CSU, SACRAMENTO

EEE 178: MACHINE VISION

3D Object Tracking Using Stereo Vision

Authors:
Curtis MUNTZ
David LARRIBAS
Thao CHAU

Instructor:
Dr. BELKHOUCHE

Abstract

A system for offline 3D tracking using stereo vision is implemented in MATLAB. Calibration of a stereo rig is explored. Various methods must be implemented to isolate and track an object. An orange ball is used as the object to simplify the isolation techniques in order to prove functionality.

Keywords. Object Tracking, Camera Calibration, Stereo Vision

May 18, 2014

I. Introduction

The goal of this project is to track a moving object in 3D space. The parameters we are concerned with are position, velocity, and acceleration. There are many methods to perform this type of analysis, but we chose to use the method of stereoscopic vision which uses two cameras to determine these parameters. This method involves synchronizing the two cameras, determining their intrinsic and extrinsic parameters, isolating the desired object, and then finally calculating the spatial parameters of the object.

II. Camera Synchronization

In order to produce meaningful data from the stereo rig, we need to be able to guarantee that the two cameras are producing synchronized data. One way to accomplish this was by finding the time stamp of the same frame being seen by the left and the right cameras. Listing 1 shows the method used to capture two images simultaneously.

```
ffmpeg -f v4l2 -r 25 -s 640x480 -i /dev/video0
    -f v4l2 -r 25 -s 640x480 -i /dev/video1 -
    filter_complex "nullsrc=size=1280x480 [
    base]; [0:v] setpts=PTS-STARTPTS, scale
    =640x480 [left]; [1:v] setpts=PTS-STARTPTS
    , scale=640x480 [right]; [base][left]
    overlay=shortest=1 [tmp1]; [tmp1][right]
    overlay=shortest=1:x=640" -c:v libx264
    output.mp4
```

Listing 1. Bash Camera Capture Code

This method was modified from the FFMPEG documentation. It allows for simultaneous overlaying of videos onto a complex filter. The /dev/video0 device is placed on the left side of the output movie, and the /dev/video1 device is placed on the right side of the output movie [1]. According to the documentation, the setpts=PTS-STARTPTS flag ensures that each frame has the same time stamp. Exploiting this feature allows us to synchronize our camera inputs. Figure 1 shows an example of the synchronized frames.

Fig. 1. *Synchronized Camera Frames*

Because these frames are overlayed on top of each other, the MATLAB script shown in Listing 2 was used to split each camera into its respective frames for later MATLAB processing [2].

```
clear all, close all, clc;

workingDir = 'C:\Users\me\Desktop\zzzGOODATA\'
    ;
fileNAME   = 'bounce.mp4';
cd(workingDir)
vid = VideoReader(fileNAME);

%convert each frame into an imag
for ii = 1:vid.NumberOfFrames
    img = read(vid,ii);
    %resolution decided to maximize frames per
     second
    %crop each image to reclaim original
    frames
    Left  = img([1:480],[1:640],:);
  Right = img([1:480],[641:1280],:);
    imwrite(Left,fullfile(workingDir, 'bounce'
    , sprintf('left%03d.jpg',ii)));
    imwrite(Right,fullfile(workingDir, 'bounce
    ', sprintf('right%03d.jpg',ii)));
    imwrite(img,fullfile(workingDir, 'bounce',
     sprintf('img%03d.jpg',ii)));
end
```

Listing 2. MATLAB Frame Splitting Code

III. Camera Calibration

Not all cameras are equal. Each camera will have specific intrinsic parameters relating to how they view the world. These parameters are pixel width/height, focal length, and pixel offset. We attempted a number of different methods to calculate these values, but ultimately ended up building our own method. Each camera will also have extrinsic parameters which relate the cameras to their position in the world plane.

A. Pre-Built Methods

Many pre-built methods exist and can be found online. One such method is the built-in camera calibration toolbox for MATLAB. This method involved taking multiple screen captures of a black and white checkerboard pattern. The module would then detect the corners of the squares and use these points to determine the intrinsic and extrinsic parameters of the camera. The output of this process was a matrix we were unfamiliar with and could not deduce its meaning. We looked at and attempted various other pre-built methods and ran into the same problem. Performing the calibration was easy enough, but the results were mostly unusable. For this reason, we decided to try to perform calibration from scratch using the equations learned in class.

B. Our Method

Our method was to measure the real-world coordinates and map them to the pixel coordinates of one camera. As we are assuming the cameras intrinsic parameters are equal and there to be zero rotation and a translation of only the baseline, our process will be much simpler than those we attempted and the final matrix will be much easier to use. We decided to use the checkerboard pattern like the other methods we attempted used. This pattern can be seen in Figure 2.

