2022

SCE4104: Practical Applications in Computer Vision

ASSIGNMENT: BALL TRACKING IN FOOTBALL GAMES

ERIKA SPITERI BAILEY

Contents

Introduction

The aim of the project was to design a vision-based system that can detect and track the football during a football match. Six video clips were provided, together with corresponding ground truth/annotations of the actual ball position on every frame. The videos were captured at 25 frames per second.

Various characteristics of these video clips, from camera position to approximate ball size were used to detect the football. Tuneable parameters were created and perfected to detect the ball as it appeared from any corner or edge of the camera, and locate it as it traverses the football ground, as consistently as possible. Numerous factors and computations were combined to reduce the number of false positives and false negatives as much as possible, while keeping the highest number of true positives and true negatives possible.

Increasing the number of measures used to determine ball location made the overall system more robust, but it also created a more complex system by depending on multiple parameters.

All the parameters taken into consideration together with the factors contributing to the method selection are described in the following report.

Method Description

The contributing parameters to the designed system, whose method is described hereunder, is shown below in Figure 1.

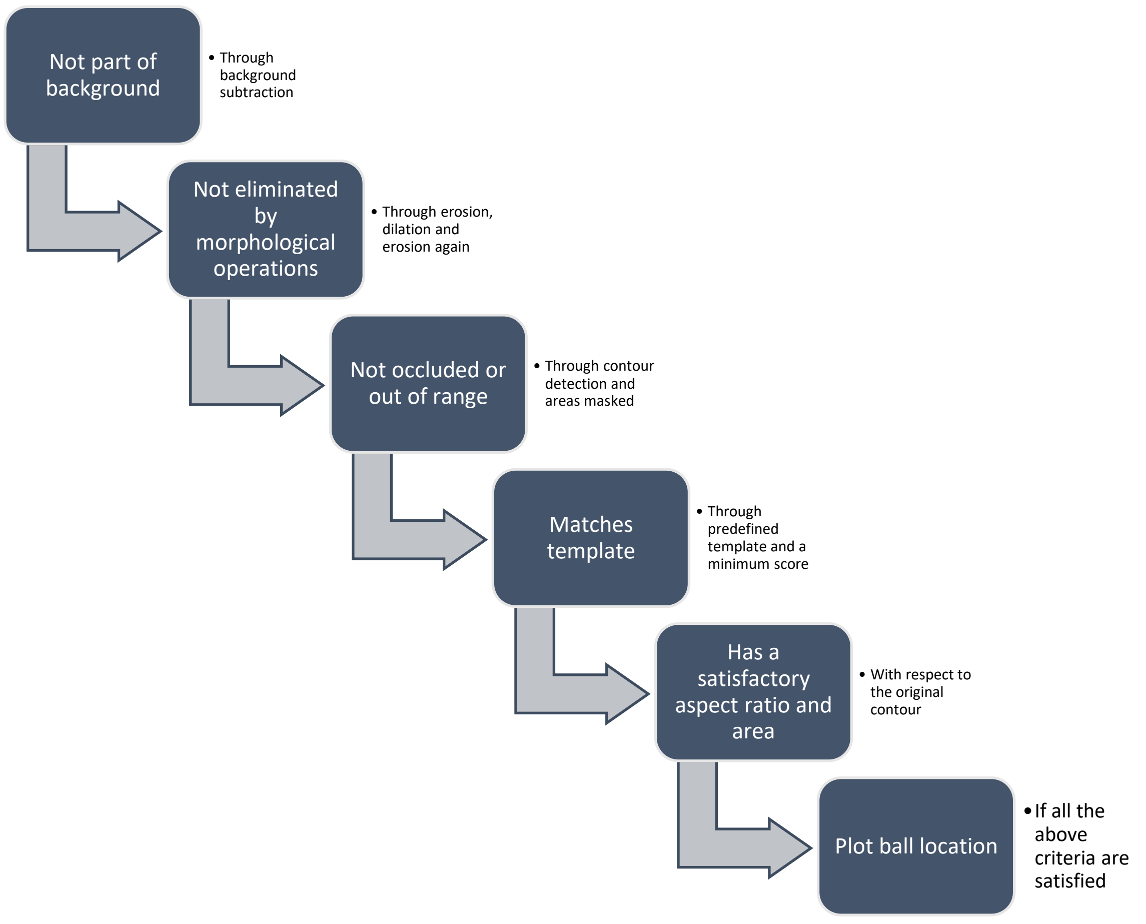

Not part of background
• Through background subtraction
Not eliminated by morphological operations
• Through erosion, dilation and erosion again
Not occluded or out of range
• Through contour detection and areas masked
Matches template
• Through predefined template and a minimum score
Has a satisfactory aspect ratio and area
• With respect to the original contour
Plot ball location
• If all the above criteria are satisfied

Figure 1: Flowchart of designed system.

