CIS 580, Machine Perception, Spring 2020
Homework 2
Due: Wednesday Feb 5 2020, 11:59pm

Instructions

- This is an individual homework and worth 100 points
- You must submit your solutions on Gradescope, the entry code is MB8ZJP. We recommend that you use LaTeX, but we will accept scanned solutions as well.
- Please complete this homework with Python 3. To run this homework successfully, you will need the following python packages:
  - numpy
  - matplotlib
  - opencv-python
- Start early! Please post your questions on Piazza or come to office hours!

Submission

- You will submit a concise report to Gradescope that includes a discussion on the algorithms implemented as well as results from multiple frames.
- You will also submit all python files you completed as .py files to Gradescope

1 Barcelona - 50 pts

1.1 Introduction

In this programming assignment, we will use the concepts of projective geometry and homographies to allow us to project an image onto a scene in a natural way that respects perspective. To demonstrate this, we will project our logo onto the goal during a football match. For this assignment, we have provided images from a video sequence of a football match, as well as the corners of the goal in each image and an image of the Penn Engineering logo. Your task is, for each image in the video sequence, compute the homography between the Penn logo and the goal, and then warp the goal points onto the ones in the Penn logo to generate a projection of the logo onto the video frame (e.g. Figure 1).

(a) Original image from video (b) Image with Penn logo projected

Figure 1: Example of logo projection

1.2 Technical Details

The Python script **project_logo.py** will be the main script to run this assignment. In this script, we provide you the following:

- A sequence of images onto which you will project a logo image.
- The corners in each video frame that the logo should project onto.
- The logo image itself.

Your goal will be to complete the functions **est_homography** and **warp_pts** in their respective .py files. **est_homography** estimates the homography that maps the video image points onto the logo points (i.e. $\mathbf{x}_{logo} \sim \mathbf{x}_{video}$), and **warp_pts** then warps the sample points according to this homography, returning the warped positions of the points (that is, the corresponding points in the logo image). We then use these correspondences to copy the points from the logo into the video image. Once you finish these two functions, change your current directory to the directory of "Part_1", and run "python project_logo.py". It will use the two functions you completed to project logo, visualize them in a video, and save sampled results.

1.3 Homography Estimation

To project one image patch onto another, we need, for each point inside the goal in the video frame, to find the corresponding point from the logo image to copy over. In other words, we need to calculate the homography between the two image patches. This homography is a 3x3 matrix that satisfies the following:

$$\mathbf{x}_{logo} \sim H\mathbf{x}_{video} \tag{1}$$

Or, equivalently:

$$\lambda\mathbf{x}_{logo} = H\mathbf{x}_{video} \tag{2}$$

Where $\mathbf{x}_{logo}$ and $\mathbf{x}_{video}$ are homogeneous image coordinates from each patch and λ is some scaling constant. To calculate the homography needed for this projection, we provide, for each

image, the corners of the patches that we would like you to warp between in each image. You must calculate the homography using the provided corner points and the technique covered in the lectures and Appendix A. You can then warp each image point using H to find its corresponding point in the logo (note that the homography equation is estimated up to a scalar, so you will need to divide $H\mathbf{x}_{image}$ by the third term, which is λ), and then return the set of corresponding points as a matrix.

1.4 Inverse Warping

You may be wondering why we are calculating the projection from the video frames to the logo image, when we want to project the logo image onto the video frames. We do this because, if we compute the inverse homography, and project all the logo points into the video frame, we will most likely have the case where multiple logo points project to one video frame pixel (due to rounding of the pixels), while other pixels may have no logo points at all. This results in 'holes' in the video frame where no logo points are mapped. To avoid this, we calculate the projection from video frame points to logo points to guarantee that every video frame gets a point from the logo.

We can then replace every point in the video frame ($\mathbf{x}_{video}$) with the corresponding point in the logo ($\mathbf{x}_{logo}$) using the correspondences ($\mathbf{x}_{image}, \mathbf{x}_{logo}$). This is already done for you in **inverse_warping**, and you should not need to change it.

1.5 Files to complete and submit

1. **est_homography.py**
   This function is responsible for computing the homography given correspondence
2. **warp_pts.py**
   This function is responsible for computing the warped points given correspondence and a set of sample points

1.6 Visualizing Results

To play the projected images as a video, run the **project_logo.py** script. First open a terminal window and change the current directory to the directory of "Part_1". Then run command:

python project_logo.py

You can also use an IDE to run the script. If you would like to play with your own data, you can edit **project_logo** and generate your own video with a set of points.

1.7 Appendix A: Calculating Homographies

As we learned in the lectures, a homography H maps a set of points $\mathbf{x} = \begin{pmatrix} x & y & 1 \end{pmatrix}^T$ to another set of points $\mathbf{x}' = \begin{pmatrix} x' & y' & 1 \end{pmatrix}$ up to a scalar:

$$\mathbf{x}' \sim H\mathbf{x} \tag{3}$$

$$\lambda \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4}$$

$$\lambda x' = h_{11}x + h_{12}y + h_{13} \tag{5}$$

$$\lambda y' = h_{21}x + h_{22}y + h_{23} \tag{6}$$

$$\lambda = h_{31}x + h_{32}y + h_{33} \tag{7}$$

In order to recover x' and y', we can divide equations (5) and (6) by (7):

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \tag{8}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{9}$$

