Report for IVR Coursework

Myrzakhmetov Bagdat - s1153100
Tarlungianu Claudiu Andrei - s1132388

1. Introduction

For this coursework we needed to track a ball/balls and other objects in a sequence of images. There are a lot of approaches for tracking objects in the sequence of images. One of the widely used methods for tracking objects is by using their position. This can be found by comparing sequences of images with a stationary (background) image. If we see any differences then we could say that there is/are object(s) in the respective frame. One other method that can be used for tracking objects is by using colour values. This is because in the some cases our position tracking may not identify an object well, so colour is also important. After detecting the objects we can identify further additional information about object: it's possible to find the path, the highest point it reaches in its 'movement', or the shape of the object, whether it is a ball or some other kind of shapes (eg. Squared-shape or triangle-shaped).

All the above questions could be implemented by doing some image processing. First of all we can take the RGB colour values of the compared images. Also we can take first take the differences between images and use RGB values; in our case, we used first used the standard RGB values for background removal (taking the differences with the binary value). We have come to realise that using normalised values of RGB is more efficient since it handles changes in illumination a lot better than with the use of standard RGB values.

To carry out image processing, we start by removing the outliers, filling the objects inside and carrying out morphological opening. For this practical we need to find the highest point of the identified objects, draw the path created by them and identify whether the object is a ball or not. For all these purposes we can use object centroids, areas and perimeters.

To find the highest point we can naively use the minimum Y coordinate position (by first looking at the full path created by the object and analysing it afterwards to determine the highest point), or we can also check the differences between the object's centroids in consecutive frames, because at the highest point the differences between centroids could be reduced and if it is below than some threshold value then, say it is the highest point of the tracking object. For identifying the ball we could use circularity/compactness, corner detections (for example if our object has a small amount of corners, then it's possible to say that this is the ball), centre of mass etc.

2. Methods

Main methods:
- Reading inputs and finding background

First of all we need to read given inputs. We have decided to use the code that was supplied for a sequence of images reading. It goes through each image, reads it and displays them consecutively. To compare objects we need to find the average background image. As was suggested in the assignment requirements, we have used the median value of each individual pixel of first 25 frames. The median means that we firstly need to sort all individual pixels in row i and column j for the 25 frame. To do so at the beginning we thought to use triple 'for' loops which goes through every rows, columns and each frames. But this was time consuming and quite complicated in the sense of coding (precessing speed greatly reduced). Then we decided to use standard Matlab functions which finds the median: X(:,:,:,k) = imread('file_names'), background = median(X,4);

- Preprocessing of images

After finding the background image, we have to compare it with other frames, so as to detect whether there are any changes from one frame to another (whether there any objects can be found in the frame); here we will first consider position. To do so, we have used the Matlab function imabsdiff(im1, im2), which takes two images and computes their absolute difference. But in some cases just taking into account the absolute differences might not be enough, especially if, as mentioned before, we firstly tried to use the RGB image of each picture. In the sequences of images there are some illumination effects. To cope with this it is possible to use normalised RGB values instead of raw RGB. We considered different methods for normalising, firstly using normalising by taking into account squared addition in the squared root(for example: r/(sqrt(r^2+g^2+b^2))) , but then decided to only use the simple normalising method: r/(r+b+g); After normalising we computed the differences of images, also converted it to binary image. Now that we have obtained the 'difference' image (difference between background and frames), there might still be be some noise due to illumination. To detect the object more precisely it's important to eliminate (or at least to reduce) this noise. We have used the Matlab function bwareaopen(im, P) which takes the binary image with specified pixel values, then removes all connected components which are below that P. Also imfill(im, 'holes') fills all holes in the binary image, so that when applying the 'edge' function, we only obtain a border around the object and not inside it as well. bwmorph(difference, 'erode', 1) command removes all small areas in the image. We used these functions to get the exact, accurate difference. Now that we have processed the binary image, we may now use it for our future tasks.