We measured out the points in the real-world where the checkerboard pattern was located in front of the camera. We used the left camera as the origin for the world plane. The z-axis was represented on the table, the surface in which the cameras were located, using electric tape and this was aligned with one of the columns on the checkerboard. The height of the

camera lens from the table was measured and translated over to the checkerboard. Using this center point on the grid and knowing the height and width of each square, we were able to define all the points of the grid in real-world coordinates. We chose 24 points to use for our calibration.

Once the real-world points were defined, we then took a picture of the grid and determined the pixel coordinates of these points. With these coordinates known, we used the equation $Ax = 0$, fully shown in Equation 1 to determine the intrinsic parameters of the camera. With these known, we are now able to deduce the real-world coordinates from the pixel coordinates read by the cameras.

$$\begin{bmatrix} rz & -z & -x \\ rc & -z & -y \end{bmatrix} \begin{bmatrix} Sx & Sy \\ u_o Sx & v_o Sy \\ \lambda & \lambda \end{bmatrix} = 0 \tag{1}$$

Using MATLAB's SVD solver, we obtained the intrinsic parameters as follows, where the $Xright_x$ is the solution for the left column of the x matrix and $Xright_y$ is solution for the right column of the x matrix.

```
Xright_x =      Xright_y =

    0.0012        0.0011
    0.2680        0.3025
    0.9634       -0.9532
```

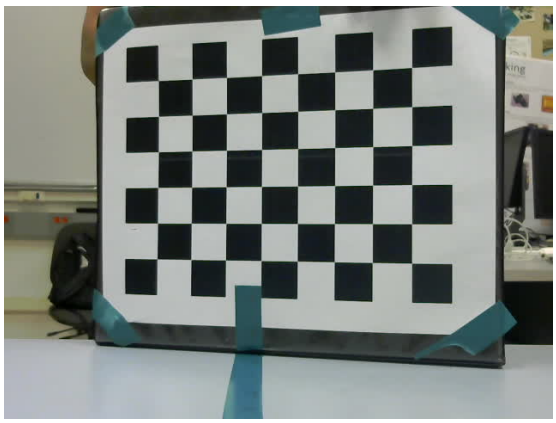

Fig. 2. *Checkerboard Calibration Image*

IV. Object Isolation

The method of object isolation selected depends heavily on the type of object being tracked without applying advanced techniques. We wanted to keep the process simple, but effective enough for us to perform the analysis. For this reason we selected the technique based upon the object we would be detecting. The goal of the entire process was to reduce an object down to a single point, so that we could compare the disparity between the stereo cameras and estimate the relative position.

A. Background Subtraction

Initially we wanted to track a person. For this we used background subtraction and the method worked well. The process involved taking 100 frames of the background, without the object, and finding the average of the pixel values. A large number of frames were used to account for the fluctuations in lighting that naturally occur from their 60Hz power source. This background image was then subtracted from each subsequent frame being analyzed. The result was a fairly clear object with a heavily faded background. A Gaussian filter was then applied to the subtracted frame to smooth out the edges of the object. Finally, thresholding was applied to the frame to produce a binary image with only the object as ones. Figure 3 shows an example of this process.

While this method was effective for isolating a person, we found that the results after detecting the centroid were rough. As the person turned sideways, there was a large amount of noise in the Y-axis. Additionally, because of the small areas that were available to us, it was difficult to have the person fully in the frame which led to issues in detecting the center of mass. Because of these issues, we decided to defer to detecting a small orange ball. We wanted to be able to hold the ball while tracking which eliminates background subtraction as an effective means of isolation.

B. HSV

HSV is a method which converts the RGB parameters of an image to hue, saturation, and value. This method of isolation was sort of a shot in the dark to find a solution to isolating the orange ball. To perform this method, we determined some threshold of each value by using imshow on the original (HSV converted) image and looking at multiple points within the object to determine the range of values the ball exists within. With these values used to isolate the image, we saw good results for the most part. We ran into false readings when there was cardboard or bright blue in the image. It became difficult to isolate the bright orange from these colors and we were seeing better results using the chroma method so we opted to used that.

