GTU

DEPARTMENT OF COMPUTER ENGINEERING

CSE 463 – Spring 2022

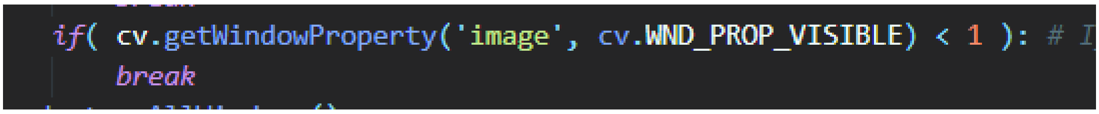
HOMEWORK 1 REPORT

SÜLEYMAN GÖLBOL

1801042656

1. PROBLEM SOLUTION APPROACH AND ALGORITHMS

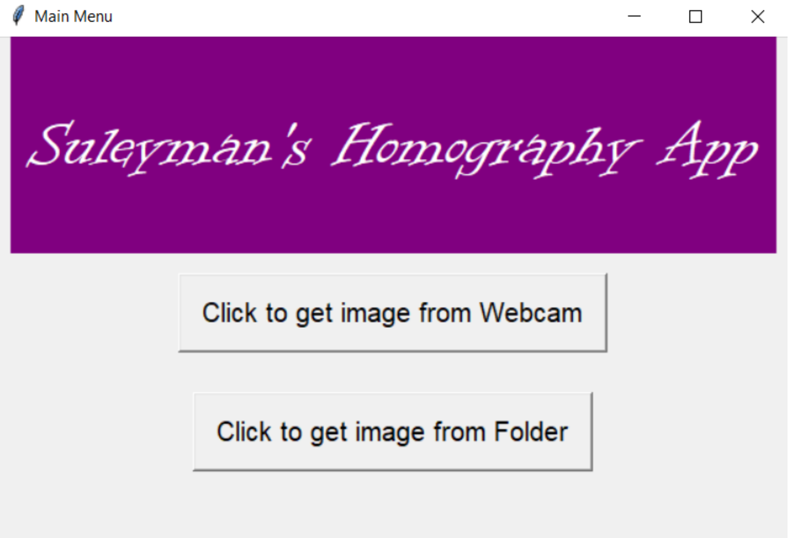
My first problem that I have encountered in application was OpenCV’s window problems. For example, OpenCV doesn’t close the window when user clicks into X button. To solve this problem I named window and used the function below.



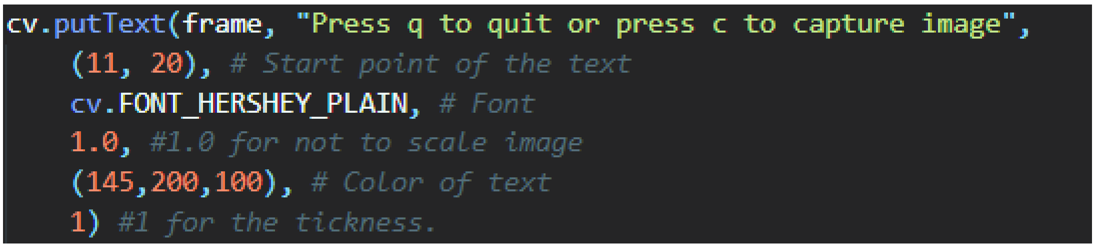
```
    if( cv.getWindowProperty('image', cv.WND_PROP_VISIBLE) < 1 ): # I
        break
```

To get the 4 points from user I used `if event == cv.EVENT_LBUTTONDOWN:` to get the mouse left clicks from user and in every click, I saved the position into a list and after 4 points, I sent these points to my Homography module.

To select button option between webcam or folder; I used tkinter module.


Main Menu
Suleyman's Homography App
Click to get image from Webcam
Click to get image from Folder

If user opens webcam to save image from it, in the left top of the screen, a text appears for the exit and capture keyboard shortcuts.

