CMP9135M — Computer Vision

1st George Davies
School of Computer Science
University of Lincoln
Lincoln, United Kingdom
27421138@students.lincoln.ac.uk

Task 1 — Image Processing

The aim of this task was to segment three different types of balls (a tennis ball, a football, and an american football) from a timeseries of images of them rolling across the frame. No deep learning solutions were allowed.

Task 1.a — Automated ball objects segmentation

1) Otsu thesholding: The first step was to binarise the image. By default, the matlab function imbinarize uses otsu to distinguish between foreground and background.
By ajusting the sensitivity, I found the value that removed the most of the background without removing the balls was 0.35. The imbinarize function only works on grayscale images so I tested parsing each colour channel separately and the red channel seemed to perform the best.
Examples of the output can be seen in Figure 1.

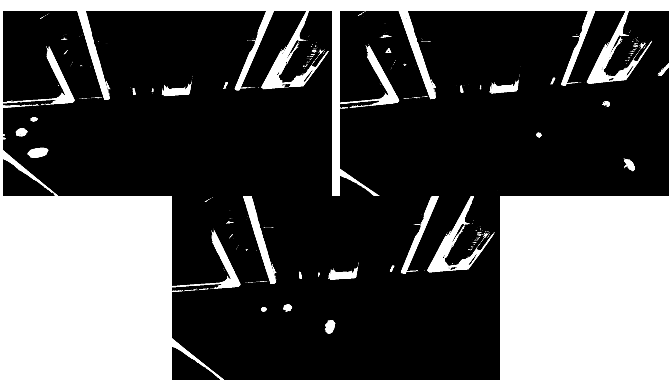

Fig. 1: Examples of the binary image output of the preliminarly segmentation using otsu

2) Converting to convex hulls: To removed the remaining background artifacts, I decided to convert all the connected components into convex hulls.
As the ball are supposed to be convex in the first place, they shouldn't change much (apart from the american football in the last few frames where the otsu segmentation was a bit aggressive).
This had the effect of massivly increasing the area of the background artifacts, making them easily distinguishable from the balls.
Examples of the output can be seen in Figure 2.

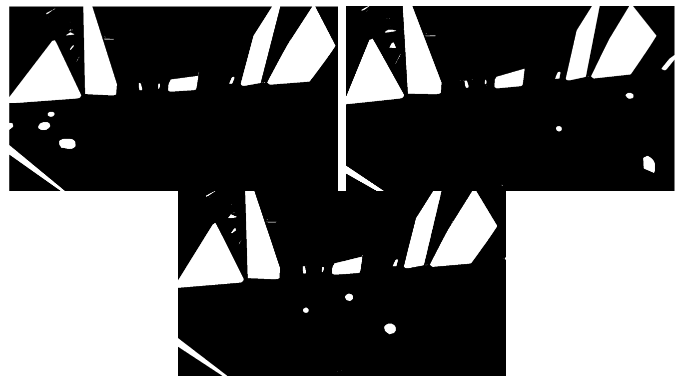

Fig. 2: Examples of the binary image output after transforming each connected component into a convex hull

3) Removing the remaining artifacts: Now, the background artifacts can be removed from the binarised images by removing every connected component that doesn't fit certain characteristics such as area and extent. After this, all that remains are the three balls.

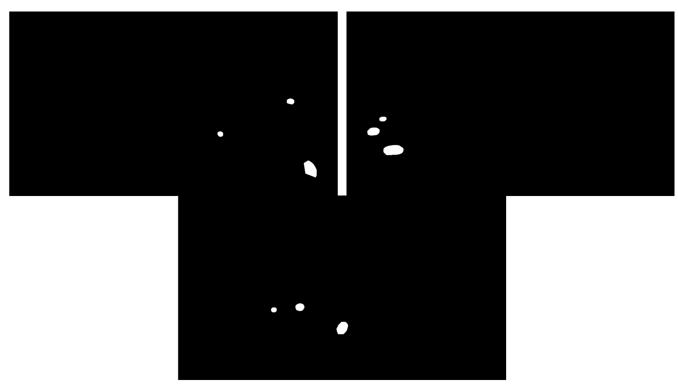

Fig. 3: Examples of the binary image output after transforming each connected component into a convex hull

More examples of the best and worst segmentations can be found in the Appendix Figures 9 and 10

Task 1.b — Segmentation evaluation

To evaluated the segmentation, the Dice Similarity score between the segmented images and the ground truth can be calculated. As each image is simply stored as a matrix of 1s and 0s, it is possible to do binary operations on the matrices.

The DS score is calculated by multiplying the number of 1s in the intersection of the two matrices (logical &) by 2 and dividing that by the total number of 1s in both matrices. If there is a perfect overlap between the images, then the intersection will incompass all the 1s and the DS score will be 1. If there is no overlap, then there is no intersection and the DS score will be 0.

$$\text{Dice}(M, S) = \frac{2|M \cap S|}{|M| + |S|} \tag{1}$$

where:

- M and S are the to sets to compared
- |M| and |S| are sizes of the sets M and S

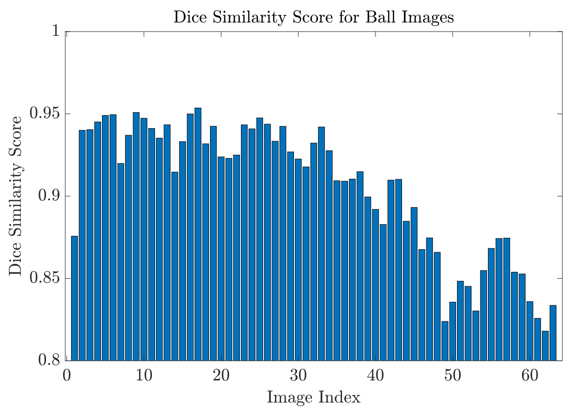

Fig. 4: Dice Similarity score for all 63 images. y-axis restricted between 0.8 and 1 for better visibility

As seen in Figure 4, the segmentation performs well from images 2 to 34, then it starts to lower in quality. This is due to the otsu segmentation of the american football being too strong near the end.
The average every image's DS score is **0.90464** and the standard deviation is **0.040538**

Task 2 — Feature Calculation

Task 2.a — Shape features

For this task, the following shape features were calculated:

1) Solidity: The proportion of the pixels in the convex hull that are also in the object.

$$\text{Solidity} = \frac{\text{Area of the object}}{\text{Area of its convex hull}}$$

2) Non-compactness: Compactness is the proportion of the region's pixels to all of the bounding box's pixels. So non-compactness is the inverse of this.

$$\text{Non-compactness} = 1 - \frac{\text{Area of the object}}{\text{Area of bounding box}}$$

3) Circularity: The roundness of the object.

$$\text{Circularity} = \frac{4\pi * \text{Area of the object}}{(\text{Perimeter of the object})^2} * \left(1 - \frac{0.5}{\text{r}}\right)$$

Where r $= \frac{\text{Perimeter of the object}}{2\pi} + 0.5$

4) Eccentricity: The eccentricity is the proportion of the distance between the foci of the ellipse and the length of its major axis. An eccentricity of 0 means its a perfect circle. An eccentricity of 1 means its a line.