C. Chroma (YCbCr)

Experimental results were obtained by isolating the blue chroma channel of the YCbCr color map spectrum. In this mapping, the color of our orange ball stood out as a bright object against a dull background. This enabled us to threshold and then filter out most of the background by using some morphological operations. Figure 4 shows the final process used to isolate an object. The resultant algorithm ended up being incredibly robust, and could track our orange ball incredibly easily against most backgrounds without having to adjust the threshold in between sample videos. The final code can be seen in Listing 6

One of the biggest benefits of this method compared to background subtraction is this method allows us to have moving objects in the image. Because this method of detection only cares about color, we are able to have people moving in the shot along with the ball and have only the ball detected. This allowed us to hold the ball and move it around as we desired. We used this advantage to hold the ball and move it along one axis at a time. We then plotted the movement of

Fig. 3. *Background Subtraction Process*

the ball and were able to verify the system was registering the balls movements in these axes.

V. Object Tracking

Now that our object has been isolated, we need to determine what points to use to determine where to track. Our first instinct was to use regionprops to located the centroids of each object. This was a good method, but in the event of the object splitting due to inefficient filtering, we would get multiple objects at times. To solve this issue we used the average of the centroids. This method worked well enough, but was still lacking. In the event of a stray pixel, the centroid would be shifted greatly. The solution to this problem was to take the average of the centroids weighted to the area of each object. This combination yields constant centroid regions with little undesired fluctuation due to object separation and background noise.

With the pixel coordinates determined, we now look to convert these values into real-world coordinates. This is done using the intrinsic parameters we found earlier and Equations 2-6. We also want to know the velocity and acceleration of the object being tracked. We used Eulers approximation to determine the discrete derivative which allows us to estimate the objects acceleration and velocity. For our presentation, we used the magnitude of velocity and acceleration to reduce clutter.

$$u_r = \frac{\lambda(x-b)}{z} \tag{2}$$

$$u_l = \frac{\lambda x}{z} \tag{3}$$

$$v_r = \frac{\lambda y}{z} \tag{4}$$

$$v_l = \frac{\lambda y}{z} \tag{5}$$

$$z = \frac{\lambda b}{u_l - u_r} \tag{6}$$

To determine that our methods were actually producing accurate results, we devised a test plan. To determine whether or not the distance equations were producing valid results, we set the ball up on a chair, level with the cameras. We then measured the distance from the ball to the cameras. The test plan can be seen in Figure 5

Fig. 5. *Distance Test*

The distance we measured was 4 feet. A short video was taken to produce a synchronized set of images. We then processed the images through our code to determine distance and found the distance to be roughly 3.9 feet. This proved the code to be a success with an error of about 2.5% at that distance.

To determine if the other two parameters were producing accurate results, we dropped the ball in front of the cameras.

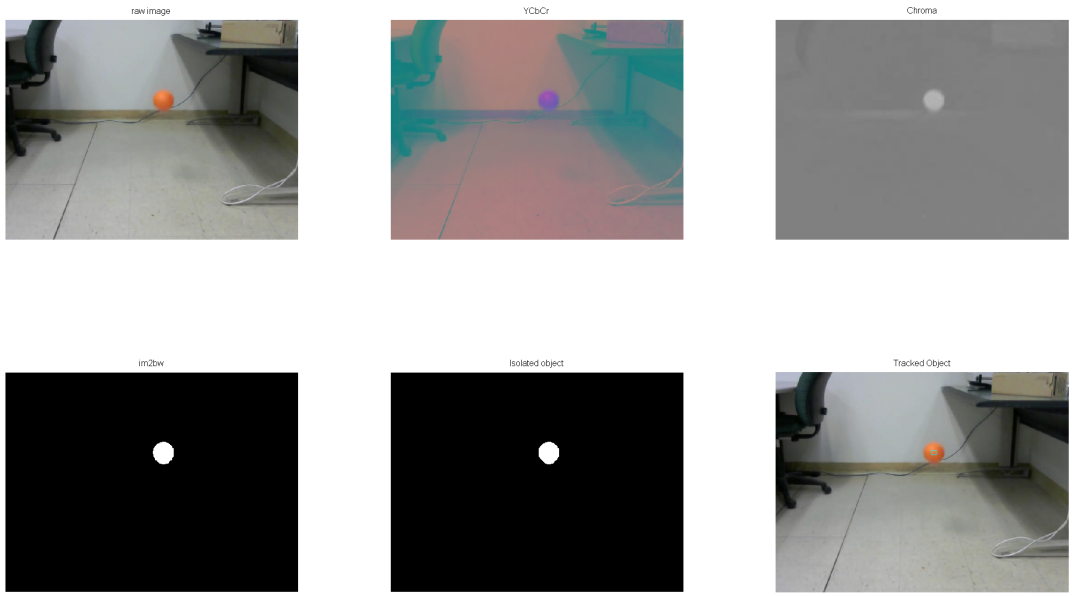

Fig. 4. *Object Isolation*

We expected to see the velocity increasing and the acceleration to be close to $9.8\ \frac{m}{s^2}$, the acceleration due to gravity. Our results showed just that. The magnitude of acceleration hovered around $9.8\ \frac{m}{s^2}$ with an impulse when the ball hit the ground. The velocity was shown to slowly increase as the ball fell. Figure 6 shows the resultant data.

