Computer Vision Approach to Football Player Mapping

Bryant Cornwell, Lucas Franz, Seth Mize

Professional sports industries continue to increase their reliance upon analytics, one of the primary being the tracking of player movement. In football (soccer) this is even more prioritized as the game relies upon player positioning and movement on a vast playing surface. Modern techniques utilize wearable technology to accomplish this. This paper argues that through the adaptation of existing computer vision techniques the same results can be achieved and democratized. Utilizing image channel thresholding, Hough Transformations, algebra, linear algebra, and a deep neural network, our results illustrate the possibilities of transforming player and ball locations from broadcast images into a 2-dimensional overhead coordinate space.

INTRODUCTION

Over the last two decades, the use of analytics has become pervasive in professional sports leagues. One component of interest is player location tracking with respect to the dimensions of the playing surface. The primary means of collecting this information is done through GPS tracking through wearable technology or manual annotations. This paper seeks to explore the application of computer vision techniques to accomplish this task.

Focusing on the sport of football (soccer), we will utilize available broadcast feeds to convert the video frames into a two-dimensional, overhead representation of the player locations based on the corresponding portion of the pitch (field) in the frame of the image. The technique used in this application is image projection. This is accomplished by first analyzing the pitch itself, extracting the universal field boundaries and markings to establish feature points of the image. These feature points are then used to solve for the projective transformation to the overhead perspective. Once the transformation has been solved for, a deep neural network is then used to identify objects on the pitch and their corresponding anchor points, which can then be translated into their overhead coordinates.

BACKGROUND

The stages of our work that were previously outlined, pitch feature extraction, projection, and object detection, all correspond to existing techniques. Beginning with pitch feature extraction, this problem is broken into 3 components, image filtering, hough transformation, and line intersection. Imaging filtering for line identification has most notably been explored in autonomous driving applications [9]. Invented in 1962 and cemented in the computer vision community in 1981, Hough Transformations have been in the toolkit of computer vision engineers for quite some time. The last step, line intersection, is a simple application of algebraic formulas.

In solving for projection, 2 mathematical techniques are again relied upon. The projection transformation matrix is found through the solving of a system of linear equations. Because of the format of our problem, the widely accepted technique of RANSAC is not applied to projection solutions to identify the optimal projection, rather this paper attempted to utilize a brute force approach with a voting method to identify the correct transformation.

The last stage of our solution focusing on object detection makes use of the industry leading deep neural network approach. Previous work on football player detection and tracking have been explored using computer vision techniques [6].

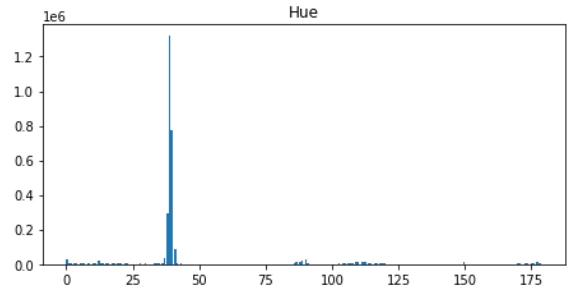
METHODS

Stage 1: Feature Points

The feature points for this given problem must be those that include the pitch lines and markings as these are the only identifiable corresponding features of the overhead image of a generic pitch.

Line Isolation

The first step in obtaining feature points from pitch lines is to identify the lines themselves. Converting the image into Hue, Lightness, and Saturation (HLS) channels and exploring the Hue channel gives a clear identification of the pitch, as the pitch makes up the majority of the pixels in the image.



Histogram of Image Hue Channel

While the exact hue values may shift, this pattern will be repeated for all pitches and broadcast qualities. Targeting the peak and neighboring values, the image can be filtered to only visualize the pitch.

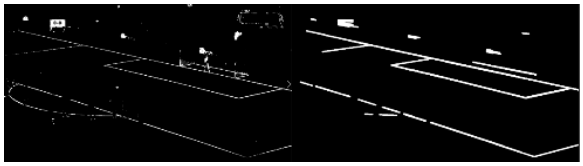
Original (Left), Hue Filter (Right)

Taking the filtered image and converting it to grayscale we have yet another distribution in which keeping only the top percentile of values will further filter the image, leaving behind only the highest values, the white lines.