- Detecting the object

For detecting the object, Matlab has some special functions. To obtain the labels of the objects for instance, we may use bwlabel(object,4). Also, edge(labelled) draws an edge around the given labelled object. So by using edge function we identify where the edges are. We then overlap the edge obtained that we have turned to a green colour with the original frame to obtain a image in which the object is outlined by the green edges created. We have left this method in because it was developed by using some computations to arrive at the final result, but we also use the 'bwboundaries' function to draw boundaries around the object because of it being easier to use and because it allows us to easier draw differently coloured boundaries (so as to see all objects in the frame with edges around them). Because of the existence of noise in some of the images, we have

encountered some false object detections.

- Drawing the path between objects

The basic idea which was used was to compare consecutive images and draw lines between the image centres. After taking the absolute differences, we need to enter the preprocessing stages. At the beginning it converts difference into gray scale by using the ‘rgb2gray’ function and then using thresholding for the object. Morphological opening can then be performed by calling imopen(Im, SE), where SE is the structuring element. In our case, we decided SE to be SE = strel('disk', 5, 8); because we need morphological element ‘strel’ with a disc shape with a specified Radius and approximated by a sequence of N periodic-line structuring elements (from the Mathworks webpage). Now we can take labels and identify the region properties with the ‘regionprops’ function. Here we need to take the maximum area object and identify it’s centre. Now we can start drawing the line between two consequence images.

- Finding the highest point

To finding the maximum point we can use the centroid points from the previous section. We are interested only in the Y coordinate, because it shows the vertical position (enabling us to be more accurate than taking the actual distance between the object in the different frames). If we were to use both X and Y coordinates and calculate the Euclidean distance, some paths might have been misleading (especially those that belong to the objects that travel a considerable distance on the X axis as well). The algorithm we have tried to implement uses the last 3 differences between 2 neighbourhood points; if the said differences are all smaller than a given threshold (1.7 seems to be fairly accurate) then say that is the maximum point. In this way, we have tried to detect the highest point without looking into the future, but only using the information we have gathered so far; this way the detection is real-time.

It is able to determine what the highest point the object in each frame is by drawing a blue ‘plus’ and the highest-point it reaches in its path by drawing a cyan star.

In order for the program not to crash when it detects no object, thus not having any highest point to search for, we only try to find the highest point if the sum of all the pixel values in our binary processed difference between background & current frame is greater than some amount (in our case, 1000).

- Identifying the ball

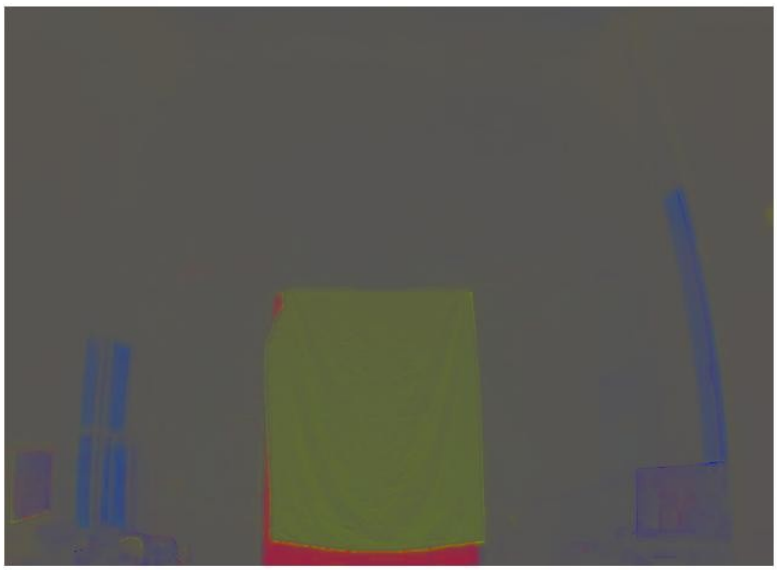
There are many ways of identifying the ball .We have decided to use an algorithm which is based on the object’s circularity: Circularity = P(perimeter)^2/(4*pi*A(area)) . If this number is close enough to 1, then we can say that the object has a circle shape, so we may deduct that it is a ball. To get the Area and Perimeter of the identified object we can again call the ‘regionprops’ function. By using this function we have have obtained the perimeter and the filled area.

