September 13, 2020

Offside or not?

Using homography via Python OpenCV library for detection of offside in the screenshots taken from real-life football matches

Final Project for Computer Vision Class
Given at Bahcesehir University in Summer 2020 Term

Hasan Fahri YETIM
1901456

Goal of the project

With this project, I aim at deciding if a position is offside using Python's OpenCV.

Screenshots are taken from a video uploaded on YouTube for explaining the rule of offside. Link to the video is as follows:
https://www.youtube.com/watch?v=D-mcLd3GAPc

Method used in the project

For the aim explained above, I created my algorithm to use the homography method.

Homography expresses the linear relationship between multiple images of a given plane to be used in translating perspectives. A 3x3 homography matrix can be used to relate different points in each image, such that:

$$x_2 = Hx_1 \tag{1}$$

where x_1 and x_2 are the points in images that show the same point in the target plane (OpenCV, n.d., "Basic theory – What is the homography matrix?"). This way, it allows translating points in an image to the ones in the other, without the need to know about positions of cameras capturing the images. The equation above can be expressed in matrix form as:

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \tag{2}$$

where u_1 and v_1 are the pixel notation of the place of the point in image 1, and u_2 and v_2 are that of the point in image 2. In order to normalize, H_{33} can be set to 1 or sum of squares of

each element of the H matrix can be set to 1. This leaves us with 8 degrees of freedom. Therefore, we can estimate homography matrix by selecting four points from each image representing same points in each pair (OpenCV, n.d., "Basic theory – What is the homography matrix?").

The algorithm used in this project is a special application of homography to images of potential offside situations captured from different places in the grandstand. In this application, four points among the corners of main lines in the football pitch are selected in each input image to serve as the focal points for homographic translation. The figure below illustrates how points in different images can be translated into related points on the actual planar surface by using the homography matrix.

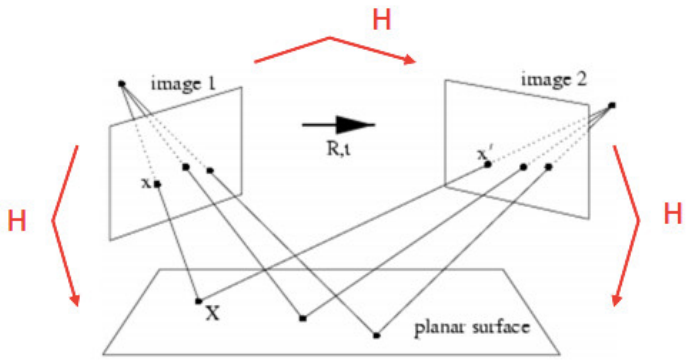


Figure 1 Translation of points to the ones in the target planer surface using the homography matrix (OpenCV, n.d., "Basic theory – What is the homography matrix?")

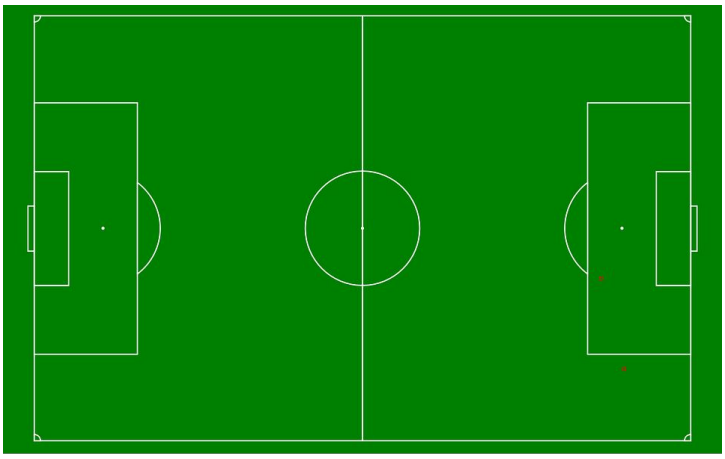
BIBLIOGRAPHY

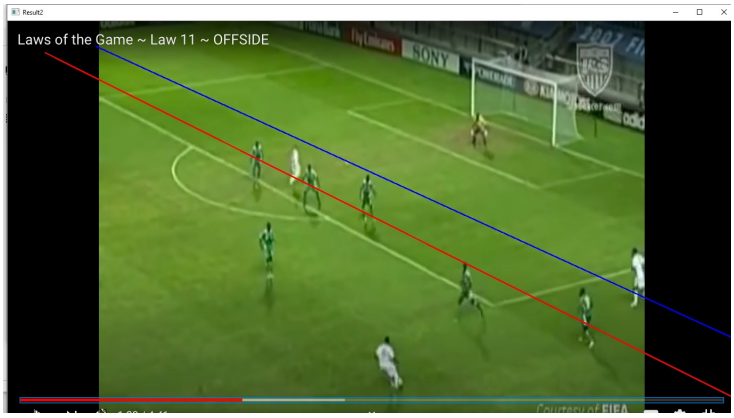
OpenCV. (n.d.). *Basic concepts of the homography explained with code*. Retrieved September 13, 2020, from https://docs.opencv.org/master/d9/dab/tutorial_homography.html

