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.

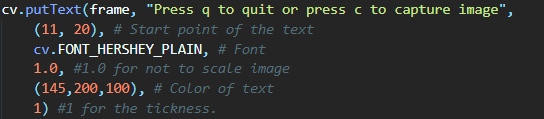


To get the 4 points from user I used if event == cv.EVENT\_LBUTTONDOWN: to get the mouse left clicks from user and in every click, I saved the position into a list and after 4 points, I sent these points to my Homography module.

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



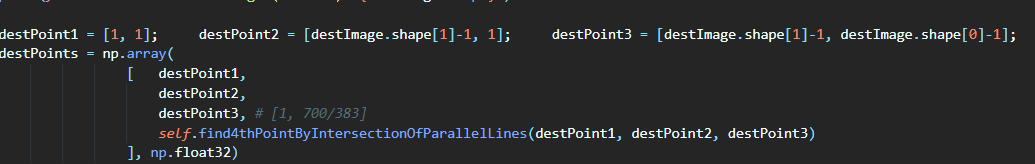
If user opens webcam to save image from it, in the left top of the screen, a text appears for the exit and capture keyboard shortcuts.



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

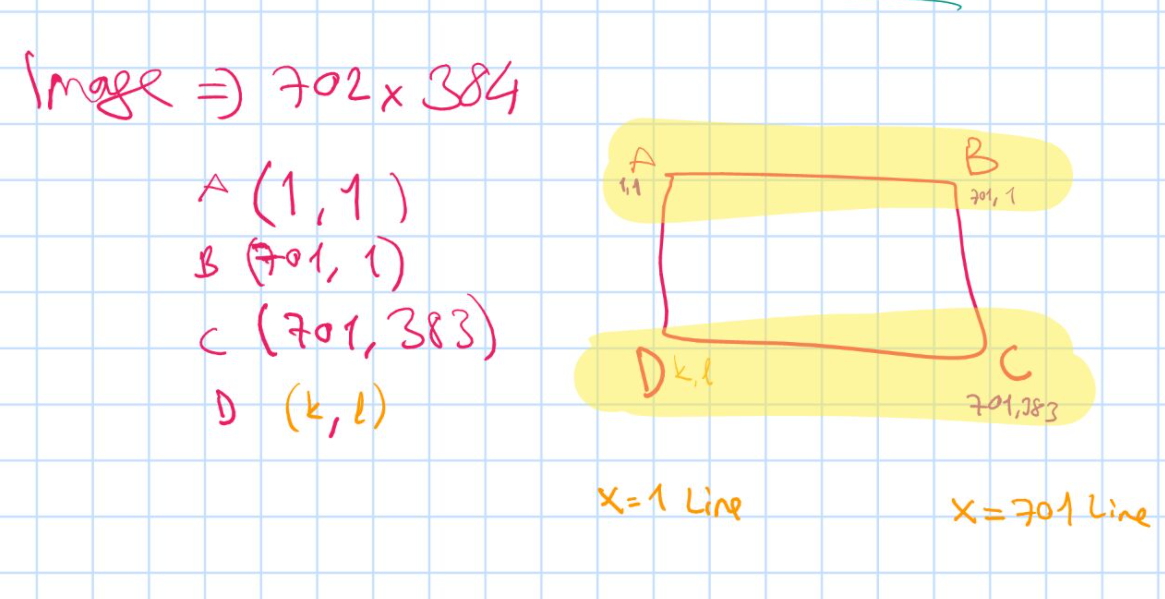
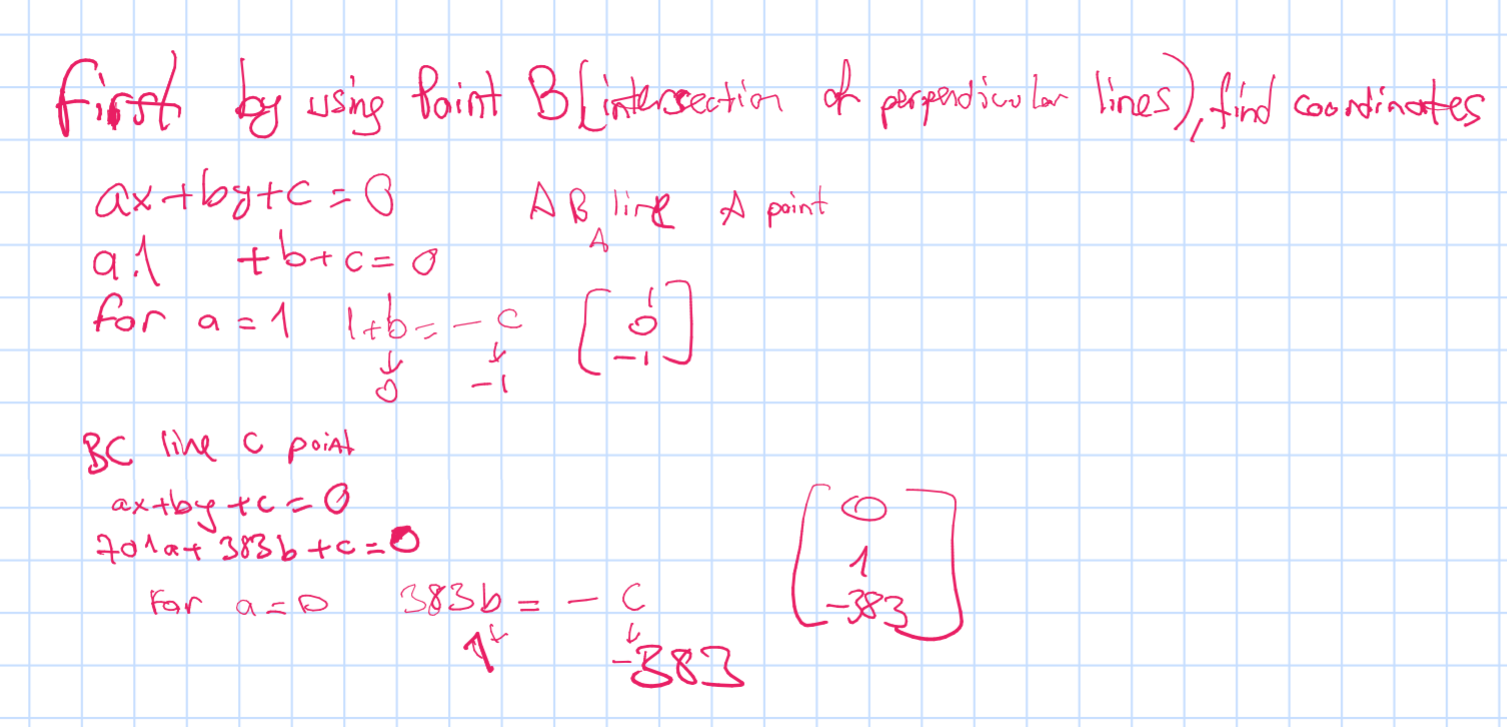
**I** **INTERSECTION OF PARALLEL LINES**

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

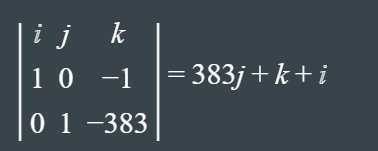


1) using P1 value from P1P2 line to get the first coordinate vector of the line  
2) using P3 value from P3P4 line to get the second coordinate vector of the line  
3) using the intersection of the two lines to get the 4th point [using determinant formula]

Example from our model soccer field:

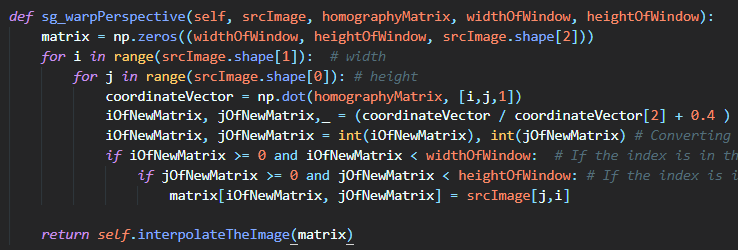
  


As you can see from our points in parallel lines, we obtained 2 vectors.  
To find the intersection point that’s on infinity, we need to use determinant. Coefficient of i is 1, and coefficient of j is 383 so I used this as 4th point of model field.

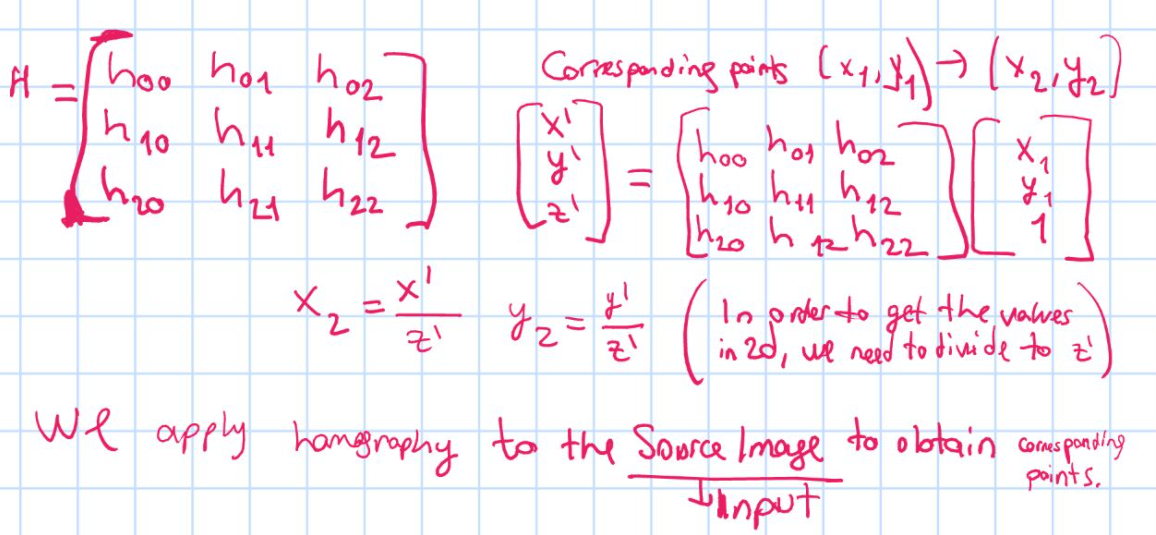


**My warpPerspective Implementation with Homography Formulas**

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>

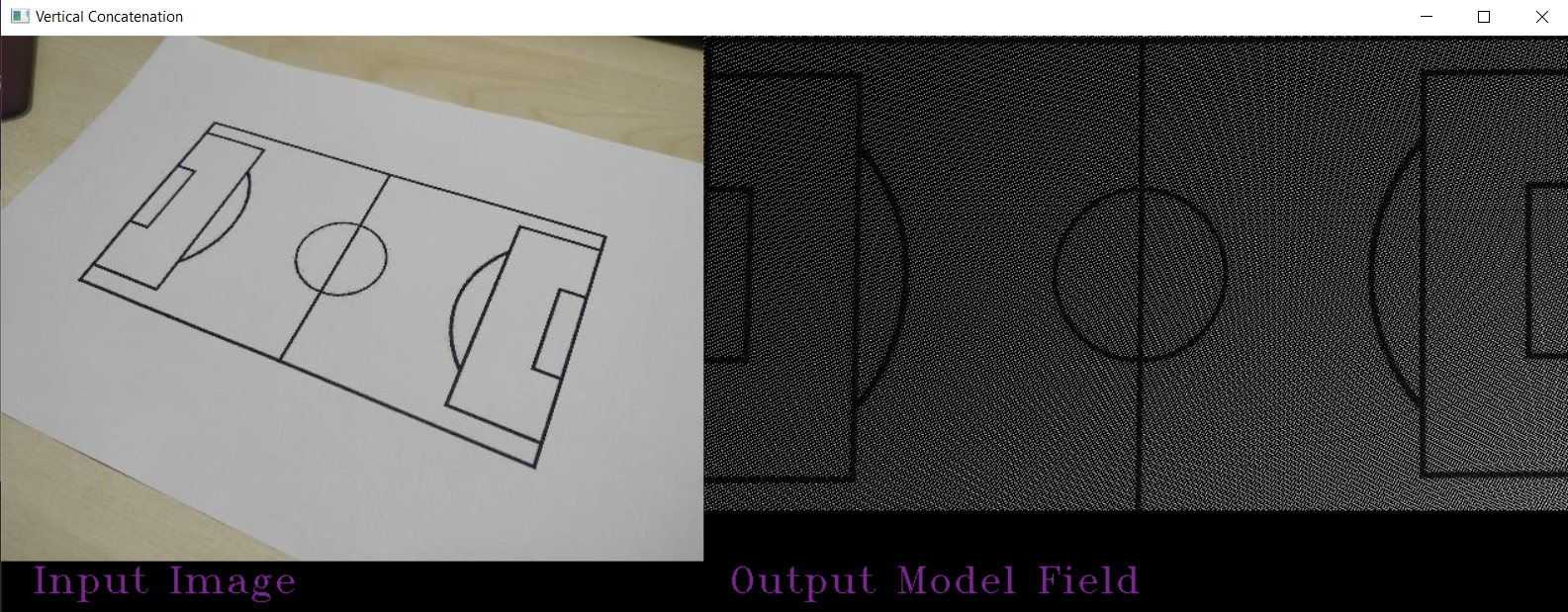
And unfortunately warpPerspective is a image processing function. So I needed to implement it.  


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



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.