3. Results

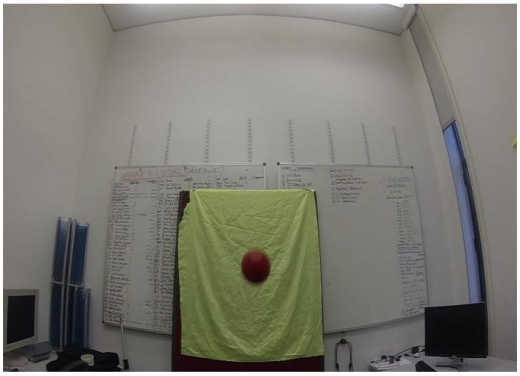
The following examples have been gathered from processing images in the ‘GOPR0002’ folder.

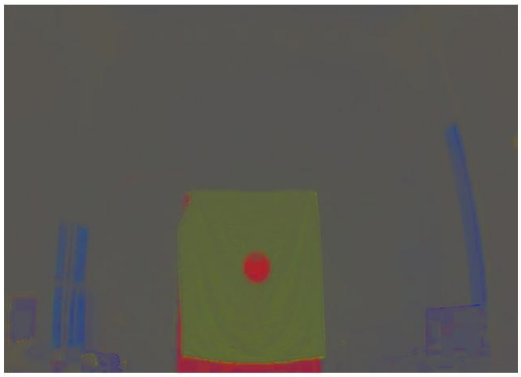
Obtaining an average background by calculating the median value of pixels from the first 25 images in the video – background (line 20):

Obtaining Normalised RGB for background – normBack(line 40):

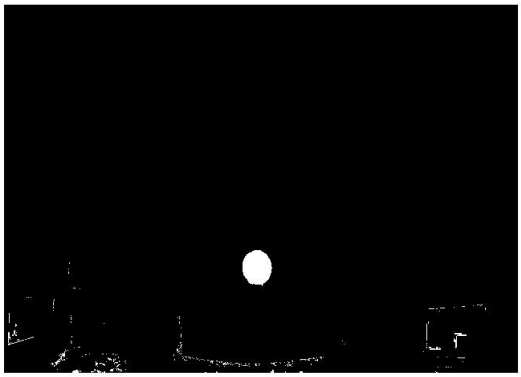


Frame 251 – frame & Obtaining Normalised RGB for frame 251 – normFrame (line 91):