Background Subtraction

All six videos were shot using static cameras – they did not move across the ground with the ball, rather, the ball moved in and out of the frame. This factor was exploited through background subtraction to distinguish moving objects from the remainder of the frame. The benefits of this were two-fold:

- Computational times for remaining processes leading to ball detection were significantly decreased by narrowing down the search area to moving objects, rather than the whole frame;
- Other methods, such as thresholding of the green turf was less accurate as it was bound to vary with lighting conditions, weather, etc.

The background subtractor by OpenCV was used to perform this task, through the below code which was applied to each frame:

```
object_detector =
cv2.createBackgroundSubtractorMOG2(history=100,varThreshold=40)
```

This background subtraction method selected used a Gaussian Mixture Model to model the background, and thus adapted to minor changes, introduced by illumination, for example. Two parameters were passed to the function; history – length of the history (100 frames), and varThreshold – defined as the threshold on the squared Mahalanobis distances between the pixel and the model to decide whether a pixel is well described by the background model. The latter is unrelated to the updating of the background, but decides which pixels are retained as background. These two parameters were adapted heuristically to suit the application.

A varying number of pixels were set for each video, to eliminate the upper portion of the videos which includes continuously changing banners on the perimeter of the ground. This sacrificed ball detections occurring within this range, for a lower number of false positives. The resulting objects were used as the 'mask' for the video.

Morphological Operations

While the players and ball were well distinguished through the background subtractor, noise pixels were observed in the vicinity of the moving objects. Furthermore, certain gaps were created which separated, for example, a player's hand from the rest of his body within the mask. The features of the arm alone were more resemblant to a ball, and lead to a false positive from the ball detection. Likewise, false positives resulted from shoes, socks, and even heads of players. Thus, a process of three morphological operations was instantiated, combining erosion, dilation and a second dilation using OpenCV. The first erosion performed noise removal. This was followed by dilation, which expanded any unfilled gaps within the players bodies, among others, and increased the size of the ball. The last erosion smoothened the edges and cleaned the edges of the moving objects. A 3×3 kernel was used for all morphological operations, since it gave better results than larger kernels. A larger kernel caused a ball moving object to be merged with a player to an undesirable extent, giving rise to a high number of missed detections. The images shown below depict nearby frames, before and after morphological operations were applied.

```
kernel = np.ones((3,3),dtype=np.uint8)
```

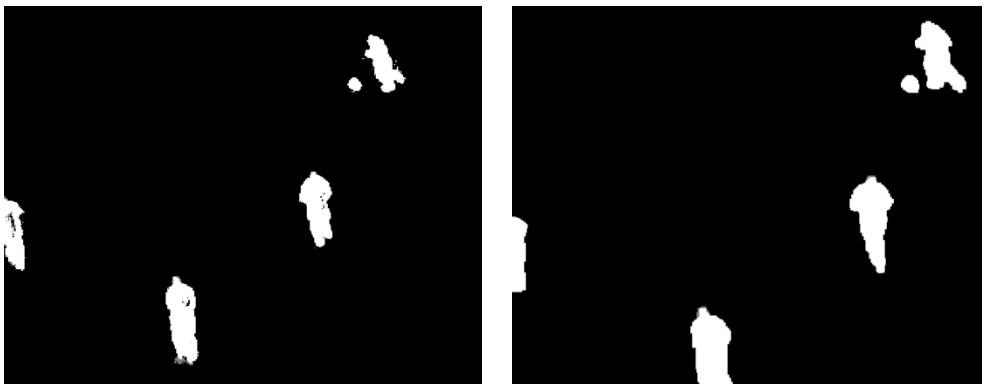


Figure 2: Mask before (left) and after (right) morphological operations are applied.

Contours

The last pre-processing step entailed obtaining the contours from the resulting mask image. This was carried out through OpenCV's findContours function, with the mode set to RETR_EXTERNAL, which retrieved only the extreme outer contours. This was desirable, since it helped in reducing the breakdown of objects into smaller ones. Furthermore, CHAIN_APPROX_NONE was used to save all the boundary points, rather than CHAIN_APPROX_SIMPLE, which excluded excessive points. This preserved as much information as possible from the contours. These parameters were implemented through the below line of code.

```
contours, _ = cv2.findContours(mask_erode, cv2.RETR_EXTERNAL,
                               cv2.CHAIN_APPROX_NONE)
```

Region of Interest

From each contour, the region of interest was generated to increase the accuracy of the results. The bounding rectangle of the convex hull was generated to retrieve properties of x-position, y-position, width and height, required to generate a rectangular region of interest from the original image. Lastly, the region of interest was converted to grayscale before being passed to detection.