VI. Conclusion

Through trial and tribulation, we were able to develop a tracking system utilizing two cameras. We can determine the distance, velocity, and acceleration of an object within a reasonable margin of error. Our stereoscopic processing procedure produces accurate results within 5% error. As of now, our system only works with certain objects and backgrounds, but advanced techniques could be implemented to produce better results.

References

[1] FFmpeg, "Create a mosaic out of several input videos." ffmpeg.org, 2013.
[2] Mathworks, "Convert between image sequences and video." Mathworks, 2014.
[3] S. Dave, "Basic image processing with matlab code." weebly.com, 2012.

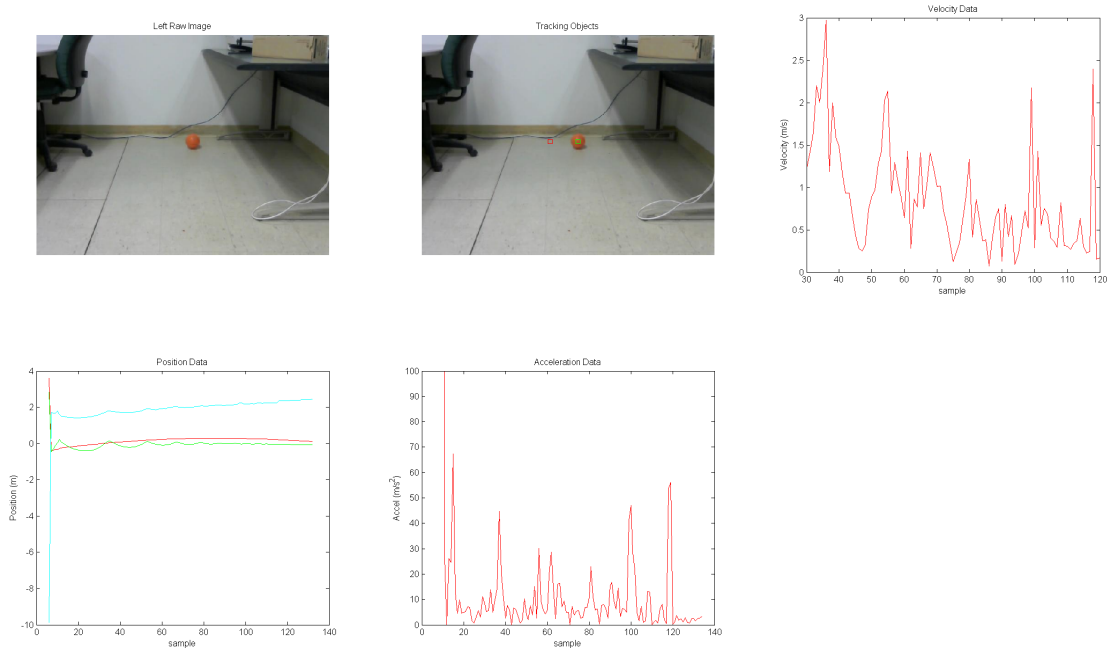

Fig. 6. *Tracking a Ballistic Object*

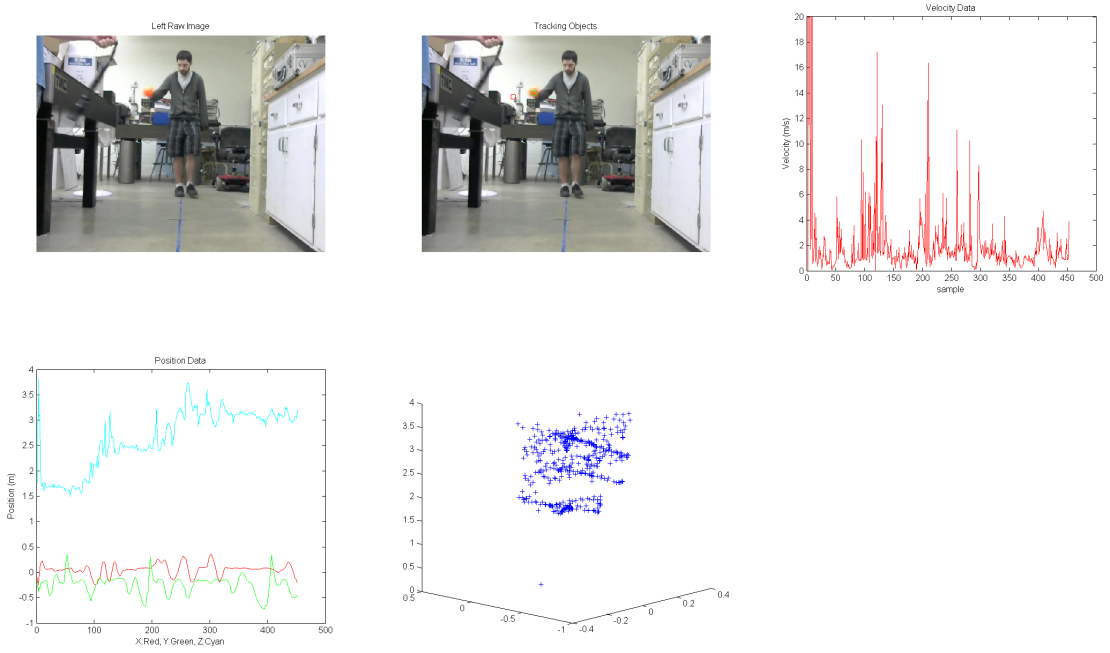

Fig. 7. *Tracking a ball against a non-ballistic trajectory*

APPENDIX
COMPLETE PROJECT CODE

```
ffmpeg -f v4l2 -r 25 -s 640x480 -i /dev/video0 -f v4l2 -r 25 -s 640x480 -i /dev/video1 -
    filter_complex "nullsrc=size=1280x480 [base]; [0:v] setpts=PTS-STARTPTS, scale=640x480 [
    left]; [1:v] setpts=PTS-STARTPTS, scale=640x480 [right]; [base][left] overlay=shortest=1 [
    tmp1]; [tmp1][right] overlay=shortest=1:x=640" -c:v libx264 output.mp4
```

Listing 3. Bash Camera Capture Code

```
clear all, close all, clc;

workingDir = 'C:\Users\me\Desktop\zzzGOODATA\';
fileNAME   = 'bounce.mp4';
cd(workingDir)
vid = VideoReader(fileNAME);

%convert each frame into an imag
for ii = 1:vid.NumberOfFrames
    img = read(vid,ii);
    %resolution decided to maximize frames per second
    %crop each image to reclaim original frames
    Left  = img([1:480],[1:640],:);
  Right = img([1:480],[641:1280],:);
    imwrite(Left,fullfile(workingDir, 'bounce', sprintf('left%03d.jpg',ii)));
    imwrite(Right,fullfile(workingDir, 'bounce', sprintf('right%03d.jpg',ii)));
    imwrite(img,fullfile(workingDir, 'bounce', sprintf('img%03d.jpg',ii)));
end
```

Listing 4. MATLAB Frame Splitting Code

```
clear all; close all; clc;

%set(0,'DefaultFigureWindowStyle','docked');

base_dir = 'C:\Users\me\Desktop\zzzGOODATA\calibration\cal_may8';
cd(base_dir);