$$\text{Eccentricity} = \frac{\text{Distance between foci of ellipse}}{\text{Length of major axis of ellipse}}$$

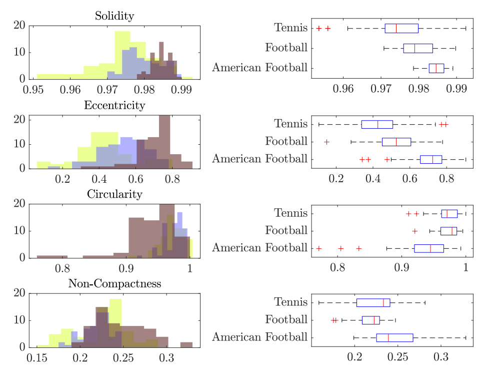

Fig. 5: Histograms and boxplots showing the Solidity, Non-Compactness, Circularity, and Eccentricity of the three ball types.
Legend:

- yellow: tennis
- blue: football
- brown: american football

Figure 5 displays histograms and boxplots representing the distribution of four different shape features (solidity, non-compactness, circularity, and eccentricity) for each ball type (tennis, football, and American football).
Going through each shape feature one by one.

There is a lot of overlap between the three ball types for solidity. The tennis ball has the lowest median solidity, followed by the football, and then the American football. The tennis ball has the highest range of solidity values, followed by the football, and then the american football.

For Eccentricity, the order of the median values is the same as for solidity but there is less overlap between the three ball types. This measure seems better at distinguishing the american football from the other two ball types as the american football is not a sphere like the other two balls.

Circularity like eccentricity is better at distinguishing the american football from the other two ball types, with the tennis ball and football having a similar Q1 and Q3 values. This is also due to the american football not being a sphere like the other two balls.

Finally, non-compactness is the worst at distinguishing between the three ball types as there is a lot of overlap between the three ball types. The mean of the non-compactness values for each ball lie within 0.025 of each other.

Task 2.b — Texture features

The goal of this task was to calculate the normalised grey-level co-occurrence matrix (glcm) in 4 orientations for each ball patch and for each colour channel.

The texture features that were extracted were:

- Angular Second Moment (ASM): also know as energy is the sum of squared elements in the glcm.

$$\text{ASM} = \sum_{i,j} p(i,j)^2 \tag{2}$$

- Contrast: is the a measure of the intensity contrast between a pixel and its neighbor over the whole image.

$$\text{Contrast} = \sum_{i,j} |i-j|^2 p(i,j) \tag{3}$$

- Correlation: is the measure of how correlated a pixel is to its neighbor over the whole image.

$$\text{Correlation} = \sum_{i,j} \frac{(i-\mu_i)(j-\mu_j)p(i,j)}{\sigma_i \sigma_j} \tag{4}$$

For each of these texture features, the average and range was calculated over the four orientations (0°, 45°, 90°, 135°). The averages can be seen in Figure 6 and the ranges can be seen in Figure 7

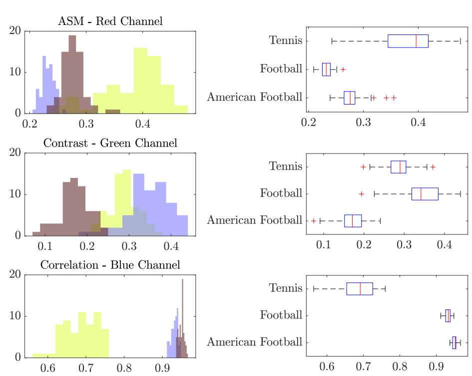

Fig. 6: Histograms and boxplots showing the average Angular Second Moment, Contrast, and Correlation associated with one colour channel for each ball type. Angular Second Moment on the red channel, Contrast on the green channel, Correlation on the blue channel. Same Legend as Figure 5.

For ASM in the red channel, there is no overlap between the ball types from their Q1 to Q3. Meaning this is a very good metric for discriminating between the ball types.
Same goes for Contrast in the green channel though there is more overlap between the tennis ball and football. The american football is very distinguished from the other ball types.
Finally for the Correlation in the blue channel, the tennis ball is completely seperated from the other two ball types with minor overlap between the football and american football.

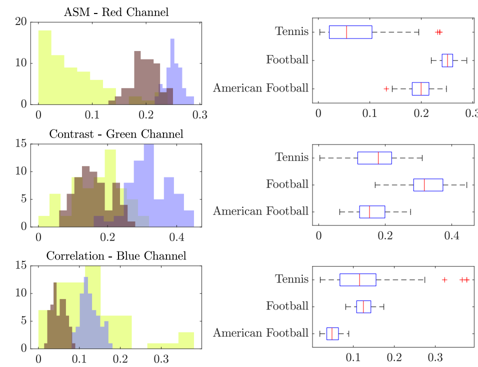

Fig. 7: Histograms and boxplots showing the ranges of Angular Second Moment, Contrast, and Correlation associated with one colour channel for each ball type. Angular Second Moment on the red channel, Contrast on the green channel, Correlation on the blue channel. Same Legend as Figure 5.

With the ranges instead of the averages, the ASM seperates the three balls relativly well with the tennis ball frequently having a value of zero and the two others with a small amount of overlap.
The Contrast is very bad at distinguishing the tennis ball from the american football, with the regular football having slightly higher contrast.
The Correction is incapable of distinguishing the tennis ball but is alright at seperating the football from the american football.

Task 2.c — Discriminative information

Overall the texture features are much better at providing discriminative information than the shape features. This makes sense for this situation as the three balls have similar shapes but their colours and patterns are more unique. If all three balls were made out of the same material, the texture features would be much less effective.

Task 3 — Object Tracking

The final task was to implement a kalman filter from scratch without using any methods from inbuilt libraries. Four trajectories were given, x.csv and y.csv contained the real coordinates and na.csv and nb.csv contained the noisy coordinates from which estimated coordinates could be calculated.

For this task, certain variables were set: F is should be a Constant Velocity motion model. i.e.

$$\text{F} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where Δt is the time interval set to 0.5.
H is a Cartisian observation model. i.e.

$$\text{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{6}$$

The covariance matices Q and R are

$$Q = \begin{bmatrix} 0.16 & 0 & 0 & 0 \\ 0 & 0.36 & 0 & 0 \\ 0 & 0 & 0.16 & 0 \\ 0 & 0 & 0 & 0.36 \end{bmatrix} R = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.25 \end{bmatrix} \quad (7)$$

Task 3.a — Kalman filter tracking

The kalman tracking algoritm iterates over every sample. For each sample, it predicts a state vector and a covariance using the constant velocity motion model and the motion noise, with which it then estimates the state and covariance with the Cartesian observation model, observation noise, and the sample.