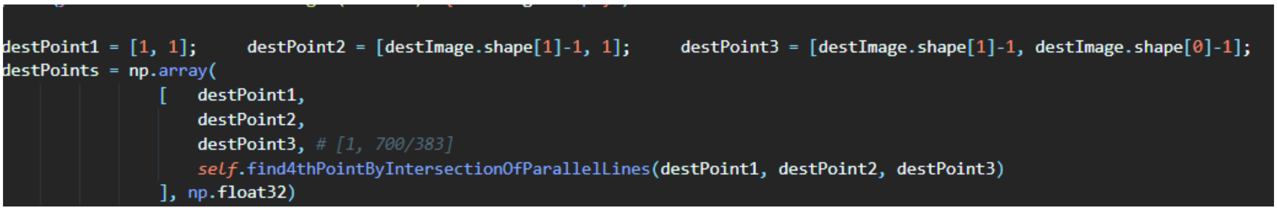


```
cv.putText(frame, "Press q to quit or press c to capture image",
    (11, 20), # Start point of the text
    cv.FONT_HERSHEY_PLAIN, # Font
    1.0, #1.0 for not to scale image
    (145,200,100), # Color of text
    1) #1 for the tickness.
```

After user selects the image and save the 4 points of input image, the Homography class is being called.

I INTERSECTION OF PARALLEL LINES

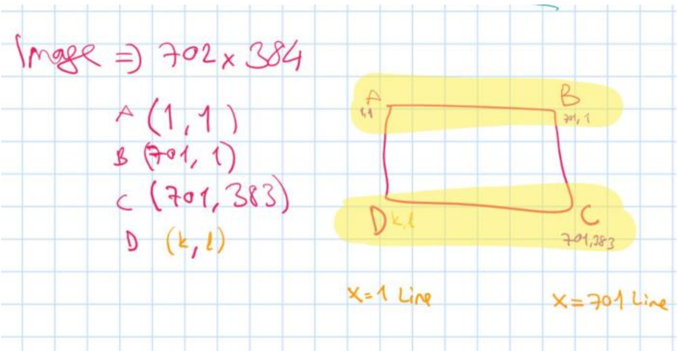
To find 4th point from the intersection of parallel lines in the soccer model field, I created a method called `find4thPointByIntersectionOfParallelLines`.

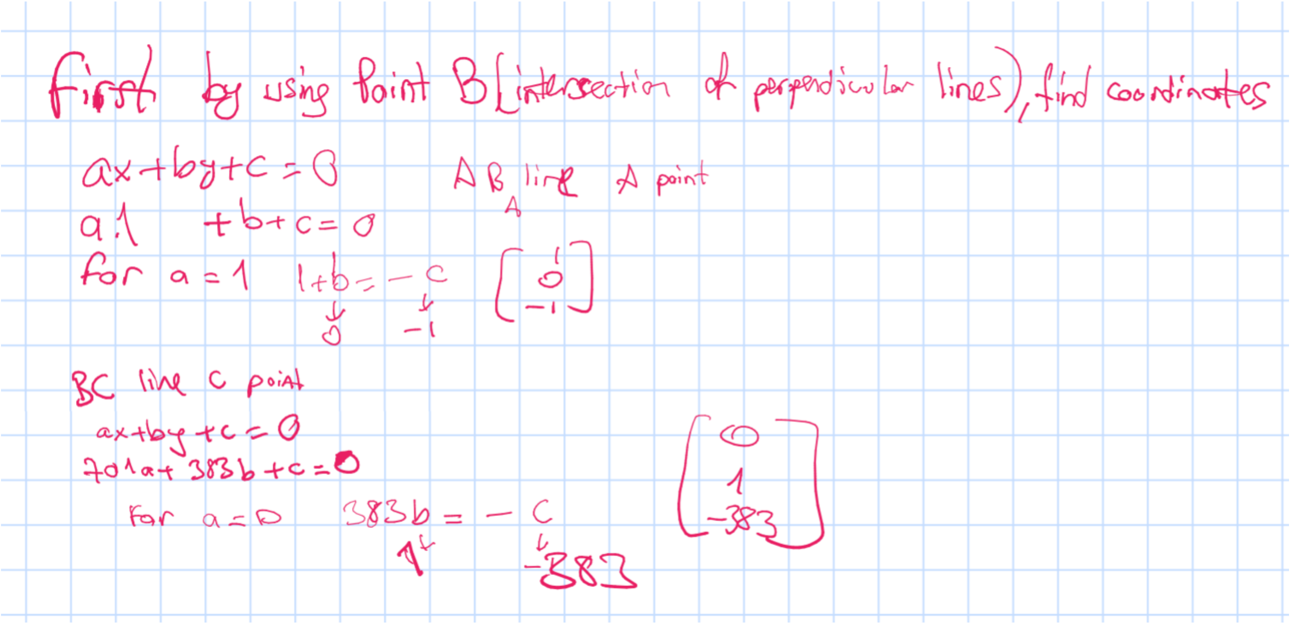


```
destPoint1 = [1, 1];     destPoint2 = [destImage.shape[1]-1, 1];     destPoint3 = [destImage.shape[1]-1, destImage.shape[0]-1];
destPoints = np.array(
                [   destPoint1,
                    destPoint2,
                    destPoint3, # [1, 700/383]
                    self.find4thPointByIntersectionOfParallelLines(destPoint1, destPoint2, destPoint3)
                ], np.float32)
```