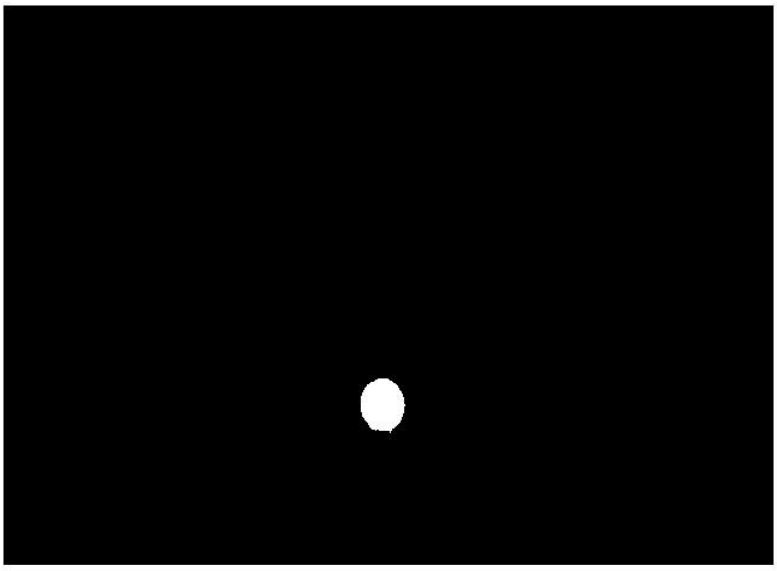




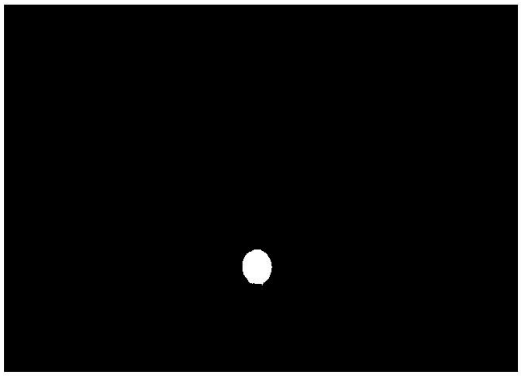
Difference between normalised background and normalised frame 251 – difference (line 94): & Difference obtained after converting the difference from normalised RGB to a binary image, using threshold 0.035 – difference (line 95):



Difference obtained after performing 'morphological opening' on the binary difference image, using parameter 250 for the area of connected components to be removed & standard connectivity (8 – two-dimensional, four-connected neighbourhood) – difference (line 96):



Difference obtained after ‘filling the holes’ in the previously obtained object – difference (line 97): & Result after eliminating noise by using ‘erode’ on the previous image – object (line 100).



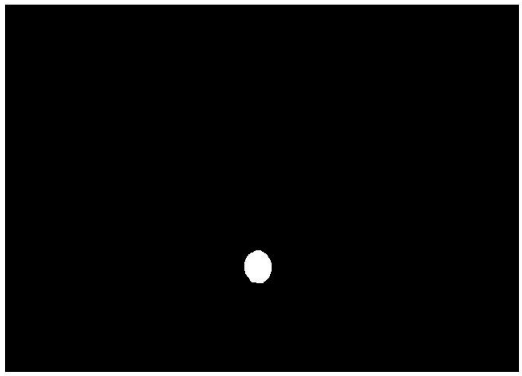
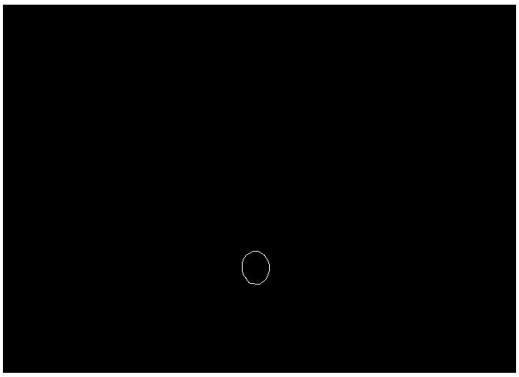
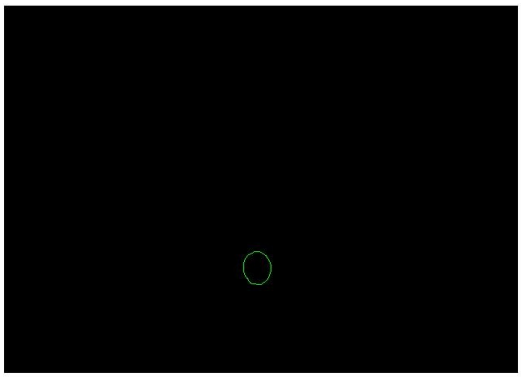
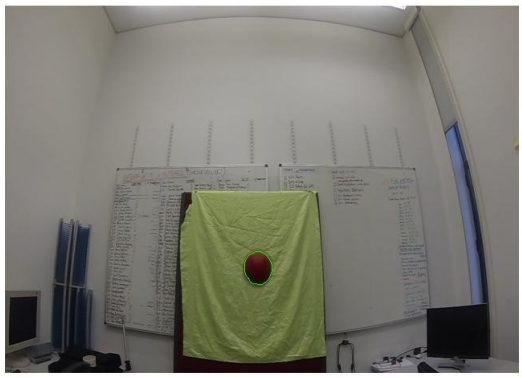


Image obtained after switching the values in the previous image between them (1s become 0s and 0s become 1s) (so as to apply the ‘edge’ function) – object (line 101). After performing labelling and applying the ‘edge function’, we obtain the following image – edges (line 106).



