Soccer Video Analysis

Kevin Cheng
Department of Electrical Engineering

***Abstract*—An algorithm to extract key features from a soccer video is proposed. In this algorithm, the soccer field, individual players and the ball are detected and tracked in each frame. The work done can be used as a basis for robust feature extraction of soccer games that can be expanded to provide automated analysis, interpretation of soccer games.**

Keywords-Soccer, analysis, field, player, ball, detection, tracking

I. Introduction

With the increase in technology in competitive sports, analysis of the game has been an ongoing growing field. In order to devise the best game plan, each aspect of the opponent is dissected to find weakness. In the competitive sports scene, every bit of information is key to victory. Also, with the growth of sports broadcasting around the world, more and video footage of sporting events is becoming available each day. An automated tool to help in analyzing sports could help facilitate this process.

In this paper, an algorithm is proposed to analyze soccer games. The algorithm is designed to detect and track the players from each team, the ball, and provide the basis for being able to detect where on the field the current camera frame lies.

For the player detector, a SIFT keypoint detector is used to scout out key potential players. These keypoints are then narrowed down and assigned to teams based on their respected RGB values[1]. In order to track the players from frame to frame, a connectivity map is used. The ball detector algorithm is based around a corner detector, combined a strategic search. In order to accurately track the ball, the corner detector is weighted by a Gaussian at the previously known position. The field detector employs a RGB mapping algorithm to detect the field, and a hough transform to extract key features about the detected field.

The player detection is very reliable, and has built in robustness to potential misdetections. From the testing, the player detection algorithm was able to detect with over 90% accuracy, the location of each player. The ball detector on the other hand is a little less reliable. Due to its design, it is possible for it to latch onto a local maximum.

The algorithm was implemented using MATLAB with help from the Image Processing Toolbox, as well as mex functions from vl_feat. The current algorithm runs at 1.2 frames per second. This is not nearly close enough to run real time, but would be ok for post-processing purposes. Not much work was done to optimize the algorithm, so many improvements could be made to improve on this timing.

With this soccer detection algorithm, further work can be done to begin interpretation of the player and ball positions. Using this algorithm as a base, this algorithm can be extended to perform automated soccer broadcasting, or just used to extract key statistics from the game.

II. Soccer Detection Algorithm

A. *Preprocessing*

Many of the videos included a network label, and so to increase the robustness of the algorithm, a mask was applied to each frame prior to any processing to remove network remnants.

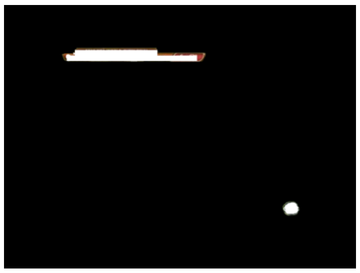


Figure 1: Sample network mask for video processing

Secondly, the proposed algorithm is optimized to work for wide field shots. So therefore, camera angles of close ups were not considered. This was by a fairly simple rule: if the detected field did not take up more than ½ the screen, then we determine the shot to be invalid. Obviously this was not a very robust approach, but worked fairly well.

Lastly, for all grayscale processing, an artificial grayscale map was created. Usually, grayscale images are taken by weighting the red, green, and blue components by 30%, 59%, and 11% respectively. For this detection, a weighting of 50%, 0%, 50%, was used instead. Since contrast with the field was desired, it was determined that removing the green component of the grayscale weighting improved performance.

B. *Field Detector*

At the core of the field detector is multidimensional MAP detector that is trained to detect the color of the field. The field detector is trained on a set of images where each pixel has been hand labeled as a "field" color, versus a "non-field" color. By creating a 16x16x16 3d histogram, training images were used to teach the detector what RGB values are considered the field. After applying this map to the image, a morphological close was used to smooth out some of the edges. Then to extract the exact bounding frame of the field, region labeling was performed, and boundary points of the largest region were extracted in the thresholded image. This was deemed to be the boundary of the field.