1) using P1 value from P1P2 line to get the first coordinate vector of the line
2) using P3 value from P3P4 line to get the second coordinate vector of the line
3) using the intersection of the two lines to get the 4th point [using determinant formula]

Example from our model soccer field:





First by using Point B [intersection of perpendicular lines), find coordinates

$ax+by+c=0$ AB line A point

$a.1 \quad +b+c=0$

for $a=1$ $\quad 1+b=-c$ $\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$

$b \to 0 \quad c \to -1$

BC line C point

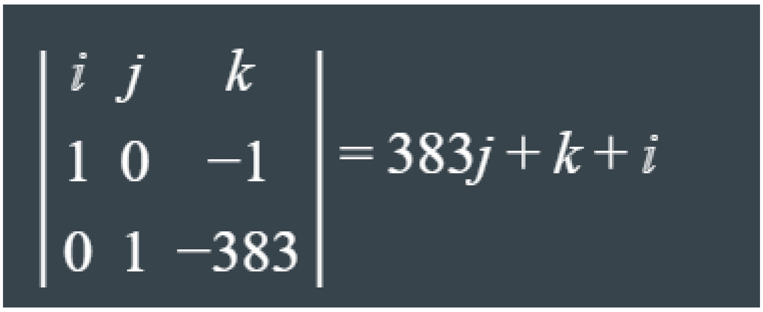
$ax+by+c=0$

$701a+383b+c=0$

For $a=0$ $\quad 383b=-c$ $\begin{bmatrix} 0 \\ 1 \\ -383 \end{bmatrix}$

$b \to 1 \quad c \to -383$

As you can see from our points in parallel lines, we obtained 2 vectors. To find the intersection point that's on infinity, we need to use determinant. Coefficient of i is 1, and coefficient of j is 383 so I used this as 4th point of model field.

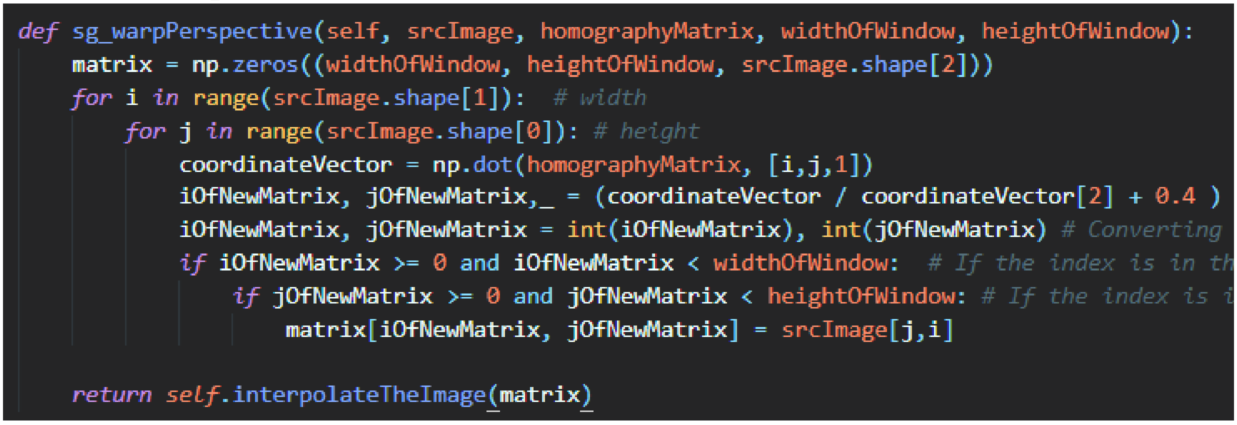


$$\begin{vmatrix} i & j & k \\ 1 & 0 & -1 \\ 0 & 1 & -383 \end{vmatrix} = 383j + k + i$$

My warpPerspective Implementation with Homography Formulas

As it says in PDF, I didn't use any img processing function from OpenCV expect in pdf's. https://docs.opencv.org/2.4/modules/imgproc/doc/imgproc.html