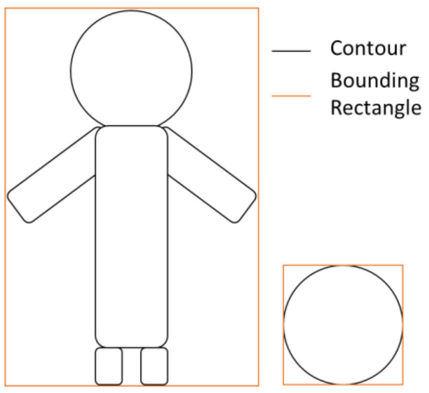


Figure 3: Sketch of contour vs. bounding rectangle.

```
hull = cv2.convexHull(cnt)

xrect, yrect, w, h = cv2.boundingRect(hull)

ROI = frame[xrect:xrect+w,yrect:yrect+h]

ROI_gray = cv2.cvtColor(src=ROI, code=cv2.COLOR_BGR2GRAY)
```

Score Generation

A score for each region of interest was generated in the form of an array having four values, as shown in the table below, and computed as per the following description.

Table 1: Score array.

0	1	2	3
Template matching	x co-ordinate	y co-ordinate	radius

Template Matching

Template matching was performed on all regions of interest generated, which were greater than the template itself. A check was implemented for the size of the region of interest to ensure that no errors occurred at run time when the template could not be convolved over the image due to dimension mismatching. When template matching could not be performed, a value of '0' was assigned to the score. A circular template was used for all images, relative to ball shape, with each video having its own template size. The template was created as black on white in grayscale, to match the region of interest. One such template is shown below.

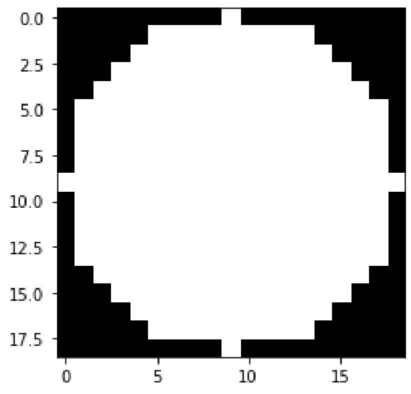


Figure 4: Example of template.

The template matching function provided by OpenCV, matchTemplate, was implemented using the squared difference method. This method indicates a perfect match using a zero score, and a bad match using a large score. This method was chosen over other available ones, such as the correlation method, as it provided a slightly more advanced computation. Furthermore, the returned match values were simpler to analyse, since when the difference was squared, all values were returned positive, while other methods did not always generate positive values.

Another function by OpenCV, minMaxLoc, was implemented following template matching. The minimum match value from the template matching was retained, to hold the best match, and place it in the score array.

```
tmpl_match =
cv2.matchTemplate(image=ROI,templ=ball_tmpl,method=cv2.TM_SQDIFF)

min_score, _, min_loc, _ = cv2.minMaxLoc(tmpl_match)

score.append(min_score)
```

Minimum Enclosing Circle

In addition to template matching, another feature was added to strengthen the results further. From the information retained of the bounding rectangle earlier, a check was added on the width-to-height ratio. This was done to restrict the possible detections to those more likely, having a minimum aspect ratio of 0.7. The aspect ratio was chosen high enough to eliminate extremities, but low enough to salvage correct detections of imperfect ball depictions in the frame, as explained in the discussion section of this report.

Furthermore, the minimum enclosing circle of the contour, through OpenCV's minEnclosingCircle, was obtained to calculate the detected ball's area. While various methods were available to retrieve a form of bounding box, circle, etc. around the contour, a circle was chosen as it distinguishes footballs from miscellaneous objects best.

One method used throughout testing was the convex hull. As shown in the sketch below, the contour of the player yielded a much smaller area than the convex hull of the same player. Furthermore, the area of the minimum enclosing circle was even larger. In the classification process, the area of the ball was set to lie within a range; keeping the area of the ball and the area of the players further apart yielded better results. If the classification was to be made on the contour area rather than the circle area, the range would have to be far more specific, and would still lead to a higher number of misclassifications.

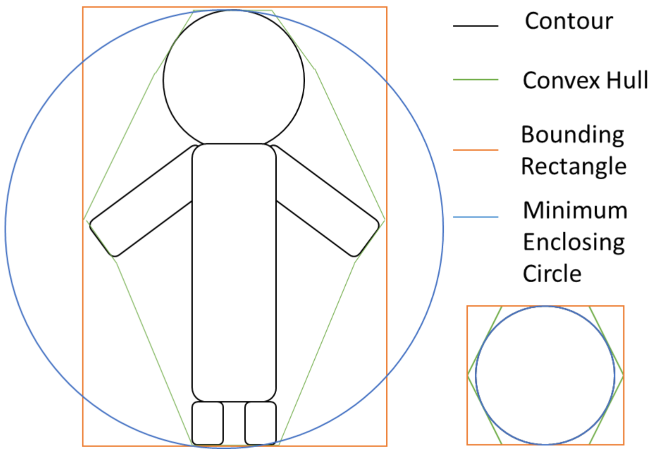


Figure 5: Sketch of minimum enclosing circle.

