Tracking Football Players

An investigation into tracking amateur football players with multiple videos.

Ryan Sewell
ID Number: 9085100

Supervised by Dr Aphrodite Galata
May 1, 2018

A report submitted as part of the Third Year Project for the Degree of BSc (Hons) Artificial Intelligence

School of Computer Science
The University of Manchester

Abstract

My aim for this project was to build a program, or a set of programs, to track football players throughout multiple video feeds which would also include the player's location on the pitch. This would allow me to create, for example, heat maps and how far a player has travelled in a game.

The data used to obtain the statistics would be multiple video recordings taken simultaneously at various angles around the pitch. The final system produced is two python programs. The first program is used to enable the transformation of the video recordings data into a single plane, obtaining a transformation matrix. This would produce a common floor plane where all of the data will be projected onto. The second program detects players in every camera. It detects individual player's locations in each camera and then projects them into the joint floor plane. The player's location on the floor plane is found by merging the individual projected player locations. Subsequently, it will then try to match the detected points with the predicted points of current tracks. Each point is allocated a track number which is then passed back to each camera, as the point and track number.

Due to the lack of availability of multi-view football data, the proposed method has been tested on surveillance data without loss of quality.

Acknowledgements

I would like to thank my parents for helping me out and keeping me going throughout this project. I would also like to thank my tutor, Dr Aphrodite Galata, for guiding me in the right direction and also keeping me going. Lastly, I would like to thank my housemates who let me ramble on about my project and helped me when I was struggling.

Contents

1. Introduction

The tracking of objects in video can be difficult to start with. However, the tracking of multiple fast moving objects, in multiple videos, becomes increasingly difficult.

In this report I shall discuss several way to achieve the goal of tracking humans in a video or multiple videos. I will also consider the strengths and weaknesses of my project. The aim of this project was to create a system that allowed the user to acquire information regarding a player's performance during a match. As the system will be used postgame live feeds are not needed. The videos must be accessible after the system has run. The problem of tracking objects through a video is a challenging one. There are several reasons for this but the main ones are:

1. It is hard to track team sports. The individual players are fast moving, and often use quick changes in direction in order to confuse opponents, which makes them very hard to track. Pedestrians are easier to track as generally they do not change direction sharply. In addition, their pace is a lot slower.
2. Tracking could be dropped when players run out of the frame. This may happen when a player is substituted as in football or basketball.
3. Occlusion happens a lot due to players moving continuously. Tracks may get lost, or players misidentified when participants cross behind each other.
4. The identification of players on a team is a really difficult part of tracking. The majority of players on a team, as in the case of football, wear the same strip so cannot be distinguished by colour. It is not really feasible to use player's faces as the image may be blurred due to low resolution.

The above are general problems that occur in sports object tracking.

However, as my system was intended for a children's game several other things had to be taken into consideration. Low cost would be advisable meaning that it should be able to run on cheaper machines and things like GPS tracking are not feasible. However, the footage will most likely be of an inferior quality. Weather conditions must also be taken into consideration. For the majority of the time it will be used outside. Weather conditions can change rapidly. In England, clear skies can be quickly followed by heavy rain and then back to clear skies during the course of a single event. In addition, spectators standing very close to the side of the pitch need to be taken into consideration for either exclusion or detection in the tracking.

In this project I am focusing on dealing with the problem of occlusion using multiple cameras and projecting data in a common plane will resolve the large number of ambiguities due to occlusion between players.

2. Background

The main reasons for the project were firstly to be able to track players in multiple low quality videos and secondly to create an accurate representation of player's locations on the pitch at any given time.

In this chapter I will describe how I looked at object tracking including techniques, algorithms and the concepts discovered during my research. In the past, many ways have been used for player tracking in sports. There are even more options in the larger subject of object tracking. Whilst detection is still a key component, the use of multiple cameras for detection and tracking allows for a little bit of leeway.

2.1 Research

Paper [1] suggested a way for player detection and tracking with multiple cameras. This involved working with a single camera first and then incorporating the results from the remaining cameras. The first step in this process is foreground detection. By using a pitch mask false alarms from the spectators are removed. This gives an output of the foreground, hence the name. Following this, the next stage is single camera tracking. By using the bounding boxes from the foreground detection it is possible to create, or update, tracks. The next step required a category measurement to detect between the players, goalkeepers and referees. After the single camera tracking is complete, the multiple camera tracking is then undertaken. The aim is to match every current track with every detection from every camera. If detections are not matched a new track is created. Un-matched detections from other cameras will then try to be combined. The final step would be track selection.
In team sports the number of players can vary. As there are a maximum number of tracks available for use, the 25 most likely tracks are utilized.

Paper [2] describes a way of tracking players on a basketball court and then mapping the players. This is done in a five step process:

Court Detection
Individual Detection
Colour Classification
Player Tracking
Position Translation onto a court map

For court detection a multitude of techniques and algorithms are used such as erosion, dilation and canny edge detection. Individual detection is done through Histogram of Oriented Gradients. This was used together with a SVM to identify players. Additionally, they used the HOG detector from OpenCV; a very accessible HOG detector with default data set for pedestrians. Players are then labelled based on colour. This is done by thresholding in HSV rather than RGB as it has higher discriminations between colours. The tracking is done by using tracks, so if detection points are under the threshold for the track it will update with the new position, or else look for one close by and finally adding tracks and dropping tracks. Mapping is achieved by affine transformation on the court giving a matrix for the transformation. By using this on the detected players, you obtain the player's location on the court.

Paper [3] uses conditional random fields to identify players with a robust way to track with a moving camera, at a medium distance. There are three steps to the player tracking in this case of player detection, team classification and then player tracking. In using three different features extraction types have been used to create visual words, of MSER, SIFT and RGB. These are used to train a classifier for an appearance model. Conditional random fields are then used to link images of the same player together. It then makes sure that no player appears twice with mutual exclusion edges between all players in a frame.

In this project I have utilised ideas from all the above papers to solve multiview player localisation and tracking.