Using the first sample and the initial state, Figure 8 is the result

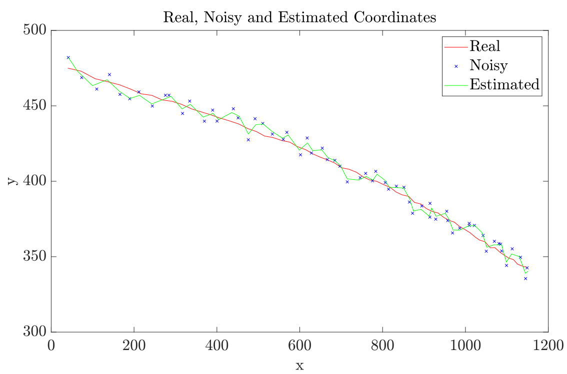

Fig. 8: plot of the estimated trajectory of coordinates $[x\_, y\_]$, together with the real $[x, y]$ and the noisy $[na, nb]$ for comparison

Task 3.b — Evaluation

The equation for Root Mean Squared Error (RMSE) is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right)} \quad (8)$$

where:

- n is the total number of observations
- x_i and y_i are the i^{th} actual coordinates
- $\hat{x}_i$ and $\hat{y}_i$ is the i^{th} predicted coordinates

This equation in matlab looks like this:

```
% Root Mean Squared Error (RMSE) calculation
rmse_n = sqrt(mean((x - na).^2 + (y - nb).^2));
rmse = sqrt(mean((x - x_).^2 + (y - y_).^2));
```

$rsme\_n$ is the RMSE of the noisy coordinates relative to the ground truth.

$rsme$ is the RMSE of the estimated coordinates realtive to the ground truth.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \quad (9)$$

where:

- σ is the standard deviation
- n is the total number of observations
- x_i is the i^{th} value
- μ is the mean of the values

This equation in matlab looks like this:

```
% Standard deviation calculation of the RMSE
std_n = std(sqrt((x - na).^2 + (y - nb).^2));
std = std(sqrt((x - x_).^2 + (y - y_).^2));
```

As the RMSE is a single number, there is no standard deviation. Therefore, I took the standard deviation of the root squared errors.

The results of these calualations are:

- RMSE for the noisy coordinates ≅ **7.416**
- RMSE for the estimated coordinates ≅ **5.877**
- STD for the noisy coordinates ≅ **2.573**
- STD for the estimated coordinates ≅ **2.181**

All these values are relative to the real coordinates and rounded to the 3^{rd} decimal. These results show that the kalman filter successfully removed some of the noise as it is closer to the real coordinates.

Better results may be found by testing different noise matrices and time intervals.

APPENDIX

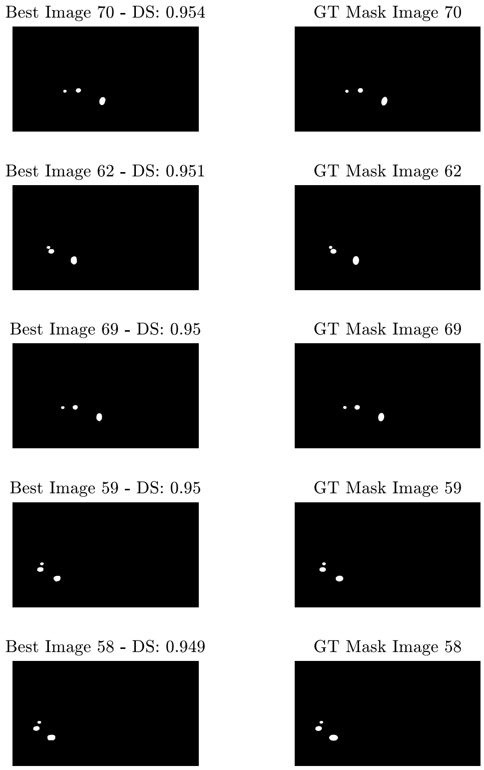

Fig. 9: Best 5 segmented ball images compared to the ground truth

A. *find_ball.m*

```
function out = find_balls(img)

    %% Close the objects to make them fuller
    se = strel('disk', 4);
    img = imclose(img, se);
    img = imfill(img, 'holes');

    %% Make the objects convex (to get rid of the holes in
        the balls)
    cc1 = bwconncomp(img);
    stats1 = regionprops(cc1, 'ConvexHull');

    convex_image = false(size(img));
    for i = 1:numel(stats1)
        % Get the convex hull for the current object
        convexHull = stats1(i).ConvexHull;
        % Convert the convex hull to a binary mask
        mask = poly2mask(convexHull(:,1), convexHull(:,2),
            size(img, 1), size(img, 2));
        % Combine the mask with the existing binary image
        convex_image = convex_image | mask;
    end

    %% Remove objects that are too small or too large, are
        too high, and have a low area to bbox area ratio
    cc2 = bwconncomp(convex_image);
```

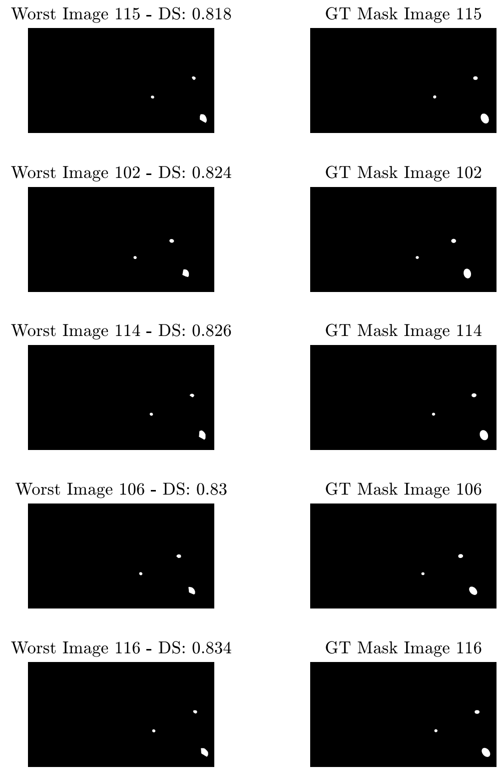

Fig. 10: Worst 5 segmented ball images compared to the ground truth

```
    stats2 = regionprops(cc2, 'Area', 'Centroid', 'Extent')
        ;

    y = [stats2.Centroid];
    y = y(2:2:end);

    % Create a new binary image where only the elliptical
        objects are included
    elliptical_objects = ismember(labelmatrix(cc2), find
        ((325 < [stats2.Area] & [stats2.Area] < 2600) & (
        height(img)*0.45 < y) & [stats2.Extent] > 0.6));
    % imshow(elliptical_objects)

    out = elliptical_objects;
end
```