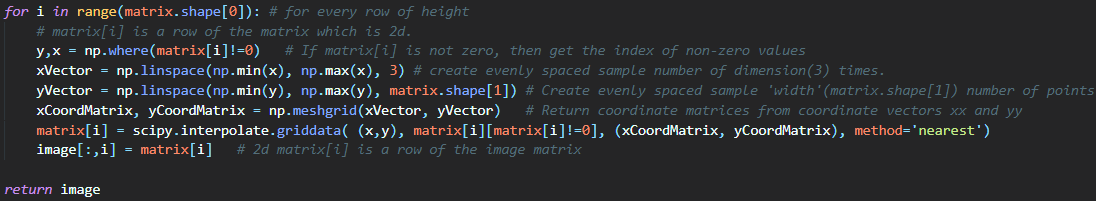
**Interpolating Problem**



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.

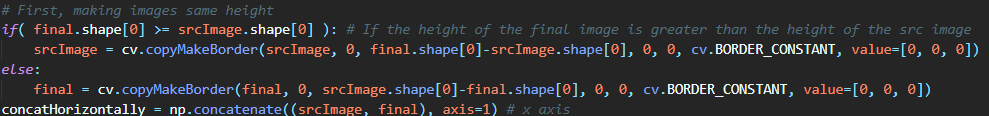
There were easy functions to interpolating the image like cv2.resize [But I didn’t use it because it’s a image processing function]. Also there were functions in PIL, scimage but they are also image processing libraries so I didn’t use them.

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



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.

**3 ) TESTS AND RESULTS**

Running python code with python main.py

# 

# Running with Jupyter Notebook

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

# nbook\_main.ipynb

# 

# 

# 

# 

# 