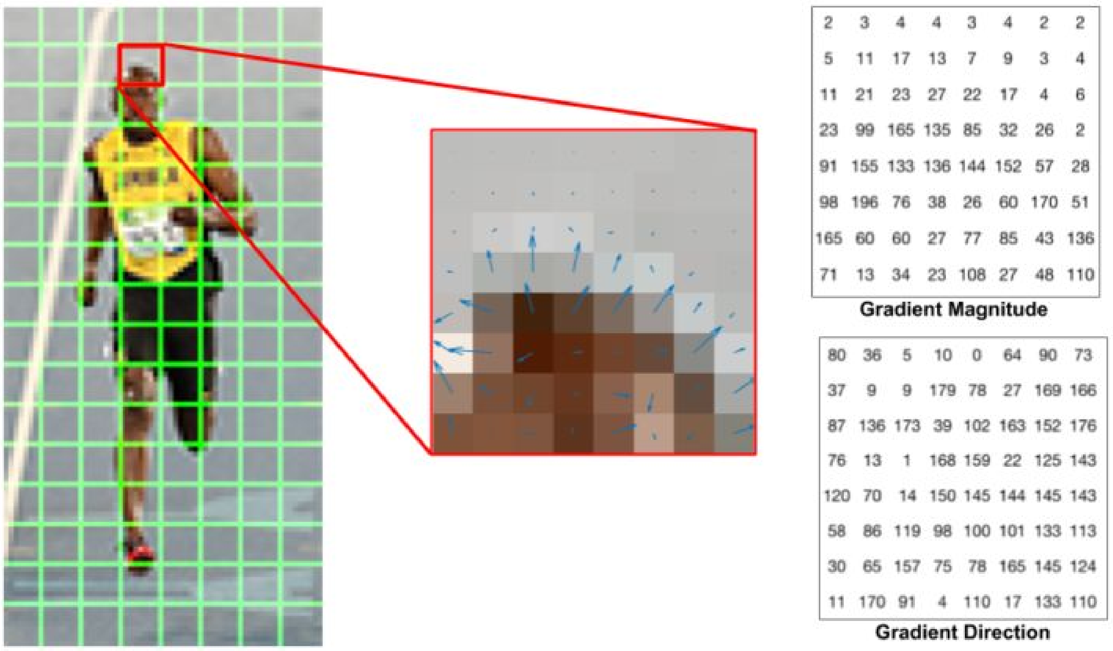
2.2 Detecting

The first step in trying to track players on a football pitch is to detect them in the image. There are multiple ways to do this; the ones I looked at, and felt were a good way to do it, are below.

2.2.1 Histogram of Oriented Gradients (HOG)

With this way of detecting people, it very closely resembles the way a human would look at an image and detect an object or a person. It can be used to great effect for detection of things like people, but with things in common shapes, like a circle or a square, it is not as good. HOG detection has been used in pedestrian detection to great effect.

A HOG feature detector uses the gradients in an image 8x8 pixels (Seen in Figure 1).
The next step is to create a histogram of gradients for each 8x8 cell, with 9 bins, 0, 20 , …,160 (Seen in Figure 2). The histogram created can then be used as a feature vector for that part of the image. Using these features you can train a SVM [10] to detect people. It is then used on the images and finds the location of people in the image.



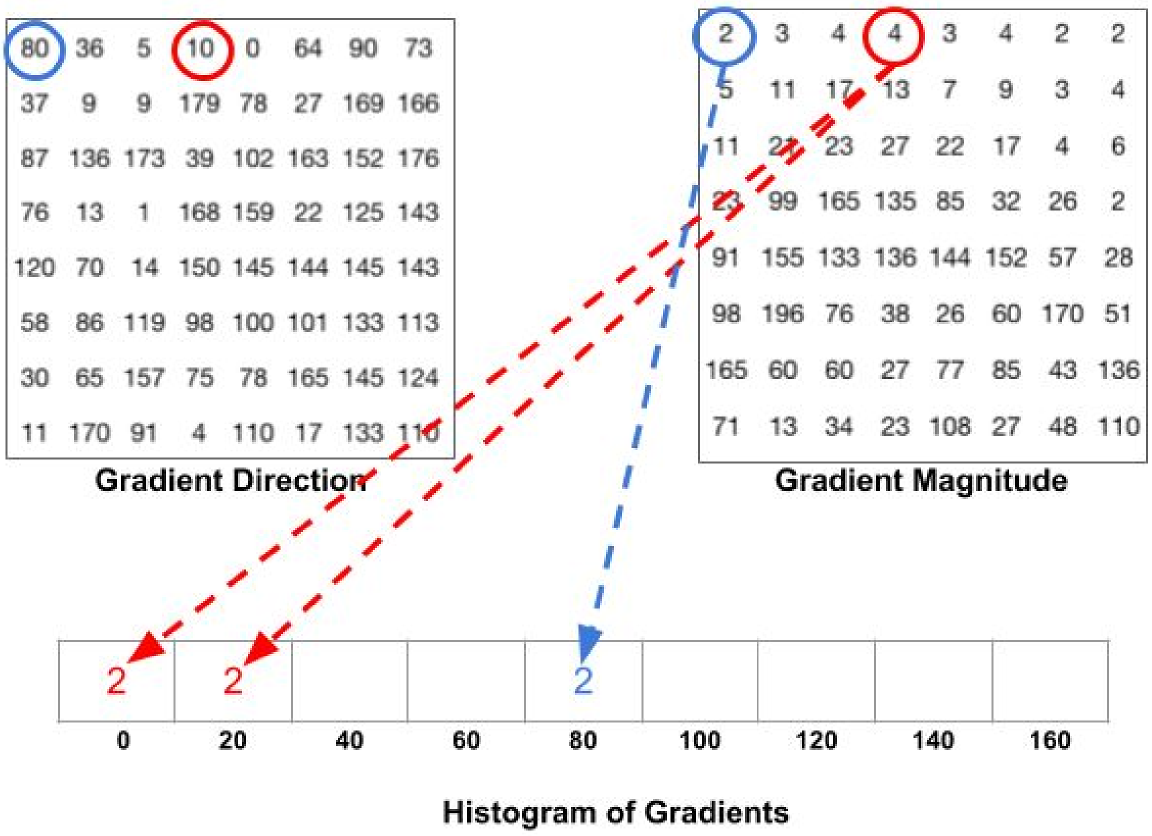
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Figure 1. Gradient Magnitude and Direction. Taken from [4]



80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Figure 2. The direction and magnitude put into the histogram. Taken from [4]

2.2.2 Background Subtraction

Background subtraction is a quick and easy way to detect motion in an image.