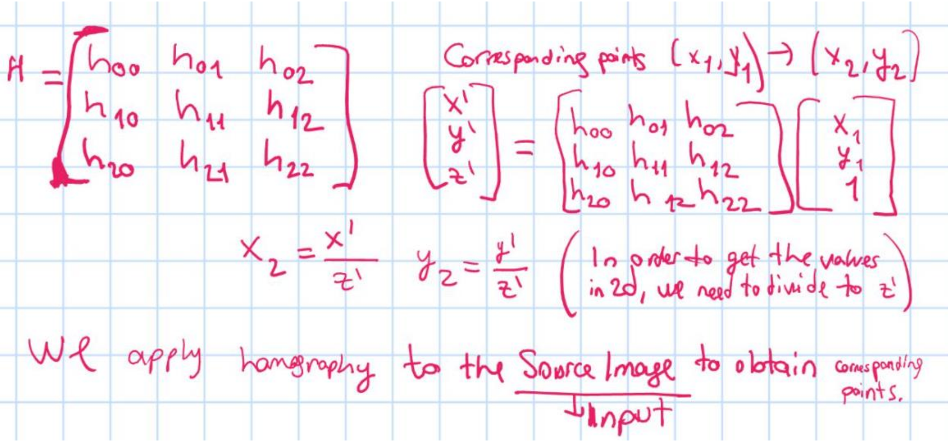
And unfortunately warpPerspective is a image processing function. So I needed to implement it.



```
def sg_warpPerspective(self, srcImage, homographyMatrix, widthOfWindow, heightOfWindow):
    matrix = np.zeros((widthOfWindow, heightOfWindow, srcImage.shape[2]))
    for i in range(srcImage.shape[1]):  # width
        for j in range(srcImage.shape[0]): # height
            coordinateVector = np.dot(homographyMatrix, [i,j,1])
            iOfNewMatrix, jOfNewMatrix,_ = (coordinateVector / coordinateVector[2] + 0.4 )
            iOfNewMatrix, jOfNewMatrix = int(iOfNewMatrix), int(jOfNewMatrix) # Converting
            if iOfNewMatrix >= 0 and iOfNewMatrix < widthOfWindow:  # If the index is in th
                if jOfNewMatrix >= 0 and jOfNewMatrix < heightOfWindow: # If the index is i
                    matrix[iOfNewMatrix, jOfNewMatrix] = srcImage[j,i]

    return self.interpolateTheImage(matrix)
```

I took the source(input) image, the homography matrix, and the size of destination image.



$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

Corresponding points $(x_1, y_1) \rightarrow (x_2, y_2)$

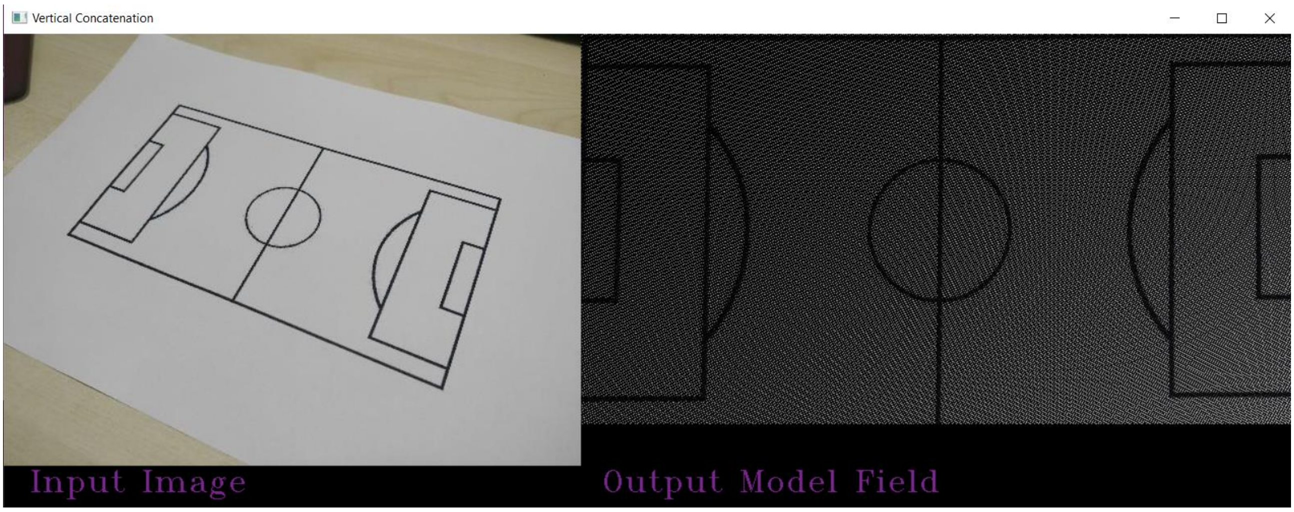
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{12} & h_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$x_2 = \frac{x'}{z'} \quad y_2 = \frac{y'}{z'}$$ (In order to get the values in 2d, we need to divide to z')

We apply homography to the $\underbrace{\text{Source Image}}_{\text{Input}}$ to obtain corresponding points.