Each video had its own allowable area range, which was configured to match approximate apparent ball size with respect to the camera distance from the ground. If the bounding rectangle did not fall under the permissible aspect ratio, or fell out of range of the permissible areas, the last three values of the score array were set to 'zero'. If all conditions were met, the centre and radius of the minimum enclosing circle were stored into the score array.

```
(x,y), rad = cv2.minEnclosingCircle(cnt)

area = math.pi*rad*rad
```

Ball Plotting

With the score array at hand, a ball detection could be produced. It was decided that the ball would be plotted, if, and only if, a score was generated by both the template matching and the minimum enclosing circle parts of the code. In doing so, the data to be analysed was cleaned up, speeding up computational time, and the number of false positives was reduced further. These were categorized into an array of 'eligible' data. Since only one detection was possible on each frame, the values corresponding to the minimum value (best) template matching were retained and used to plot the detected ball, of all the eligible data. The ball was plotted using a blue circle, with centre and radius as per the minimum enclosing circle.

For frames where no ball was detected, the centre of the ball was set to (2000,2000). These values were chosen as they are pixel values greater than the dimensions of the video and can never be mistaken as true coordinates. These values also facilitated the results generation process.

Ground Truth Plotting

To facilitate the monitoring of results in real time, the ground truth of the data set was also plotted on each frame. The data was loaded using the pandas library to read the csv and convert it to a numpy array. The annotations were in the format (frame, x, y); corresponding to frame number, and the x and y coordinates of the ball. Where a frame contained no ball, the annotations contained the symbol '-' (declared as *char*), for both coordinates. Therefore, during video processing, the frame number was stored to plot the annotations to the frame, where they did not contain the '-' symbol, i.e. where they had been provided. Since the ball radius was not provided, this parameter was arbitrarily chosen 15. The ball was plotted in red.

```
if not any(c in x_ann for c in char):

    x_ann = int(x_ann)

    y_ann = int(y_ann)

    cv2.circle(frame, (x_ann,y_ann), 15, (0,0,255), 3)

    cv2.putText(frame, "Actual", (x_ann,y_ann+(2*15)),
     cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)
```

Saving Results to a csv File

The last step in the ball detection part of the code was to save the results to a csv file, using the csv library, in the same format as the ground truth was presented. This made results analysis more straightforward by consistent formats. File paths were declared early in the code when video-dependent parameters were declared. To ensure that only the actual detections were saved, only values which did not have the coordinates (2000,2000) previously described, were saved to the csv file. When no true positive was declared by the detection, the results were saved as (frame, '-', '-'), as the ground truth was presented.

Displaying of Frame and Mask

Lastly, two images were displayed; the output frame, and the mask upon which computations are based. These lines of code, together with the csv saving lines can be toggled between comment and code, as required.

Code Structure

All code was separated into various functions to facilitate any required corrections. The first function, called *preprocessing*, computed background subtraction, morphological operations and contours. The *gen_score* function computed template matching and minimum enclosing circle, together with any relevant computations, and saved the score array. The *plot_ball* function analysed the resulting data and plotted detections. Similarly, the *plot_groundtruth* function analysed the ground truth data and plotted where necessary. Lastly, the *results_tocsv* function saved the results in the aforementioned format in the corresponding csv files. Indentation was done carefully to minimise computational time. The only function called for every contour is the *gen_score* function. The rest need only be carried out once per frame.

Generation of Metrics

A separate notebook was created to distinguish football tracking from results analysis, and to break down the task into individual components. To run this code, the path to the results generated, the path to where the true positives were to be saved and the path where the ground truth was located were to be provided prior to running the code. To ensure the data was correctly collected and prevent errors at a later stage, the number of rows and columns in the generated results data, and the ground truth data, were checked. If they were not equal, an error message was displayed.

A threshold was set for the false positives, being located 50 pixels away. This meant that any detection further than 50 pixels from the ground truth was eliminated from the data. This new data of true positives was saved to a new csv file for future reference. Again, the data structures were analysed to prevent unnecessary errors. The scikit learn library mean_absolute_error function was used. This function did not accept the '-' symbols, and therefore, to prevent skewed results, the data had to be altered. On every instance of a '-' symbol in the true positive data, both the true positive data entries, and the ground truth entries were changed to 'zero' values. The mean absolute error (MAE) for the x co-ordinates was calculated by:

$$MAE_x = \frac{\sum_{i=0}^{n} \operatorname{abs}\left(x_{ann_i} - x_{res_i}\right)}{n}$$

Where x_{ann} is the co-ordinate provided by the annotations, and x_{res} is the co-ordinate generated by the results, over n total results. The same can be computed in the y-direction.

Furthermore, the sensitivity and specificity were calculated. While the MAE was the recommended metric for the assignment, the architecture chosen proved that missed detections and false detections were a greater issue then inaccurate detection. In fact, the following section shows that the MAE proved to be very low. The sensitivity is the ratio of the number of true positives in the original data to the number of true positives in the ground truth, while the specificity is the ratio of the number of true negatives in the original data to the number of true negatives in the ground truth. These two metrics provide much greater insight into the effectiveness of the ball detection of the designed system by focusing on the number of missed detections, as well as the number of predicted detections.

