Ertugrul Kasikci 200104004097

CSE 463 HW1

This homework is consist of two parts.

Part A:

Taking homography of the given images

Part B:

Taking homography of the given images, detecting players and drawing circles below them then converting the images back to the original format.

Part A:

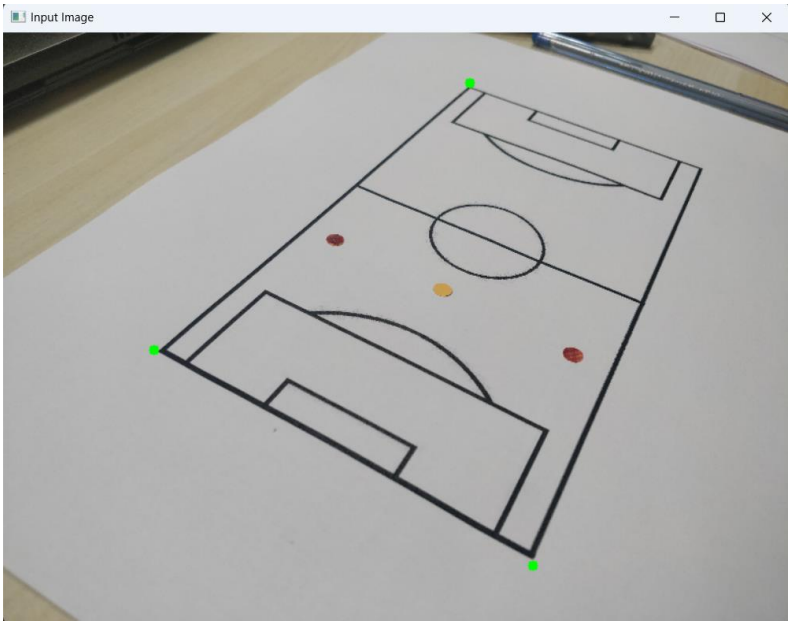
I took homography using three techniques;

- Homography with mouse clicks.
- Homography with corner detection.
- Homography with color detection.

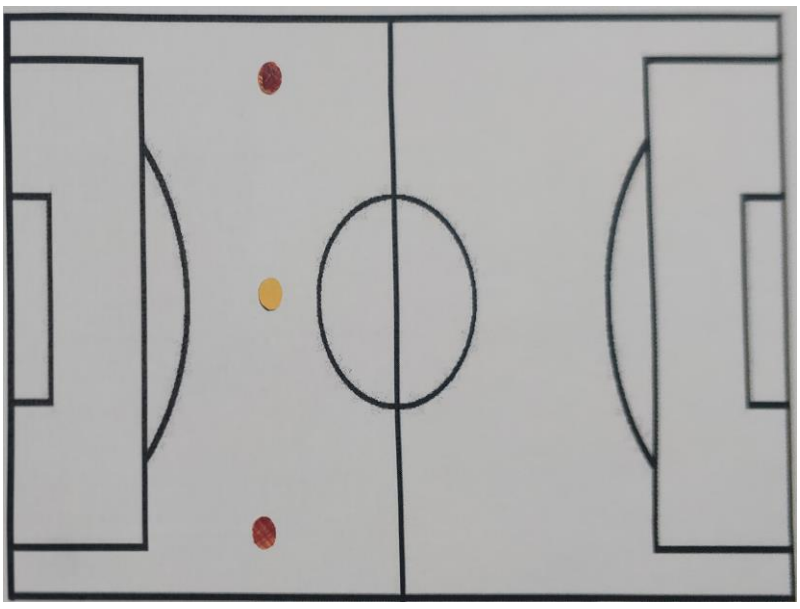
Homography with Mouse Clicks:

After running the code, the image is shown on the screen and user expected to select points to determine the corners of the output homography image. The clicks should be upper left, upper right, bottom left, bottom right corners in the order.

