Image Analysis and Computer Vision

Automatic Trajectory, Ball Spin and Velocity Tracking Applied to Bowling

From single view stationary video

Authors:
Aria Alinejad, Thomas Bruflot

February, 2023

POLITECNICO
MILANO 1863

Abstract

The automatic detection of different performance parameters in sports has been found useful for athletes, trainers and supporters of different sports. These techniques are now widely used in sports like football, tennis and golf. This paper will focus on the detection of performance parameters in bowling through the use of computer vision. We will use established computer vision techniques to detect the spin vector, velocity vector and trajectory of a bowling ball using a single view stationary video of the bowling lane. The aim is to provide an easy to use and accurate system that bowlers can use to gain insight into, and improve their performance. The system can also be used to serve the dual purpose of providing a better viewing experience for supporters.

Table of Contents

1 Introduction

Given a video of a bowling game our implementation aims at providing useful information to the player and the viewers about each delivery. The goal is to provide as accurate as possible information about the spin, velocity and trajectory of the ball at a processing speed that is as low as possible. These three factors give the most important information about each delivery. We believe that this information can help players improve their game, and give viewers more insight into what goes into each delivery.

The information is to be retrieved using computer vision techniques on a stationary single view video, of high resolution and frame rate. The high resolution and frame rate is a requirement to be able to accurately analyze the video. The desired output of the system is a real time assessment of the described performance parameters, and a trajectory of each delivery seen from above (with correct proportions), after each delivery.

2 Implementation

In this section we will discuss the methods and tools used to arrive at the final solution. In addition to the camera setup used.

The whole system is implemented in Matlab using the Computer Vision Toolbox.

2.1 The scene

We have analyzed several different videos of bowling play. A snapshot of one of these videoes are shown in figure 1.

Figure 1: Snapshot from a video of bowling play.

The video is captured with a stationary camera that is placed with an angle in the center of the bowling lane behind the first line.

As mentioned it is desired to have as high resolution and and frame rate as possible. It is also desired to have the camera placed as high as possible and preferably facing straight towards the lane. Although both these perturbations are accounted for in the implementation a very low angle or very shifted view can cause worse results. In addition to this it is a requirement that the line present at the start of the lane is visible and that the lighting conditions are satisfactory. Lastly the camera should be as stable as possible as major movements in the video causes increased error in the results.

2.2 Ball detection

To detect the ball we use Hough Circle transform (CHT) on each frame of the video. We use the CHT based algorithm because it is fairly robust to noise, occlusion and varying illumination settings. The algorithm first finds foreground pixels with high gradient as candidate pixels which can be voted on at a later stage. Then circles are fitted as shown in figure 2 and by seeing where the circles intersect we find the maxima which is the circle center. The radius is then estimated from the candidate pixels and the circle center.

Since the radius of the ball becomes smaller for each frame we have to check for a range of different radii. This is one of the main factors that slows down the processing of each frame. Since we know that the radius of the circle can't change too much between frames we choose the circle that is most similar to the last one found as the output circle. To get improved performance several different

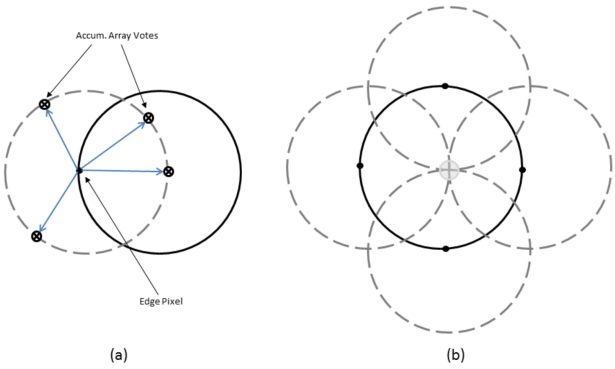

Figure 2: CHT algorithm where dashed lines are part of the voting pattern.

image processing techniques were tested. We found that using adaptive histogram equalized gives the best results.

2.3 Velocity vector

To find the direction and magnitude of the velocity of the ball, we use an Optical flow algorithm proposed by Gunnar Farneback. The algorithm was presented in the paper "Two-Frame Motion Estimation Based on Polynomial Expansion" [1]. The core idea of optical flow analysis is to find the rate of change from one image to another as shown in figure 3. The vectors are pointing in the direction of change. The more the image points move per frame the larger magnitude the vectors will have. The Farneback approach approximates neighbourhoods in frames by quadratic polynomials with polynomial expansion transform and then uses it to find displacements fields. The algorithm finds this displacement field between two successive frames.

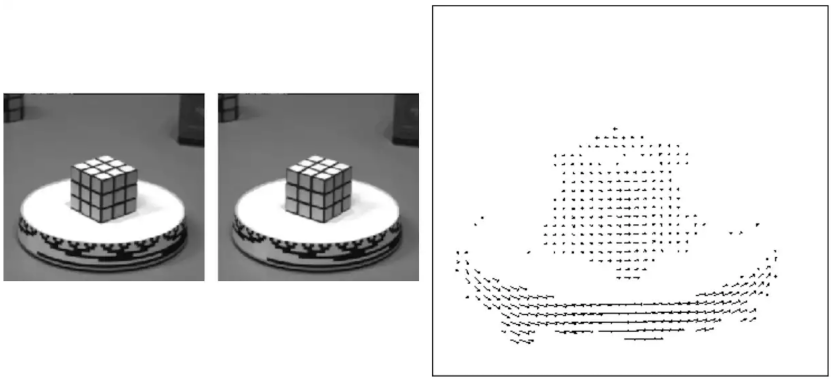

Figure 3: Optical flow core idea visualized.