Hue Filter (Left), Grayscale Filter (Right)

Hough Transformation

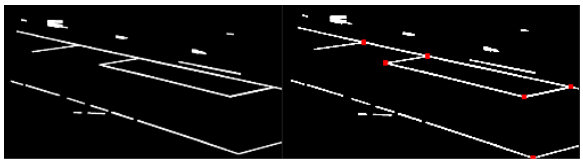
Once the only remaining pixels are those corresponding to pitch lines, we can apply a Hough Transformation to get solid and complete lines.



Grayscale Filter (Left), Hough Transform (Right)

Line Intersection

In OpenCV [15], hough lines are reported as 2 sets of coordinates. From these coordinates we can solve for the equation of the line. With 2 equations of lines we can solve for their intersection. Ensuring the intersection occurs within the bounds of both hough lines, we can plot and save the coordinates.



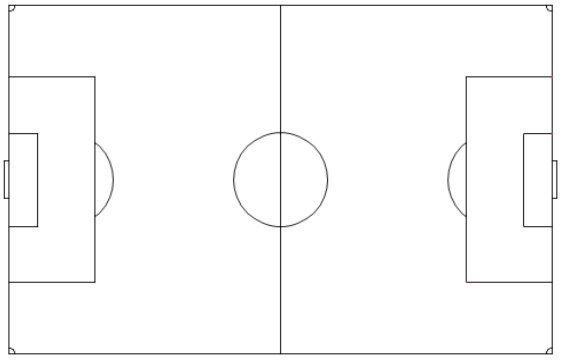
Hough Transform (Left), Line Intersection (Right)

Stage 2: Solving Projection

The feature points from stage 1 are delivered to stage 2 as x, y coordinate pairs. A minimum of 4 coordinate pairs are required for projective mapping into the overhead destination image.

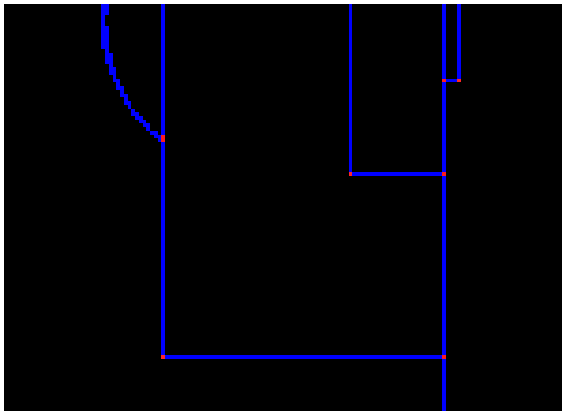
Destination Image

The overhead pitch image consists of the minimum required lines and markings, as well as the minimum pitch dimensions, scaled proportionally.



Overhead Image Space

Because our feature points from the source image target line intersections, we are able to predetermine the destination feature points, creating a static list of possible matching coordinates.



Overhead Pitch Lines (Blue) and Line Intersections (Red)

Transformation Matrix

To solve for the transformation matrix, 4 of the feature points returned from the source image are converted into 3 dimensional coordinates. The converted coordinates are then used to structure a linear system of equations representing the solution of the projective transformation matrix.. Solving

the linear system of equations supplies the transformation matrix.

To find a matrix

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix}$$

such that

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x^*z^* \\ y^*z^* \\ z^* \end{pmatrix} \rightarrow \begin{pmatrix} x^* \\ y^* \end{pmatrix}$$

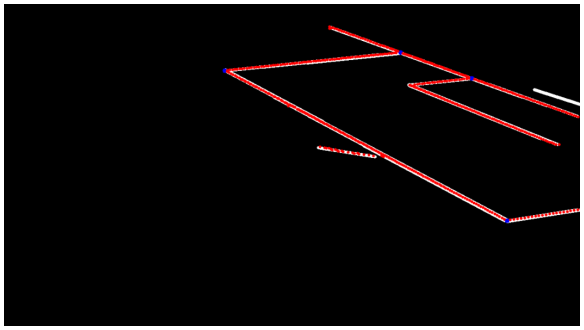
we can solve the following equation for a, b, c, d, e, f, g, h given four sets of x_i, y_i, x_i^*, y_i^* values.