The background is a model of everything in the frame that does not move too much. The things of interest, in this case the movement of people as an easily identifiable difference between the two. The model is constantly updated with things that have not moved much from the background, or that stop moving after a while and become the background. If we have a model of the background and a new image to test it against, what you would need to do is subtract the background from the new image. This would leave you with the difference between the two, with a greater value if there has been a bigger difference between the new image and the background. The next step is to use a threshold on the new image. This gives a two tone or binary image, so the difference is clearly visible. With this binary image it will be easy to see the cases of slight movement, such as leaves moving. So to get rid of this you put the image through a filter. This will get rid of patches that are a few pixels wide.

There are many types of filters to use, all of which give similar but different results:

The normalized box filter makes the value of the new pixel to be the mean of the surrounding pixels.

The Gaussian Filter is seen as the filter that gives the best results but does not work as fast as the others. It looks at the pixel and those surrounding it. Those that are further away have less weight.

The final main filter used is the median filter. It takes the median of the surrounding pixels and uses that value.

All taken from [5].

When you have used one of these filters, the next step is to do a threshold again as it will not be a binary image anymore. What is now left in the second binary image is where there is a big difference between the new image and the background model. You now need to find where these are in the image. So you do blob detection (OpenCV library). You then have the x, y, width and height of the blobs. You then use these values to draw the detection on the original image.

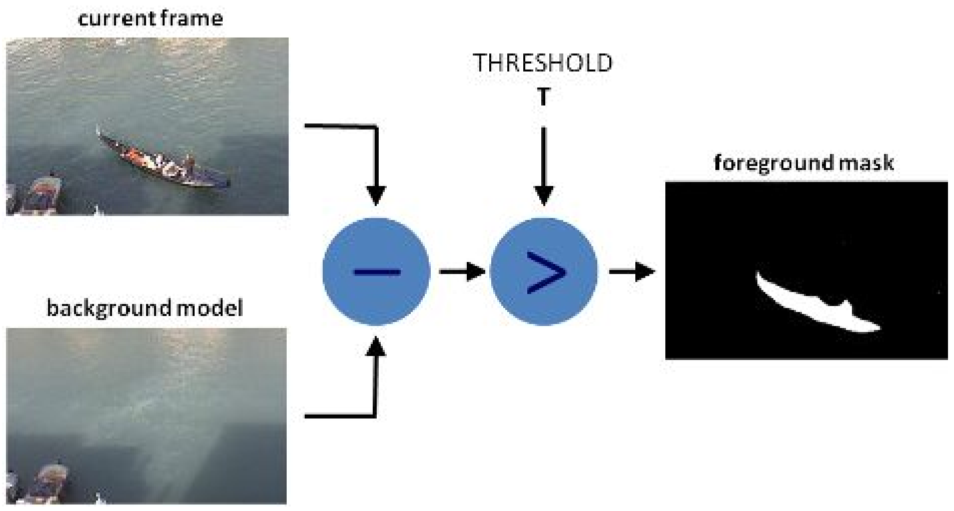


Figure 3. Background Subtraction. Taken from [6]

2.3 Mapping to Floor

When working with multiple cameras you need to be able to make them work with each other. The way to do this is to put them all onto a local plane i.e. the floor/pitch. There are quite a few ways in which this can be achieved as shown below.

2.3.1 Camera Calibration

The first thing I looked at to try and get the camera detections working with each other, was to try and calibrate the cameras to automatically get their own position relative to the scene and to each other. This would be useful as it would mean that the program would not have to be edited each time it is used at a new location. This works by holding up, for example, a checkerboard of size 7x8. You say how many columns and rows there are as well as the size of the boxes. The camera can then work out its position and the other values as seen in Figure 4. Looking at Figure 4, the point I would work out would be the point at the bottom of the tree in the 3-D object. This is a fast way to do this, as once the cameras have their locations it is just one equation to map the location onto the pitch.

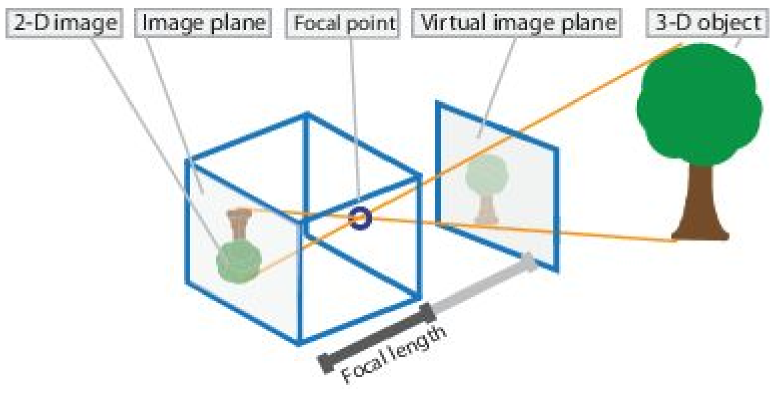


Figure 4. Finding object point with camera location. Taken from [7]

2.3.2 Geometric Transformations

Another way to map the detections onto a local plane is to do a perspective transformation. This is where you transform each camera onto a plane that is the same for all of them. With the created transformation matrix, you can then just use it on the detections made putting all detections into the same plane. This is a very simple and quick way to get a floor map. A simple perspective change on a Sudoku image is shown in Figure 5.