The algorithm is implemented in The Computer Vision Toolbox of Matlab.

We look at the image frame by frame and analyze the part of the image where there is a lane using Optical flow. We then select the most relevant part of the flow, which is the flow found where the bowling ball is present on the image. To find the mask used for selecting the most relevant flow we use the ball's previously detected center and radius.

When we have selected the relevant velocity vectors for pixels inside the ball radius we find the average of the vectors and output the resulting vector on the image. We also present the magnitude

and direction of the vector in text form. The velocity in m/s v is found using the following equation:

$$v = pixels/frame \cdot framerate \cdot \frac{r_{std}}{r_{pix}} \tag{1}$$

Where r_{pix} is the radius of the ball in pixels, and $r_{std} = 0.12$ m is the standard radius of a bowling ball. By using r_{std}/r_{pix} we find the mapping between pixel distance and distance in the real world in that single frame, thus accounting for the fact that the ball progressively gets smaller.

To account for the fact that we are seeing the lane from an angle and thus finding the velocity along this projective mapping of the real lane we look at the angle at which the parallel lines on the side of the court are projected in the image. We do this since this angle is proportional to the shortened lane length present in the image. But it does not in fact account for the actual error, it is merely an approximation that together with accounting for the ball size gives good results. The parallel lines will be as shown in figure 4.

To further get the velocity along the bowling lane and not along a projected court we can approximate the angle at which the parallel lines on the side of the court are projected in the image. The parallel lines will be as shown in figure 4.

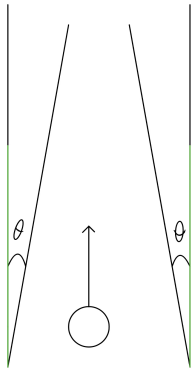

Figure 4: Court and the approximation of velocity along the bowling lane.

To get the velocity along the green lines in the image you divide the velocity by $cos(\theta)$. In our case we approximated θ to be 75 degrees. Another maybe more precise way to find the velocity along the lane is to calculate a metric rectification of the image and find the lengths in the resulting new image compared to real worlds lengths and then estimate velocity from that instead.

is to use the image that is the similaryty mapping of the real scene to find the ratio of lengths we need to find the real distance traveled by the ball. This will in fact also be an approximation, but maybe a more precise one.

2.4 Spin vector

To find the spin of the ball we used the Optical flow Farneback algorithm implemented using the Computer Vision Toolbox in Matlab. We extract a square image around the position of the ball for each frame of the video. The position of the center of the square is set to the position of the ball center found by the CHT algorithm. The size of the square is set to the radius of the first detected ball throughout the sequence. This is because the Optical flow needs images of the same size throughout the sequence.

When computing the flow we keep the values only in the positions where a ball is detected, as before. The average magnitude of vectors is again computed and displayed on the frame. The RPM

and direction is also shown in text format. To calculate the RPM the following equation is used:

$$RPM = \frac{60 \cdot v_s}{r_{std} \cdot 2\pi} \tag{2}$$

Where the v_s is the spin velocity given in m/s, and we multiply by 60 to go from seconds to minutes.

2.5 Trajectory

The trajectory is found using the collection of center positions for the detected balls found throughout the video. In cases where no circle is found in a frame the next trajectory point set to the same as the previous one. This trajectory is then transformed using a projective transformation, and shown along side the same transformation performed on the sidelines of the lane. This then results in the trajectory as seen from above with correct ratios. The output is an approximation of a similariy transformation of the real scene.

The homography H is found using sets of two and two lines known to be have a right angle real world. Since the bowling lane is know to posses such orthogonal lines, these can be used. We know that each lane possesses at least 8 independent right angles, as shown in figure 5. These points are given by the user.

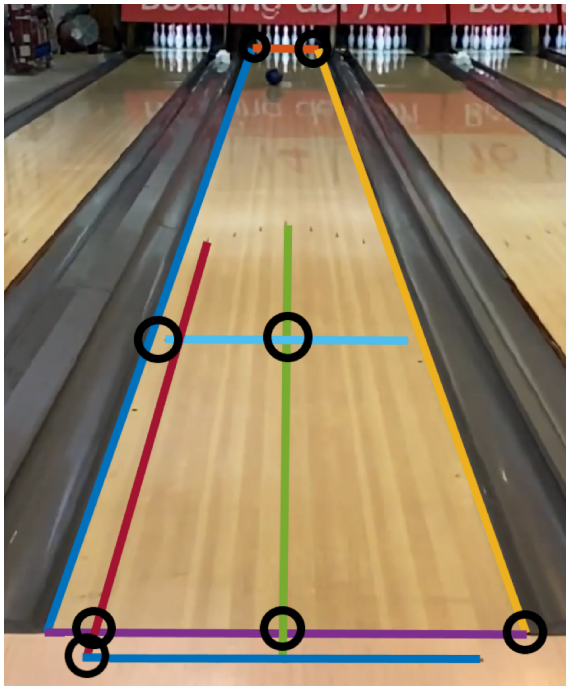

Figure 5: Lines known to be orthogonal in the real scene.