Results and Discussion

The table below shows the metrics extracted from the second Jupyter notebook which was used to generate results. The true positive results are saved to a new csv file, and the below metrics are printed at the end of the notebook.

```
print("Mean Absolute Error for x co-ordinates: ", x_mae)
print("Mean Absolute Error for y co-ordinates: ", y_mae)
print("Sensitivity (TP rate): ", sens,"%")
print("Specificity (TN rate): ", spec,"%")
```

Table 2: Table of results.

	MAE (x co-ordinates) (px)	MAE(y co-ordinates) (px)	Sensitivity (%)	Specificity (%)
Video 1	1.571	3.110	30.000	98.413
Video 2	2.761	3.006	19.900	95.316
Video 3	3.364	4.376	10.529	89.036
Video 4	2.670	2.420	15.864	90.296
Video 5	1.762	2.533	33.228	92.024
Video 6	1.840	2.600	38.911	84.251

The table above shows that the MAE, computed over the true positives, is relatively low, as desired. The largest error is of 4.376 pixels. The frame height is 1080 pixels for video 3, and thus the error is of only 0.41% over the frame height, which is practically invisible to the human eye. Furthermore, the specificity of all the videos is quite satisfactory, indicating that true negatives are mostly correctly identified. The sensitivity, on the other hand is quite low for all the videos, with the highest sensitivity being 38.911% for video 6. There are several contributing factors to such a low sensitivity, which will be the focus of the remainder of this text.

Low Number of True Positives

The specificity of the video was better than the sensitivity of the video because the majority of the frames of most of the videos did not contain a ball. By analysing the annotation files, the percentage of frames containing a ball, and those not containing a ball can be found:

Table 3: Proportion of frames containing a ball.

	Frames containing ball (%)	'Blank' frames (%)
Video 1	24.35	76.65
Video 2	26.65	73.35
Video 3	52.32	47.68
Video 4	47.15	52.85
Video 5	10.54	89.46
Video 6	8.57	91.43

Comparing Table 3 with the Table 2 shows that the higher the number of frames which contain a ball, the lower the sensitivity. This was attributed to the way the ball appears within the frames. Having such a high proportion of the frames containing a ball corresponded to a higher variance in the characteristics of the ball, in terms of its size and shape, as the ball traversed the ground multiple times from varying distances from the camera at varying speeds. This meant that the parameters were harder to fine tune for better results across all instances of ball location, as explained further in the respective sections of this report.

Additionally, the lowest sensitivity results were obtained from Videos 3 and 4, which also contained the highest number of video frames which included a ball. These two videos recorded the match from the centre of the football ground, which were the busiest areas of the game, containing most of the players for a lengthy time. The ball often traverses the centre point of the football ground as the match takes its course.

Ball Appearance: Shape

As the ball traverses over the ground, it's appearance is more elliptical than circular. One such instances are shown in Figure 6 below. This frame was extracted by observing the annotation files and noting the most drastic changes in the x direction between successive frames. It was then manually cropped to show the ball clearly. In fact, the ellipse's longer axis is visibly in the horizontal direction, due to its rapid change in that direction. It was due to these ellipses that template matching did not always perform well when the ball moved very quickly. In fact, a template having an ellipse shape was created, but not used. A circular template was preferred over an elliptical template because the elliptical shape of the ball was not fixed. Proportion and direction varied as the ball traversed the ground. A horizontal ellipse did not satisfy most of the elliptical instances of the ball. For a comprehensive ball detection using an elliptical template, more than one template would have been required, having various orientations. This would have increased the computational time greatly. On the other hand, the circular template reduced the computational time at the expense of some missed detections since only one template matching exercise needed to be performed. Furthermore, the circular template did not necessarily miss all elliptical instances as it was convolved over the images.

To decrease the number of missed detections, or false negatives from the enclosing circle, the minimum permissible aspect ratio $(width/height)$ of a ball was set to 0.7 for a score to be generated. This allowed elliptical shapes to pass through, within limits.

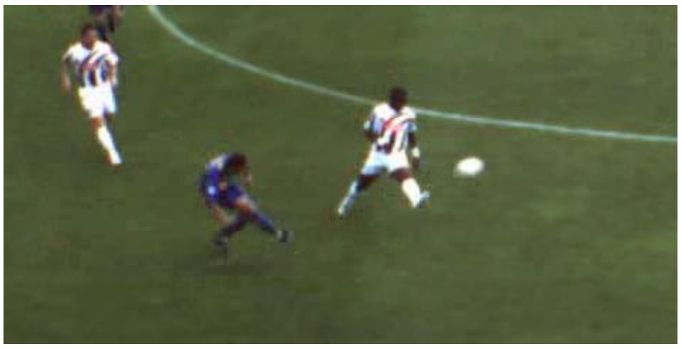


Figure 6: Elliptical appearance of ball.