Rearranging the terms above, we can get a set of equations that is linear in the terms of H:

$$-h_{11}x - h_{12}y - h_{13} + h_{31}xx' + h_{32}yx' + h_{33}x' = 0 \tag{10}$$

$$-h_{21}x - h_{22}y - h_{23} + h_{31}xy' + h_{32}yy' + h_{33}y' = 0 \tag{11}$$

Finally, we can write the above as a matrix equation:

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} h = 0 \tag{12}$$

Where:

$$a_x = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \end{pmatrix} \tag{13}$$

$$a_y = \begin{pmatrix} 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{pmatrix} \tag{14}$$

$$h = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{pmatrix}^T \tag{15}$$

Our matrix H has 8 degrees of freedom, and so, as each point gives 2 sets of equations, we will need 4 points to solve for h uniquely. So, given four points (such as the corners provided for this assignment), we can generate vectors a_x and a_y for each, and concatenate them together:

$$A = \begin{pmatrix} a_{x,1} \\ a_{y_1} \\ \vdots \\ a_{x,n} \\ a_{y,n} \end{pmatrix} \tag{16}$$

Our problem is now:

$$Ah = 0 \tag{17}$$

As A is a 8x9 matrix, there is a unique null space. Normally, we can use MATLAB's **null** function, however, due to noise in our measurements, there may not be an h such that Ah is exactly 0. Instead, we have, for some small $\vec{\epsilon}$:

$$Ah = \vec{\epsilon} \tag{18}$$

To resolve this issue, we can find the vector h that minimizes the norm of this $\vec{\epsilon}$. To do this, we must use the SVD, which we will cover in week 3. For this project, all you need to know is that you need to run the command:

```
[U, S, V] = svd(A);
```

The vector h will then be the last column of V, and you can then construct the 3x3 homography matrix by reshaping the 9x1 h vector.

2 Invictus vs Harleysville - 50 pts

2.1 Introduction

In this programming assignment, we will use the concepts of projective geometry and vanishing points to project a virtual line that would be parallel as seen from above. We have provided a video sequence, as well as the relevant annotations. The task is to compute the vanishing point then find the line segment in the image that represents the referee's location on the field. (e.g. Figure 2)

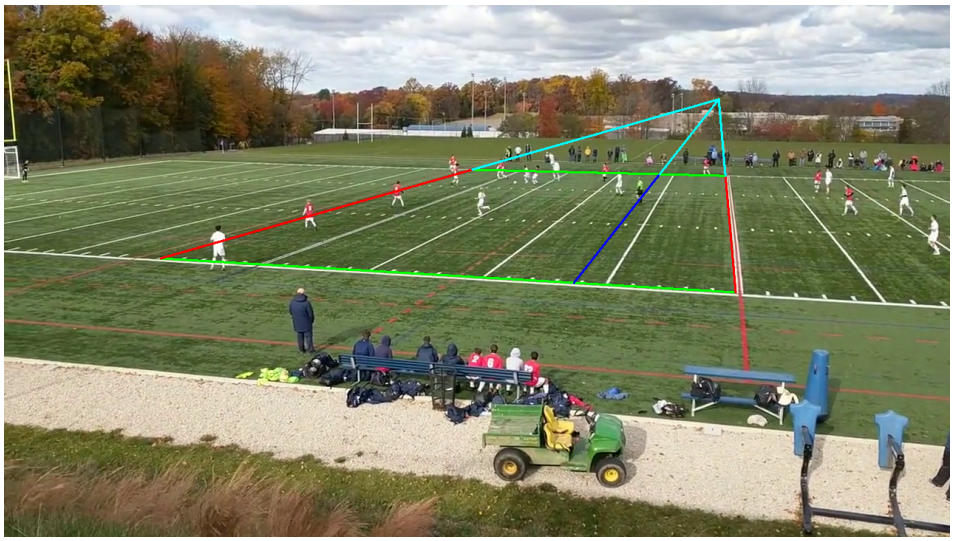

Figure 2: Example of the referee's line (blue) projected onto the field.

2.2 Technical Details

The Python script **visualize_ref_line.py** will be the main script for this question. We provide the following:

- A sequence of images onto which you will draw the referee's line
- All annotated points (referee and 4 field points)
- Drawing utilities for all points

2.3 Data given

Figure 3 shows the layout of the provided keypoints as seen from a top down perspective. The mappings from figure to code are as follows:

A lower_left_keypoint

B lower_right_keypoint

C upper_right_keypoint

D upper_left_keypoint

E referee_keypoint

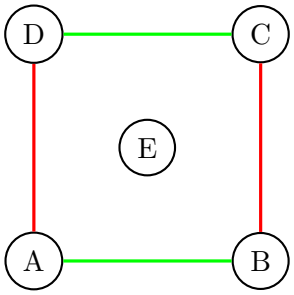

Figure 3: Supplied keypoints for each frame.

2.4 Files to complete and submit

1. **line_from_pts.py**
   This function is responsible for computing the line that passes through two points in $\mathbb{P}^2$

2. **line_intersection.py**
   This function is responsible for computing the intersection between two lines in $\mathbb{P}^2$

3. **compute_ref_line_segment.py**
   This function is responsible for computing the referee's position on the field, represented as two end points (on either side of the field).

2.5 Visualizing Results

To visualize the results as a video, run the **visualize_ref_line.py** script. First open a terminal window and change the current directory to the directory of "Part_2". Then run command:

python visualize_ref_line.py

You can also use an IDE to run the script.