Here are the images showing process;



After user clicked for the last corner, the homography image is saved. Here it is;

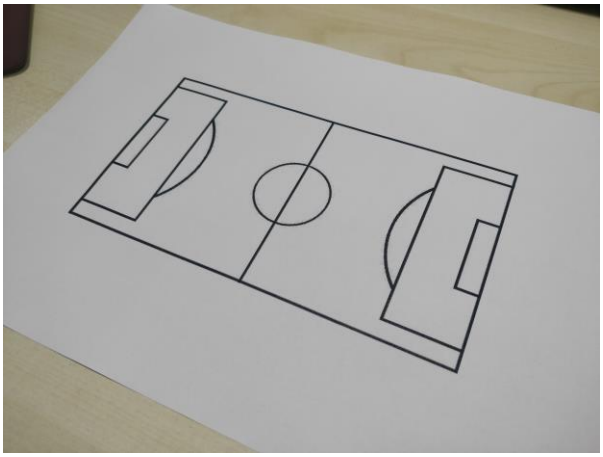


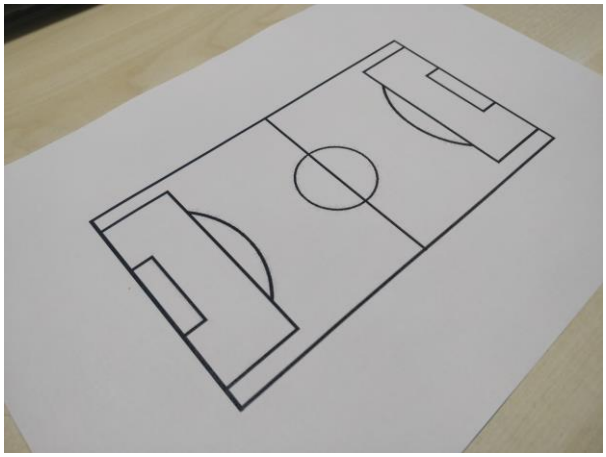
This method gives good results because the user can select whichever points they like.

Homography with Corner Detection:

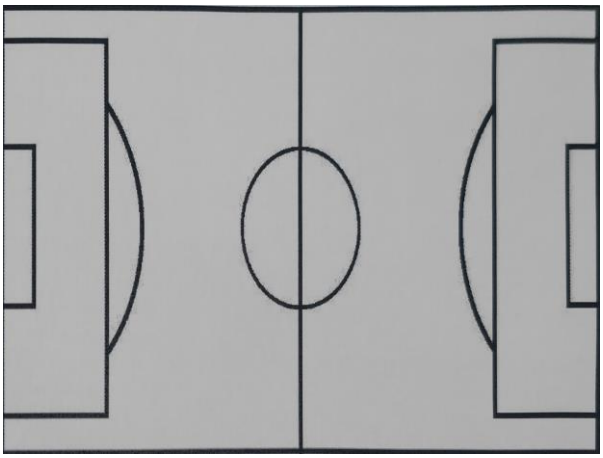
For the second method I used Harris Corner Detector to find the corners. There are some assumptions in this implementation. The upmost corner found is assigned to upper left corner, the leftmost corner found is assigned to below left corner etc. for the homography image. But we can't exactly sure which points should be which corner in homography image. It is better to show the outputs to explain it.

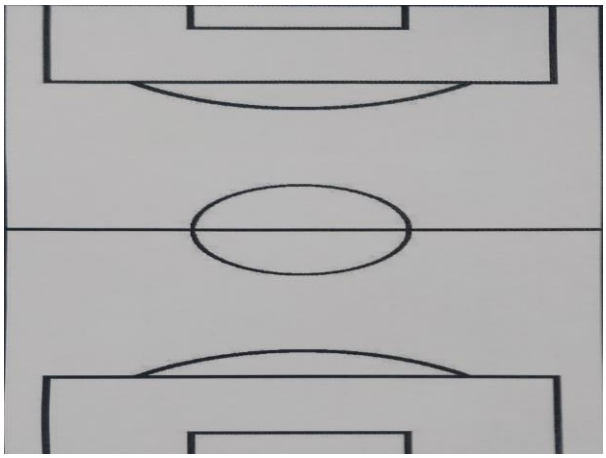
We have these two images;