Ball Appearance: Size

In a similar manner to how the ball's shape changed due to its velocity, the ball's size also varied. While at a far distance from the camera, the ball occupied x number of pixels, when it was nearer to the camera, the same ball occupied y number of pixels, where $x < y$. This made the detection process harder, in terms of template matching, and through area. The former was more problematic since the size of the circular template was predefined. If the template was defined too large, it would miss smaller instances of the ball, while if it was defined too small, more false positives occurred on players' shoes, etc. The area of the ball was easier to adjust, since a range was set, attempting to leave out very small objects, and very large objects. The range was adjusted for each video to compensate for minor differences between video clips.

To put the above into perspective, two frames from Video 1 can be observed. On the following page, Figure 7 correspond to frame numbers 117 and 278 and show the ball at two different locations in the ground. In frame number 117, the ball is significantly closer the camera then it is in frame number 278. To the naked eye the difference may appear minor and thus a yellow circle, approximately the same size as the ball in frame number 117, has been manually placed over both frames. There is a visible difference in the size of the ball between the two frames, when comparing the yellow circle with the ball in frame number 278. Furthermore, the frames' widths and heights are 1920 and 1088 pixels respectively, therefore when considering that the templates radii range from 6 to 9 pixels over the whole dataset, any small adjustment will have a remarkable effect on the outcome.

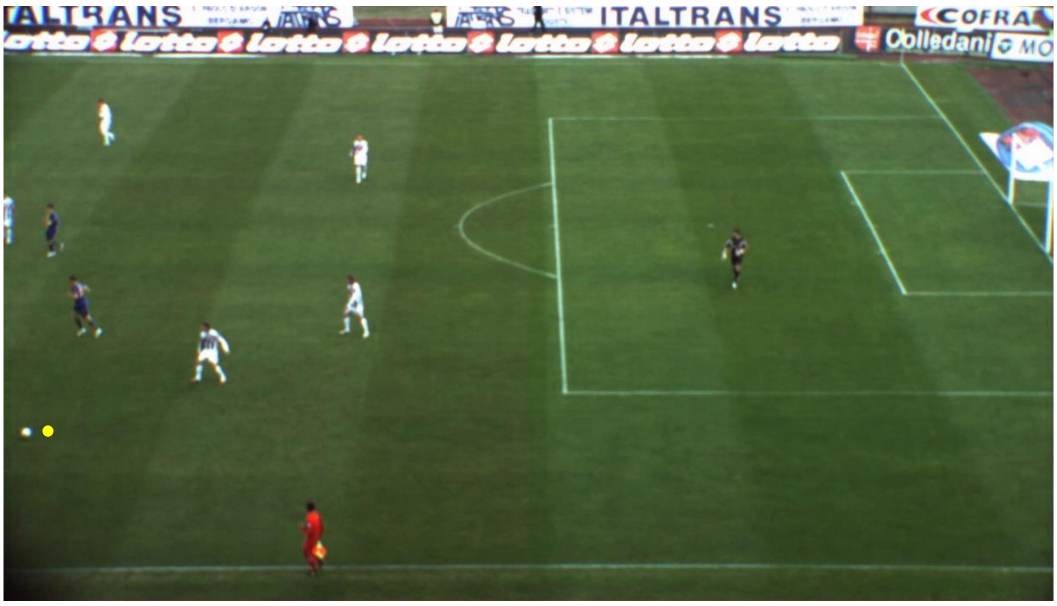
This was especially problematic for those video clips which contained most instances of a ball within a frame. More specifically, it was problematic when the ball appeared in various parts of the frame; close to the camera, and far from it, since a good template was harder to deduce.

Occlusions

The pre-processing performed on the image was beneficial in merging broken objects into one, but also resulted in some occlusions of the ball when it was too close to the player's feet, but not visibly occluded on the frame. One such occlusion, taken from Video 1, is shown in Figure 8 on the following page. While the ball is easily detectable to the human eye in both the frame and the mask, the computations could not be carried out since it was merged with the player. Had the objects been distinguishable, the true positive rate would have increased, as this issue was repeatedly observed.

Masked Image

A mask was extracted from the original frames which included a parameter which varied with each video. The intention of this parameter was to mask out the side line banners which are constantly changing, and lead to a high number of false positives. While the false positives were decreased, this also eliminated possible detections occurring in this band. It was implemented since the reduced number of false positives outweighed the reduced number of true positives.