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 & 0 & -x_1^* & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & 0 & 0 & -y_1^* & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & -1 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -x_2^* & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 & 0 & 0 & -y_2^* & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & 0 & -1 & 0 & 0 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_3^* & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & 0 & 0 & 0 & 0 & -y_3^* & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_3 & y_3 & 0 & 0 & -1 & 0 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_4^* \\ 0 & 0 & 0 & x_4 & y_4 & 1 & 0 & 0 & 0 & 0 & 0 & -y_4^* \\ 0 & 0 & 0 & 0 & 0 & 0 & x_4 & y_4 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ z_1^* \\ z_2^* \\ z_3^* \\ z_4^* \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

Equation to solve for projection

Transformation Matrix Selection

Using the predefined coordinates from the destination image, each combination of 4 points will generate a unique projection matrix. To judge the validity of the projection we apply the inverse projection from the overhead projection image (/football-mapper/Seth/overhead-space.png) to the source Hough transformed image. We can then tally the number of line pixels from the destination image that were mapped to a line pixel in the source image. The projection resulting in the highest number of matches is deemed the true projection.



Matching Over Line Pixels

Stage 3: Detecting Objects

The model training dataset of 12 images was created from four different FIFA World Cup football games hosted on the FIFA youtube channel. The images were labeled using "LabelImg" [1] to manually apply bounding boxes to an input image, label them, and export a xml file that consists of box locations and associated labels for the given image. The players were labeled as "Person" and the football as "Ball" for each of the images.

The TensorFlow Object Detection API (Version 2) and Models repository were utilized for generating this project's object detection model [5]. A label map was generated for the corresponding two classes from the labeled image dataset to train the model and detect objects. To train and test the model, the dataset (.xml and .png files) was separated into an 83/17 percent split train and test sets. These sets were converted into .record files.

To generate the model with TensorFlow 2 API, a pre-trained model was required. Since a person is a class within the COCO dataset [2], models for this dataset were priority. The model selected for this project was the Faster R-CNN ResNet50 V1 1024x1024, since faster-rcnn models tend

to be more accurate [3]. The model file consists of a .config, checkpoint (.cpkt) files, and the .pb model file.

The paths for the train/test .record files, label map file, and pre-trained checkpoint files were added to the config file to prepare for training a new model. Other minor modifications to the config file are detailed in this project's README.md. The new model was trained and exported model file (.pb) for object detection on images found in the results.

A guide for replicating the model and running the code for applying the object detection model to an image can be found on the project's github and README.md (https://github.iu.edu/samize/football-mapper).

Stage 4: Applying Projection

The applying projection stage is fairly straightforward. This stage of the program receives two pieces of information. The first comes from stage 2 in the form of the projection matrix. The second is the list of anchor coordinates provided by the object detection in stage 3. Stage 4 code then executes by applying the projection matrix to the anchor coordinates, and plotting the resulting coordinates in the overhead image space.

DISCUSSION OF METHODS

Stage 1: Feature Points

Our original approach to extracting the pitch lines sought to exclusively implement grayscale filtering but never produced a viable solution. There were 2 reasons for this, crowd noise and pixel bleeding. The pitch is surrounded by the crown and advertisements, all of which variably included white pixels. Filtering exclusively for white pixels retained too much noise. We also found that the white pixels of the pitch lines were influenced by the neighboring grass pixels, bringing up the green values associated with the lines.

Pivoting our approach to a multi-step filter alleviated these issues but did present the program with 2 hyperparameters in need of tuning. Due to the variability of pitch coloring, lighting, and broadcast quality, it was necessary to tune the range of the hue filter, as well as restrict or relax the grayscale threshold.

For our current program, this is done manually given 1 frame from a given broadcast. A future extension of our solution would involve a very restrictive starting point for both parameters and an iterative relaxation, testing for viability in number and quality of feature points at each step. This would negate the need for manual tuning and create a more dynamic and robust program.

Stage 2: Solving Projection

The largest issues encountered in solving for the projection matrix came in runtime constraints. Testing all possible combinations, which was a necessity in the original setup of our program, proved to be time consuming to the extent that only a single frame could be processed per 62 hours.

Due to this time constraint, many of our initial tests reduced the number of feature points from the overhead space that we

were searching through manual selection. The results of the small search space seemed promising, so we decided to proceed with generalizing our approach in a more quantifiably testable and efficient way.

This method attempted to use symmetries of the combinations of 4-point-pairs to reduce the number of combinations of point-pairs explored. Then, we iterate through each combination.