To find the values in H we first find the image of the dual conic $C^{*'}_{\infty}$. We need 4 constraints as there are 4 degrees of freedom in a similarity projective transformation. In principle lines l', m' which are orthogonal in the image provide an equation that can be a constraint on $C^{*'}_{\infty}$ given by $lC^{*'}_{\infty}m' = 0$. If we get multiple of these equations we get a linear system $Ac = 0$ which is solved by $c = RNS(A)$. Then H is calculated by $HC^{*}_{\infty}H^T = C^{*'}_{\infty}$ by doing singular value decomposition.

2.6 Other features

We give the user the opportunity to crop the video frame at the start, so that the frame only includes the most relevant parts of the image frame, see figure 6. Then the user is prompted to mark the 4 edge points of the lane. The area of the lane is then used as a mask so that all processing is only done where there is a lane present, see figure 7. This drastically increased speed and also accuracy, especially because of the HCT.

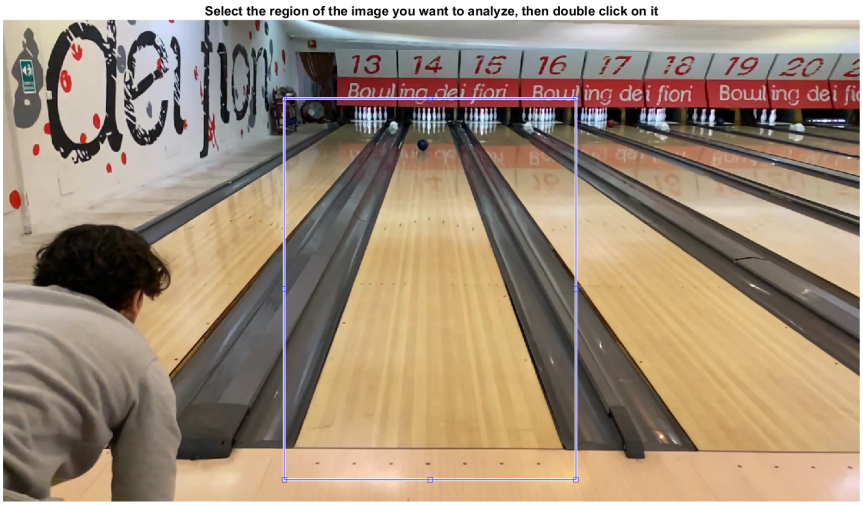

Figure 6: GUI for cropping the frame.

Figure 7: The region of the video used for all processing.

We also take the processed frames with all the information on them and output a video containing all the desired information of the delivery.

2.7 Remarks

We decided to use slow motion footage to get more frames per second and in turn get more accurate spin estimates as the pixel differences from one image to another is smaller with a higher fps. Then we multiplied resulting numbers for velocity and spin with the slow motion factor which for us was $\frac{240}{30} = 8$. The flow motion factor is the factor at which the frame rate is artificially brought down.

3 Results

The video used was captured using a single hand held camera recording at 4k resolution and 240 frames per second. The captured video can be approximated as stationary. The video filmed a single lane head on, at an angle close to the ground (at human height). The ball used was a dark blue ball with white text on one side of the ball. The player was also present in the video, and partly obstructed the view to the ball at the start.

We also tested the system on other videos and got similar results.

3.1 Ball detection

We found the Hough transform to perform quite well in detecting the correct center and radius of the ball in most cases. But it is worth noting that most tests were run on dark balls, and that light colored balls were harder to detect because of the light lane background. An image of the ball being detected with radius and center drawn onto the image is shown in figure 8.

With the video used the HCT gave very good results up until the end, at which points it made several errors. This is most likely because of the low resolution present at this point. The HCT was also able to detect the ball when it was partly obstructed, as seen in the image.

Figure 8: Ball detected using the circle hough transform on the frame.

3.2 Velocity vector

The obtained velocity vector of the bowling ball is shown in blue in figure 9. In the figure we can also see the calculated direction and magnitude. The velocity vector is pointing in a direction that corresponds well with the perceived direction from the video and also the trajectory found at a later stage. The magnitude on the other hand is a little less accurate, but still yields plausible values. Both are subject to noise and vary a little along the video.

3.3 Spin vector

Since the implemented calculation of the spin is highly dependent on the center position found by the HCT the results become inaccurate when the assumed center position is wrong. In addition to this, slight errors in center detection between frames also results in the Optical flow algorithm calculating the error as motion of the ball. Luckily the HCT is quite accurate throughout the video