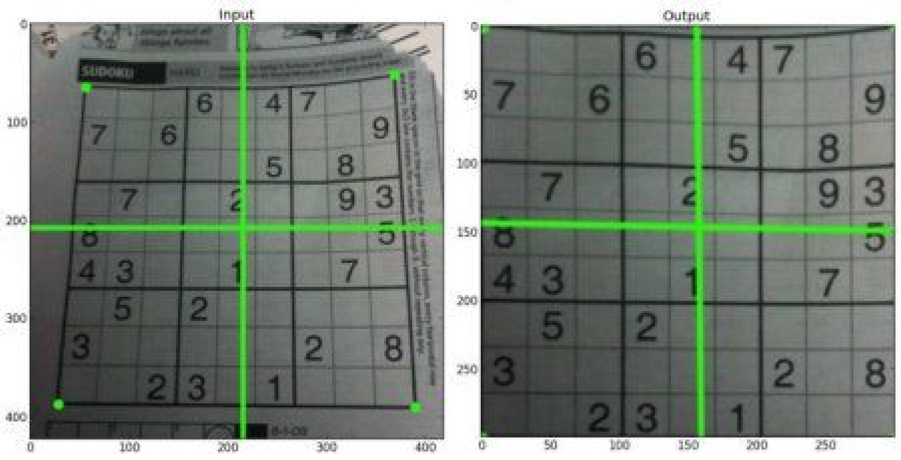


Figure 5. A simple perspective change. Taken from [8]

2.3.3 Epipolar Geometry

The final way I looked at was epipolar geometry. This is where you have two cameras looking at the same image from different angles. With these two cameras we can then make the images into 3D space. With the multiple cameras, you would have to do this multiple times, so would not be as efficient. This would solve the problem of mapping onto a singular plane as well as merging points in one as it should give one detection for one person.

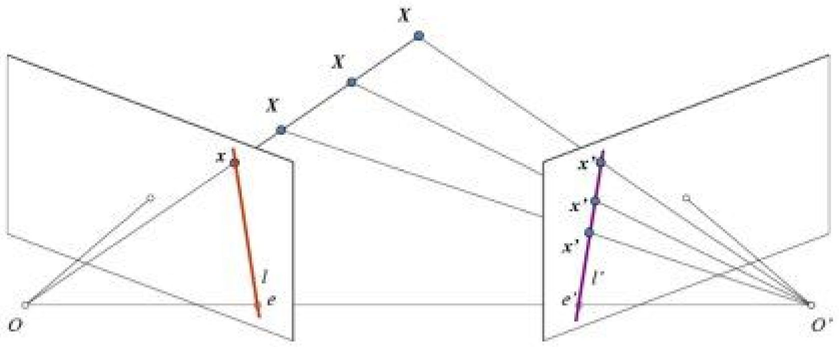


Figure 6. An example of Epipolar Geometry. Taken from [9]

2.4 Merging and Tracking

With managing to get the detections onto a local plane there is then the problem of having multiple detections per object. So somewhere during the process of tracking you need to be able to merge the detections from each camera.

For the merging process you will first put all detections onto the local plane. Then go through the detections from one camera, one by one and match them up with the detections from another. You do this via a threshold, so if it is close enough find the midpoint between the two and delete both of the originals.

The tracking is a bit different as you will have a list of all the tracks. In this you will have an array of all positions the track has been at, if it is active and how many frames since it had a detection. You have the array of all positions so you can draw the track on the image as well as using it to make a predicted position. You need to know if it is active as if it is not you can just skip that track as it is not needed anymore. You need to keep how many frames it has been since there has been a detection as you drop them after so many frames. Once this is all done on the local plane, you will do a transformation back to each camera with the inverse matrix.

2.4.1 Track then Merge

The first option is to do tracking per camera. Then using the tracks transform them onto the local plane and merge tracks with the same track number together and then try to merge the leftover tracks together.

2.4.2 Merge then Track

The other option is to do the merging process first. So you will transform all of the detections onto the local plane and then try to match up with the detections from the other cameras. Once this is done you will then start the tracking.

There are advantages and disadvantages of both options. However, I felt that merging the points onto a local plane and then tracking (2.4.2) is the better option for a few reasons. The main reason I choose to merge then track is that if the tracking in between the objects is not the same for one camera compared to the other, it means that the tracks might go in different directions giving incorrect results.

3. Development

There were several ways to split this project up. The ones I chose were the sub problems of player detection and player tracking. At the start of the year I created a plan for the project. I outlined how I thought the system development would progress. I knew that the plan would probably change as my ideas progressed and I discovered new ways to achieve tasks.

3.1 Player Detection

The detection of players is a key component of this project as nothing else would be able to work without it. In order for it to work to its full potential good detection is necessary.

Throughout the course of the project I used two main ways of detection:

Background Subtraction
Google's TensorFlow

3.1.1 Background Subtraction

Background Subtraction is a viable option in this case as all the cameras are static. This allows for a representation of the background to be made. When a person enters the frame there is considerable difference to the background. When the two are subtracted we obtain the difference between them. By then applying a threshold a very clear difference is obtained.

After this has been done, since the cameras are not fixed solidly, there can be motion of the camera to take into consideration as this will be seen as movement from the subtraction and thresholding. With the resulting image, a filter needs to be applied. This is used to remove those parts of the image which have very little movement or noise.

Then you threshold the image again, as with the blur it will not be in black and white. It is necessary to remove the grey. What remains on the image will be various sized blobs where a lot of movement has occurred between the background image and the current image. It is then necessary to find the location of the blobs. By finding the contours, this enables you to find the encompassing rectangles of each blob. Then, by checking the size of each blob you are able to discard the ones that are of no use. Now you have the location and size of every major change in the frame, so go back to the original image and draw the boxes found on the image. The process is seen below in Figure 7. (a) is the original image, (b) is when the background has been subtracted, (c) is after the filter has been applied and (d) is with the boxes drawn around the detection points on the original image.

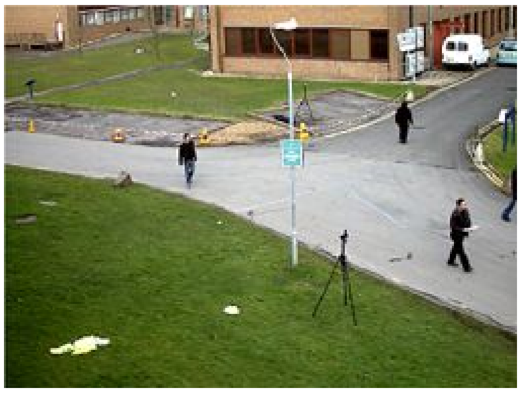


Figure 7(a)