First, during the iteration, we use the four points to solve for the transition matrix. Then, we map each of the four feature points from the overhead image to the hough line image, and ensure that the pixels are within a distance of 10 pixels of where they should be mapped to.

Next, we transform each pixel representing a pitch line from the overhead space and map it to the hough line image using our transition matrix. Any pixel that maps to a line pixel on the hough image is counted, casting a vote for that transformation. When searching, we select the transformation with the most votes as a percentage of the total number of pixels that landed inside the hough line image as the transition matrix to use.

The code we wrote for this method ended up having some issues with it, such that it failed to properly identify the correct transformation using the voting image. We ran out of time to identify the issue before transitioning to writing the report and producing our poster, but we look forward to continuing this approach.

Since we had proved earlier that it conceptually functioned, we opted to manually provide the program the correct subset of matching source and destination points, allowing us to produce mappings to demonstrate this proof of concept.

Future adaptations to our solution can come in 2 forms, the first being inference about the source feature points and the second being a heuristic based search algorithm for subsequent frames.

Given the source feature point coordinates, we believe it is possible to infer information from the geometric relationships in the source coordinate space that would in turn help filter the combinations needing to be tested from the destination image. This information could be, but not limited to, do the points exist on a shared line, could the points be ordered sequentially, do the points indicate an end of the pitch, or do the points correspond to a rectangle.

The second evolution of our program related to the processing of sequential frames of gameplay. Given that we are able to identify the matching destination coordinates for the first frame, information can be extrapolated from those coordinates to inform which coordinates have the highest probability of being the matching coordinate in the subsequent frame. Devising a heuristic for a search algorithm that implements these probabilities would drastically improve running time.

Stage 3: Detecting Objects

Initially, the objects planned for detection are team A players, team B players, referees, goalies and the soccer ball, but were reduced to person and ball to break down the object detection algorithm. The image dataset was relabeled using ‘LabelImg’ to reduce the code for detecting

the manually labeled bounding box locations.

To get an idea, we referenced a video to get an idea of how to implement object detection [4]. Ultimately, we did not use code from the video.

A football player detection and tracking research paper and project by Samuel Hurault, Coloma Ballester, Gloria Haro provided a model trained on a large football image dataset [6,7]. After failed attempts to use the model, we did not use the model as we did not have the tools to convert the PyTorch model to a different format to load it for testing.

The TF2 Object Detection API tutorial outlined building a model utilizing a pre-trained model and after many failed attempts to train a model without a pre-trained model, we found success using a TF2 pre-trained model from the COCO dataset [3]. Due to the limited image dataset, the faster-rcnn model type was chosen for its higher accuracy [3].

Many out-of-memory errors that occurred while training the model were fixed by modifying the batch_size and number of steps within the pipeline.config file. Additional software was installed to allow and implement GPU computation for training the model to reduce runtime and increase the memory allocated. Most information for loading models for object detection on images is currently out-dated using techniques for the first version of the TF2 API. We were able to utilize the code provided within the TF2 tutorial as a precursor for our final object detection code [8]. A 30% threshold was used to reduce the amount of detected objects shown in the images.

Additional details for the issues encountered can be found within this project's README.md.

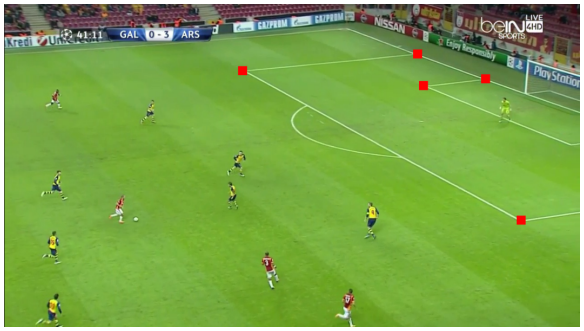
Stage 4: Applying Projection

Because this stage only involves matrix multiplication, and the matrix components are delivered from previous stages, there is no room for improvement we were able to identify for Stage 4.