As in the formula above, when we create a vector of [i,j,1] and we multiply with our homography matrix, we get some values, and if we divide these to z’, we get corresponding points.

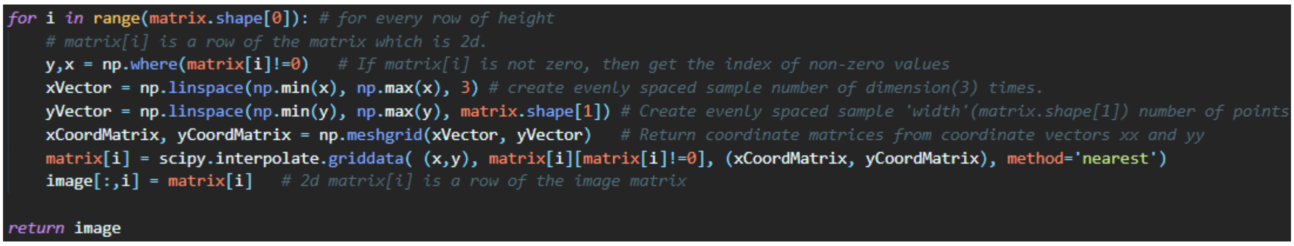
Interpolating Problem



If I don’ use a function to interpolating the image, that’s what it’s gonna look like. But we don’t want that. So, I needed to interpolating the matrix.

There were easy functions to interpolating the image like cv2.resize [But I didn’t use it because it’s a image processing function]. Also there were functions in PIL, scimage but they are also image processing libraries so I didn’t use them.

At the end I used array interpolating function griddata() from scipy.



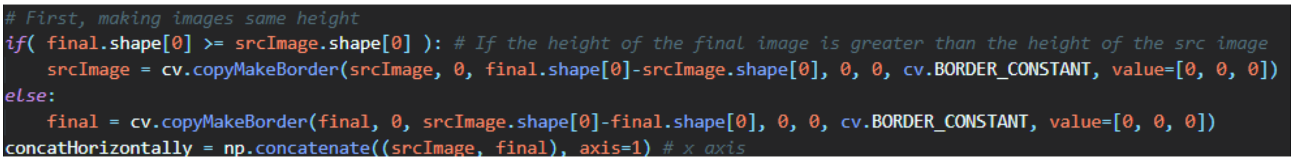
```
for i in range(matrix.shape[0]): # for every row of height
    # matrix[i] is a row of the matrix which is 2d.
    y,x = np.where(matrix[i]!=0)   # If matrix[i] is not zero, then get the index of non-zero values
    xVector = np.linspace(np.min(x), np.max(x), 3) # create evenly spaced sample number of dimension(3) times.
    yVector = np.linspace(np.min(y), np.max(y), matrix.shape[1]) # Create evenly spaced sample 'width'(matrix.shape[1]) number of points
    xCoordMatrix, yCoordMatrix = np.meshgrid(xVector, yVector)   # Return coordinate matrices from coordinate vectors xx and yy
    matrix[i] = scipy.interpolate.griddata( (x,y), matrix[i][matrix[i]!=0], (xCoordMatrix, yCoordMatrix), method='nearest')
    image[:,i] = matrix[i]   # 2d matrix[i] is a row of the image matrix

return image
```

This function, checks every 2d array inside my image matrix to find non zero points and puts them into y variable. Then between these y’s interval I create sample data width times. Then I create coordinate matrices from these samples. And I used griddata() to obtain 2d matrix with nearest interpolation method. I applied this for every row of height matrix. After I put this to my image’s ith column, that’s it with interpolation.