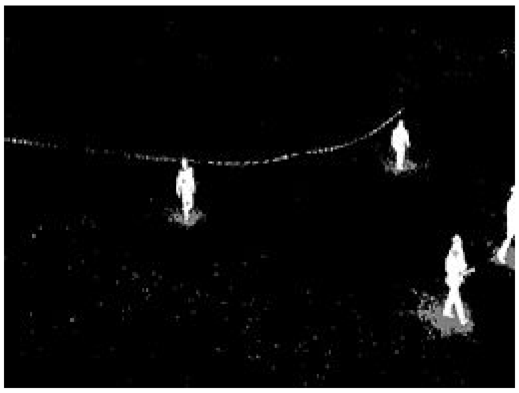


Figure 7(b)

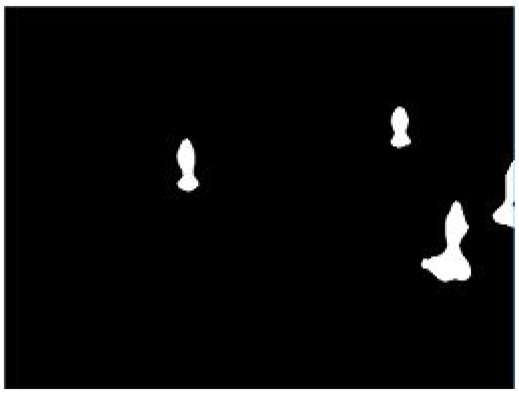


Figure 7(c)

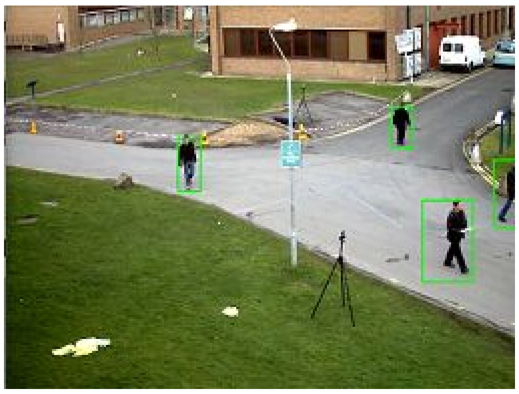


Figure 7(d)

Figure 7. Background subtraction process taken from a test video

3.1.2 TensorFlow

As the project progressed, problems occurred when using the Background Subtraction such as occlusion and losing objects if they did not keep moving. I tried many ways to correct these problems but I felt they were not good enough. I needed the accurate location of an object. This led me to seek alternatives.

I looked at things such as HOG detection but eventually decided that TensorFlow [11] was the best way to go as it gives a very good outline of where people are. This is required for the tracking of a player as I will need to use the middle point between the lowest points. This would be the player's location on the floor. Whereas, in the case of a single camera the middle point would be used for detection as this would be the most stable point and the most consistent.

TensorFlow is an open source software library for high performance numerical computation. TensorFlow computations are expressed as stateful dataflow graphs. Where nodes are the operations and the edges are tensors (multi dimensional arrays). For the training with TensorFlow there are a lot of pre trained models for use of varying speed, quality and the number of things that can be detected [12]. I chose to use the fastest model as it included the detection for humans and once it has started takes approximately 5 minutes to process a minute of video on my laptop. Meaning a full training session of an hour would take five hours to process; a 90 minute game would take seven and half hours to process. Another benefit of using TensorFlow is that you can easily add in your own images to train as a new object for the model to detect or to add to an existing model.

When the process has gone through the stateful dataflow graphs it will produce detections with a certainty degree and the name of the detection. If the highest certainty percentage for that detection is above a threshold it will enclose the detection, identify the object and the level of certainty. Figure 9 shows how TensorFlow was working on footage. The TensorFlow elements of the program are in green. The blue dots are from the tracking described later in the report.

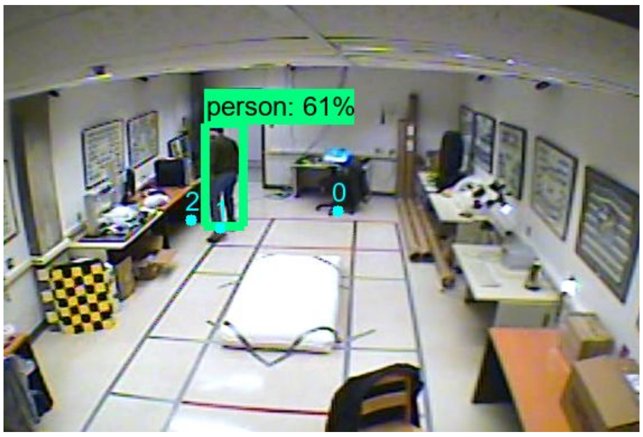


Figure 9 (a). Correct Detection.

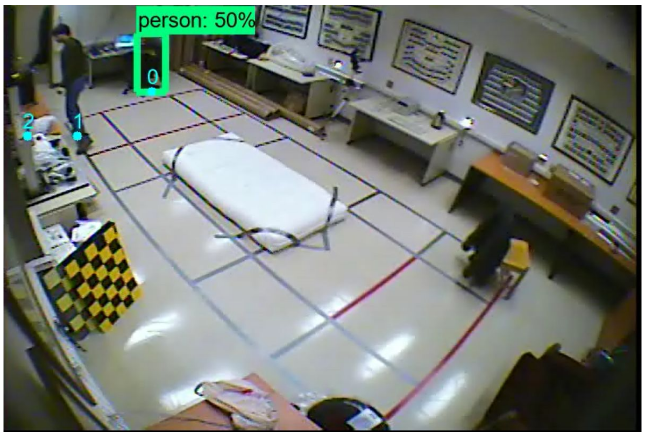


Figure 9 (b). Incorrect Detection

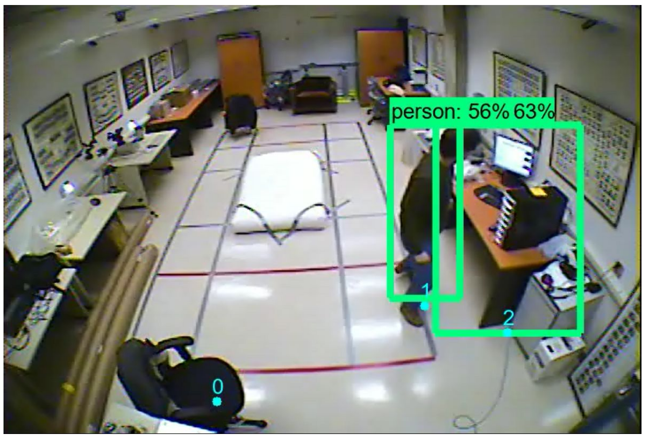


Figure 9 (c). Correct and Incorrect Detection - 1 and 2 respectively.