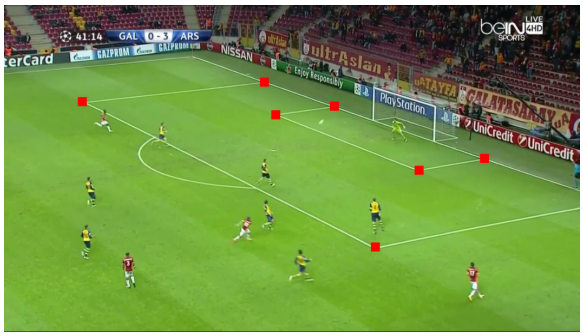
RESULTS

Stage 1: Feature Points

The stage 1 techniques produce adequate, although imperfect, results when the goalie box is within the frame. The goalie box produced the highest number of feature point sets that included a minimum of 4 points. This was due to the availability of line intersections within the central focus of the frame. Although the correct number of feature points, the multi-pixel width of the lines created the opportunity for intersection pixels to be captured at inconsistent locations of the lines.



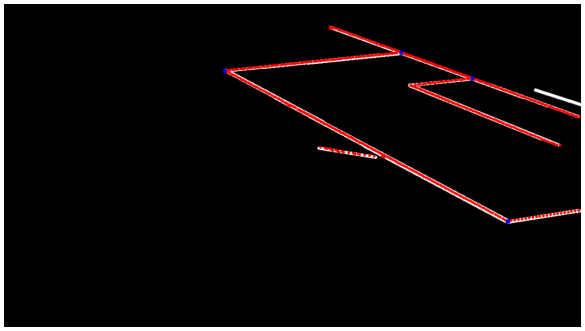
Frame 0 of 807-2 Test Clips Feature Points



Frame 124 of 807-2 Test Clips Feature Points

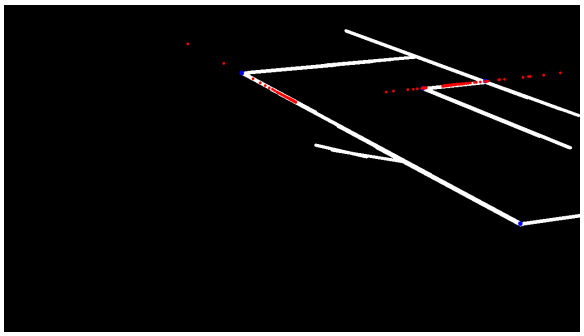
Stage 2: Solving Projection

Given no runtime constraints, the projection solver will find the best fitting projection. Unfortunately, because the delivered feature points are imperfect, the best fitting projection is also imperfect. When conducting the inverse projection for voting by pixel match we observed that the best projection map all available points to available hough lines drawn with a 5 pixel width.



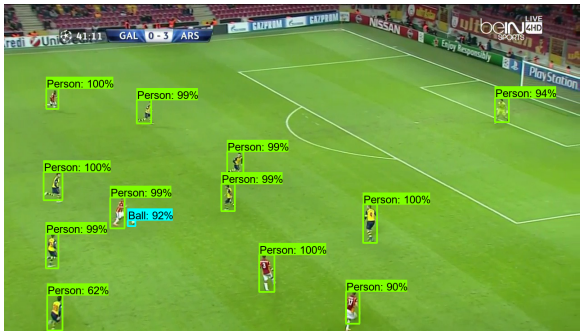
Matching Over Line Pixels

The alternative method that we attempted that failed, even after exploring a reduced feature point space, was returning results similar to the image below with vote accuracies in the upper 99% 's, but with results that were clearly not representative of that. We were not able to identify the source of the issue by the deadline. It could be issues with false positives, the code could be skipping one of the correct mappings, or it could have been an overlooked bug in the process.



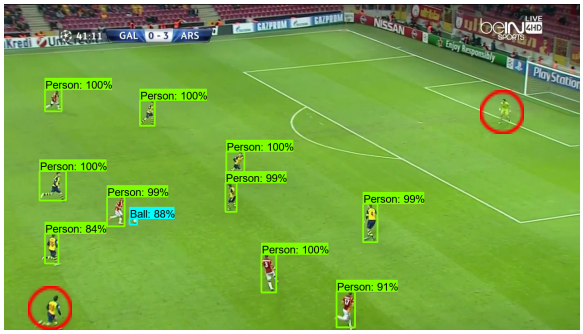
Example Failure with > 99% "vote accuracy"