Showing Images Together

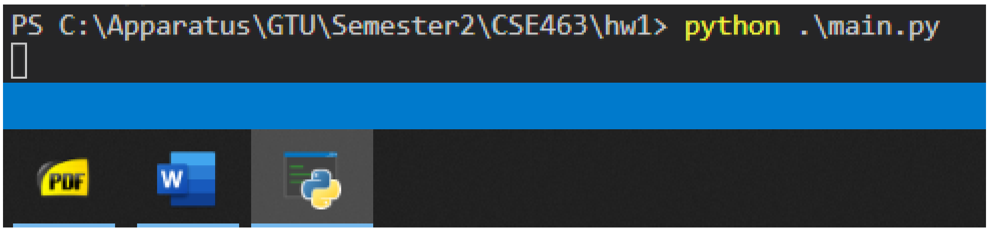
Normally, in order to show images together we can concatenate matrices with numpy.concatanate() function, but it’s only working for same height images. So I checked the height sizes, and created a border with black color in order to match the sizes of height. Then I used concatenate() to show them in the same window.



```
# First, making images same height
if( final.shape[0] >= srcImage.shape[0] ): # If the height of the final image is greater than the height of the src image
    srcImage = cv.copyMakeBorder(srcImage, 0, final.shape[0]-srcImage.shape[0], 0, 0, cv.BORDER_CONSTANT, value=[0, 0, 0])
else:
    final = cv.copyMakeBorder(final, 0, srcImage.shape[0]-final.shape[0], 0, 0, cv.BORDER_CONSTANT, value=[0, 0, 0])
concatHorizontally = np.concatenate((srcImage, final), axis=1) # x axis
```

3) TESTS AND RESULTS

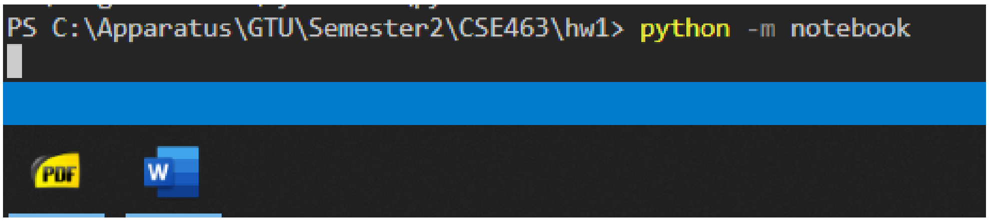
Running python code with `python main.py`



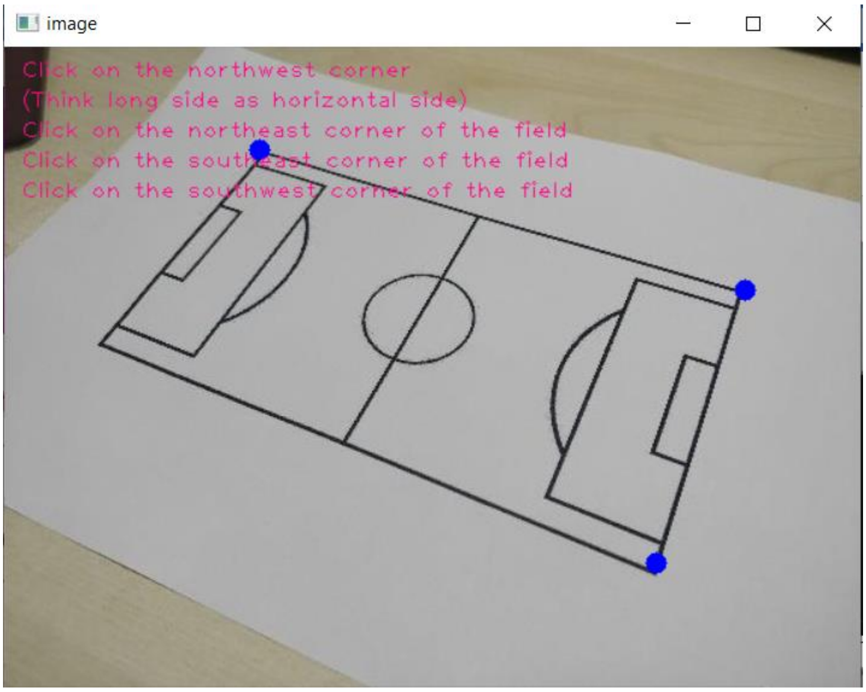
```
PS C:\Apparatus\GTU\Semester2\CSE463\hw1> python .\main.py
```