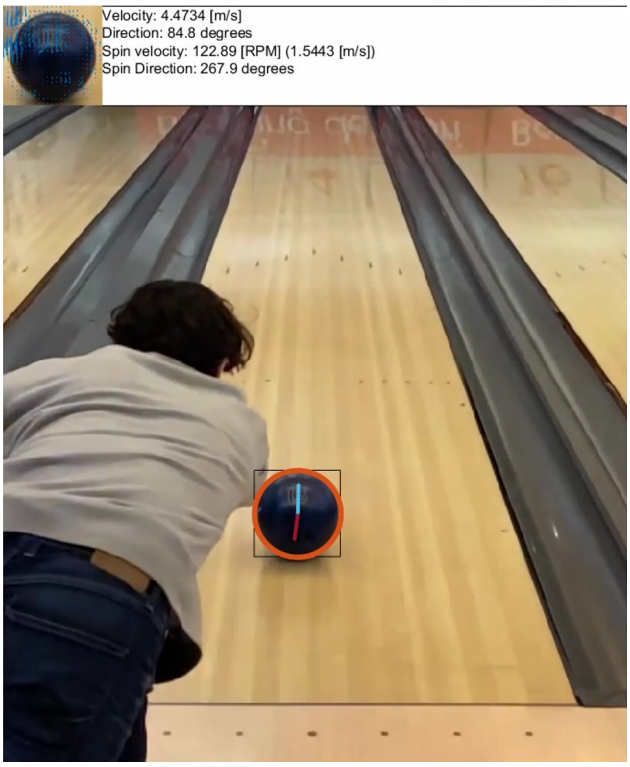

Figure 9: A single frame of the output video.

until the end. In addition to this the image of the ball gets progressively smaller throughout the video, it gets as little as 7 pixels at the end of the video we used for testing. This naturally means that the spin detected will get progressively more inaccurate as the ball moves down the lane. In the video used one can conclude that the spin values are meaningful until approximately half way down the lane. The results also show that the values are more accurate when the ball has easily distinguishable patterns on it. The algorithm finds more spin when the holes of the ball or the white text on the ball are visible. When a part of the ball with no pattern is the only part visible the algorithm sees this as no spin.

The acquired results can be seen in figure 9. Here we can see the spin vector in red and the output values at the top. We can also see the region at which the algorithm looks when calculating the spin flow. The acquired spin flow is also printed on top of this image.

3.4 Trajectory

As one might expect the trajectory found is accurate as long as the centers found by the Hough transform are correct. This means that the trajectory is accurate until the last part when the resolution is low. The acquired results are shown in figure 10.

From the figure we see that the results deviate at he very end. Figure 11 shows the transformed lane and trajectory, side by side with the original lane and trajectory. This gives us a better sense of the trajectory of the ball in the real world. As we can see the untransformed lane at the left is much more compressed at the end. We can see that the four lines of the transformed lane are almost perpendicular to each other, indicating that the transformation is quite accurate even though the original angle was quite low.

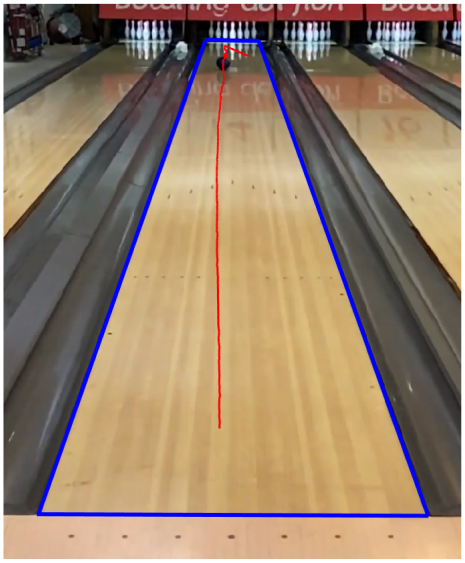

Figure 10: Obtained trajectory.

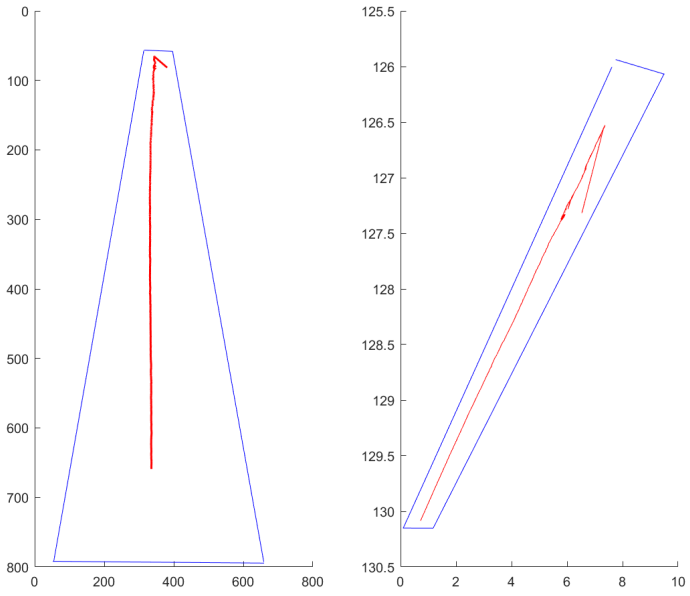

Figure 11: Transformed trajectories. The one on the left is untransformed, while the one on the right is transformed.

4 Discussion

4.1 Ball detection

As one might imagine there are many possible ways to improve the detection of the circles. There is room for improvement both in the accuracy and speed of the detection found using solely HCT.

Neural Network: It is possible to use a trained Neural Network to detect the circle positions and their radius. This can be used in addition to the HCT to speed up the process, allowing for testing fewer more relevant circle centers and radii. Depending of the accuracy of the network it can also provide more accurate results.

One could use a segmentation network (eg. something resembling U-net) to segment the ball form the background thus acquiring the pixels belonging to the ball. From this we can use morphological operators to improve the solution and then find a more accurate center and radius.

It is also possible to use a detection network (eg. YOLO) to detect the bounding box around the image. After attaining the boxes we can perform the HCT, on the image contained in the box, and only on radii smaller then the bounding box size and not bigger then what is reasonable. This would give significantly faster processing time and more accurate results.