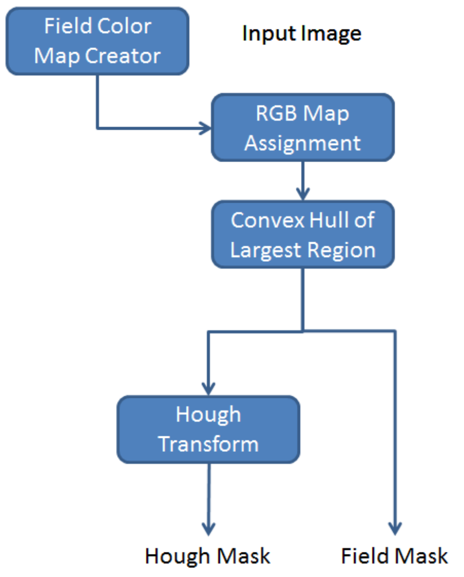


Figure 2: Flowchart of field detector algorithm

Also coupled in the field detector is a hough transform. The goal of the hough transform is to detect the field lines painted. The 10 strongest hough lines are saved for each frame and are used in eliminating noise the other detection algorithms. Although it was not implemented, the foresight was to utilize knowledge of the boundaries to correctly determine which part of the field the frame was taken at.

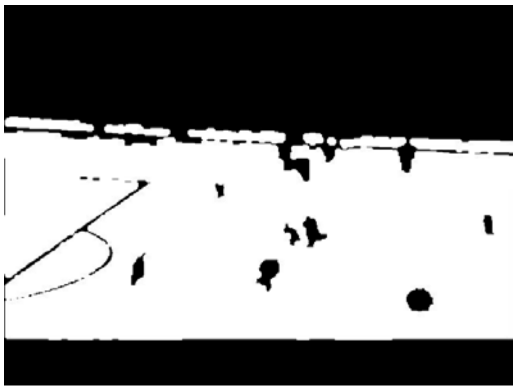


Figure 3: Example frame with RGB field map applied detecting "field" and "non-field"

C. Player

The way that the players are identified in the image is by their jersey color. The first frame of the video is designated as a keyframe, and information about the two competing team's jersey color is extracted from this frame. Similar to the field detector, a 16x16x16 histogram is created from the region of the image that has been returned by the field detector. From this, all the pixels whose RGB values have been labeled as "field" are removed, as well as all the pixels that lie on any of the detected hough transform lines. From this histogram, the top two peaks are saved as the RGB values of the 2 respective teams.

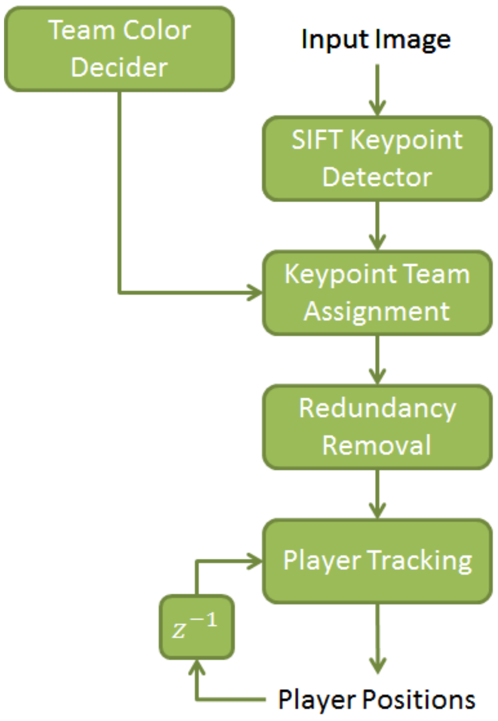


Figure 4: Flowchat of Player Detection algorithm

1) Detection

After the initial player color map has been generated, each subsequent image is compared using this map. To narrow down on the search regions, a SIFT keypoint detector is used to identify potential player regions. For each SIFT keypoint, the mean RGB value for the surrounding region is compared to the RGB value of each corresponding team. If the RGB value is considered close enough to that team color, then that particular point is assigned to that team.