Running with Jupyter Notebook

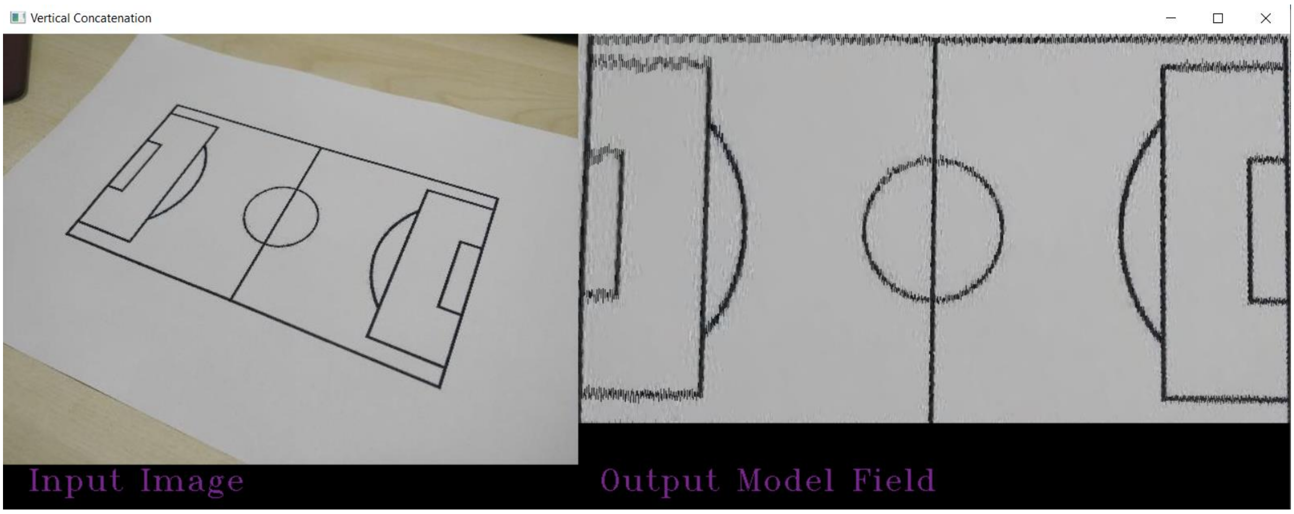
Homograph.py and PointSaving.py should be in the same folder with notebook file nbook_main.ipynb