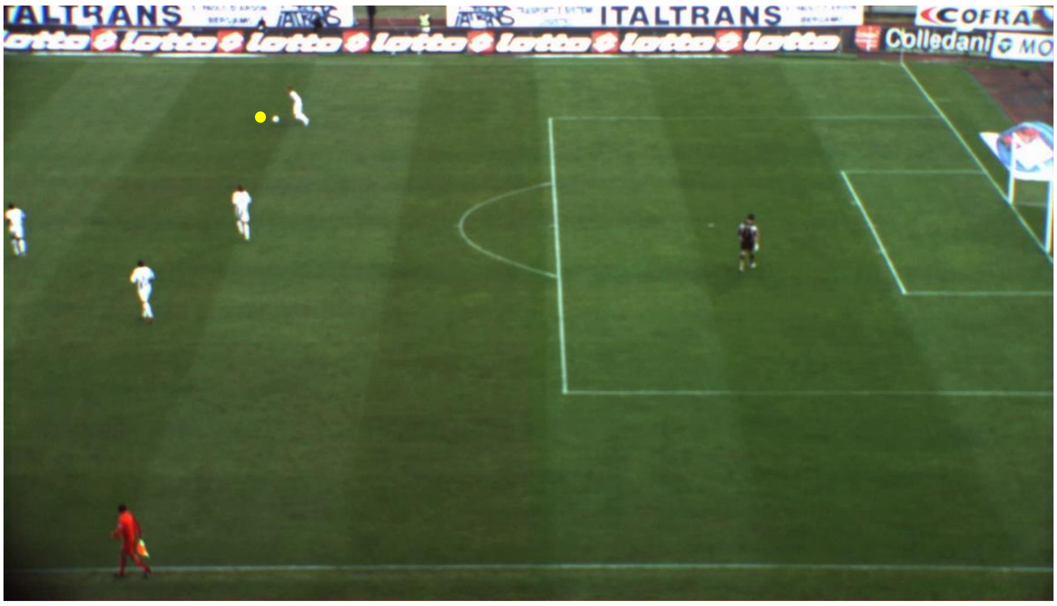


Figure 7: Ball close to camera (above) and far from camera (below)

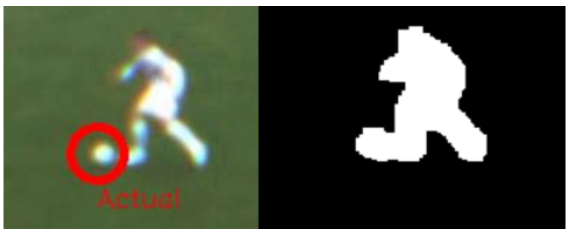


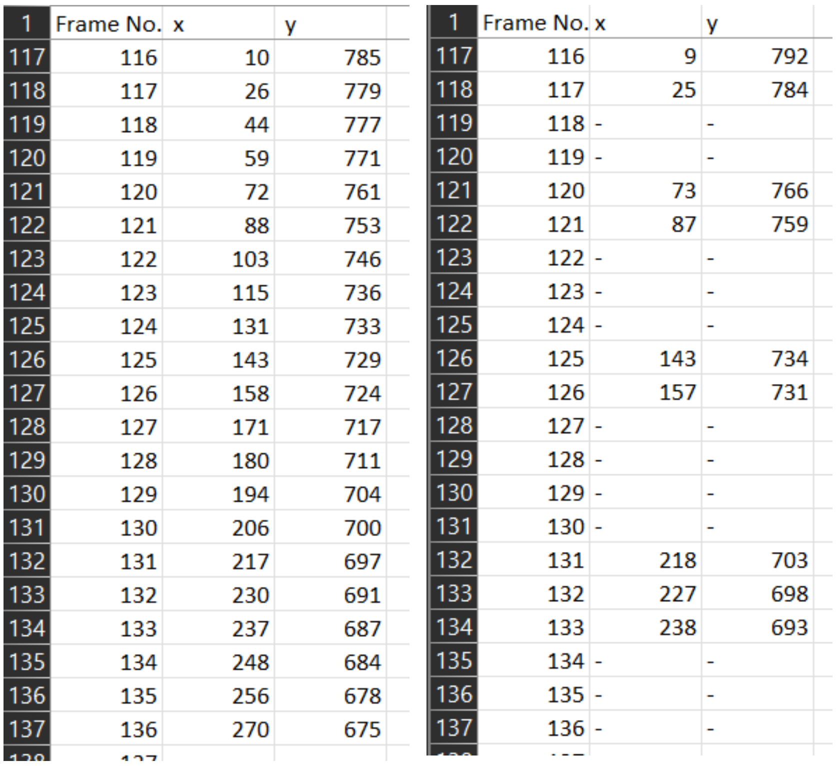
Figure 8: Occluded ball.

Future Improvements

To combat some of the issues determined with the code, some improvements can be made to improve the sensitivity. These suggestions were not implemented into the presented assignment due to time constraints but are considered suitable given the existing flaws.

Interpolation of Ball Position

A glimpse at the true positive results generated showed that while ball detections were quite frequent, they were picked up and missed within the close proximity of a small number of frames. Demonstrating this, a snapshot of the ground truth and true positives of Video 1 are shown in Figure 9 below.



