```
clc;
close all;
clear all;

% Importing video
vid = VideoReader('pingpongvid3.mp4');
%  Converting each frame into an individual image & saving as a "stack"
frames = read(vid);
numFrames = vid.NumFrames;
frameRate = vid.FrameRate;

% Cropping the first image to establish the region of interest
frame = frames(:,:,:,1);
imshow(frame);
title('Select Region of Interest');
croppedArea = round(getrect); % Select ROI manually

% Creating a vector to store particle positions
posVec = [];

% Processing frames to track the ball
for i = 1:numFrames
    % Cropping frame
    frameCropped = imcrop(frames(:,:,:,i), croppedArea);

    % Converting to grayscale
    frameCropped = rgb2gray(frameCropped);

    % Binarizing the image
    frameBW = imbinarize(frameCropped, 0.66);

    % Removing noise
    frameBW = imopen(frameBW, strel('disk', 3));

    % Finding the ball position with an improved radius range
    [pos, rad] = imfindcircles(frameBW, [10 50]); % Adjusted min radius

    % Check if ball was detected before storing data
    if ~isempty(pos)
        posVec = [posVec; pos(1,:) i]; % Store (x, y) and frame number
    end

    % Display processed frame
    imshow(frameBW);
    title(sprintf('Frame: %d', i));
    hold on;
    if ~isempty(pos)
        viscircles(pos(1,:), rad(1));
    end
    hold off;
    drawnow;
end

% Ensure there are detected positions before proceeding
if isempty(posVec)
    error('No ball positions detected. Try adjusting the imfindcircles
parameters.');
end
```

```
% Extracting height vs time data
timeVec = posVec(:,3) / frameRate; % Time in seconds
heightVec = max(posVec(:,2)) - posVec(:,2); % Convert y-coordinates to height
(pixels)

% Compute Velocity (Central Difference Approximation)
velocityVec = zeros(size(heightVec));
for j = 2:length(heightVec)-1
    velocityVec(j) = (heightVec(j+1) - heightVec(j-1)) / (timeVec(j+1) - timeVec(j-
1));
end
velocityVec(1) = velocityVec(2);
velocityVec(end) = velocityVec(end-1);

% Identify Impact Points (Bounces) using findpeaks
[~, impactIndices] = findpeaks(-velocityVec, 'MinPeakDistance',
floor(frameRate/5));

% Check if enough bounces are detected
if length(impactIndices) >= 2
    eValues = [];

    for i = 1:length(impactIndices)-1
        impactIndex = impactIndices(i);

        % Ensure valid range
        if impactIndex > 5 && impactIndex < length(velocityVec)-5

            % Get pre-impact velocity: Maximum falling speed before impact
            vi = min(velocityVec(impactIndex-5 : impactIndex));

            % Get post-impact velocity: Maximum rising speed after impact
            vf = max(velocityVec(impactIndex : impactIndex+5));

            % Calculate restitution coefficient
            e = abs(vf / vi);
            eValues = [eValues, e];

            % Output Results
            fprintf('Bounce %d:\n', i);
            fprintf('Velocity before impact: %.3f m/s\n', vi);
            fprintf('Velocity after impact: %.3f m/s\n', vf);
            fprintf('Coefficient of Restitution: %.3f\n\n', e);
        end
    end

    % Compute the average coefficient of restitution
    eAvg = mean(eValues);
    fprintf('Average Coefficient of Restitution: %.4f\n', eAvg);
else
    fprintf('Not enough bounces detected to compute velocity before/after impact.\
n');
end

% Plotting Height vs. Time
figure;
plot(timeVec, heightVec, '-o', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Height (pixels)');
```

```
title('Height vs. Time');
grid on;

% Plotting Velocity vs. Time
figure;
plot(timeVec, velocityVec, '-o', 'LineWidth', 1.5, 'Color', 'r');
xlabel('Time (s)');
ylabel('Velocity (pixels/s)');
title('Velocity vs. Time');
grid on;
```