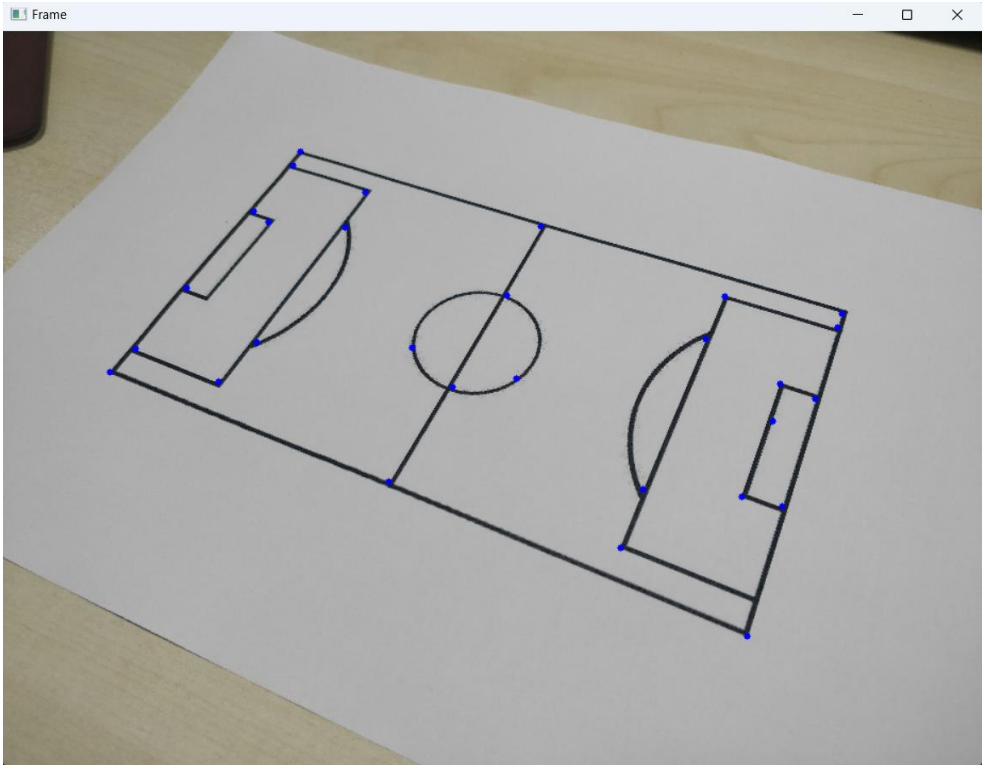
As can be seen, we can't sure where the upper left corner of the homography image should be selected in the input image (Upmost one?, leftmost one?, what if the image is turned more than 90 degrees?). So the homography of these images are as follows;





This method work fine as soon as the input image is not rotated too much. It this is the case, for example the point which should be the upper right corner can be taken as below right corner.

The corners to be selected are detected as follows:



Here we need to select 4 points to determine homography (**The code producing this image is not inclueded in the final version. I put the image to show the corner detection process**).

Homography with Color Detection:

In this implementation, I put different colors for every corner in the input image to detects their locations and use them while taking the homography of the image. The colors represent;