```

```
imageLeft=imread('video0_calib_0.jpg');
imageRight=imread('video1_calib_0.jpg');

figure('name','Left Image');
imshow(imageLeft)

figure('name','Right Image');
imshow(imageRight)


close all
% values gathered manually by inspection:
% array is in form (r,c)
LEFTCAM=[181,294;
    219,295;
    261,295;
    300,296;
    344,297;
    385,297;
    431,298;
    475,299;

    180,253;
    221,253;
    260,253;
    302,253;
    343,253;
    387,253;
    430,253;
    477,253;

    182,211;
    220,211;
    262,210;
    301,210;
    344,210;
    387,210;
    431,209;
    476,208;

    182,170;
    221,170;
    261,169;
    302,168;
    344,167;
    387,166;
    431,165;
    477,163;]

RIGHTCAM=[101,302;
    138,302;
    177,303;
    216,303;
    256,304;
    297,304;
    340,305;
    383,305;

    102,261;
    139,261;
    177,260;
    217,260;
    256,260;
    298,260;
    340,260;
    384,260;

    103,219;
```

```
    140,219;
    178,219;
    217,218;
    257,217;
    298,216;
    341,215;
    384,214;

    103,179;
    141,178;
    178,177;
    217,176;
    257,174;
    298,173;
    340,171;
    384,169;]
figure('name','r,c plane');
plot(LEFTCAM(:,1),LEFTCAM(:,2),'s'), title('r,c plane');

%close all;
Re=zeros(32,3);
square=25.87;
y=-21.53-square;

for k=1:4
    x=-103.48;
    z=508;
    y=y+square;

    for n=1:8
        z=z-(55.56/232.13);
        x=x+square;
            Re(8*(k-1)+n,1)=x;
            Re(8*(k-1)+n,2)=y;
            Re(8*(k-1)+n,3)=z;
    end
end
figure('name','real world');
plot3(Re(:,1),Re(:,2),Re(:,3),'s')


%Ax=0;

close all;
rl=[(LEFTCAM(:,1).*Re(:,3)),-Re(:,3),-Re(:,1)];
cl=[(LEFTCAM(:,2).*Re(:,3)),-Re(:,3),-Re(:,2)];
A=[rl];
[U,D,V]=svd(A);
Xleft_x=V(:,end);

A=[cl];
[U,D,V]=svd(A);
Xleft_y=V(:,end);

rr=[(RIGHTCAM(:,1).*Re(:,3)),-Re(:,3),-Re(:,1)];
cr=[(RIGHTCAM(:,2).*Re(:,3)),-Re(:,3),-Re(:,2)];
A=[rr];
[U,D,V]=svd(A);
Xright_x=V(:,end)

A=[cr];
[U,D,V]=svd(A);
Xright_y=V(:,end)


% %% AX=B
% close all;
% Br=LEFTCAM(:,1).*Re(:,3); % r*Z
% Bc=LEFTCAM(:,2).*Re(:,3); % c*Z
```

```
% A1=[Re(:,3),Re(:,1)];
% A2=[Re(:,3),Re(:,2)];
%
% X1=(inv(A1'*A1)*A1'*Br)
% X2=(inv(A2'*A2)*A2'*Bc)
```