B. *main.m*

```
%loop through all the images in ../../data/ball_frames
images_GT = dir('../../data/ball_frames/ground_truth/*.png'
    );
images = dir('../../data/ball_frames/original/*.png');
out_dir = '../../data/ball_frames/mask/';

for i = 1:length(images)
    %read the image
    img = imread(images(i).folder + "/" + images(i).name);

    %% theshold the balls individually by colour then
        combine them (DS = 0.84)
    % hsv_img = rgb2hsv(img);
```

```matlab
    % tennis_mask = mask_tennisball(hsv_img);
    % football_mask = mask_football(img);
    % american_fb_mask = mask_american_football(hsv_img);

    % balls_mask = tennis_mask | football_mask |
        american_fb_mask;

    %% theshold the entire image and find the balls (DS =
        0.90)
    % binarise the image using ostu thesholding (only use
        the red channel as it works best for the balls)
    otsu_mask = imbinarize(img(:, :, 1), 'adaptive', '
        Sensitivity', 0.35);

    % find the balls in the image
    balls_mask = find_balls(otsu_mask);

    %% save the mask
    imshow(balls_mask)
    % imwrite(balls_mask, fullfile(out_dir, images(i).name)
        );
end
close("all")
```

C. mask_american_football.m

```matlab
function american_fb = mask_american_football(img)
    % Create a binary mask for the american football
    mask = img(:, :, 1) > 0.03 & 0.05 > img(:, :, 1) & img
        (:, :, 3) > 0.70 & 1 > img(:, :, 3);

    % Close the objects to make them fuller
    se = strel('disk', 4);
    mask = imclose(mask, se);
    mask = imfill(mask, 'holes');

    %% Make the objects convex (to get rid of the holes in
        the balls)
    cc1 = bwconncomp(mask);
    stats1 = regionprops(cc1, 'ConvexHull');

    convex_image = false(size(mask));
    for i = 1:numel(stats1)
        % Get the convex hull for the current object
        convexHull = stats1(i).ConvexHull;
        % Convert the convex hull to a binary mask
        mask = poly2mask(convexHull(:,1), convexHull(:,2),
            size(img, 1), size(img, 2));
        % Combine the mask with the existing binary image
        convex_image = convex_image | mask;
    end

    %% Remove objects that are too small or too large, and
        are too high
    cc2 = bwconncomp(convex_image);
    stats2 = regionprops(cc2, 'Area', 'Centroid');

    y = [stats2.Centroid];
    y = y(2:2:end);

    american_fb = ismember(labelmatrix(cc2), find([stats2.
        Area] < 2600 & [stats2.Area] > 900 & (height(img)
        *0.45 < y)));
end
```

D. mask_football.m

```matlab
function football = mask_football(img)
    % Create a binary mask for the football
    mask = img(:, :, 1) > 150 & img(:, :, 2) > 150 & img(:,
         :, 3) > 150;

    se = strel('disk', 4);
    mask = imclose(mask, se);
    mask = imfill(mask, 'holes');

    %% Make the objects convex (to get rid of the holes in
        the balls)
    cc1 = bwconncomp(mask);
    stats1 = regionprops(cc1, 'ConvexHull');

    convex_image = false(size(mask));
    for i = 1:numel(stats1)
        % Get the convex hull for the current object
        convexHull = stats1(i).ConvexHull;
        % Convert the convex hull to a binary mask
        mask = poly2mask(convexHull(:,1), convexHull(:,2),
            size(img, 1), size(img, 2));
        % Combine the mask with the existing binary image
        convex_image = convex_image | mask;
    end

    %% Remove objects that are too small or too large, are
        too high, and have a low area to bbox area ratio
    cc2 = bwconncomp(convex_image);
    stats2 = regionprops(cc2, 'Area', 'Circularity', '
        Centroid');

    y = [stats2.Centroid];
    y = y(2:2:end);

    football_mask = ismember(labelmatrix(cc2), find([stats2
        .Area] < 900 & [stats2.Area] > 400 & [stats2.
        Circularity] > 0.85 & (height(img)*0.45 < y)));

    %% Draw a disk with center point at the centroid and
        diameter as the major axis
    cc3 = bwconncomp(football_mask);
    stats3 = regionprops(cc3, 'Centroid', 'MajorAxisLength'
        );
    football = false(size(football_mask));
    for i = 1:numel(stats3)
        centroid = stats3(i).Centroid;
        maj_AL = stats3(i).MajorAxisLength;
        radius = maj_AL / 2;
        centerX = centroid(1);
        centerY = centroid(2);

        % Create a grid of points within the disk
        [x, y] = meshgrid(1:size(img, 2), 1:size(img, 1));
        distance = sqrt((x - centerX).^2 + (y - centerY)
            .^2);

        % Create a binary mask for the disk
        football = distance <= radius;
    end
end
```

E. mask_tennisball.m

```matlab
function tennis = mask_tennisball(img)
    % Create a binary mask for the tennis ball
    mask = img(:, :, 1) > 0.08 & img(:, :, 1) < 0.16 & img
        (:, :, 2) > 0.7 & img(:, :, 3) > 0.7;

    % Close the objects to make them fuller
    se = strel('disk', 4);
    mask = imclose(mask, se);
    mask = imfill(mask, 'holes');

    % Draw a disk with center point at the centroid and
        diameter as the major axis
    cc = bwconncomp(mask);
    stats = regionprops(cc, 'Centroid', 'MajorAxisLength');
    tennis = false(size(mask));
    for i = 1:numel(stats)
        centroid = stats(i).Centroid;
        maj_AL = stats(i).MajorAxisLength;
        radius = maj_AL / 2;
        centerX = centroid(1);
        centerY = centroid(2);

        % Create a grid of points within the disk
        [x, y] = meshgrid(1:size(img, 2), 1:size(img, 1));
        distance = sqrt((x - centerX).^2 + (y - centerY)
            .^2);

        % Create a binary mask for the disk
        tennis = distance <= radius;
    end
end
```

F. plot_DS.m

```
function plot_DS(DS)
    % Plotting the bar graph
    hfig = figure;
    bar(DS);
    xlabel('Image Index');
    ylabel('Dice Similarity Score');
    title('Dice Similarity Score for Ball Images');
    ylim([0.8, 1]);

    fname = '../../report/figures/DS_bar_graph.pdf';

    picturewidth = 30; % set this parameter and keep it
        forever
    hw_ratio = 0.65; % feel free to play with this ratio
    set(findall(hfig,'-property','FontSize'),'FontSize',
        22) % adjust fontsize to your document

    set(findall(hfig,'-property','Interpreter'),'
        Interpreter','latex')
    set(findall(hfig,'-property','TickLabelInterpreter'),'
        TickLabelInterpreter','latex')
    set(hfig,'Units','centimeters','Position',[3 3
        picturewidth hw_ratio*picturewidth])
    pos = get(hfig,'Position');
    set(hfig,'PaperPositionMode','Auto','PaperUnits','
        centimeters','PaperSize',[pos(3), pos(4)])
    print(hfig,fname,'-dpdf','-vector','-fillpage')
end
```