3.2 Player Tracking

There are several ways to carry out player tracking, as described in section 2.4. Firstly, I looked at the tracking in each individual camera then merging these points. The tracks in each camera might move in opposite directions to each other but still be seen as the same track. This means that the track would be wrong and with this interference others would be as well. It could go quite badly wrong. The second option was to merge the points from each individual camera and then do the tracking. With this option it would be a lot harder for things to go wrong. I decided to choose the second option.

3.2.1 Merging

There are a few ways to merge the detection points from each camera on the floor plane. This first is to merge the detection from one camera with another cameras detection point, within a certain threshold. This new point is then merged with the detection from the final camera if it is under the threshold. The other way is to look at the detections from the first camera and see if there are any detection points from the other cameras within range. If there are, you then look for detection points within range of that point excluding the first point. Then find the average position of all three. The problem I found with this way was that if the second and third points were in the opposite direction to each other it would not as likely be picked up. Whereas, with the first option if the first two points merge first it has a higher chance of being in range for the third point.

The way I chose to go, was to obtain all of the points from every camera and then transform them all onto a local plane i.e. the pitch. In doing so it allows the points from all of the cameras to be combined together. With them all together on one plane, this allows me to merge points from different cameras that are close to each other as one new point. So it finds all the points from one camera on the plane and then one by one tries to match them with points from another camera. If a match is made the middle point is taken as a new point and the matched pair are deleted.

Any points that have not been matched are added to a list of merged points. This list is the final merged list for those two cameras. This happens for all cameras two at a time. So if there were four cameras, the first two cameras would look for points to merge. Then the second two cameras would do the same thereby leaving two new lists of merged points. The final two are merged together which leaves a single list of all the points.

This is shown in diagram form in Figure 10. With (a) being the original three points, (b) being the merged point between one and two and (c) being the merged point of three and the first merged point.

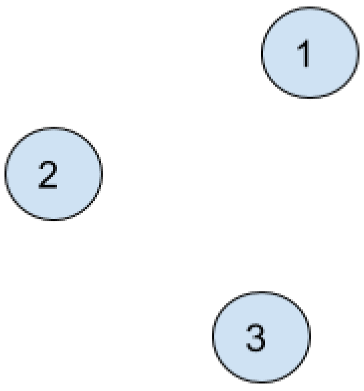


Figure 10(a)

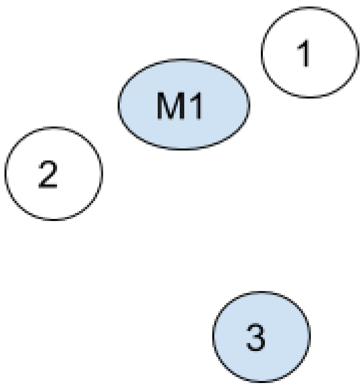


Figure10(b)

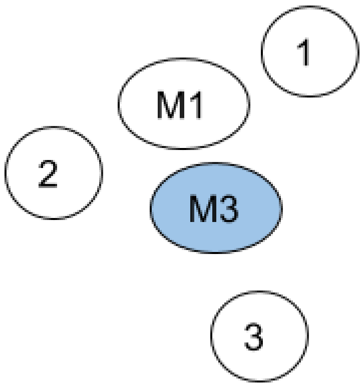


Figure 10(c)

After we have obtained the final points we need to look at the tracking. A single track would be a list of coordinates of where one object is from frame to frame. Also, whether the track has not ended, if it has not been used and if it is close to the predicted position of the track. The predicted point of the track is the difference between the most recent track position and the position before that. The calculation I used was two times the most recent position minus the previous position. If a match is not found for five frames the track will end. Additionally, if there is no track to match to the merged detection point a new track is created. When a new track is made it is not possible to produce a predicted point for the next frame, so it just uses that first position.

So to recap, detections from each camera are put onto a local plane; the playing area. These points are merged to give an overall position for the detection on the plane. This is used to compare with the current tracks which are updated where possible or new ones are created. The current tracks are passed back to each camera; the location is shown with the track number.

Figure 11 shows one track for the one person as well as all the previous positions of that track. Part (a) shows it on the floor plane. Parts (b), (c) and (d) show the track points in each camera view after it has been transformed back into the original plane for each camera.

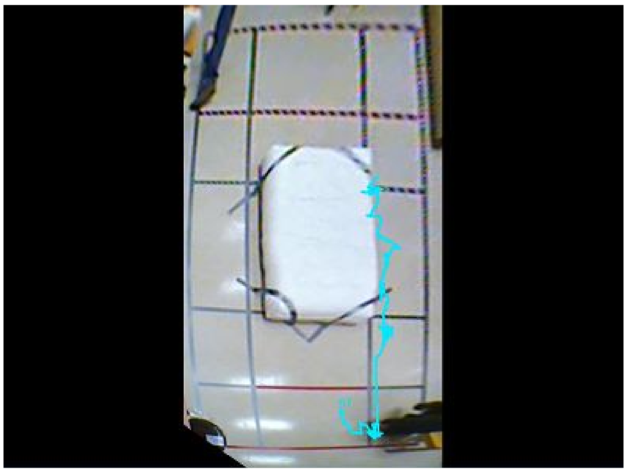


Figure 11(a)

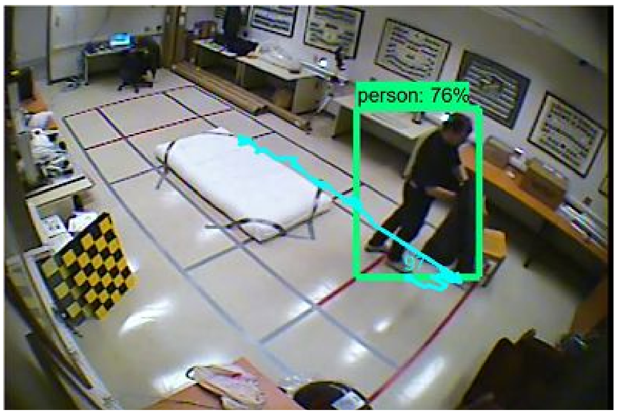


Figure 11(b)