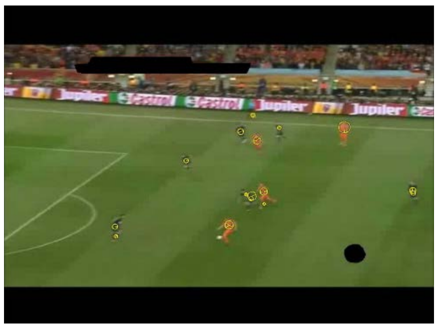


Figure 5: Filtered SIFT keypoints based on RGB values compared to team RGB values

$$d_{rgb} = \sqrt{(R - R_{team})^2 + (R - R_{team})^2 + (R - R_{team})^2}$$

Through testing, it was noticed that one individual player could produce multiple keypoints. To handle the case, a connectivity map was created for the set of keypoints pertaining to each team. The *(i,j)* entry of the connectivity map would be the pixel distance between the *i*th and *j*th keypoints. Since in general, players in the frame are about three times taller than they are wide, the distance calculated was weighted such that the x distance was weighted less.

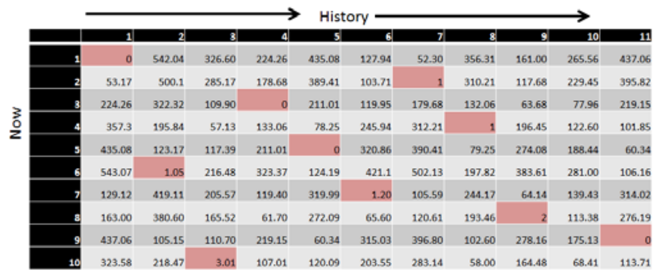
$$d = \sqrt{\left(\frac{x_1 - x_2}{3}\right)^2 + (y_1 - y_2)^2}$$

This map was then thresholded so that only values keypoints 40 pixels apart were kept. In order to make sure that clusters of three or more were properly handled, an iterative approach was taken where the keypoint that had the most neighbors was kept, while all of its neighbors were removed from the list. Then a new connectivity map was produced, and the process iterated until there were no two keypoints that were connected. The result is a duplicate free array of points containing the position of each player.

2) *Tracking*

To mitigate the issue of false positive as well as false negative player detections, a tracking algorithm was implemented to add some robustness to player detection. The player detection algorithm was also useful in being able track the movements of each individual player from frame to frame, although this is not clearly seen in the videos.

The tracking algorithm again takes advantage of a connectivity map. For each frame, there are two arrays of points containing the position of players in the two teams, as well as a history array containing the tracking information from the previous frame. A *i x j* connectivity map is produced where the *(i,j)* entry is the pixel distance between the *i*th current frame player position and *j*th history element. Note that this matrix is neither square nor symmetric. Then each history element is matched with its closest current element in a one-to-one mapping function.


History
Now

	1	2	3	4	5	6	7	8	9	10	11
1	0	542.04	326.60	224.26	435.08	127.94	52.30	356.31	161.00	265.56	437.06
2	53.17	500.1	285.17	178.68	389.41	103.71	1	310.21	117.68	229.45	395.82
3	224.26	322.32	109.90	0	211.01	119.95	179.68	132.06	63.68	77.96	219.15
4	357.3	195.84	57.13	133.06	78.25	245.94	312.21	1	196.45	122.60	101.85
5	435.08	123.17	117.39	211.01	0	320.86	390.41	79.25	274.08	188.44	60.34
6	543.07	1.05	216.48	323.37	124.19	421.1	502.13	197.82	383.61	281.00	106.16
7	129.12	419.11	205.57	119.40	319.99	1.20	105.59	244.17	64.14	139.43	314.02
8	163.00	380.60	165.52	61.70	272.09	65.60	120.61	193.46	2	113.38	276.19
9	437.06	105.15	110.70	219.15	60.34	315.03	396.80	102.60	278.16	175.13	0
10	323.58	218.47	3.01	107.01	120.09	203.55	283.14	58.00	164.48	68.41	113.71

Figure 6: Example tracking connectivity map results. History keypoint #10 does not have a match.