Listing 5. MATLAB Calibration Code

```
Xright_x =

    0.0012
    0.2680
    0.9634


Xright_y =

    0.0011
    0.3025
   -0.9532
```

```
clear all; close all; clc;

set(0,'DefaultFigureWindowStyle','docked');

%Curtis' directory
base_dir = 'C:\Users\me\Desktop\zzzGOODATA\bounce';

%David's directory
% base_dir = 'C:\Users\David\Desktop\SHAREDPROJECT\zzzGOODATA\bounce';
cd(base_dir);

listLeft =  dir('left*');
listRight= dir('right*');

close all;
figure('name','colorspace');
i=430;
for i=460:2:500%length(listLeft)

    Im_raw=(imread(listLeft(i).name));
    Im=rgb2ycbcr(imread(listLeft(i).name));
    Im_cr=Im(:,:,3);


    subplot(231);
    imshow(Im_raw);
    title('raw image');

    subplot(232);
    imshow(Im);
    title('YCbCr');

    subplot(233)
    imshow(Im_cr);
    title('Chroma');



    S=strel('disk',9); %bounce
    subplot(234)
    Bw=im2bw(Im_cr,0.57);
    imshow(Bw);
    title('im2bw');



    subplot(235)
```

```
    Bw=imerode(Bw,S);
    Bw=imdilate(Bw,S);

    imshow(Bw);
    title('Isolated object');

    stats=regionprops(Bw);
    index=[0 0];
    totArea=0;
    for j=1:length(stats)
        totArea=totArea+stats(j).Area;
    end

    for j=1:length(stats)
        index=index+stats(j).Centroid*(stats(j).Area/totArea);
    end

    subplot(236)
    imshow(Im_raw), title('Tracked Object'); hold on;
    plot(index(1), index(2), 's', 'color', 'cyan'); hold off;

    pause(0.01);
end
```

