    # Viewing a video and determines,when video reaches end
    if oframe is None:
        break
    count = count+1
    frame = oframe[::-1, :, :]  # Flips image so bottom left is origin
    # Resizes the frame, blurs it, and convert it to HSV color gamut
    frameR = cv2.resize(frame, dsize=(450, 650))
    blurred = cv2.GaussianBlur(frameR, (11, 11), 0)
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

    # Constructs a mask for the ball color, then performs a series
    # of dilations and erosions to clean up the mask
    maskR = cv2.inRange(hsv, Lower, Upper)
    maskE = cv2.erode(maskR, None, itr)
    maskD = cv2.dilate(maskE, None, itr)

    # find contours in the mask and initialize the circle from center
    contours, hierarchy = cv2.findContours(maskD.copy(),
                    mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_NONE)

    if len(contours) > 0:  # only proceed if at least one contour is found
        # in the maskD, and then use it to create an enclosing circle
        c = max(contours, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

        if track == 1:  # allows live tracking on video
            if radius > 3:  # only proceed if the radius meets a minimum size
                # Draws circle on the frame
                cv2.circle(frameR, (int(x), int(y)),
                           int(radius), (0, 255, 255), 2)
                cv2.circle(frameR, center, 5, Upper, -1)

            for i in range(1, len(pts)):
                if pts[i - 1] is None or pts[i] is None:
                    continue
                # Computes the thickness draws connecting lines
                thickness = int(np.sqrt(buffer / float(i + 1)) * 2.5)
                cv2.line(frameR, pts[i - 1], pts[i],
                         (0, 0, 255), thickness)
            # show the frame to screen
            cv2.imshow("Frame", frameR[::-1, :])
            if cv2.waitKey(1) & 0xFF == ord('q'):
                vs.release()
                break
        # Adds value from obtained from curent frame to list
        pts.append(center)
        ct.append(count)
        yt.append(y)
        xt.append(x)
cv2.destroyAllWindows()  # close all windows

# %%
# Identifies the positions in which the y position arrays have None values
inl = np.argwhere(np.array(yt) == None)
if len(inl) == 0:
    inl = [1]
yt = np.array(yt[int(inl[-1]+1):])
xt = np.array(xt[int(inl[-1]+1):])
ct = np.array(ct[int(inl[-1]+1):])

# Uses drop height to determine the relative conversion from pixels to m
r = height/(yt[0]-yt[-1])
# Calculates angle relative t horizontal using Pythagoras Theorem
thetad = np.rad2deg(np.arctan(yt[0]/xt[0]))
# Converts Pixel coordinates to values in m
ypm = (yt-(yt[-1]))*r
xpm = xt*r
# Converts frame counter to time values by multiplying by dt
tp = (ct-ct[0])*dt
# Calculates velocity of particle between two frames, by dividing the
# change in its dispalcement with the dt
vxp = np.asarray([xpm[ti+1]-xpm[ti] for ti in range(len(xpm)-1)])/dt
vyp = np.asarray([ypm[ti+1]-ypm[ti] for ti in range(len(ypm)-1)])/dt
vp = np.sqrt((vxp**2)+(vyp**2))
# Calculates acceleration by dividing the velocity of particle between two
# points with the dt
axp = np.asarray([vxp[ti+1]-vxp[ti] for ti in range(len(vxp)-1)])/dt
ayp = np.asarray([vyp[ti+1]-vyp[ti] for ti in range(len(vyp)-1)])/dt

# Calculate Gravitational Potential Energy, Kinetic Energy, and Total Energy
GPE = m*g*ypm[1:]
KE = (0.5)*m*(vyp**2)
TE = GPE+KE

# %% Restitution Coefficient Calculations
# Determines the index in y arrays where particle has made contact with
# ground and changed direction. Note: ymin must be manually defined
Bts = np.argwhere(yt < (ymin))

# As the Velocity before impact is most negative and after most positive,
# the COR for the velocity method can be found by dividing the max velocity
# over the min, albeit only for the first bounce
REv = max(vyp)/min(vyp)

# Uses the indexed position Bts, to create two height arrays one of before
# bounce and one of after bounce. Use the max heights of each to find COR
REy = np.sqrt(
    abs(max(ypm[int(Bts[0]):int(Bts[1])])/max(ypm[0:int(Bts[0])])))

REs = [abs(REv), abs(REy)]
r_xyi = [xpm[0], ypm[0]]
Data_Table = [tp, xpm, ypm, vxp, vyp, axp, ayp, GPE, KE, TE]
Initial_Conditions = {"Coordinates": r_xyi,
                      "Velocity": vp[0],
                        "Launch Angle": thetad}

print('Coefficient of Restitution based on Velocity:', np.round(REs[0], 2),
      'Coefficient of Restitution based on Height:', np.round(REs[1], 2))
return(Data_Table, Initial_Conditions, REs)
```

5.4 Empirical Model

Table 4: SUVAT Equations used for Empirical DEM Model:

$y_{max} = \frac{v_y^2}{2*g} + y_i$	Determines max height based on velocity
$t_{fall} = \sqrt{\frac{2*y_{max}}{g}}$	Determines time to fall to ground
$t_{total} = t_i + t_{fall}$	Adds time to fall to total time travelled by ball yet
$v_y = t_{fall} * g$	y velocity when ball reaches ground
$v_x = v_{xi}$ and $x = x_i + (v_x * t_{fall})$	Calculates horizontal distance travelled
$y_{grd} = 0$	New y position is ground
$v_y = -COR * v_{yi}$	Velocity after bounce
$t_{max} = \frac{v_y}{g}$	Time to reach max height
$t_{total} = t_i + t_{max}$	Adds time to reach max height to total time travelled by ball yet
$v_x = v_{xi}$ and $x = x_i + (v_x * t_{max})$	Calculates horizontal distance travelled

5.5 Experimental Design

Table 5: Experimental Design

Run	Ball Type	Height	Surface
7,8	Green Tennis	75cm (-)	Cardboard(-)
11,12	Green Tennis	75cm (-)	Carpet(0)
3,4	Green Tennis	75cm (-)	Wood(+)
5,6	Green Tennis	125cm(+)	Cardboard(-)
9,10	Green Tennis	125cm(+)	Carpet(0)
1,2	Green Tennis	125cm(+)	Wood(+)