The other element of the tracking algorithm is keeping track of the detection count for each player. Each time a player is tracked, that is, the there is a match from the player history set with the current player detection set, then its track counter is incremented by two until a maximum of fifteen. Each time a history element fails to match a current player, the track counter is decremented by 1, to a minimum of 0. The rule of display is that the track counter has to be greater than 5. This was implemented in order to provide smoother frame to frame tracking of players. If we do not track a player for a particular frame, it is likely due to detection error, since players do not just randomly disappear. Similarly, if our detection picks up a false positive, it shouldn't be immediately assumed that a player has appeared. The implemented method provides a balanced outcome.

D. *Ball*

1) *Detection*

The ball has 3 main features that allow us to detect it with reasonable effectiveness: it is small, white, and like a point. Since the ball is small and like a point, it will have high special derivatives, which makes it easy to detect using the Harris corner detector. Since the ball is white, all corners are filtered out based on their original RGB values. One major category of points that match both these criteria are the corners of the painted field lines. In order to mitigate this, the hough transform of the previous field detector is used to mask out these potential false detections.

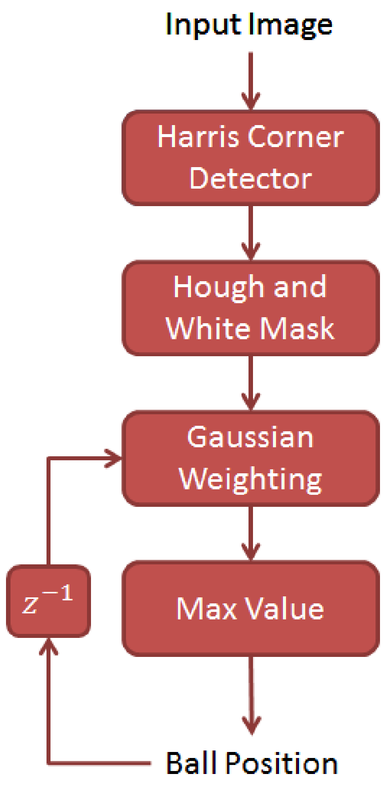


Figure 7: Flowchart of Ball Detector

2) Tracking

The detection of the ball is generally most susceptible to misdetections. In a typical soccer field, there are lots of other small white objects that may be mistaken for the ball, things like the player's white clothing, faces, or just any other object that may suddenly be very reflective for a single frame. To improve tracking from frame to frame, the corner detector is weighted by a Gaussian at the last known location of the ball.

To improve tracking from frame to frame, the corner detector is weighted by a Gaussian at the last known location of the ball.

$$g(x,y) = e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}}$$

By applying this Gaussian, the search space for the ball is localized to the area around the previously detected spot. This is effective assuming that we are able to initially detect the ball, otherwise the ball tracker may be stuck on the wrong object. However, since generally the ball will be detected in every frame, whereas the other sources of noise may not be, the ball tracker will typically find its way and latch actual ball.

III. Results

The 2010 world cup final between the Netherlands and Spain was used as a testing video for this algorithm. To increase the robustness of the player detector, the mean RGB team colors were pre-computed using various frames to be (48, 64, 48) for Spain and (224, 112, 96) for the Netherlands. To deal with the change in camera angles and close up shots that are associated with any soccer broadcast, all the history and tracking buffers were reset when a change occurred. Also, since the algorithm is designed specifically for the far camera view, close up camera views were ignored.