After this, we colour the edges green – edges2 (line 109) & overlap with the original frame – detected Object (line 111)

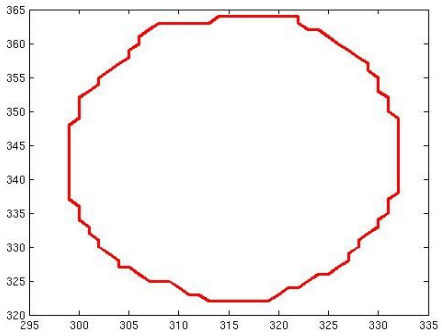




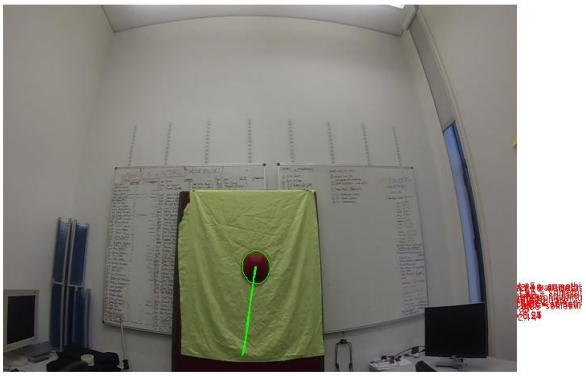
We now obtain all the object's properties in 'blobMeasurements' (lines 144-147):
perimeters = 126.5685
filledAreas = 1154.

Once this has been done, we can now calculate the object's circularities (main method of ball detection, can be found at line 150):
circularities = 1.1047
Since the value is between 0.3 and 1.2, we now classify the detected object as a ball (line 174).

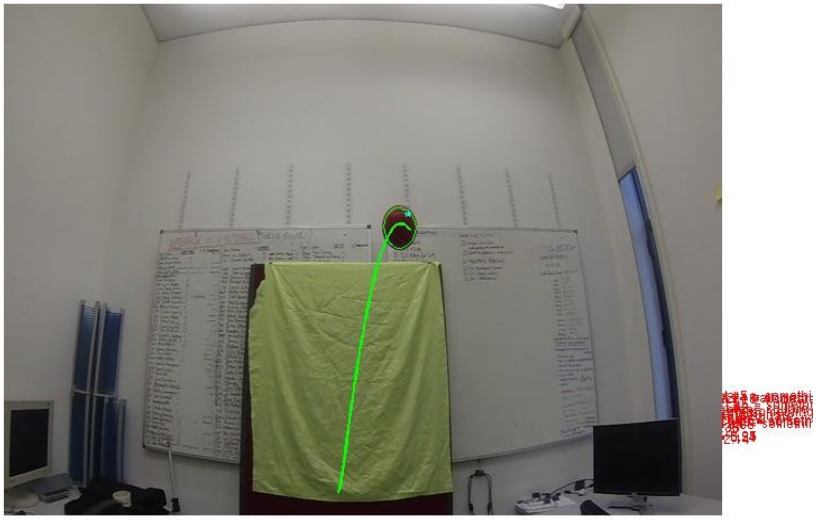
By using the built-in Matlab method, 'bwboundaries' and displaying it by using 'thisBoundary', we obtain the following image (line 170):



Since we are storing the values of the X & Y coordinates where the ball had previously been detected in the vectors Xcent and Ycent, we may use them to draw the path line (line 218)



Identifying the highest point (frame 284) (line 221) by drawing a cyan-coloured star



4. Discussion

- Detecting the object

Due to noise in some of the frames, mainly due to the change in illumination, it sometimes falsely detects pixel-patches as false objects. The algorithm works much better now in this respect when using normalised RGB than it did previously with raw RGB, however there still are small problems. This is one of the reasons for the program running slowly when it sees noise (looks at all the objects and outlines them). A quick fix would be to only look for one object, but we would loose the ability of tracking two objects when there actually are two in the frame. More experimenting with thresholds might also improve the algorithm.