Stage 3: Detecting Objects

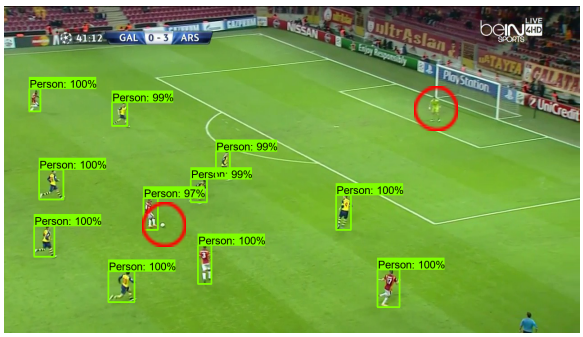


Frame 3 of 807-2 Test Clips Detected Objects

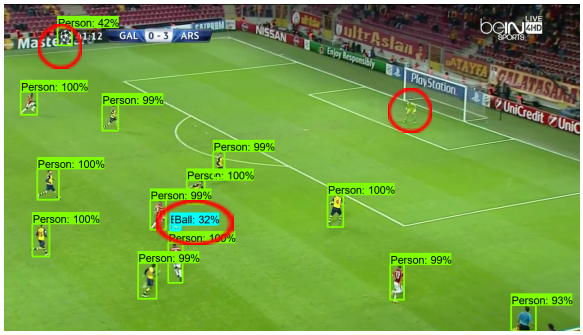
Frame #3 of the test clips was the best image detected out of the 125 frames of a short segment of football broadcast footage where each object was detected correctly. All object detection frames for this footage can be found within the documentation on the github (direct link). The accuracy percentage next to the class names for the detections were mostly high when detecting the correct class.



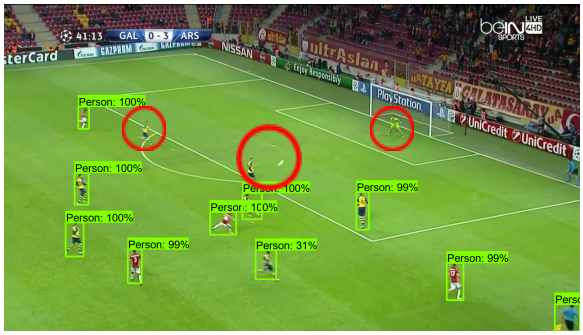
Frame 0 of 807-2 Test Clips Detected Objects



Frame 48 of 807-2 Test Clips Detected Objects



Frame 65 of 807-2 Test Clips Detected Objects



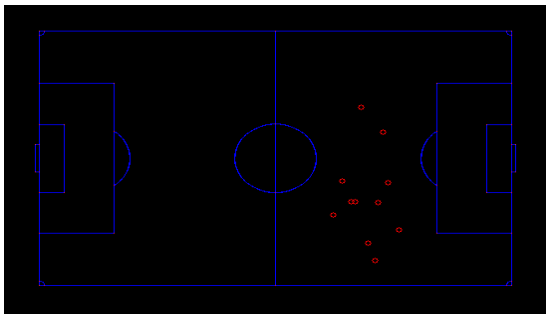
Frame 112 of 807-2 Test Clips Detected Objects

Person detection seems to do very well most of the time. The exceptions seem to occur when the person is near the edge of the image (frame #0) or when the jersey is similar to the color of the field. In this case, the goalie's uniform seems to cause issues for detections. One outlier that was often detected as a person was the football icon next to the game time seen in frame #65 of the test clips. Another outlier was the two missing person detections in frame #112.

Ball detection seems to be difficult when it is close to players (frame #48) and as the shape of the ball is warped as it is captured at a high velocity or airborne (frame #112). There were a handful of frames where the algorithm detected the ball multiple times resulting in multiple bounding boxes around it seen in frame #65 of the test clips.

Stage 4: Applying Projection

The technique of applying the projection matrix to the anchor points of the detected objects works to an accuracy of +/-2.5 pixels, extrapolating from the success of the inverse projection voting conducted in stage 2. Knowing that our solved projections matrix is not a 100% matching solution, our projected anchor points cannot be in their true location. Given the high accuracy of our projection solution under no time constraints, we do feel confident that the overhead placement of objections is an accurate representation, although not exact.



Overhead Image of Tracked Feature Points

DISCUSSION OF RESULTS

Stage 1: Feature Points

The greatest improvement to results for Stage could come from an improvement in the broadcast image angle and framing. Built to provide an ideal viewing experience, the broadcast image is typically zoomed in slightly, leaving the nearest boundary line out of view. The camera is also restricted by the stadium in its placement height, with the lower it is placed in the stadium creating a shallower angle which obscures the opposite boundary line. Cameras at a higher angle and zoomed out to include a higher number of the pitch lines would create more opportunity for pitch line intersection identification.