Also, to accommodate the broadcast scoreboard and logo, a network mask was created in which those areas of the image were permanently blocked out from the algorithm. It was noticed that when the broadcast added additional overlays displaying some statistics, the soccer detection algorithm started to misbehave.

Since the team colors are pretty distinct (orange and black), the player detector and tracker operated with a near perfect hit rate. Out of a segment of 1,800 frames, the player detection rate was over 80%. However, with the integration of the player tracking, the detection rate rose to well over 95%. One issue with the player detector algorithm is even though the player is correctly detected from frame to frame, the player tracking tends to jump quite sharply to different parts of the player's body. One large source of false positives for the player detector were the referees, whose clothing sometimes matched that of the team jerseys.

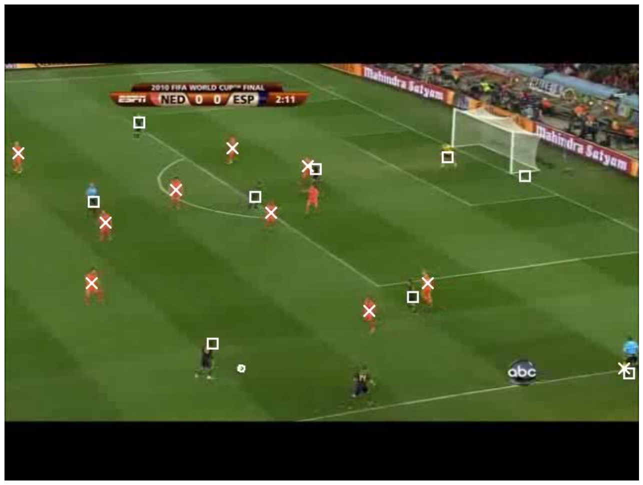
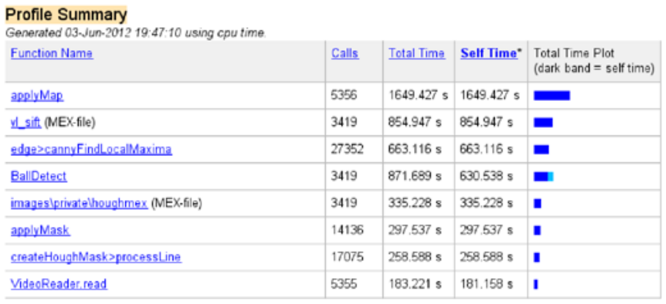


Figure 8: Sample output frame with 2 false negatives and 2 false positive (referee)

The ball tracking was also reasonably successful, but as robust as the player tracker. With the algorithm proposed in this report, when the detection latches onto the ball, it would maintain track of the ball until one of three things happened. Firstly, if the camera angle changed, then all the tracking buffers reset, and we would have the advantage of using the previous location. Secondly, since in the ball detection algorithm, we mask off the field lines, if the ball crossed a field line, then the ball could potentially latch to another point. Lastly, if the ball was obscured or blocked in a particular frame, the ball tracker would potentially latch onto another point. In certain circumstances, once the ball became visible again, it would redetect the ball. However, in other cases, it would find another local maximum and fail to find the real ball. Such examples of local maxima included the player's numbers, their socks, or the goal posts.

In terms of the timing, the proposed algorithm runs in MATLAB at about 2.63 seconds per frame. This is nowhere close to what it needs to be to run real time, however, it would be potentially suitable for post processing applications. Furthermore, not a lot of work was done in optimizing the algorithm, and definitely implementing the approach in other languages could potentially yield large timing benefits.