Listing 6. MATLAB Object Isolation Code

```
clear all; close all; clc;

set(0,'DefaultFigureWindowStyle','docked');

%Curtis' directory
base_dir = 'C:\Users\me\Desktop\zzzGOODATA\bounce';

%David's directory
% base_dir = 'C:\Users\David\Desktop\SHAREDPROJECT\zzzGOODATA\bounce';
cd(base_dir);

listLeft =  dir('left*');
listRight= dir('right*');

close all
figure('name','thresholded data')
threeDtracker=[0,0,0];
threeDVel=[0,0,0];
threeDAccel=[0,0,0];
threshR=0.57; %bounce
threshL=0.57; %bounce
% threshL=0.3; % testDATA
% threshR=0.3; % testDATA
prevP=[0,0,0];
prevV=0;
S=strel('disk',9);
for i=460:1:length(listLeft)
%for i=490:1:(490+50)%length(listLeft)
%for i=610:(610+450)%length(listLeft)
    I_l=imread(listLeft(i).name);
    Image_left=rgb2ycbcr(I_l);
    Image_left_cr=Image_left(:,:,3);

    I_r=imread(listRight(i).name);
    Image_right=rgb2ycbcr(I_r);
    Image_right_cr=Image_right(:,:,3);

    Bw_left=im2bw(Image_left_cr,threshL);
    Bw_right=im2bw(Image_right_cr,threshR);

    Bw_left=imerode(Bw_left,S);
    Bw_left=imdilate(Bw_left,S);
    Bw_right=imerode(Bw_right,S);
```

```
    Bw_right=imdilate(Bw_right,S);

    %track LEFT
    stats=regionprops(Bw_left);
    index_left=[0 0];
    totArea=0;
    for j=1:length(stats)
        totArea=totArea+stats(j).Area;
    end

    for j=1:length(stats)
        index_left=index_left+stats(j).Centroid*(stats(j).Area/totArea);
    end

    stats=regionprops(Bw_right);
    index_right=[0 0];
    totArea=0;
    for j=1:length(stats)
        totArea=totArea+stats(j).Area;
    end

    for j=1:length(stats)
        index_right=index_right+stats(j).Centroid*(stats(j).Area/totArea);
    end

    % LEFT IMAGE IS CHOSEN BECAUSE IT IS THE ORIGIN
    subplot(231)
    imshow(I_l); title('Left Raw Image');


    subplot(232)
    imshow(I_l), title('Tracking Objects'); hold on;
    plot(index_left(1), index_left(2), 's', 'color', 'green'); hold on;
    plot(index_right(1), index_right(2), 's', 'color', 'red'); hold off;
    [p, v, a, prevP, prevV] = getPhysics(index_right(1), index_right(1),index_left(1),
    index_left(2),prevP,prevV);

    subplot(233)
    threeDVel=[threeDVel,v];
    %plot3(threeDtracker(:,1),threeDtracker(:,2),threeDtracker(:,3));
    % X plot
    plot(threeDVel(2:end),'-','Color','r'); hold on;
%     plot(threeDVel(2:end,2),'-','Color','g'); hold on;
%     plot(threeDVel(2:end,3),'-','Color','c'); hold on;
    title('Velocity Data')
    xlabel('sample')
    ylabel('Velocity (m/s)')
    xlim([30,120]);
    hold off;

    subplot(234)
    threeDtracker=[threeDtracker;p];
    %plot3(threeDtracker(:,1),threeDtracker(:,2),threeDtracker(:,3));
    % X plot
    plot(threeDtracker(2:end,1),'-','Color','r'); hold on;
    plot(threeDtracker(2:end,2),'-','Color','g'); hold on;
    plot(threeDtracker(2:end,3),'-','Color','c'); hold on;
    title('Position Data')
    xlabel('sample')
    ylabel('Position (m)')
%     xlim([10,30]);
    %legend('X','Y','Z');
    hold off;

    subplot(235)
    threeDAccel=[threeDAccel,a];
    %plot3(threeDtracker(:,1),threeDtracker(:,2),threeDtracker(:,3));
    % X plot
    plot(threeDAccel(2:end),'-','Color','r'); hold on;
```

```
    title('Acceleration Data')
    xlabel('sample')
    ylabel('Accel (m/s^2)')
%     xlim([10,30]);
    ylim([0,100]);
    hold off;


    pause(0.01);
end
```