G. plot_imgs.m

```
function plot_imgs(imgs, DS, name)
    GT_path = '../../data/ball_frames/ground_truth/';
    mask_path ='../../data/ball_frames/mask/';
    hfig = figure;

    for i = 1:5
        if strcmp(name, 'best')
            subplot(5, 2, i+(i-1));
            imshow([mask_path 'frame-' num2str(imgs(end-i
                +1)+53) '.png']);
            title(['Best Image ' num2str(imgs(end-i+1)+53)
                ' - DS: ' num2str(round(DS(imgs(end-i+1)),
                3))]);

            subplot(5, 2, i*2);
            imshow([GT_path 'frame-' num2str(imgs(end-i+1)
                +53) '_GT.png']);
            title(['GT Mask Image ' num2str(imgs(end-i+1)
                +53)]);
        elseif strcmp(name, 'worst')
            subplot(5, 2, i+(i-1));
            imshow([mask_path 'frame-' num2str(imgs(i)+53)
                '.png']);
            title(['Worst Image ' num2str(imgs(i)+53) ' -
                DS: ' num2str(round(DS(imgs(i)), 3))]);

            subplot(5, 2, i*2);
            imshow([GT_path 'frame-' num2str(imgs(i)+53) '
                _GT.png']);
            title(['GT Mask Image ' num2str(imgs(i)+53)]);
        end
    end

    fname = strcat('../../report/figures/', name ,'.pdf');

    picturewidth = 30; % set this parameter and keep it
        forever
    hw_ratio = 1.4; % feel free to play with this ratio
    set(findall(hfig,'-property','FontSize'),'FontSize',
        22) % adjust fontsize to your document

    set(findall(hfig,'-property','Interpreter'),'
        Interpreter','latex')
    set(findall(hfig,'-property','TickLabelInterpreter'),'
        TickLabelInterpreter','latex')
    set(hfig,'Units','centimeters','Position',[3 3
        picturewidth hw_ratio*picturewidth])
    pos = get(hfig,'Position');
    set(hfig,'PaperPositionMode','Auto','PaperUnits','
        centimeters','PaperSize',[pos(3), pos(4)])
    print(hfig,fname,'-dpdf','-vector','-fillpage')
end
```

H. seg_eval.m

```
GT_path = '../../data/ball_frames/ground_truth/';
mask_path ='../../data/ball_frames/mask/';

DS = zeros(1, 63);

for i = 54:116
    % Load the segmented ball region (M) and the ground-
        truth binary ball mask (S)
    M = imread([mask_path 'frame-' num2str(i) '.png']);
    S = imread([GT_path 'frame-' num2str(i) '_GT.png']);

    % Convert the grayscale images to binary images
    M = imbinarize(uint8(M));
    S = imbinarize(uint8(S));

    % Calculate the intersection between M and S
    intersection = sum(sum(M & S));

    % Calculate the size of M and S
    size_M = sum(sum(M));
    size_S = sum(sum(S));

    % Calculate the Dice Similarity Score (DS)
    DS(i-53) = 2 * intersection ./ (size_M + size_S);
end

% Calculate the average and standard deviation of the Dice
    Similarity Score
avg_DS = mean(DS);
std_DS = std(DS);
disp(['The average Dice Similarity Score for the ball
    images is ' num2str(avg_DS)]);
disp(['The standard deviation of the Dice Similarity Score
    for the ball images is ' num2str(std_DS)]);

plot_DS(DS);


% Calculate the mean and standard deviation of DS
mean_DS = mean(DS);
std_DS = std(DS);

% Sort the DS array in ascending order
[sorted_DS, sorted_indices] = sort(DS);

% Display the worst 5 segmented ball images and their
    corresponding GT mask images
plot_imgs(sorted_indices, DS, 'worst');

% Display the best 5 segmented ball images and their
    corresponding GT mask images
plot_imgs(sorted_indices, DS, 'best');
```

I. show_video.m

```
images = dir('../../data/ball_frames/mask/*.png');

while true
    for i=1:length(images)
        img = imread(images(i).folder + "/" + images(i).
            name);
        imshow(img);
    end
end
```

J. box_and_hist.m

```
function box_and_hist(data_r, data_g, data_b, name)
    tennis_r = data_r{1};
    football_r = data_r{2};
    american_r = data_r{3};

    tennis_g = data_g{1};
```

```
football_g = data_g{2};
american_g = data_g{3};

tennis_b = data_b{1};
football_b = data_b{2};
american_b = data_b{3};

%(102, 56, 49)
tennis_colour = [0.87 1.0 0.31];
football_colour = [0.5 0.5 1];
american_colour = [0.4 0.2 0.19];

hfig = figure;  % save the figure handle in a variable

subplot(3, 2, 1);
histogram(tennis_r, 10, 'FaceColor', tennis_colour, '
    EdgeColor', 'none');
hold on;
histogram(football_r, 10, 'FaceColor', football_colour,
     'EdgeColor', 'none');
histogram(american_r, 10, 'FaceColor', american_colour,
     'EdgeColor', 'none');
title('ASM - Red Channel');

subplot(3, 2, 2); % Create a subplot for boxplots
boxplot([american_r, football_r, tennis_r], '
    Orientation', 'horizontal', 'Labels', {'American
    Football', 'Football', 'Tennis'});
hold off;

subplot(3, 2, 3);
histogram(tennis_g, 10, 'FaceColor', tennis_colour, '
    EdgeColor', 'none');
hold on;
histogram(football_g, 10, 'FaceColor', football_colour,
     'EdgeColor', 'none');
histogram(american_g, 10, 'FaceColor', american_colour,
     'EdgeColor', 'none');
title('Contrast - Green Channel');

subplot(3, 2, 4); % Create a subplot for boxplots
boxplot([american_g, football_g, tennis_g], '
    Orientation', 'horizontal', 'Labels', {'American
    Football', 'Football', 'Tennis'});
hold off;

subplot(3, 2, 5);
histogram(tennis_b, 10, 'FaceColor', tennis_colour, '
    EdgeColor', 'none');
hold on;
histogram(football_b, 10, 'FaceColor', football_colour,
     'EdgeColor', 'none');
histogram(american_b, 10, 'FaceColor', american_colour,
     'EdgeColor', 'none');
title('Correlation - Blue Channel');

subplot(3, 2, 6); % Create a subplot for boxplots
boxplot([american_b, football_b, tennis_b], '
    Orientation', 'horizontal', 'Labels', {'American
    Football', 'Football', 'Tennis'});
hold off;


%https://www.youtube.com/watch?v=wP3jjk1O18A
fname = strcat('../../report/figures/', name, '.pdf');

picturewidth = 30; % set this parameter and keep it
    forever
hw_ratio = 0.65; % feel free to play with this ratio
set(findall(hfig,'-property','FontSize'),'FontSize',
    15) % adjust fontsize to your document

set(findall(hfig,'-property','Interpreter'),'
    Interpreter','latex')
set(findall(hfig,'-property','TickLabelInterpreter'),'
    TickLabelInterpreter','latex')
set(hfig,'Units','centimeters','Position',[3 3
    picturewidth hw_ratio*picturewidth])
pos = get(hfig,'Position');
set(hfig,'PaperPositionMode','Auto','PaperUnits','
    centimeters','PaperSize',[pos(3), pos(4)])
print(hfig,fname,'-dpdf','-vector','-fillpage')
end
```