Profile Summary

Generated 03-Jun-2012 19:47:10 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
applyMap	5356	1649.427 s	1649.427 s	▬
vl_sift (MEX-file)	3419	854.947 s	854.947 s	▬
edge>cannyFindLocalMaxima	27352	663.116 s	663.116 s	▬
BallDetect	3419	871.689 s	630.538 s	▬
images\private\houghmex (MEX-file)	3419	335.228 s	335.228 s	▬
applyMask	14136	297.537 s	297.537 s	▬
createHoughMask>processLine	17075	258.588 s	258.588 s	▬
VideoReader.read	5355	183.221 s	181.158 s	▬

Figure 9: MATLAB profiler results, showing most time consuming functions.

	Field Detector	Player Detector	Ball Detector	Overhead	Total
%	46.6%	16%	10.9%	26.5%	100%
t (s)	1.15	.396	.27	.81	2.63

Figure 10: Timing breakdown by algorithm by % total time, and seconds per frame

Overall, the field detector is by far the most time consuming detector. The largest contributor to timing is the function that iterates over the initial image, and labels each pixel as “field” and “non-field” based on the training map that was generated. This takes about .37 seconds per frame. This seems to be more of an issue with implementation within MATLAB. Other large contributors to the time in the field detector are the canny edge detector as well as the hough transform.

The second highest single function contributor to the time is the SIFT keypoint detector which takes about .28 seconds per frame.

IV. Discussion

One of the areas for improvement for the proposed algorithm lies in the ball detector. Given its current architecture, as seen from the results, it is possible to latch onto a false positive, and not be able to regain track of the ball for an extended period of time. Other stabilization techniques can be implemented such that if the ball is not detected for one particular frame, then the tracker will not immediately latch onto another local maximum. Another potential method is to have multiple ball detection objects that follow slightly different rules when handling mis-detections. These can then be inputted into a decider engine that would find the absolute maximum across the different inputs. This would help mitigate the effect of mis-detections and local maxima.

Another potential approach is to use a circular hough transform like that proposed by D’Orazio[3]. This however, would not solve the issue of misdetection, since the circular hough transform would also fail if the ball was slightly obscured by a player or a field line.

Also, some work may be needed to look into alternatives to the SIFT based player detector. One of the largest contributors to time is the SIFT based player detector. Color based team matching seems to be quick and effective, but perhaps an approach use of the field map found in the field detector, combined with region analysis would be a faster approach. However the currently proposed player detection algorithm is very robust, and so changes may sacrifice effectiveness for speed.

Another undesired artifact of the SIFT based player detector is the large amount of jitter from frame to frame, since the detected SIFT keypoint can jump from the legs to the upper body in subsequent frames. Other non-keypoint based approaches could be implemented to mitigate this issue.

One last improvement that would make this into a fully-capable soccer analysis tool is to improve on the field detector. Currently the field detector accurately determines where in the field the is in each frame, but is unable to fully map the field in the frame to the greater. Ekin and Tekalp exhibit some potential advanced field detection, as well as game interpretation in their report[2].

V. Conclusion

Given the results and the testing data, automated soccer detection and analysis is a very feasible. With the proposed algorithm, both the players and the ball can be detected with a high degree of accuracy. Using the methods described as a basis, improvements can be made to soccer video detection and extension into analysis and interpretation.

VI. Awknowledgements

For the work on this project, I would like to acknowledge my advisor André Araujo for his help. I would also like to thank the entire Stanford EE368 course staff for their support and guidance on this project.

VII. References

[1] Youness, Tabii & Rachid, Oulad Haj Thami. A Framework for Soccer Video Processing and Analysis Based on Enhanced Algorithm for Dominant Color Extraction

[2] Ahmet Ekin, A. Murat Tekalp, Fellow, IEEE, and Rajiv Mehrotra. Automatic Soccer Video Analysis and Summarization. IEEE Transactions on Image Processing, vol. 12, no. 7, July 2003

[3] T. D’Orazio, N. Ancona, G. Cicirelli, M. Nitti. A Ball Detection Algorithm for Real Soccer Image Sequences. Istituto Elaborazione Segnali ed Immagini - C.N.R.