Since acquiring the dataset required for training a network can pose challenging the most relevant alternative is either to find an already trained network on the correct data or to find a similar enough pretrained network and use transfer learning.

Foreground Detection: It is also possible to use the ForegroundDetector already present in the Computer vision toolbox of Matlab. When testing, this had worse performance then the HCT method, but presented much higher speeds of detection. Thus this can be used together with the HCT. We can look for circles solely in the parts of the image where there has been detected movement. One can also look at the parts of the image with both circles and detection, find the radii of the objects detected by the foreground detector, and then only check the most relevant radii and centers with the HCT. This would again speed up the process and give more accurate results.

4.2 Velocity vector

The magnitude of the found velocity is a little lower then what would be expected. This is, at least in part, due to the angle at which the video is shot. Since the camera does not see the ball from above, but rather from close to the ground every pixel moved on the image represents several more pixels moved seen from a bird angle. Thus the calculated velocity in meters per second is lower then the actual velocity. This has been partly accounted for, but it still has an affect.

4.3 Spin vector

As mentioned, the spin vector is often quite inaccurate. Part of this comes from the low resolution and frame rate of the image as it is moving fast and far away form the camera. Focus is also an important factor as the ball most often is not in focus throughout the whole motion.

We know that the ball size decreases with respect to the region we are looking at for each consecutive frame. Although this effect is present we have not found it to have significant effect on the calculated spin, as long as the frame-rate of the video is sufficiently high. Furthermore the color of the ball was also found to be significant as a lighter color makes the holes for the fingers more clear, thus giving easier detection of spinning motion. As mentioned, the center detection is also very important for accurate results using our method. Methods of improving this is discussed above.

An error source was the light in the ceiling in the bowling hall used for filming. Due to flickering from the light the image had very different brightness levels on the surface of the ball for different frames. This an issue because the optical flow algorithm is related to the rate of change of color values on the surface of the ball.

Most of all the accuracy of the spin found is dependent on the resolution, frame rate and color difference on the ball. Three factors we can't do too much about in post processing.

4.4 Trajectory

When computing H we used 8 angles known to be perpendicular in the real scene. The more angles used and the more carefully selected they are the more accurate is the computed H. Since we want the system to be generally implementable we have only used lines present on any bowling lane, but the user can also use other lines, eg. lines on neighboring lanes, to get a more accurate transformation.

5 Conclusion

Using a single view stationary video of a bowling lane our implementation finds the ball velocity vector, ball spin vector and the trajectory as seen from above. This is done with the use of Hough circle transform, Farneback Optical flow and Projective transformations.

The acquired results are best for the first frames and deviate more as the ball's distance from the camera increases. The results are good enough to give an indication of the velocity, spin and trajectory at the start. This can be useful for finding patterns in a players game. The results are not stable or accurate enough in order to use the acquired values as the true values.

The results are most accurate using a dark ball with distinct pattern, shot at high frame rate and resolution.

References

[1] Gunnar Farneback. Two-frame motion estimation based on polynomial expansion. https://link.springer.com/chapter/10.1007/3-540-45103-X_50?awc=26429_1675857263_07b2579a863a24b2df8c78b6d336b8e0&utm_medium=affiliate&utm_source=awin&utm_campaign=CONR_BOOKS_ECOM_DE_PHSS_ALWYS_DEEPLINK&utm_content=textlink&utm_term=101248, 2003.