K. *box_and_hist.m*

L. *main.m*

```
% This part of the assignment will deal with
    feature extraction, more specifically you will be
% examining texture and shape features. Using the
    provided GT ball masks to obtain the
% corresponding ball patches from original RGB images,
    carrying out the following tasks.

GT_path = '../../data/ball_frames/ground_truth/';
RGB_path = '../../data/ball_frames/original/';

%% Shape features

% a) (shape features) For each of the ball patches,
     calculate four different shape features
% discussed in the lectures (solidity, non-compactness,
    circularity, eccentricity). Plot the
% distribution of all the four features, per ball type.

[ball1, ball2, ball3] = shape_features(GT_path);
plot_shapes(ball1, ball2, ball3);

%% Texture features

% b) (texture features) Calculate the normalised grey-level
     co-occurrence matrix in four
% orientations (0, 45, 90, 135) for the patches from the
    three balls, separately for each of the
% colour channels (red, green, blue). For each
    orientation, calculate the first three features
% proposed by Haralick et al. (Angular Second Moment,
    Contrast, Correlation), and produce per-
% patch features by calculating the feature average and
    range across the 4 orientations. Select
% one feature from each of the colour channels and plot the
     distribution per ball type.

[features, averages, ranges] = texture_features(RGB_path,
    GT_path);
plot_features(averages, ranges);

%% Discriminative information

% c) (discriminative information) Based on your
    visualisations in part a) and b), discuss which
% features appear to be best at differentiating between
    different ball types. For each ball type,
% are shape or texture features more informative?
    Which ball type is the easiest/hardest to
% distinguish, based on the calculated features? Which
    other features or types of features would
% you suggest for the task of differentiating between the
    different ball types and why?
```

M. *plot_features.m*

```
function plot_features(avg, ranges)
   %% split the averages and ranges into the 3 balls
    avg_tennis = avg(:, 1:3);
    avg_football = avg(:, 4:6);
    avg_american = avg(:, 7:9);

    range_tennis = ranges(:, 1:3);
    range_football = ranges(:, 4:6);
    range_american = ranges(:, 7:9);

   %% split into the 3 texture features
    avg_tennis_asm = avg_tennis(:, 1);
    avg_tennis_cont = avg_tennis(:, 2);
    avg_tennis_corr = avg_tennis(:, 3);

    avg_football_asm = avg_football(:, 1);
    avg_football_cont = avg_football(:, 2);
    avg_football_corr = avg_football(:, 3);
```

```
    avg_american_asm = avg_american(:, 1);
    avg_american_cont = avg_american(:, 2);
    avg_american_corr = avg_american(:, 3);

    range_tennis_asm = range_tennis(:, 1);
    range_tennis_cont = range_tennis(:, 2);
    range_tennis_corr = range_tennis(:, 3);

    range_football_asm = range_football(:, 1);
    range_football_cont = range_football(:, 2);
    range_football_corr = range_football(:, 3);

    range_american_asm = range_american(:, 1);
    range_american_cont = range_american(:, 2);
    range_american_corr = range_american(:, 3);

    %% split into the 3 colour channels
    avg_tennis_asm_r = avg_tennis_asm(1:3:end);
    avg_tennis_cont_g = avg_tennis_cont(2:3:end);
    avg_tennis_corr_b = avg_tennis_corr(3:3:end);

    avg_football_asm_r = avg_football_asm(1:3:end);
    avg_football_cont_g = avg_football_cont(2:3:end);
    avg_football_corr_b = avg_football_corr(3:3:end);

    avg_american_asm_r = avg_american_asm(1:3:end);
    avg_american_cont_g = avg_american_cont(2:3:end);
    avg_american_corr_b = avg_american_corr(3:3:end);

    range_tennis_asm_r = range_tennis_asm(1:3:end);
    range_tennis_cont_g = range_tennis_cont(2:3:end);
    range_tennis_corr_b = range_tennis_corr(3:3:end);

    range_football_asm_r = range_football_asm(1:3:end);
    range_football_cont_g = range_football_cont(2:3:end);
    range_football_corr_b = range_football_corr(3:3:end);

    range_american_asm_r = range_american_asm(1:3:end);
    range_american_cont_g = range_american_cont(2:3:end);
    range_american_corr_b = range_american_corr(3:3:end);

    box_and_hist({avg_tennis_asm_r, avg_football_asm_r,
        avg_american_asm_r}, {avg_tennis_cont_g,
        avg_football_cont_g, avg_american_cont_g}, {
        avg_tennis_corr_b, avg_football_corr_b,
        avg_american_corr_b}, 'averages');
    box_and_hist({range_tennis_asm_r, range_football_asm_r,
         range_american_asm_r}, {range_tennis_cont_g,
        range_football_cont_g, range_american_cont_g}, {
        range_tennis_corr_b, range_football_corr_b,
        range_american_corr_b}, 'ranges');

end
```

N. plot_shapes.m

```
function plot_shapes(ball1, ball2, ball3)
    tennis_colour = [0.87 1.0 0.31];
    football_colour = [0.5 0.5 1];
    american_colour = [0.4 0.2 0.19];

    hfig = figure;

    subplot(4, 2, 1);
    histogram(ball1.Solidity, 10, 'FaceColor',
        tennis_colour, 'EdgeColor', 'none');
    hold on;
    histogram(ball2.Solidity, 10, 'FaceColor',
        football_colour, 'EdgeColor', 'none');
    histogram(ball3.Solidity, 10, 'FaceColor',
        american_colour, 'EdgeColor', 'none');
    title('Solidity');
    hold off;

    subplot(4, 2, 2); % Create a subplot for boxplots
    boxplot([ball3.Solidity, ball2.Solidity, ball1.Solidity
        ], 'Orientation', 'horizontal', 'Labels', {'
        American Football', 'Football', 'Tennis'});
    hold off;

    subplot(4, 2, 3);
    histogram(ball1.Eccentricity, 10, 'FaceColor',
        tennis_colour, 'EdgeColor', 'none');
    hold on;
    histogram(ball2.Eccentricity, 10, 'FaceColor',
        football_colour, 'EdgeColor', 'none');
    histogram(ball3.Eccentricity, 10, 'FaceColor',
        american_colour, 'EdgeColor', 'none');
    title('Eccentricity');
    hold off;

    subplot(4, 2, 4); % Create a subplot for boxplots
    boxplot([ball3.Eccentricity, ball2.Eccentricity, ball1.
        Eccentricity], 'Orientation', 'horizontal', 'Labels
        ', {'American Football', 'Football', 'Tennis'});
    hold off;


    subplot(4, 2, 5);
    histogram(ball1.Circularity, 10, 'FaceColor',
        tennis_colour, 'EdgeColor', 'none');
    hold on;
    histogram(ball2.Circularity, 10, 'FaceColor',
        football_colour, 'EdgeColor', 'none');
    histogram(ball3.Circularity, 10, 'FaceColor',
        american_colour, 'EdgeColor', 'none');
    title('Circularity');
    hold off;

    subplot(4, 2, 6); % Create a subplot for boxplots
    boxplot([ball3.Circularity, ball2.Circularity, ball1.
        Circularity], 'Orientation', 'horizontal', 'Labels'
        , {'American Football', 'Football', 'Tennis'});
    hold off;


    subplot(4, 2, 7);
    histogram(ball1.NonCompactness, 10, 'FaceColor',
        tennis_colour, 'EdgeColor', 'none');
    hold on;
    histogram(ball2.NonCompactness, 10, 'FaceColor',
        football_colour, 'EdgeColor', 'none');
    histogram(ball3.NonCompactness, 10, 'FaceColor',
        american_colour, 'EdgeColor', 'none');
    title('Non-Compactness');
    hold off;

    subplot(4, 2, 8); % Create a subplot for boxplots
    boxplot([ball3.NonCompactness, ball2.NonCompactness,
        ball1.NonCompactness], 'Orientation', 'horizontal',
         'Labels', {'American Football', 'Football', '
        Tennis'});
    hold off;

    %https://www.youtube.com/watch?v=wP3jjk1O18A
    fname = strcat('../../report/figures/shape_feats.pdf');

    picturewidth = 30; % set this parameter and keep it
        forever
    hw_ratio = 0.65; % feel free to play with this ratio
    set(findall(hfig,'-property','FontSize'),'FontSize',
        15) % adjust fontsize to your document

    set(findall(hfig,'-property','Interpreter'),'
        Interpreter','latex')
    set(findall(hfig,'-property','TickLabelInterpreter'),'
        TickLabelInterpreter','latex')
    set(hfig,'Units','centimeters','Position',[3 3
        picturewidth hw_ratio*picturewidth])
    pos = get(hfig,'Position');
    set(hfig,'PaperPositionMode','Auto','PaperUnits','
        centimeters','PaperSize',[pos(3), pos(4)])
    print(hfig,fname,'-dpdf','-vector','-fillpage')


end
```