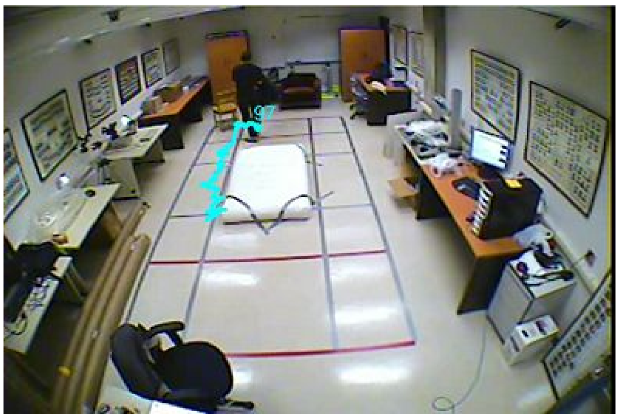


Figure 11(c)

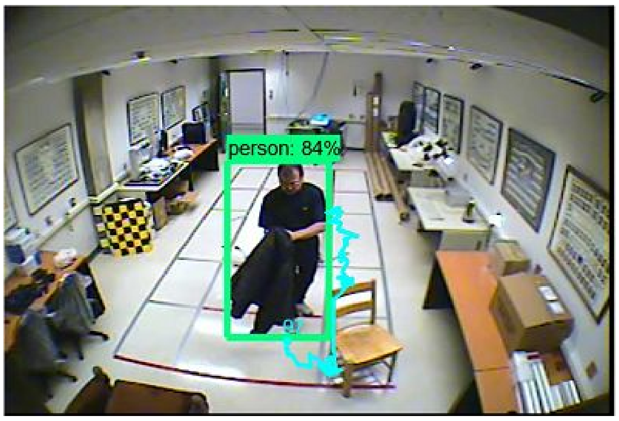


Figure 11(d)

4. Evaluation

In order to evaluate my program's performance I need to collect data on how well my program manages to detect and track people in each camera. This is a hard task as to hand annotate each frame, from 3 sources in a 40 second video, would take up too much time. So, an efficient way to collect the data is required.

4.1 Extraction of Data

The first thing I feel would be useful, when extracting the data, would be to split the videos up into separate parts so that I am able concentrate on certain things at a time. To start with, I could look at the detections in each frame for one person. Then, in another, look for multiple people in each frame.

In order to evaluate how well this system works, there was information I needed to collect and consider regarding the scene:

- Number of people in the frame
- Person Location

The first point will be useful in discovering how well the detection worked. I can use that as a quick way to determine if there was the right amount of detections per frame. Then, look through the video and find if any detections were wrong when there was only one frame.

The second point will be useful for when I wish to see how accurate it is for the tracking. I can do this by looking at the three sources I used and manually inputting the location frame by frame. However, in order to speed up the process, I will look at every ten frames as ground truth data.

When collecting the data there are three thresholds which I can change. So, when looking at each section the other two thresholds will not change. To start with the merge distance is set at 50 pixels and the tracking distance is set at 70 pixels.

4.2 Player Detection

The way I carried out person detection gives a value for the accuracy of that detection. When detecting it will only accept above a certain threshold. To test what the best values would be I decided to increment the threshold by 5% each time. I also have to take into consideration that I have multiple cameras so this must be done for all of them. By looking at both precision and recall, this can give a good understanding on how well this worked. The certainty of the detection is what will be used as a threshold, as the higher the certainty the more likely it is to be correct but will not pick up as many detections.

4.2.1 Precision

Precision is the number of correct detections over the course of the video. So, the formula for it would be the number of correct detections divided by the number of frames.

Precision = (Correct Detections / Number of frames) * 100

Threshold (Certainty)	Camera 1 Accuracy	Camera 2 Accuracy	Camera 3 Accuracy	Overall Accuracy
50%	79.6%	62.1%	97.5%	98.6%
55%	75.0%	55.8%	95.0%	98.6%
60%	69.6%	53.3%	91.3%	97.5%
65%	62.1%	49.2%	86.7%	96.7%
70%	55.0%	43.8%	82.5%	95.0%
75%	42.1%	37.5%	75.0%	91.7%
80%	31.7%	32.1%	61.7%	85.8%

Table 1

From these results you can see it stays very accurate overall. Even if two cameras drop into a low accuracy, the third still allowed for the overall accuracy over the three cameras to be higher. In the case at 80%, camera 1 and camera 2 have an accuracy of just over 30% but when added in with camera 3, which has an accuracy of double the other two, brings the overall accuracy up to 86%.

4.2.2 Recall

Recall is a bit different to precision as you now look at the total number of detections and the correct detections. In order to obtain the recall accuracy you need to divide the number of correct detections by the total number of detections.

Recall = (Correct Detections/ Total Detections) * 100

Threshold (Certainty)	Camera 1 Accuracy	Camera 2 Accuracy	Camera 3 Accuracy	Overall Accuracy
50%	189/199, 95.0%	145/168, 86.3%	233/258, 90.3%	90.5%
55%	178/185, 96.2%	131/147, 89.1%	225/246, 91.5%	92.3%
60%	165/169, 97.6%	124/139, 89.2%	218/230, 94.8%	93.9%
65%	148/151, 98.0%	115/125, 92.0%	206/213, 96.7%	95.6%
70%	131/132, 99.2%	102/108, 94.4%	197/199, 99.0%	97.5%
75%	103/103, 100%	90/90, 100%	179/180, 99.4%	99.8%
80%	76/76, 100%	77/77, 100%	148/148, 100%	100%

Table 2

When looked at together, these results give a good understanding of how well the detection worked. This data shows that when the threshold is higher you get a greater recall i.e. there are a lot less false detections. Whereas, a higher threshold does loose precision as there are frames where no detections are made. These values are a very close approximation of the accuracy. If it finds a false positive and doesn't detect anything else it will say that this detection was correct. I have attempted to mitigate this as much as possible by going through all three videos at each percentage by hand, in order to find as many of these occurrences as possible and input them into the results.