Appendix

A Matlab Implementation

```
clc;close all;clear all;
% Access video file
%v = VideoReader('Videos/IMG_0923.MOV');
v = VideoReader('Videos/Aria_1.MOV');

% Choose frame of video
im = read(v,v.NumFrames/2);
imshow(im)
title('Select the region of the image you want to analyze, then double click on
    it');
hold on
[J, rect] = imcrop(im);
rect = floor(rect);
hold off
%------Load values during testing to save time----
%rect = load('R/vars.mat', 'rect').rect;
%J = im(rect(2):rect(2)+rect(4), rect(1):rect(1)+rect(3), :);
%-------------------------------------------------------------------

imshow(J);

%---------Select region we are interested in (the lane)----------
title('Select the four angles of the lane, then press enter');
[xi, yi] = getpts;
%-----Load values during testing to save time----
%yi = load('R/vars.mat', 'yi').yi;
%xi = load('R/vars.mat', 'xi').xi;
%-------------------------------------------------------------------
hold on

lane_x = [xi(1), xi(2), xi(3), xi(4)];
lane_y = [yi(1), yi(2), yi(3), yi(4)];

U = roipoly(J,lane_x,lane_y);
U = bwareafilt(U,1);
U = bwconvhull(U);
croppedImg = immultiply(J,repmat(U,[1 1 3]));
imshow(croppedImg);

%%

% Define figure and starting frame
h = figure();
movegui(h);
im = read(v,1);

% Preallocate structure to store video frames
% Note that this is just an initial guess for preallocation based on
% duration and framerate, the video may have fewer or more frames
nFrames = ceil(v.FrameRate*v.Duration);
s(nFrames) = struct('cdata',[],'colormap',[]);


% Loop through video, grabbing frames and updating plots
k = 1;
iter = 0;
movVector = zeros(nFrames,3); % Vector storing centers and radii
rectBall = [0,0,0,0]; % rectangle plotted around ball
```

```
initSize = 0; % radius of the first detected all
% Set initial textbox values
v_ball_text = sprintf('Velocity: - [m/s]');
dir_ball_text = sprintf('Direction: - degrees');
v_spin_text = sprintf('Spin velocity: - [RPM] (- [m/s])');
dir_spin_text = sprintf('Spin Direction: - degrees');
% Define Optical flow variables
opticFlow_v = opticalFlowFarneback; % Optical flow for velocity
opticFlow_s = opticalFlowFarneback; % Optical flow for spin

% Parameters
MaxRadius = 60; % Found from testing, radius of biggest ball
MinRadius = 7; % Found from testing, radius of smallest ball
slowMotionFactor = 8; % Found from knowledge og original video frame rate

while hasFrame(v)
    reFrame = readFrame(v);
    iter = iter+1;
    im = reFrame(rect(2):rect(2)+rect(4), rect(1):rect(1)+rect(3), :);
    imshow(im)
    hold on

    % Only look at area of image where the lane is
    imCrop = immultiply(im,repmat(U,[1 1 3]));

    % Plot the previous spin and velocity data (so that the values will flicker less)
    rectangle('Position',[1+initSize*2,1,size(im,2),1+initSize*2], 'FaceColor', [1 1 1],'EdgeColor','k');
    text(1+initSize*2, 10, v_ball_text, 'FontSize', 8, 'Color', 'k')
    text(1+initSize*2, 30, dir_ball_text, 'FontSize', 8, 'Color', 'k')
    text(1+initSize*2, 50, v_spin_text, 'FontSize', 8, 'Color', 'k')
    text(1+initSize*2, 70, dir_spin_text, 'FontSize', 8, 'Color', 'k')

    % Use hough transform to find circles (darker then the background)
    % We use adapthisteq on the image for better results
    if (iter>1)
        [centers, radii, metric] = imfindcircles(adapthisteq(rgb2gray(imCrop)),[MinRadius MaxRadius], 'ObjectPolarity','dark');
        % Keep circle most similar to last circle
        [val, idx] = min(abs(radii-movVector(iter-1,3)));
        center = centers(idx,:);
        radius = radii(idx);
    else
        [centers, radii, metric] = imfindcircles(adapthisteq(rgb2gray(imCrop)),[MaxRadius*0.8 MaxRadius], 'ObjectPolarity','dark');
        % Keep the best circle
        [val, idx] = max(metric);
        center = centers(idx,:);
        radius = radii(idx);
    end


    % Estimate the flow on the image
    flow_v = estimateFlow(opticFlow_v,rgb2gray(imCrop));

    pause(10^-3)
    if(radius) % checks if we found a circle or not
        % record the movement of the center and radius
        movVector(iter,1) = center(1);
        movVector(iter,2) = center(2);
        movVector(iter,3) = radius;