Listing 7. MATLAB Ball Bounce Tracking Demo

```
clear all; close all; clc;

set(0,'DefaultFigureWindowStyle','docked');

%Curtis' directory
base_dir = 'C:\Users\me\Desktop\zzzGOODATA\lastTest';

%David's directory
% base_dir = 'C:\Users\David\Desktop\SHAREDPROJECT\zzzGOODATA\lastTest';
cd(base_dir);


listLeft =  dir('left*');
listRight= dir('right*');


close all
figure('name','thresholded data')
threeDtracker=[0,0,0];
threeDVel=[0,0,0];
threeDAccel=[0,0,0];
threshR=0.57; %bounce
threshL=0.57; %bounce

prevP=[0,0,0];
prevV=0;
S=strel('disk',9);

for i=610:1:(610+450)%length(listLeft)
    I_l=imread(listLeft(i).name);
    Image_left=rgb2ycbcr(I_l);
    Image_left_cr=Image_left(:,:,3);

    I_r=imread(listRight(i).name);
    Image_right=rgb2ycbcr(I_r);
    Image_right_cr=Image_right(:,:,3);

    Bw_left=im2bw(Image_left_cr,threshL);
    Bw_right=im2bw(Image_right_cr,threshR);

    Bw_left=imerode(Bw_left,S);
    Bw_left=imdilate(Bw_left,S);
    Bw_right=imerode(Bw_right,S);
    Bw_right=imdilate(Bw_right,S);

    %track LEFT
    stats=regionprops(Bw_left);
    index_left=[0 0];
    totArea=0;
    for j=1:length(stats)
        totArea=totArea+stats(j).Area;
    end

    for j=1:length(stats)
        index_left=index_left+stats(j).Centroid*(stats(j).Area/totArea);
    end
```

```
    stats=regionprops(Bw_right);
    index_right=[0 0];
    totArea=0;
    for j=1:length(stats)
        totArea=totArea+stats(j).Area;
    end

    for j=1:length(stats)
        index_right=index_right+stats(j).Centroid*(stats(j).Area/totArea);
    end

    % LEFT IMAGE IS CHOSEN BECAUSE IT IS THE ORIGIN
    subplot(231)
    imshow(I_l); title('Left Raw Image');


    subplot(232)
    imshow(I_l), title('Tracking Objects'); hold on;
    plot(index_left(1), index_left(2), 's', 'color', 'green'); hold on;
    plot(index_right(1), index_right(2), 's', 'color', 'red'); hold off;
    [p, v, a, prevP, prevV] = getPhysics(index_right(1), index_right(1),index_left(1),
    index_left(2),prevP,prevV);

    subplot(233)
    threeDVel=[threeDVel,v];
    %plot3(threeDtracker(:,1),threeDtracker(:,2),threeDtracker(:,3));
    % X plot
    plot(threeDVel(2:end),'-','Color','r'); hold on;
%     plot(threeDVel(2:end,2),'-','Color','g'); hold on;
%     plot(threeDVel(2:end,3),'-','Color','c'); hold on;
    title('Velocity Data')
    xlabel('sample')
    ylabel('Velocity (m/s)')
    ylim([0,20]);
%     xlim([30,120]);
    hold off;

    subplot(234)
    threeDtracker=[threeDtracker;p];
    %plot3(threeDtracker(:,1),threeDtracker(:,2),threeDtracker(:,3));
    % X plot
    plot(threeDtracker(2:end,1),'-','Color','r'); hold on;
    plot(threeDtracker(2:end,2),'-','Color','g'); hold on;
    plot(threeDtracker(2:end,3),'-','Color','c'); hold on;
    title('Position Data')
    xlabel('X:Red, Y:Green, Z:Cyan')
    ylabel('Position (m)')
%     xlim([10,30]);
    %legend('X','Y','Z');
    hold off;

    subplot(235)
    threeDAccel=[threeDAccel,a];
    plot3(threeDtracker(:,1),threeDtracker(:,2),threeDtracker(:,3),'+');
    % X plot
%     plot(threeDAccel(2:end),'-','Color','r'); hold on;
%     title('Acceleration Data')
%     xlabel('sample')
%     ylabel('Accel (m/s^2)')
% %     xlim([10,30]);
%     ylim([0,100]);
%     hold off;


    pause(0.01);
end
```

Listing 8. MATLAB Ball Hold Tracking Demo

```
function [p, v, a, prevP, prevV] = getPhysics(Rr, Cr, Rl, Cl, prevPos, prevVel);
    %knowns (from calibration)
    lambda=1e-3;
    Sx=1.2e-6;
    Sy=1.1e-6;
    u0=303;
    v0=260;
    %baseline=76e-3; %mm
    baseline=177.8e-3; %7 inches ish
    framerate=25;
    sampleTime=1/framerate;

    %triangulations
    Ul=(Rl-u0)*Sx;
    Ur=(Rr-u0)*Sx;
    Vl=(Cl-v0)*Sy;
    %Vr=(Cr-v0)*Sy;


    X=(baseline*Ul)/(Ul-Ur);
    Y=(baseline*Vl)/(Ul-Ur);
    Z=(lambda*baseline)/(Ul-Ur);


    p=zeros(1,3);
    v=zeros(1,3);
    a=zeros(1,3);

    dX=(X-prevPos(1))/sampleTime;
    dY=(Y-prevPos(2))/sampleTime;
    dZ=(Z-prevPos(3))/sampleTime;
    p=[X,Y,Z];

    %only measures magnitude of velocity, estimated using Euler's
    %approximation.
    v=sqrt((X-prevPos(1))^2+(Y-prevPos(2))^2+(Z-prevPos(3))^2)/sampleTime;

    %only measures magnitude of acceleration, estimated using Euler's
    %approximation.
    a=abs((v-prevVel)/sampleTime);


    prevP=p;
    prevV=v;

end
```

Listing 9. MATLAB custom getPhysics function