- Drawing the path between objects

Due to the fact that after many object detections, the Xcent and Ycent vectors (which store the path to draw) become very big and there is a lot more to display, the program starts running very slowly. An option would be to only display the paths of the objects at the very end of the file processing. We can do this by commenting the code for drawing the path in each frame (line 218), and de-commenting the code which starts at line 246. Another way to work around this issue would be to change the code that displays the object in such a way that it normally draws the path of the ball in each frame as long as an object can continuously be detected, but if an object detection has not taken place in, say 10 frames, then stop drawing path and reset the Xcent and Ycent vectors as empty. This way, it could both run smoothly and display the path in each frame.

Because of the fact that we are using only one pair of Xcent and Ycent vectors, the path-finding algorithm does not behave well the moment when there are 2 different objects in the frame; it usually jumps from one object's movement to the other (depending on which of the two it thinks is more of an 'object'). We could work around this by giving Xcent and Ycent an extra dimension, in which it could store the path values for each object in the frame individually (might react badly because of noise). In our code we are only using the centroid of the object with maximum area for getting Xcent & Ycent values that we use to draw the path. Instead of doing this, we could store the centroid values of all the objects in different vectors (i.e. Xcent(1), Ycent(1), Xcent(2) etc.) and to detect the correspondence between the two vectors so as to classify the objects to their paths, we look at the area & Euclidean distances between centroid points of the objects.

- Finding the highest point

It is able to determine what the highest point in the path of the object is fairly accurately. It could be improved by storing the point in a variable to compare it to further detections on the same path/object, so as to eliminate confusion because of points of local minimum (sometimes, after starting its decent, it thinks the object is going down and back up again for a while – usually not true for balls –, after which it recomputes the highest point – which we don't want).

The same problem as with the path-finding is present when there are two objects in the frame; it could be fixed as well by looking for a highest point for each object.

Furthermore, by looking at more differences in the past & using Bayesian classification would improve the accuracy of the detection.

- Identifying the ball

It has no real issue with identifying whether an object is a ball or not, but it does need the

whole of the object to be in frame so as to say it is a ball or not (first frames when the ball appears, it identifies it as ‘something else’).