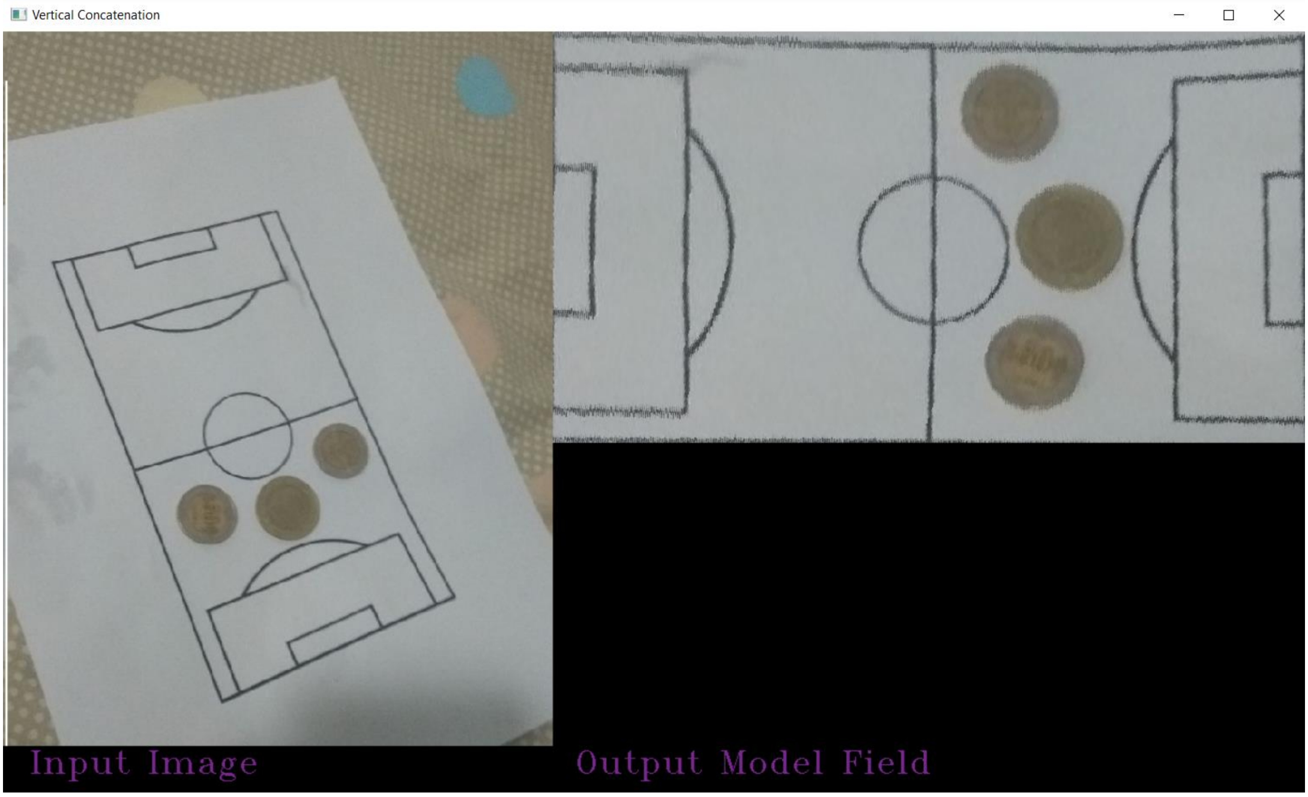


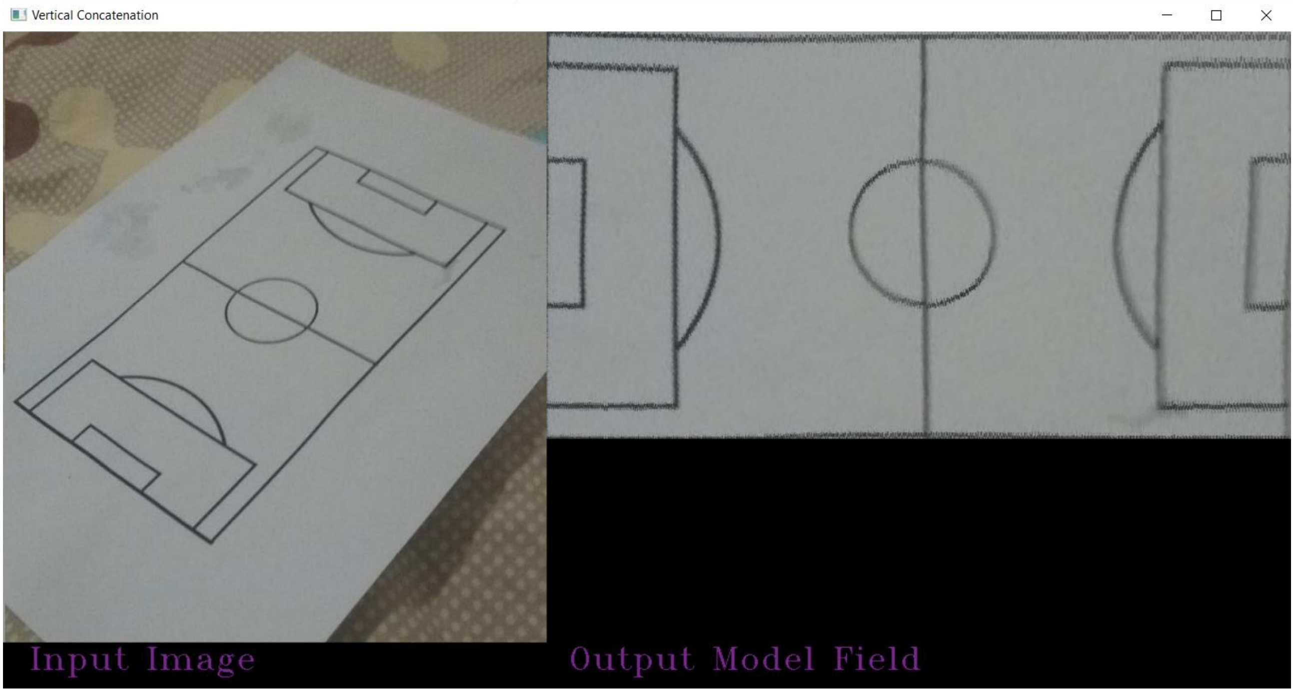
```
PS C:\Apparatus\GTU\Semester2\CSE463\hw1> python -m notebook
```



Click on the northwest corner
(Think long side as horizontal side)
Click on the northeast corner of the field
Click on the southeast corner of the field
Click on the southwest corner of the field


Vertical Concatenation
Input Image
Output Model Field


Vertical Concatenation
Input Image
Output Model Field


Vertical Concatenation
Input Image
Output Model Field