Inputs and Outputs

Input #1

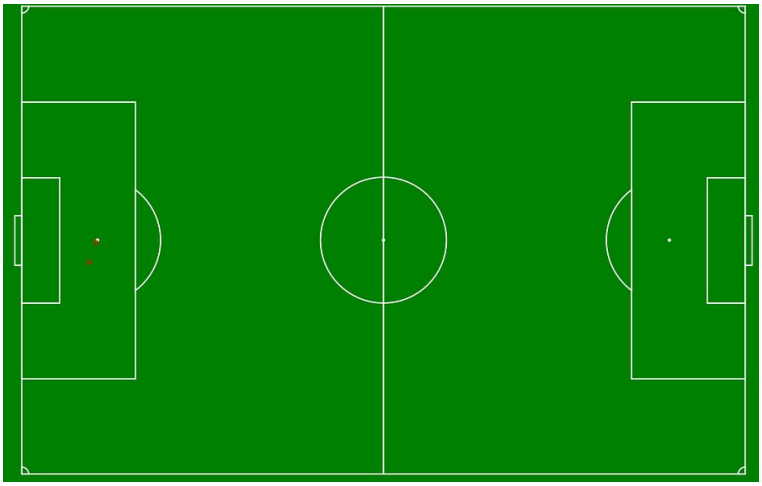
Outputs of Input #1





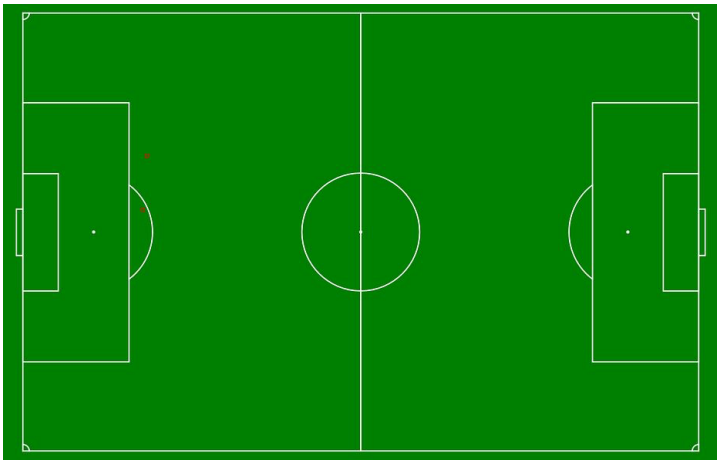
Input #2

Outputs of Input #2



Input #3

Outputs of Input #3



Source code

```
import cv2
import numpy as np

cursor_coordinates = []
field_coordinates = []
screenshot_coordinates = []


def get_matchings(screenshot, field):
    print("\n\nIn this step, you are given two screenshots and asked to click on the
points"
          "that are visible and mutual in two screenshots. You can click either one by
one or"
          "group by group, but the order is important. \n!!! Press ESC when you are done
!!!\nIf "
          "pairs are not equal in length you are going to be prompted again.")
    input("Press enter when ready")

    while 1:
        cv2.namedWindow('field coordinates')
        cv2.setMouseCallback('field coordinates', get_field_coordinates, param=field)
        cv2.imshow('field coordinates', field)

        cv2.namedWindow('screenshot coordinates')
        cv2.setMouseCallback('screenshot coordinates', get_screenshot_coordinates,
param=screenshot)
        cv2.imshow('screenshot coordinates', screenshot)
        if cv2.waitKey(20) & 0xFF == 27:
            break

    cv2.destroyAllWindows()
    return screenshot_coordinates, field_coordinates


def get_user_points(screenshot, num_of_points):
    print("\n\nOn the following screenshot, click on the attacker's and the defender's
positions"
          "in same order specified.")
    input("Press enter when ready")

    while 1 and len(cursor_coordinates) < num_of_points:
        global field_coordinates, screenshot_coordinates
        field_coordinates = []
        screenshot_coordinates = []
        cv2.namedWindow('Player coordinates')
```

```
        cv2.setMouseCallback('Player coordinates', get_cursor_coordinates,
param=screenshot)
        cv2.imshow('Player coordinates', screenshot)
        if cv2.waitKey(20) & 0xFF == 27:
            break

    print("Points selected!")
    cv2.destroyAllWindows()
    return cursor_coordinates


def determine_offside(x1, x2):
    offside = False
    # Attack is to the right
    if x1 > 480:
        offside = True if x1 > x2 else False
    else:
        offside = True if x2 > x1 else False
    return offside


def draw_on_field(field, output_points):
    line_points = []
    normals = []
    for point in output_points:
        x, y = point
        x = int(x)
        y = int(y)
        line_points.append([x, 17])
        line_points.append([x, 584])
        normals.append([x, y, 0])
        normals.append([x, y, 10])
        cv2.rectangle(field, (x, y), (x + 4, y - 4), [0, 0, 255])
    return line_points, normals


def draw_from_field_to_screenshot(screenshot, reverse_output_points):
    for i in range(0, len(reverse_output_points), 2):
        x1, y1 = reverse_output_points[i]
        x2, y2 = reverse_output_points[i + 1]

        x1 = int(x1)
        y1 = int(y1)
        x2 = int(x2)
        y2 = int(y2)

        color = [255, 0, 0] if i < 1 else [0, 0, 255]
        cv2.line(screenshot, (x1, y1), (x2, y2), color, 2)
```

```
def main():
    print('This program reads a screenshot file and takes multiple point pairs from '
          'two football field screenshots. Then asks for the attacker and defender\'s
position.'
          'Finally it draws perpendicular and parallel lines to field ground with '
          'the distance and offside decision on bottom left.')

    file = input('Type file path including file name: ')
    filename = file.split('/')[-1]
    field_file = './field.png'
    screenshot = cv2.imread(file)
    field = cv2.imread(field_file)

    # Get user's points
    user_points = np.array(get_user_points(screenshot, 2), dtype='float32')
    cv2.destroyAllWindows()