5. Code

```
function background = IVRvideoread(file_dir)

    % Storring files
    filenames = dir([file_dir '*.jpg']);


    frame = imread([file_dir filenames(1).name]);
    figure(1); h1 = imshow(frame);

    % Read first 25 frames to obtain background.
    for k = 1 : 25
        X(:,:,:,k) = imread([file_dir filenames(k).name]);
        figure(1);
        imshow(X(:,:,:,k));
        h1 = imshow(X(:,:,:,k));
        disp(['showing frame ' num2str(k)]);
    end

    % Compute background by taking median of pixel values in the first 25 frames
    background = median(X,4);

    %Initialise Vectors for path drawing
    Xcent = [];
    Ycent = [];

    % Initialise disc for opening
    SE = strel('disk', 5, 8);

    % Perform Normalise RGB for background
        Im = im2double(background);

    imR = squeeze(Im(:,:,1));
    imG = squeeze(Im(:,:,2));
    imB = squeeze(Im(:,:,3));

    imRNorm = imR./(imR+imG+imB);
    imGNorm = imG./(imR+imG+imB);
    imBNorm = imB./(imR+imG+imB);

    normBack = cat(3, imRNorm, imGNorm, imBNorm);

    % Flag for drawing star (used for highest point highlight)
    flag = 2;

    % X & Y coordinates of Star (highest-point)
    starX = 0;
    starY = 0;

    %Read remaining frames to look for objects
    %GOPR0002 - Objects start: 230, 495, 618, 820, 1063, 1261, 1432, 1548, 1704.

    for k = 26 : size(filenames,1)
```

```
        disp(['tracking obj in frame ' num2str(k)]);
        isHighestPoint = false;



        %Code for path detection

        frame_4 = imread([file_dir filenames(k-4).name]);
        %frame_1 = imread([file_dir filenames(k-1).name]);
        frame = imread([file_dir filenames(k).name]);
        consecutive_difference = abs(frame_4 - frame);
        Igray = rgb2gray(consecutive_difference);
        Ithresh = Igray > 3;
        BW = imopen(Ithresh, SE);
        [labels,number] = bwlabel(BW, 4);
        Istats = regionprops(labels, 'basic', 'Centroid');
        [maxVal, maxIndex] = max([Istats.Area]);
        Xcent = [ Xcent Istats(maxIndex).Centroid(1) ];
        Ycent = [ Ycent Istats(maxIndex).Centroid(2) ];
        if (size(Ycent,2))
            disp(['Y coordinate: ' num2str(Ycent(size(Ycent,2)))]);
        end



        %Code for object highlighting

        %Perform Normalise RGB on current frame
        Im = im2double(frame);

        imR = squeeze(Im(:,:,1));
        imG = squeeze(Im(:,:,2));
        imB = squeeze(Im(:,:,3));

        imRNorm = imR./(imR+imG+imB);
        imGNorm = imG./(imR+imG+imB);
        imBNorm = imB./(imR+imG+imB);

        normFrame = cat(3, imRNorm, imGNorm, imBNorm);

        % Take difference between background & current frame; threshold it; open
it; fill holes in object
        difference = imabsdiff(normBack, normFrame);
        difference = im2bw(difference, 0.035);
        difference = bwareaopen(difference,250);
        difference = imfill(difference, 'holes');

        % Get rid of noise in image
        object = bwmorph(difference, 'erode', 1);
        object = ~object;

        labeled = bwlabel(object, 4);

        % Create image with only the edges around object
        edges = edge(labeled);

        % create RGB image with green edges
        edges2 = cat(3,zeros(size(edges, 1), size(edges, 2)), edges,
zeros(size(edges, 1), size(edges, 2)));
```

```
        % 'Overlap' the RGB frame with the greep edges image to obtain contour
around the ball
        detectedObj = frame;

        for i = 1 : size(edges2,1)
            for j = 1 : size(edges2,2)
                if (edges2(i,j,2) ~= 0)
                    detectedObj(i,j,1) = 0;
                    detectedObj(i,j,2) = 255;
                    detectedObj(i,j,3) = 0;
                    end
            end
        end



        % Code for highest point detection (still needs a considerable amount of
work, although works to a certain extent)

        % Analyses differences between Y coordinates fo the ball position at
previous 3 frames
        i = size(Ycent,2);
        if (i >= 4)
            if (Ycent(i) - Ycent(i-1) <= 1.7 && Ycent(i-1) - Ycent(i-2) <= 1.7
&& Ycent(i-2) - Ycent(i-3) <= 1.7)
                isHighestPoint = true;
            end
        end



        %Code for Ball Detection

        % Get each object in frame and it's index
        [labeledImage numberOfObjects] = bwlabel(~object);

        % Get the properties of each object in the frame
        blobMeasurements = regionprops(labeledImage, 'Perimeter', 'FilledArea',