```

```
        % Find the square used for estimating the spin flow
        if (initSize == 0)
            initSize=floor(radius);
        end

        % use try catch to prevent error when part of the ball is out of frame
        % imball is the image around the ball used for spin detection
        try
            imball = im(floor(center(2))-initSize:floor(center(2))+initSize,
    floor(center(1))-initSize:floor(center(1))+initSize, :);%im(floor(center(1)
    )-initSize:floor(center(1))+initSize, floor(center(2))-initSize:floor(
    center(2))+initSize, :);
            imshow(imball); % Show region used for spin flow in upper right

            % Find and plot the spin flow
            flow_s = estimateFlow(opticFlow_s,rgb2gray(imball));
            plot(flow_s,'DecimationFactor',[5 5],'ScaleFactor',2);
        catch
            flow_s = 0;
        end

        %Define the rectangle around the ball
        rectBall(1) = floor(center(1)-radius); %x init top left val
        rectBall(2) = floor(center(2)-radius); %y init val top left
        rectBall(3) = floor(2*radius); %how much to the right x will move
        rectBall(4) = floor(2*radius); %How much down the y will move
        %The above code creates a rectangle around the ball with r length
        %from center at all sides. so really it is a square.
        rectangle('Position', rectBall); %plots the rectangle


        %Make a binary mask of the ball and take out indecies
        mask = zeros([size(im,1), size(im,2)]);
        try
            for r=1:radius
                th = 0:pi/50:2*pi;
                xunit = r * cos(th) + center(1);
                yunit = r * sin(th) + center(2);

                for i=1:length(xunit)
                    mask(round(yunit(i)),round(xunit(i))) = 1;
                end
                SE = [1 1 1; 1 1 1; 1 1 1];
                mask = imclose(mask, SE);
                [x_idx, y_idx] = find(mask == 1);
            end
        catch
            mask = ones(size(im));
        end

        %--------Find avg velocity vector--------------
        % Only use the flow present in the ball
        % try catch to prevent error when part of ball if out of frame
        try
            Vx_v = flow_v.Vx(x_idx', y_idx');
            Vy_v = flow_v.Vy(x_idx', y_idx');
        catch % if part of the ball is outside frame, simply use all values
            Vx_v = flow_v.Vx;
            Vy_v = flow_v.Vy;
        end


        % We find the x and y components of the average vectors
```

```
        avg_vx = mean2(Vx_v);
        avg_vy = mean2(Vy_v);

        % Find the direction and magnitude of the average vector
        averageMagnitude_v = sqrt(avg_vx^2 + avg_vy^2);

        % Change axis so that they are as we are used to
        if(avg_vx>0 && avg_vy>0)
            averageOrientation_v =  atan(avg_vy/avg_vx);
        elseif (avg_vx<0 && avg_vy>0)
            averageOrientation_v =  pi - atan(avg_vy/(-avg_vx));
        elseif (avg_vx<0 && avg_vy<0)
            averageOrientation_v =   pi + atan((-avg_vy)/(-avg_vx));
        elseif (avg_vx>0 && avg_vy<0)
            averageOrientation_v =  2*pi - atan((-avg_vy)/(avg_vx));
        end
        averageOrientation_v = 2*pi - averageOrientation_v;


        % -------Calculate the velocity and direction of the ball--------
        v_ball= ((averageMagnitude_v*v.FrameRate*0.12)/radius)*slowMotionFactor
    ;
        v_ball = v_ball/0.2588190451; % cos(75 degrees) division
        dir_ball = (averageOrientation_v/pi)*180;

        %--------Find avg spin vector--------------
        try
            % Only use the flow present in the ball (values have to be adjusted
     wrt. the changed image dimentions)
            x_idx = x_idx - (floor(center(2))-initSize);
            y_idx = y_idx - (floor(center(1))-initSize);
            % In case parts of the ball are outside of the box we are
            % looking at index is set to one or end value
            x_idx(x_idx < 1) = 1;x_idx(x_idx > initSize*2-1) = initSize*2+1;
            y_idx(y_idx < 1) = 1;y_idx(y_idx > initSize*2-1) = initSize*2+1;
            % Find flow only at binary ball mask
            Vx_s= flow_s.Vx(x_idx', y_idx');
            Vy_s = flow_s.Vy(x_idx', y_idx');
            M_s = flow_s.Magnitude(x_idx', y_idx');
        catch % assume error in choosing frame around ball, and set to zero
            Vx_s = 0;
            Vy_s = 0;
            M_s = 0;
        end

        % Only use the highest (most prominant spin values in the calculation)
        M_ss = maxk(M_s, 10, 1);
        [~, m_I] = maxk(M_ss, 10, 2);
        Vx_s_m = Vx_s(m_I);
        Vy_s_m = Vy_s(m_I);

        % We find the x and y components of the average vectors
        avg_sx = mean2(Vx_s_m);
        avg_sy = mean2(Vy_s_m);

        % Find the direction and magnitude of the average vector
        averageMagnitude_s = sqrt(avg_sx^2 + avg_sy^2);

        % Change axis so that they are as we are used to
        if(avg_sx>0 && avg_sy>0)
            averageOrientation_s =  atan(avg_sy/avg_sx);
        elseif (avg_sx<0 && avg_sy>0)
            averageOrientation_s =  pi - atan(avg_sy/(-avg_sx));
```

```
        elseif (avg_sx<0 && avg_sy<0)
            averageOrientation_s =   pi + atan((-avg_sy)/(-avg_sx));
        elseif (avg_sx>0 && avg_sy<0)
            averageOrientation_s =  2*pi - atan((-avg_sy)/(avg_sx));
        else
            averageOrientation_s = 0;
        end
        averageOrientation_s = 2*pi - averageOrientation_s;


        % -------Calculate the amplitude and direction of the ball spin--------
        v_spin= ((averageMagnitude_s*v.FrameRate*0.12)/radius)*slowMotionFactor
