**PROJECT REPORT:**

**Major Steps of the ALGORITHM:**

1. Find at least four corresponding/matching points for the Image and Field.
2. Find the Homography by using cv.findHomography function.
3. Select Any two points (They can be defender and attacker players) in the Image.
4. Find the Mapping of above two selected points to the Field.
5. Now find the Points, in the Field, which represent the LINE.
6. Then Map those 4 Field points, which represent the Two Lines, to the Image plane.
7. After getting the Line points. Draw the Lines on Image Plane.
8. Find the Distance b/w those two Parallel Lines which are on the Image Plane.

Although I have used multiple functions but the major two functions which are cv2.findHomography and cv2.perspectiveTransform.

**cv2.findHomography:**

It is used to find the Homography parameters. Homography shows the Relation of One Image with another Image. Homography parameters show us the Transformation of One image with another Image.

cv.findHomography takes input of corresponding or matching points from Field and the Image. Its output is 3 by 3 matrix which represent the Transformation Parameters (9 parameters). These are indicative of how much Transformation happened for both the Image (Field and Real Image)

**cv2.perspectiveTransform:**

This function helps to MAP the points from Image to its corresponding points in Field. Or to map the points from Field to its corresponding points in Image.

The input attribute to this function are Points which we want to Map to another plain. Another input is Homography Matrix which helps to Map the Input points.

For example: cv2.perspectiveTransform(inputPoints, h)

Let say the above code is to transform/map Points from Image1 to Image2. Now if we want to Map points from Image2 to Image1 then we should use the INVERSE OF HOMOGRAPHY in cv2.perspectiveTransform code. As shown below:

cv2.perspectiveTransform(inputPoints, np.linalg.inv(h)) [0]

These techniques I have used in my code which I have submitted.

**Other functions which I have used in the code and their uses:**

cv2.namedWindow('Player Coordinates'): To create a new Window named ”Player Coordinates”.

cv2.setMouseCallback(): A call back function to fetch mouse x and y coordinate.

cv2.imshow(): To show an Image in specified window

cv2.putText(): To write Text on an Image or generally to put a Text

append(): To append a value to a variable.

cv2.waitKey(0): Image Window waits until we press **any** button

cv2.destroyAllWindows(): All opened Windows Disappear/Delete

range (): It makes a list of values from 0 to specified value minus in the functions attribute. For Example, range (5) will give values from 0,1,2,3,4

cv2.line(image, (x1, y1), (x2, y2), color, 2): Function to draw a Line b/w (x1, y1), (x2, y2) points on Image named “image”.

cv2.imread(): to Read the Image and it takes input as the “Path of the image”

print(): To print some text/string on Output Screen

np.array(): To convert a variable data type to **array data type**

**Code Details:**

I have imported that necessary libraries like numpy and cv2.

At first, I have defined a function named **get\_mouse\_coordinates()** which checks if we called for an Event of Left Button by using condition if Event == cv2.EVENT\_LBUTTONDOWN:

This condition along with cv2.setMouseCallback() function helps to fetch the points. To store the mouse points, we have used a variable name mouse \_coordinates = [] and then to add/append our selected points we have mouse\_coordinates.append([x, y]) function.

Then have defined another function named **fetch\_points(Img, NumberofPoints)** which takes input as an image in order to display it and to select Two points (b/w which we will find the Distance).In order to display the image we have used cv2.imshow(‘ImageWindow', Img). Further to show the image in a separate Window we have used cv2.namedWindow('ImageWindow'). To destroy all open windows, we have use cv2.destroyAllWindows() function.

It also uses cv2.setMouseCallback('ImageWindow', get\_mouse\_coordinates, param=Img) function which helps to fetch Selected points coordinates when user press Left Mouse Button in the Image named “Img”.

Used **cv.findHomography** to compute Homography of the 10 corresponding points from both Field and Image FIGURES.

Then I have used **cv2.perspectiveTransform** to find the Mapping of 2 points(Player A and Player B) which we selected by using **fetch\_points(Img, NumberofPoints)** function.

Then have defined **find\_LinePointsForPitch(pitch, output\_points) ,** where **output\_points** are mapped points we got after applying **Homography** onPlayer A and Player B Points. This function helps to **find the POINTS** which will help to make **PARALLEL LINES** and to make **Normal Line**.

Then I have defined a function named **draw\_line\_on\_ImagePlane(image,** **mapped\_points)**.Basically we apply **INVERSE HOMOGRAPHY** on the Line points (which we got from above function) and then we have applied **cv2.perspectiveTransform** on them in order to find the mapped POINTS on Image plane**.** These mapped Points will help us to Draw a Parallel Lines on Image plane**.**

**cv2.line(image, (x1, y1), (x2, y2), color, 2)** This function we have used in above defined function, in order to Draw Lines . It takes two points **(x1, y1), (x2, y2)** and thus makes a Straight Line b/w those Two Points. Where 2 represents thickness .

Then I have defined the Main Body of the Program where I have used **cv2.imread(input("Write file name with its path: "))** . It takes user input and **cv2.imread()** is used to read the File.

We gave these ten corresponding points

MatchingImagePoints = [ [843, 421], [762, 216], [857, 239], [853, 281], [861, 191], [160, 118], [238, 167], [80, 146], [325, 62], [129, 190]]

MatchingFieldPoints = [ [782, 469], [750, 300], [781, 303], [781, 361], [781, 240], [480, 225], [556, 301], [481, 300], [481, 18], [530, 358]]

**if len(MatchingImagePoints) > 3 and len(MatchingImagePoints) == len(MatchingFieldPoints):**

**break**

We have used it make sure We DIDN'T GAVE LESS THAN 4 POINTS and that Same number of Image and Filed Points are Selected. Else it will Break the loop.

Then I used all the above defined functions ,in proper order , to get our desired result.

To compute the distance , used this formula:

**Distance = np.absolute(line\_points[0][0] - line\_points[2][0]) \* width\_per\_pixel**

We subtracted the x coordinates of the two lines to find the Distance b/w them . Then apply np.absolute to make it positive .And Then Multiplied by width\_per\_pixel to get Distance in meter or cm units but not in pixel units.

At last, we use **cv2.imshow('Output', image)** to Show the Image and use

**if cv2.waitKey(0):**

**cv2.destroyAllWindows()**

To destroy all windows WHEN THE USER Press any Button.