'Centroid');
        boundaries = bwboundaries(~object);
        perimeters = [blobMeasurements.Perimeter];
        filledAreas = [blobMeasurements.FilledArea];

        % Calculate circularities:
        circularities = perimeters .^2 ./ (4 * pi * filledAreas);

        % Print to command window.
        fprintf('#, Perimeter, Filled Area, Circularity\n');
        for blobNumber = 1 : numberOfObjects
            fprintf('%d, %9.3f, %11.3f, %11.3f\n', blobNumber,
perimeters(blobNumber), filledAreas(blobNumber), circularities(blobNumber));
        end

        % Go through each object in frame
        for blobNumber = 1 : numberOfObjects
            % Outline object
            thisBoundary = boundaries{blobNumber};
            hold on;

            % Display prior boundaries in blue
```

```
            for l = 1 : blobNumber-1
                thisBoundary = boundaries{l};
                plot(thisBoundary(:,2), thisBoundary(:,1), 'b', 'LineWidth', 3);
            end

            % Display current bounary in red
            thisBoundary = boundaries{blobNumber};
            plot(thisBoundary(:,2), thisBoundary(:,1), 'r', 'LineWidth', 3);

            % Determine the shape
            if (circularities(blobNumber) < 1.25 && circularities(blobNumber) >
0.75)
                message = sprintf('The circularity of object #%d is %.3f,\nso
the object is a ball',...
                blobNumber, circularities(blobNumber));
                shape = 'ball';
            elseif circularities(blobNumber) < 1.6
                message = sprintf('The circularity of object #%d is %.3f,\nso
the object is a square',...
                blobNumber, circularities(blobNumber));
                shape = 'something else';
            elseif circularities(blobNumber) > 1.6 && circularities(blobNumber)
< 1.8
                message = sprintf('The circularity of object #%d is %.3f,\nso
the object is an isocoles triangle',...
                blobNumber, circularities(blobNumber));
                shape = 'something else';
            else
                message = sprintf('The circularity of object #%d is %.3f,\nso
the object is something else.',...
                blobNumber, circularities(blobNumber));
                shape = 'something else';
            end

            % Display overlay message close to the object
            overlayMessage = sprintf('Object #%d = %s\ncirc = %.2f', blobNumber,
shape, circularities(blobNumber));
            text(blobMeasurements(blobNumber).Centroid(1) + 50,
blobMeasurements(blobNumber).Centroid(2) - 50, overlayMessage, 'Color', 'r');
            %pause(0.01);

            % Button to go frame-by-frame
            %button = questdlg(message, 'Continue', 'Continue', 'Cancel',
'Continue');
            %if strcmp(button, 'tats(maxIndex).Centroid(1),2) ~= 0) ||Cancel')
            %    break;
            %end

        end




        %Code for displaying image, path & highest point markers

        % Image with green border
        h1 = imshow(detectedObj);
        set(h1, 'CData', detectedObj);
        hold on;
```

```
        % Path drawer
        line(Xcent, Ycent, 'color', 'green', 'LineWidth', 2);

        %Highest point highlighter
        if (sum(sum(difference))<=1000)
            if (isHighestPoint)
                q = size(Istats(maxIndex).Centroid(1),2);
                if(q > 0)
                    plot(Istats(maxIndex).Centroid(1),
Istats(maxIndex).BoundingBox(2), '-+b');
                    %pause(0.01);
                        flag = 1;
                end
            elseif(flag == 1)
                starX = Istats(maxIndex).Centroid(1);
                starY = Istats(maxIndex).BoundingBox(2);
                plot(Istats(maxIndex).Centroid(1),
Istats(maxIndex).BoundingBox(2), '-pc');
                pause(0.1);
                flag = 0;
            end
        end

        if(starX && starY)
            plot(starX, starY, '-pc');
            drawnow;
        end

    end


    %Code for drawing path at the end
    %{
    h1 = imshow(background);
    set(h1, 'CData', background);
    hold on;
    line(Xcent, Ycent, 'color', 'green', 'LineWidth', 2);
    %}

end
```

References

Mathworks
for describing the different Matlab functions used in our code

http://www.mathworks.com/matlabcentral/answers/42587-detect-certain-shapes-in-binary
determining whether the object is ball or not (used some ideas)

http://www.youtube.com/watch?v=WWtZDmW2vB0
motion tracking