Further work can also be applied to how the point of intersection for multi-pixel lines is determined. This is a difficult proposition without additional logic that can determine which intersecting lines are being analyzed.

Stage 2: Solving Projection

Given that without time constraints the Stage 2 solution will find the best projection matrix, improvement in results depends on the quality of the feature points produced in Stage 1.

We would like to continue researching and debugging our failed attempt to automate the mapping between the overhead image space and the hough line image space. The current runtime of this method is relatively small, running in the magnitudes of minutes compared to hours, so if we can solve the selection issue, it is promising.

Ultimately though, the manual point pairing proved that it is conceptually possible to perform this stage.

Stage 3: Detecting Objects

Object detection performed quite well for the purpose of detecting the two classes. The dataset used for training the model is quite small, since the goal for this project was not to create a cutting-edge object detection algorithm. The results suggest that a larger train and test image datasets would improve the model to detect objects that are currently missed by the model as well as reduce the risk of detecting outliers.

Stage 4: Applying Projection

Any further improvement in applying the projection results relies on outside factors, the most important being the quality of the projection matrix. We could also see further improvement in standardizing the pixel density of the broadcast images being processed to consistently map to a proportionally dense destination image.

CONCLUSION

In conclusion, we believe we have demonstrated a viable proof of concept for obtaining and mapping football player locations using computer vision techniques. We are able to inject an image from a broadcast angle of a football match, extract the pitch lines to identify feature points, solve for the projection matrix to the overhead space, identify objects in the original image, and project those objects into the overhead space.

While encapsulating all of our goals, our solution does leave room for improvement in most respects. Line detection can be improved in the robustness of the program. We've identified the necessary techniques but failed to implement a fully dynamic solution. Working to iteratively identify the thresholds would accomplish this.

In solving for the projection matrix, our largest hurdle we were unable to overcome was that of runtime. Evolving our solution to implement a more intelligent search technique in identifying the correct matching set of points to the identified feature points would greatly improve the efficiency of our program.

Future work for object detection could add the support to recognise both team players, goalies, and the referee for adding value towards tracking events and statistics.

SOURCE CODE

https://github.iu.edu/samize/football-mapper

REFERENCES

[1] LabelImg Github
https://github.com/tzutalin/labelImg
[2] Coco Dataset Website
http://cocodataset.org/
[3] Configuring TF2 Object Detection Pipeline
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md
[4] Object Detection OpenCV Video
https://www.youtube.com/watch?v=HXDD7-EnGBY
[5] TF2 Object Detection API Tutorial
https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html
[6] Football Player Detection and Tracking Paper https://arxiv.org/abs/2011.10336
[7] Football Player Detection and Tracking Github
https://github.com/samuro95/Self-Supervised-Small-Soccer-Player-Detection-Tracking
[8] Importing TF2 Saved Model Tutorial
https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/auto_examples/plot_object_detection_saved_model.html
[9] Hough Transform: Underestimated Tool in the Computer Vision Field
https://www.researchgate.net/profile/Simon-Karpenko/publication/228573007_Hough_Transform_Underestimated_Tool_In_The_Computer_Vision_Field/links/0fcfd51487c0d13691000000/Hough-Transform-Underestimated-Tool-In-The-Computer-Vision-Field.pdf
[10] A Computer Vision based Lane Detection Approach
https://www.kuet.ac.bd/webportal/ppmv2/uploads/1569514224155525147 6A%20Computer%20Vision%20based%20Lane%20Detection%20Approach.pdf
[11] Tracking soccer players aiming their kinematical motion analysis
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.6699&rep=rep1&type=pdf
[12] Semantic annotation of soccer videos: automatic highlights identification
https://doi.org/10.1016/j.cviu.2003.06.004
[13] Soccer video and player position dataset
https://doi.org/10.1145/2557642.2563677
[14] Advanced Lane Detection Using Computer Vision:
https://towardsdatascience.com/teaching-cars-to-see-advanced-lane-detection-using-computer-vision-87a01de0424f
[15] OpenCV Hough Lines Documentation
https://docs.opencv.org/4.x/d9/db0/tutorial_hough_lines.html