1	Frame No.	x	y
117	116	10	785
118	117	26	779
119	118	44	777
120	119	59	771
121	120	72	761
122	121	88	753
123	122	103	746
124	123	115	736
125	124	131	733
126	125	143	729
127	126	158	724
128	127	171	717
129	128	180	711
130	129	194	704
131	130	206	700
132	131	217	697
133	132	230	691
134	133	237	687
135	134	248	684
136	135	256	678
137	136	270	675

1	Frame No.	x	y
117	116	9	792
118	117	25	784
119	118	-	-
120	119	-	-
121	120	73	766
122	121	87	759
123	122	-	-
124	123	-	-
125	124	-	-
126	125	143	734
127	126	157	731
128	127	-	-
129	128	-	-
130	129	-	-
131	130	-	-
132	131	218	703
133	132	227	698
134	133	238	693
135	134	-	-
136	135	-	-
137	136	-	-

Figure 9: Annotations and true positive detections of video 1 (extract)

Consider frame numbers 118, 119, 122-124, 127-130 of the true positives. These show no detected ball, however, follow and precede correct detections. Since the movement of the ball between successive frames is considerably small, interpolating the values such that the true positive detections are maintained should increase the true positive rate. This comes at the cost of displaying the video with a delay of a small number of frames. Given that the videos are recorded at 25 frames per second and considering a maximum interpolation across five frames, this would only result in a 0.2 second delay. The linear interpolation between frames i and $(i + n)$ for the x co-ordinate was derived and can be computed through:

$$x_{i+j} = x_i + j\left(\frac{x_{i+n} - x_i}{n+1}\right)$$

Where $0 \leq j \leq n$. The same can be applied for the y co-ordinate.

While this would increase the MAE slightly due to error introduced by interpolation, the number of true positives, based on the same threshold of 50 pixels would be improved. This interpolation was manually applied to the above data to evaluate expected improvement. The last three frames which contained missed detections could not be interpolated since there were no correct detections after the missed ones.

Table 4: Interpolations carried out on data extract

	Ground Truth		**Interpolated**		**Difference**		**True Positive**
Frame No.	**x co-ordinate**	**y co-ordinate**	**x co-ordinate**	**y co-ordinate**	**x**	**y**	**Yes/No**
116	10	785	9	792	1	7	Yes
117	26	779	25	784	1	5	Yes
118	44	777	41	778	3	1	Yes
119	59	771	57	772	2	1	Yes
120	72	761	73	766	1	5	Yes
121	88	753	87	759	1	6	Yes
122	103	746	101	752.75	2	6.75	Yes
123	115	736	115	746.5	0	10.5	Yes
124	131	733	129	740.25	2	7.25	Yes
125	143	729	143	734	0	5	Yes
126	158	724	157	731	1	7	Yes
127	171	717	169.2	725.4	1.8	8.4	Yes
128	180	711	181.4	719.8	1.4	8.8	Yes
129	194	704	193.6	714.2	0.4	10.2	Yes
130	206	700	205.8	708.6	0.2	8.6	Yes
131	217	697	218	703	1	6	Yes
132	230	691	227	698	3	7	Yes
133	237	687	238	693	1	6	Yes
134	248	684	-	-	N/A	N/A	No
135	256	678	-	-	N/A	N/A	No
136	270	675	-	-	N/A	N/A	No

The resulting metrics of the extract of data before and after interpolation are compared in Table 5 below. The MAE was computed over the resulting true positives, as before, and a change of less than 1 pixel was observed. However, the overall sensitivity increased to more than double its original value. The specificity was not computed as it was beyond the scope of this experiment. The results obtained from this interpolation are promising, and can be added as an additional function to the existing code in the future.

Table 5: Metrics of extract before and after interpolation

	MAE_x **(pixels)**	MAE_y **(pixels)**	**Sensitivity (%)**
Before interpolation	1.111	6.000	42.857
After interpolation	1.267	6.472	**85.714**

Multiple Templates

A system employing different templates can be utilised to improve sensitivity by selecting the best template for the scenario. This decision can be carried out through some deep learning techniques, or through other heuristic aspects. This should be carried out with caution, as it may bias the detections by choosing a template that would most likely result in any detection, rather than one likely to deduce a correct detection.

There are numerous ways this can be implemented:

- Use two different sized circular templates, one slightly larger than the other. For contours located some distance considered 'far' from the camera, the smaller is used. For contours located some distance considered 'near' to the camera, the larger is used.
- Use multiple elliptical templates, having various orientations, together with circular templates. The decision on which template is used is based on the trajectory of the contour (faster speed is indication that elliptical templates are ideal), while circular templates are reserved for more slow moving objects.
- A combination of the above the techniques.

Concluding Remarks

The task of ball detection in football games is a very relevant and interesting topic which can be approached in various ways. The main conclusions drawn from the above analysis are explained below.

The static cameras made background subtraction relevant. Had the cameras been moving, the approach would have to be different, perhaps by thresholding some intensity levels, or colour channels (such as the green channel), out of the frame.

Template matching is not as straightforward as detecting a circular ball. As the ball presents itself in different forms, the template must be adjusted.

While the results yield an excellent MAE and specificity, the sensitivity can be improved through linear interpolation, as demonstrated previously.

Combining numerous factors to produce a detection created a more robust system, however tuning it became an increasingly complex task. The system was perfected for the n^{th} parameter, and then require re-tuning of all *n* parameters upon the addition of the $(n+1)^{th}$ parameter.

The OpenCV documentation website was referred to for information on the functions it employs: https://docs.opencv.org/4.x/index.html. Last accessed 27/12/2021.