O. shape_features.m

```
function [ball1, ball2, ball3] = shape_features(GT_path)
    ball1 = struct('Solidity', [], 'Eccentricity', [], '
        Circularity', [], 'NonCompactness', []);
```

```
    ball2 = struct('Solidity', [], 'Eccentricity', [], '
        Circularity', [], 'NonCompactness', []);
    ball3 = struct('Solidity', [], 'Eccentricity', [], '
        Circularity', [], 'NonCompactness', []);

    for i=54:116
        GT = imread([GT_path 'frame-' num2str(i) '_GT.png'
            ]);

        cc = bwconncomp(GT);

        % calculate the shape features for each ball patch
            (solidity, non-compactness, circularity,
            eccentricity)
        stats = regionprops(cc, 'Solidity', 'Eccentricity',
             'Circularity', 'Extent', 'Area');

        for j=1:cc.NumObjects
            solidity = stats(j).Solidity;
            eccentricity = stats(j).Eccentricity;
            circularity = stats(j).Circularity;
            extent = stats(j).Extent;
            area = stats(j).Area;

            non_compactness = 1-extent;

            if area < 500
                ball1.Solidity = [ball1.Solidity; solidity
                    ];
                ball1.Eccentricity = [ball1.Eccentricity;
                    eccentricity];
                ball1.Circularity = [ball1.Circularity;
                    circularity];
                ball1.NonCompactness = [ball1.
                    NonCompactness; non_compactness];
            elseif area > 1350
                ball3.Solidity = [ball3.Solidity; solidity
                    ];
                ball3.Eccentricity = [ball3.Eccentricity;
                    eccentricity];
                ball3.Circularity = [ball3.Circularity;
                    circularity];
                ball3.NonCompactness = [ball3.
                    NonCompactness; non_compactness];
            else
                ball2.Solidity = [ball2.Solidity; solidity
                    ];
                ball2.Eccentricity = [ball2.Eccentricity;
                    eccentricity];
                ball2.Circularity = [ball2.Circularity;
                    circularity];
                ball2.NonCompactness = [ball2.
                    NonCompactness; non_compactness];
            end
        end
    end
end
```

P. texture_features.m

```
function [features, averages, ranges] = texture_features(
    RGB_path, GT_path)
    % warning suppression (it doesn't like the use of NaN to
        ignore black pixels)
    warning('off', 'images:graycomatrix:
        scaledImageContainsNan')
    warning('off', 'MATLAB:uistring:alternateprintpath:
        FigureMayHaveBeenClosed')


    %constants
    NUM_IMGS = 63;
    ORIENTATIONS = [0, 45, 90, 135];
    NUM_ORIENTATIONS = length(ORIENTATIONS);
    NUM_CHANNELS = 3;
    NUM_BALLS = 3;
    NUM_FEATURES = 3;

    %output matrices size
    num_cols = NUM_IMGS * NUM_CHANNELS;
    num_rows = NUM_FEATURES * NUM_BALLS;
    features = zeros(num_cols * NUM_ORIENTATIONS, num_rows)
        ;
    averages = zeros(num_cols, num_rows);
    ranges = zeros(num_cols, num_rows);

    % go through all the images (i for image)
    for i = 54:116
        % read the images
        img_RGB = imread([RGB_path 'frame-' num2str(i) '.
            png']);
        img_GT = imread([GT_path 'frame-' num2str(i) '_GT.
            png']);

        % extract the coordinates of the of the balls from
            the ground truth
        cc = bwconncomp(img_GT);
        stats = regionprops(cc, 'BoundingBox', 'Area');


        % for each ball in the image (b for ball)
        for b = 1:cc.NumObjects
            % extract the ball from the image
            bbox = stats(b).BoundingBox;
            ball = imcrop(img_RGB, bbox);

            % normalise the values
            ball = double(ball);
            ball = ball ./ 256;

            % set all the not ball pixels to NaN
            ball_gt = imcrop(img_GT, bbox);

            ball_r = ball(:,:,1);
            ball_g = ball(:,:,2);
            ball_b = ball(:,:,3);

            ball_r(ball_gt == 0) = NaN;
            ball_g(ball_gt == 0) = NaN;
            ball_b(ball_gt == 0) = NaN;

            ball = cat(3, ball_r, ball_g, ball_b);

            % get the area for identifying the ball type
            area = stats(b).Area;

            % for each colour channel (c for channel)
            for c = 1:3
                channel_features = zeros(numel(ORIENTATIONS
                    ), 3);

                % for each of the 4 orientations (o for
                    orientation)
                for o = 1:length(ORIENTATIONS)
                    % rotate the grayscale image and
                        calculate the GLCM
                    rotated_img = imrotate(ball(:,:,c),
                        ORIENTATIONS(o));
                    glcm = graycomatrix(rotated_img, '
                        Symmetric', true);

                    % extract the texture features (angular
                         second moment (energy), contrast,
                        and correlation)
                    stats2 = graycoprops(glcm, {'energy', '
                        contrast', 'correlation'});
                    asm = stats2.Energy;
                    contrast = stats2.Contrast;
                    correlation = stats2.Correlation;

                    % store the features in the features
                        matrix
                    channel_features(o, :) = [asm, contrast
                        , correlation];
                end

                feat_idx = (i-54) * NUM_ORIENTATIONS *
                    NUM_CHANNELS + (c-1) * NUM_ORIENTATIONS
                     + 1;
                idx = (i-54) * NUM_CHANNELS + (c-1) + 1;

                if area < 500 %tennis ball
                    features(feat_idx:feat_idx+
                        NUM_ORIENTATIONS-1, 1:3) =
```

```
                    channel_features;
                averages(idx, 1:3) = [mean(
                    channel_features, 1)];
                ranges(idx, 1:3) = [range(
                    channel_features, 1)];
            elseif area > 1350 % american football
                features(feat_idx:feat_idx+
                    NUM_ORIENTATIONS-1, 7:9) =
                    channel_features;
                averages(idx, 7:9) = [mean(
                    channel_features, 1)];
                ranges(idx, 7:9) = [range(
                    channel_features, 1)];
            else % football
                features(feat_idx:feat_idx+
                    NUM_ORIENTATIONS-1, 4:6) =
                    channel_features;
                averages(idx, 4:6) = [mean(
                    channel_features, 1)];
                ranges(idx, 4:6) = [range(
                    channel_features, 1)];
            end
        end
    end
  end
end
```