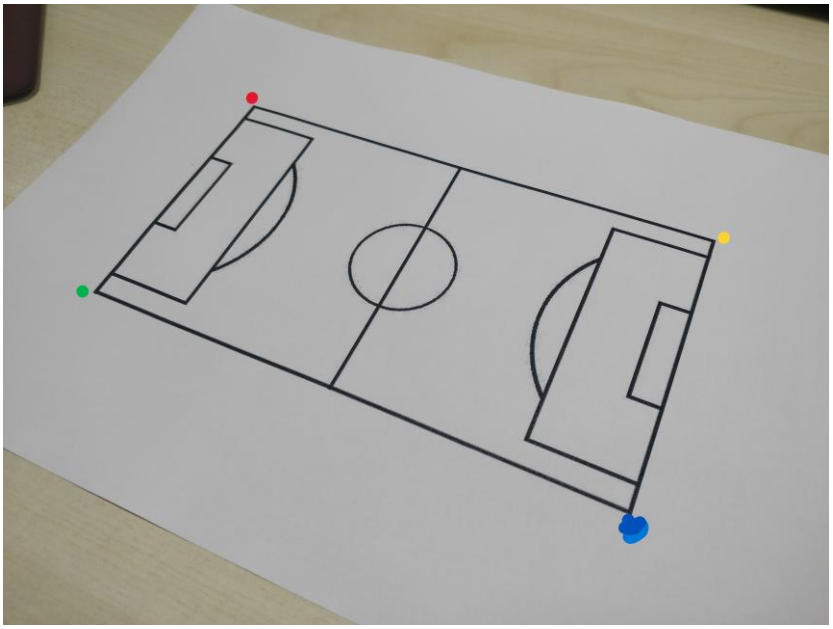
Red -> upper left corner

Yellow -> upper right corner

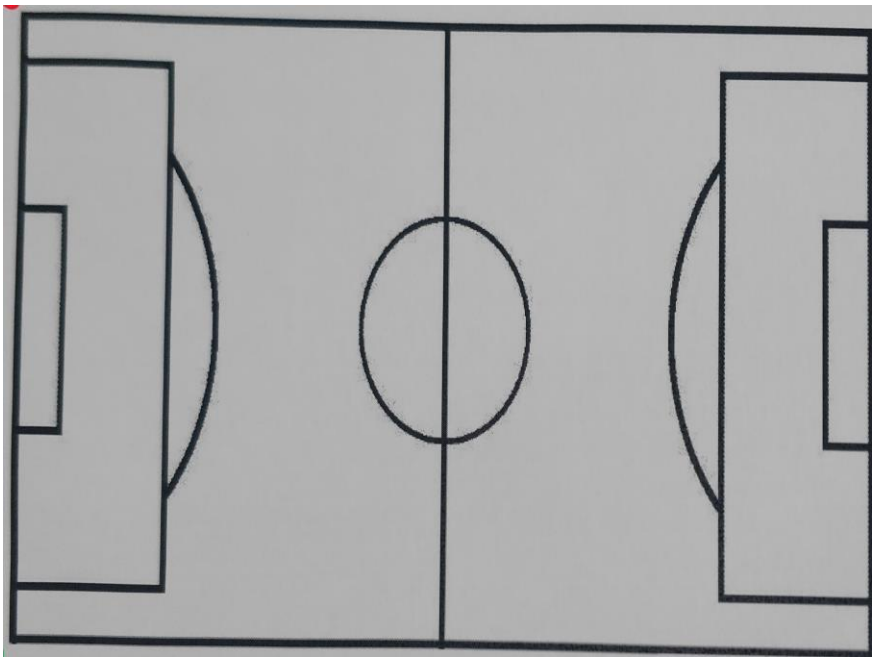
Green -> below left corner

Blue -> below right corner

The input image as it is shown;



and the homography image;



Note: In the all implementations, for destination points of the homography, I used

dst_points = np.array([[0, 0], [image.shape[1], 0], [0, image.shape[0]], [image.shape[1], image.shape[0]]], dtype=np.float32)

which means the selected 4 source points will be the corners of the homography image (as can be seen from the images provided).

For mouse click implementation, I needed to resize the input image because it didn't fit into screen. I used the ratio between the resized image and original image to get correct homography results later.

scale_x = input_image.shape[1] / resized_image.shape[1]

scale_y = input_image.shape[0] / resized_image.shape[0]

*src_points = src_points_resized * np.array([scale_x, scale_y])*

here *src_points* returned back to their original location to get correct homography.

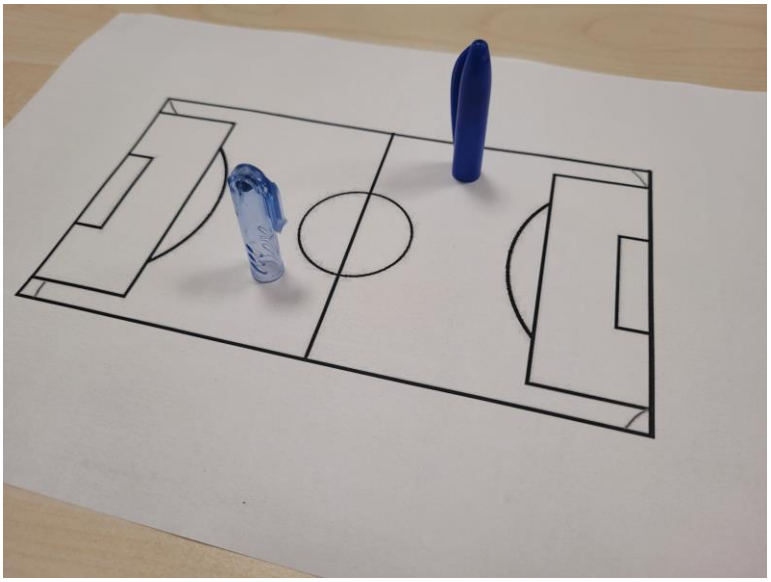
Part B:

For player detection I have tried different openCV functions like SIFT, SURF, ORB but none of them give me the result I want. I was finding too many results other than players. When I set threshold values to detect fewer features, I couldn't find the players, just some corners of the soccer field.