    ;
        rpm_spin = (v_spin*60)/(0.12*2*pi);
        dir_spin = (averageOrientation_s/pi)*180;

        % ----------PLOTS-----------
        %plot of the circle
        th = 0:pi/50:2*pi;
        xunit = radius * cos(th) + center(1);
        yunit = radius * sin(th) + center(2);

        plot(xunit,yunit, 1,3, 'r', 'LineWidth',3);
        plot(center(1),center(2),1,3, '.', 'MarkerSize', 5);

        %plot velocity vector
        quiver(center(1), center(2), avg_vx*20, avg_vy*20, 'LineWidth', 2);
        %plot spin vector
        quiver(center(1), center(2), avg_sx*20, avg_sy*20, 'LineWidth', 2);

        % Plot a box with the magnitude and orientation of the velocity
        % vector
        v_ball_text = sprintf('Velocity: %.4f [m/s]', v_ball);
        dir_ball_text = sprintf('Direction: %.1f degrees',dir_ball);

        % Plot a box with the magnitude and orientation of the spin
        % vector
        v_spin_text = sprintf('Spin velocity: %.2f [RPM] (%.4f [m/s])',rpm_spin
    , v_spin);
        dir_spin_text = sprintf('Spin Direction: %.1f degrees',dir_spin);
    elseif(iter>1)
        % record the movement of the center and radius
        % if no circle is detected record the previous value over
        movVector(iter,1) = movVector(iter-1,1);
        movVector(iter,2) = movVector(iter-1,2);
        movVector(iter,3) = movVector(iter-1,3);
    end


     % Save the frame in structure for later saving to video file
    s(k) = getframe(h);
    k = k+1;
    hold off

end

v2 = VideoReader('Videos/Aria_1.MOV');
v2 = readFrame(v2);
figure;
imshow(v2(rect(2):rect(2)+rect(4), rect:rect(1)+rect(3), :))
hold on
plot(movVector(:,1),movVector(:,2),'r-o','LineWidth',2);
legend('Measured trajectory');
```

```
hold off


% Remove any unused structure array elements
s(k:end) = [];

% Open a new figure and play the movie from the structure
hFig2 = figure;
movie(hFig2,s,1,v.FrameRate);

% Write to a video file
% This could be done within the original loop, but I wanted to show it
% separately
vOut = VideoWriter('Videos/slowmoCut3','MPEG-4');
vOut.FrameRate = v.FrameRate;
open(vOut)
for k = 1:numel(s)
    writeVideo(vOut,s(k))
end
close(vOut)


%% HERE WE FIND AND PLOT THE TRAJECTORY SEEN FROM ABOVE

% Read a single video frame
imshow(J);

%-------To allow for testing without running the whole code------
%movVector = load('R/vars.mat', 'movVector').movVector;
%----------------------------------------------------------------

% Remove potental zeros
mV = reshape(movVector(movVector>0), [size(movVector(movVector>0),1)/3, 3]);

%title('Select points (eg. 32 pointsthat form 8 right angles), first 4 must be
    around lane edges!, then press enter');
%[xii, yii] = getpts;
%---To save you the hastle of choosing all of the points, here you can load
% them (given that the same video crop is used)
xii = load('R/vars.mat', 'xii').xii;
yii = load('R/vars.mat', 'yii').yii;
%----------------------------------------------------------------------
hold on

% Initialize
N = size(xii,1); % num points
a = zeros(N,3); % points in homogeneous coordinates
l = zeros(3,N/4); % lines l
m = zeros(3,N/4); % lines m (orthogonal to l)
A = zeros(N/4,6);

for i=1:N
    a(i,:) = [xii(i); yii(i); 1];
end
for i=1:2:(N/2-1)
    plot([xii(i) xii(i+1)], [yii(i), yii(i+1)], 'linewidth', 5);
end
for i=1:4:N
    l(:,i) = cross(a(i,:)', a(i+1,:)');
    m(:,i) = cross(a(i+2,:)', a(i+3,:)');
end
for i=1:N/4
    A(i, :) = [l(1,i)*m(1,i),0.5*(l(1,i)*m(2,i)+l(2,i)*m(1,i)),l(2,i)*m(2,i)
```

```
    ,...
361             0.5*(l(1,i)*m(3,i)+l(3,i)*m(1,i)),  0.5*(l(2,i)*m(3,i)+l(3,i)*m(2,i
    )), l(3,i)*m(3,i)];
362 end
363
364 %%
365
366 [~,~,v] = svd(A);
367 sol = v(:,end);
368 imDCCP = [sol(1)  , sol(2)/2, sol(4)/2;...
369     sol(2)/2, sol(3)  , sol(5)/2;...
370     sol(4)/2, sol(5)/2  sol(6)];
371
372
373 [U,D,V] = svd(imDCCP);
374 D(3,3) = 1;
375 A = U*sqrt(D);
376
377 C = [eye(2),zeros(2,1);zeros(1,3)];
378 min(norm(A*C*A' - imDCCP),norm(A*C*A' + imDCCP))
379
380 H = inv(A);
381 min(norm(H*imDCCP*H'./norm(H*imDCCP*H') - C./norm(C)),norm(H*imDCCP*H'./norm(H*
    imDCCP*H') + C./norm(C)))
382
383
384 tform = projective2d(H');
385 K = imwarp(J,tform);
386
387 figure;
388 imshow(K);
389
390 hold off
391 %%
392
393 b = zeros(3,8); % transformed points
394
395 for i=1:size(b,2)
396     b(:,i) = H*a(i,:)';
397     b(:,i) = b(:,i)/b(3,i); % normalize
398 end
399
400 figure;imshow(J);
401 hold on
402 for i=1:2:size(b,2) % assume 4 first selected lines as lines around lane
403     line([a(i,1),a(i+1,1)], [a(i,2),a(i+1,2)], 'LineWidth', 3, 'Color', 'blue')
    ;
404 end
405 plot(mV(:,1),mV(:,2), 'Color','red', 'LineWidth', 1.5); % plot trajectory on
    lane image
406 hold off
407
408 % Plot the lines and path of original video
409 figure;
410 subplot(1,2,1);
411 hold on
412 for i=1:2:size(b,2)
413     line([a(i,1),a(i+1,1)], [a(i,2),a(i+1,2)], 'Color', 'blue');
414 end
415 plot(mV(:,1),mV(:,2), 'Color','red');
416 set(gca, 'YDir','reverse'); % We invert the axis since a normal image has y
    axis inverted when shown
417 hold off
```

```
418
419 % Find transformed trajectory
420 mV_new = mV;
421 mV_new(:,3) = 1;
422 mV_new = (H*mV_new')';
423 mV_new = mV_new./mV_new(:,3);
424
425 % Plot transfromed lines and trajectory
426 b = -b; % The values are set to negative just to make the plot more plesant to
        campare
427 mV_new = -mV_new;
428 subplot(1,2,2);
429 hold on
430 for i=1:2:size(b,2)
431     line([b(1,i),b(1,i+1)], [b(2,i),b(2,i+1)], 'Color', 'blue');
432 end
433 plot(mV_new(:,1),mV_new(:,2), 'Color', 'red');
434 set(gca, 'YDir','reverse');
435 hold off
```