    field_height = 567
    field_width = 876

    screenshot_matching = None
    field_matching = None

    isMathingsOk = False
    while not isMathingsOk:
        copy_screenshot = np.copy(screenshot)
        copy_field = np.copy(field)
        screenshot_matching, field_matching = get_matchings(copy_screenshot,
copy_field)
        if len(screenshot_matching) > 3 and len(screenshot_matching) ==
len(field_matching):
            print("Got {0} pairs.".format(len(screenshot_matching)))
            break
        global field_coordinates, screenshot_coordinates
        field_coordinates = []
        screenshot_coordinates = []

        print("Need at least four pairs")

    print("\n\n{0} pairs selected.".format(len(screenshot_matching)))

    # Get field dimensions
    length = 105
    height = 68

    #while length is None or height is None:
```

```
    #    try:
    #        length_string = input("Please write the field's width (default: 90) [90m,
120m]: ")
    #        length_string = "90" if not length_string else length_string
    #        length = int(length_string)

    #        height_string = input("Please write the field's height (default: 45) [45m, 90m]:
")
    #        height_string = "45" if not height_string else height_string
    #        height = int(height_string)

    #        break
    #    except ValueError:
    #        print('Wrong value type :(, try again.')
    #        pass

    # Get distance per pixel values
    width_per_pixel = length / field_width
    height_per_pixel = height / field_height

    # Convert python arrays to numpy arrays
    screenshot_matching = np.array(screenshot_matching)
    field_matching = np.array(field_matching)

    # Find matching between given points
    h, status = cv2.findHomography(screenshot_matching, field_matching)

    # Map user selected players' coordinates to field screenshot
    input_coor = np.array([user_points])
    output = cv2.perspectiveTransform(input_coor, h)[0]
    line_points, normals = draw_on_field(field, output)

    length = np.absolute(line_points[0][0] - line_points[2][0]) * width_per_pixel
    offside = determine_offside(line_points[0][0], line_points[2][0])

    # Get line points from field screenshot
    reverse_output = cv2.perspectiveTransform(np.array([line_points], dtype='float32'),
np.linalg.inv(h))[0]
    draw_from_field_to_screenshot(screenshot, reverse_output)

    # Show field with point locations
    cv2.imshow('Result', field)

    # Show screenshot with lines, offside decision and distance
    font = cv2.FONT_HERSHEY_SIMPLEX
    screenshot_h, screenshot_w, ch = screenshot.shape

    cv2.putText(screenshot, 'Distance: {0:.2f} metres'.format(length),
```

```
            (10, screenshot_h-32), font, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
    offside_color = [255, 0, 0] if not offside else [0, 0, 255]
    cv2.putText(screenshot, 'Offside: {0}'.format(offside),
            (10, screenshot_h-15), font, 0.5, offside_color, 1, cv2.LINE_AA)
    cv2.imshow('Result2', screenshot)
    cv2.waitKey(0)
    cv2.imwrite('./outputs/' + filename, screenshot)
    cv2.imwrite('./outputs/field_'+filename, field)
    print("Offside: {0}\tDistance between lines: {1}".format(offside, length))


def get_cursor_coordinates(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        cursor_coordinates.append([x, y])


def get_field_coordinates(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.rectangle(params, (x-2, y-2), (x + 2, y + 2), [0, 0, 255])
        field_coordinates.append([x, y])


def get_screenshot_coordinates(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.rectangle(params, (x-2, y-2), (x + 2, y + 2), [0, 0, 255])
        screenshot_coordinates.append([x, y])


if __name__ == '__main__':
    main()
```