4.3 Merging Points

For merging the points on the floor plane I used a distance threshold to see if the detections from another camera were within range to be merged or not. If points are able to be merged the midpoint is found and used as a new detection and the two original points are deleted.

The way I decided to test how well I managed to merge the points to the actual location of the player was to go through the video looking at every 10 frames and manually getting the location. This was then saved in an array and used for all of the testing. This gives reliable results as when trying to do it manually for every test the results were not very dependable. This was because with the location changing for every run, even minutely, when looking at a difference of 1 pixel between results it changes it drastically.

So when changing the threshold this will give a smaller range for merging, meaning there will be more detections made. However with there being more detections the overall distance will be down. This is because it is likely, if it does not merge, one might be slightly closer than the other.

Merging Value (Pixels)	Average Distance (Pixels)	Highest Track Number
30	29.3	36
40	30.7	34
50	31.6	33
60	32.7	35
70	34.3	28
80	36.4	25
90	37.7	25

Table 3

From these results you can clearly see that with increasing the threshold, more tracks are kept active giving a better overall tracking. However with increasing the threshold this also increases the average distance between the track and the actual location. This is obviously not what we want as the main aim was for tracking accurately.

4.4 Tracking

I carried out the tracking from frame to frame in a similar way to the merging, with a threshold for the distance in which tracks can be updated. This is obviously is a crucial point for the project.

The way I decided to test how well the tracking performs was to change the threshold on which the tracks look for detection points. How I tested how close they were to the actual point was the same as in 4.3 and used the same points. I used the actual point and worked out the distance to the tracked point. I also took into consideration the track number as a higher track number would mean more tracks have been made instead of old ones being updated.

Track Distance (Pixels)	Average Distance (Pixels)	Highest Track Number
30	34.3	61
40	34.3	43
50	34.3	39
60	34.3	32
70	34.3	28
80	34.3	27
90	34.3	25

Table 4

From this table we can see that the average distance does not change when changing the threshold value of how close the tracks are matched with a detection. At first this might seem surprising. However, when you take into consideration that the detection points are all still there, when it is lower what might have been picked up at a higher threshold is not now.

This means that a new track is created, meaning that the distance will not have changed but will become a new track. For the number of tracks as expected, the number becomes lower as the threshold becomes bigger. As with the bigger thresholds it can pick up detections from further away so are less likely to be dropped. This helps when the detections are not consistent and jump around a bit.

5. Reflection

In the context of my project, I feel that it has been a success with the way in which players are detected and tracked. I feel that many of the initial goals of the project have been accomplished.

- With the background subtraction it accurately located players over the 3 cameras. When I switched to TensorFlow it became a lot more accurate.
- People are identified really well with precision and recall at the most optimal 95% and 97.5% respectively.
- In the case of projecting the points onto a local plane and merging detections from each camera it works effectively, as having a average track distance of under 35 pixels in a 900x500(450,000) pixel image is great.
- The tracking is conducted very well with tracks staying consistent throughout the running of the videos.
- With the use of multiple cameras it has greatly increased the accuracy of the results as well as the consistency.

The project could be improved in a few ways to help accomplish the goal.

- To be able to identify the players would be a good next step as this would greatly improve the tracking. This would occur by identifying people from each camera and seeing if they are able to merge on the floor plane and have a separate active track for each person.
- Another improvement would be to smooth out the tracks; they can be a bit jumpy as the detections points can move around a bit. This is because the points transformed are at the bottom of the detection and not at the centre which would alleviate some of this movement.
- You could also get the program to automatically work out where the pitch was and what orientation it has according to its location.

6. Conclusion

The main project aim was to create a system that managed to track football players through multiple views and have an accurate pitch description of player location. With static multiview video data the program is able to detect and track players through multiple views, as well as creating a floor plan of the objects detected. There are three steps in the process of doing this. The detection of people, the merging of detection points from each camera onto the floor plane and the tracking of the detections. The data extracted shows a great detection rate and a very good tracking system.

References

[1] "TRACKING FOOTBALL PLAYERS WITH MULTIPLE CAMERAS Ming ...." https://pdfs.semanticscholar.org/5e9c/856bb932e62e5d4d36632e99bc81d2da852b.pdf. Accessed 23 Apr. 2018.
[2] "Player Tracking and Analysis of Basketball Plays." https://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Cheshire_Halasz_Perin.pdf. Accessed 23 Apr. 2018.
[3] "Identifying Players in Broadcast Sports Videos using Conditional ...." https://www.cs.ubc.ca/~murphyk/Papers/cvpr2011-camera.pdf. Accessed 24 Apr. 2018.
[4] "Histogram of Oriented Gradients | Learn OpenCV." 6 Dec. 2016, https://www.learnopencv.com/histogram-of-oriented-gradients/. Accessed 28 Apr. 2018.
[5] "Smoothing Images — OpenCV 2.4.13.6 documentation." https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html. Accessed 1 May. 2018.
[6] "How to Use Background Subtraction Methods — OpenCV 3.0.0-dev ...." 10 Nov. 2014, https://docs.opencv.org/3.0-beta/doc/tutorials/video/background_subtraction/background_subtraction.html. Accessed 1 May. 2018.
[7] "What Is Camera Calibration? - MATLAB & Simulink - MathWorks." https://www.mathworks.com/help/vision/ug/camera-calibration.html. Accessed 1 May. 2018.
[8] "OpenCV: Geometric Transformations of Images." 22 Dec. 2017, https://docs.opencv.org/3.4.0/da/d6e/tutorial_py_geometric_transformations.html. Accessed 1 May. 2018.
[9] "OpenCV: Epipolar Geometry." https://docs.opencv.org/3.2.0/da/de9/tutorial_py_epipolar_geometry.html. Accessed 1 May. 2018.
[10] "Introduction to Support Vector Machines — OpenCV 2.4.13.6 ...." https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html. Accessed 5 May. 2018.
[11] "TensorFlow." https://www.TensorFlow.org/. Accessed 7 May. 2018.
[12] "TensorFlow detection model zoo - GitHub." https://github.com/TensorFlow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Accessed 7 May. 2018.