Q. kalman_predict.m

```
% https://uk.mathworks.com/help/control/ug/kalman-filtering
    .html

function [xp, Pp] = kalman_predict(x, P, F, Q)

    % Prediction step of Kalman filter.
    % x: state vector
    % P: covariance matrix of x
    % F: matrix of motion model
    % Q: matrix of motion noise
    % Return predicted state vector xp and covariance Pp
    xp = F * x; % predict state
    Pp = F * P * F' + Q; % predict state covariance
end
```

R. kalman_tracking.m

```
function [px, py] = kalman_tracking(z)
    % Track a target with a Kalman filter
    % z: observation vector
    % Return the estimated state position coordinates (px,
        py)

    dt = 0.5; % time interval
    N = length(z); % number of samples

    F = [1 dt 0 0; 0 1 0 0; 0 0 1 dt; 0 0 0 1]; % Constant
        Velocity motion model
    H = [1 0 0 0; 0 0 1 0]; % Cartesian observation model

    Q = [0.16 0 0 0; 0 0.36 0 0; 0 0 0.16 0; 0 0 0 0.36]; %
        motion noise
    R = [0.25 0; 0 0.25]; % observation noise

    x = [z(1,1); 0; z(2,1); 0]; % initial state
    % x = [0 0 0 0]'; % initial state
    P = Q; % initial state covariance

    s = zeros(4,N);

    for i = 1 : N
        [xp, Pp] = kalman_predict(x, P, F, Q);
        [x, P] = kalman_update(xp, Pp, H, R, z(:,i));
        s(:,i) = x; % save current state
    end

    px = s(1,:); % NOTE: s(2, :) and s(4, :), not
        considered here,
    py = s(3,:); % contain the velocities on x and y
        respectively
end
```

S. kalman_update.m

```
function [xe, Pe] = kalman_update(x, P, H, R, z)
    % Update step of Kalman filter.
    % x: state vector
    % P: covariance matrix of x
    % H: matrix of observation model
    % R: matrix of observation noise
    % z: observation vector

    % Return estimated state vector xe and covariance Pe
    S = H * P * H' + R; % innovation covariance
    K = P * H' * inv(S); % Kalman gain
    zp = H * x; % predicted observation

    xe = x + K * (z - zp); % estimated state
    Pe = P - K * S * K'; % estimated covariance
end
```

T. main.m

```
%% Implement a Kalman filter from scratch (not using any
    method/class from pre-built libraries)
% that accepts as input the noisy coordinates [na,nb
    ] and produces as output the estimated
% coordinates [x*,y*]. For this, you should use a Constant
    Velocity motion model F with constant
% time intervals dt = 0.5 and a Cartesian observation model
     H. The covariance matrices Q and R
% of the respective noises are the following:

%       |- 0.16    0     0     0  -|
%  Q =  |     0  0.36    0     0   |   R = |  0.25    0  |
%       |     0    0   0.16    0   |       |    0   0.25 |
%       |-    0    0     0   0.36 -|

% real coordinates [x, y] of the football
x = readmatrix('../../data/x.csv');
y = readmatrix('../../data/y.csv');

% noisy version provided by some segmentation and
    recognition for the football
na = readmatrix('../../data/na.csv');
nb = readmatrix('../../data/nb.csv');

[x_, y_] = kalman_tracking([na; nb]);

%% 1) You should plot the estimated trajectory of
    coordinates [x*,y*], together with the real [x,y]
% and the noisy ones [a,b] for comparison. Discuss how you
    arrive to the solution.

plot_fig(x, y, na, nb, x_, y_);


%% 2) You should also assess the quality of the
    tracking by calculating the mean and standard
% deviation of the Root Mean Squared error (include the
    mathematical formulas you used for
% the error calculation in your report). Compare both noisy
     and estimated coordinates to the
% ground truth. Adjust the parameters associated with the
    Kalman filter, justify any choices
% of parameter(s) associated with Kalman Filter that
     can give you better estimation of the
% coordinates that are closer to the ground truth.
    Discuss and justify your findings in the
% report.

% Root Mean Squared Error (RMSE) calculation
rmse_n = sqrt(mean((x - na).^2 + (y - nb).^2));
rmse = sqrt(mean((x - x_).^2 + (y - y_).^2));

% Standard deviation calculation of the RMSE
std_n = std(sqrt((x - na).^2 + (y - nb).^2));
std = std(sqrt((x - x_).^2 + (y - y_).^2));

fprintf('RMSE for noisy coordinates: %f\n', rmse_n);
fprintf('RMSE for estimated coordinates: %f\n', rmse);
fprintf('Standard deviation for noisy coordinates: %f\n',
    std_n);
fprintf('Standard deviation for estimated coordinates: %f\n
    ', std);
```

U. plot_fig.m

```
function plot_fig(x, y, na, nb, x_, y_)
    hfig = figure;
    plot(x, y, 'r', 'LineWidth', 0.5);
    hold on;
    plot(na, nb, 'bx', 'MarkerSize', 5);
    plot(x_, y_, 'g', 'LineWidth', 0.5);
    legend('Real', 'Noisy', 'Estimated');
    xlabel('x');
    ylabel('y');
    title('Real, Noisy and Estimated Coordinates');
    hold off;

    fname = strcat('../../report/figures/kalman.pdf');

    picturewidth = 30; % set this parameter and keep it
        forever
    hw_ratio = 0.6; % feel free to play with this ratio
    set(findall(hfig,'-property','FontSize'),'FontSize',
        20) % adjust fontsize to your document

    set(findall(hfig,'-property','Interpreter'),'
        Interpreter','latex')
    set(findall(hfig,'-property','TickLabelInterpreter'),'
        TickLabelInterpreter','latex')
    set(hfig,'Units','centimeters','Position',[3 3
        picturewidth hw_ratio*picturewidth])
    pos = get(hfig,'Position');
    set(hfig,'PaperPositionMode','Auto','PaperUnits','
        centimeters','PaperSize',[pos(3), pos(4)])
    print(hfig,fname,'-dpdf','-vector','-fillpage')
end
```