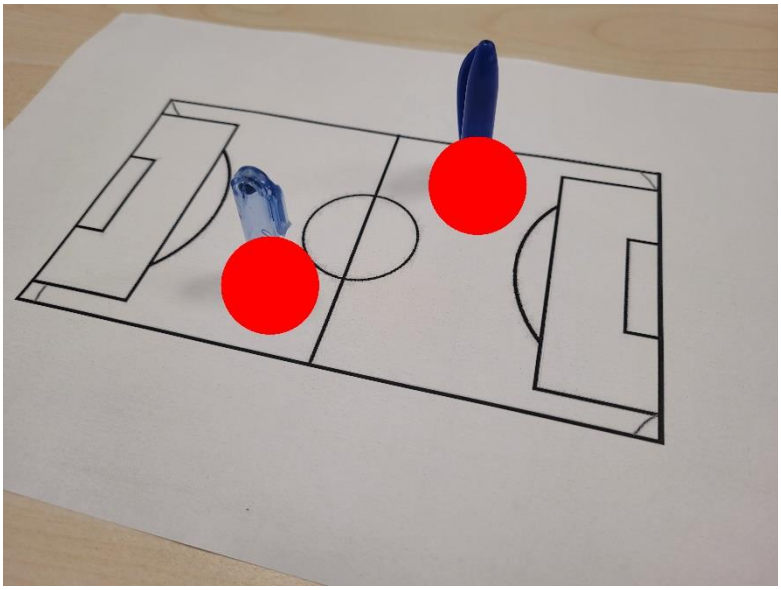
But I accomplished to find the players with color detection method. Eveything is black and white other than the players, so it is a good way to approach the problem.

In this method again I found too many features on a single player. I eliminated those features so that I have one point for a player. But there is no built-in method to eliminate nearby features as in the SIFT, so I created a function called ***eliminate_close_features*** to accomplish it.

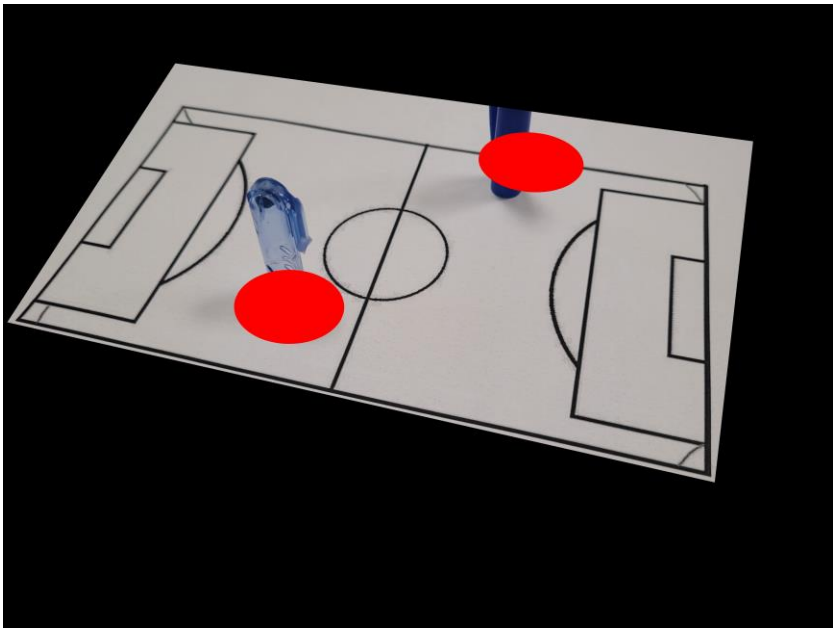
Here is the input image



The player detection algorithm without taking homography



The player detection algorithm after taking homography and returned back to the original format



Note:

- The images are saved inside the project folder with self explanatory names.
- Which code section does what is described at the beginning of each section with comment lines.