CS 498: Assignment 1: Perspective Projection

Due by 5:00pm Friday, February 11 2022

January 28, 2022

Submission

In this assignment you will be modifying the files homography.py perspective.py. Please put together a single PDF with your answers and figures for each problem, and submit to Gradescope (Course Code: JBXJVZ). We recommend you to add your answers to the latex template files we provided. For code submission, make sure you use the provided ".py" files with your modification and the dumped ".npz" file. The graders will check both your PDF submission and your code submission if needed.

Homography [8pts]

In this question, we will examine properties of homography transformations and see how to estimate a planar homography from a set of correspondences.

Question 1 [1pt]: You are given a photo of the State Farm Center (`uiuc.png`), UIUC's indoor arena that hosts our basketball teams. We marked and plotted the four corners of the court, as shown in Fig. 1 (left). A standard basketball court is 28.65 meters long and 15.24 meters wide, as shown in Fig. 1 (right). Now let's consider a 2D coordinate system defined on the planar surface of the basketball court. The origin of this coordinate system is point #3; the long edge is x axis and short edge is y axis. Please write down the four corner's coordinate in this 2D coordinate frame (in meters) and fill in your answer in the numpy array "corners_court".

corners_court = np.array([(0, 15.24), (28.65, 15.24), (28.65, 0), (0, 0)])

Question 2 [3pts]: Given your Q1's answer, now we could establish four pairs of point correspondence between two planes: the 2D basketball court plane and the 2D image plane. Using what we have learned in Lecture 3, complete the function `findHomography` and use it to estimate the homography matrix from the court to the image. Please briefly explain what you do for homography estimation and report the resulting matrix in this document. Hints: 1) you might find the function `vstack, hstack` to be handy for getting homogenous coordinate; 2) you might find `numpy.linalg.svd` to be useful; 3) lecture 3 described the homography estimate process.

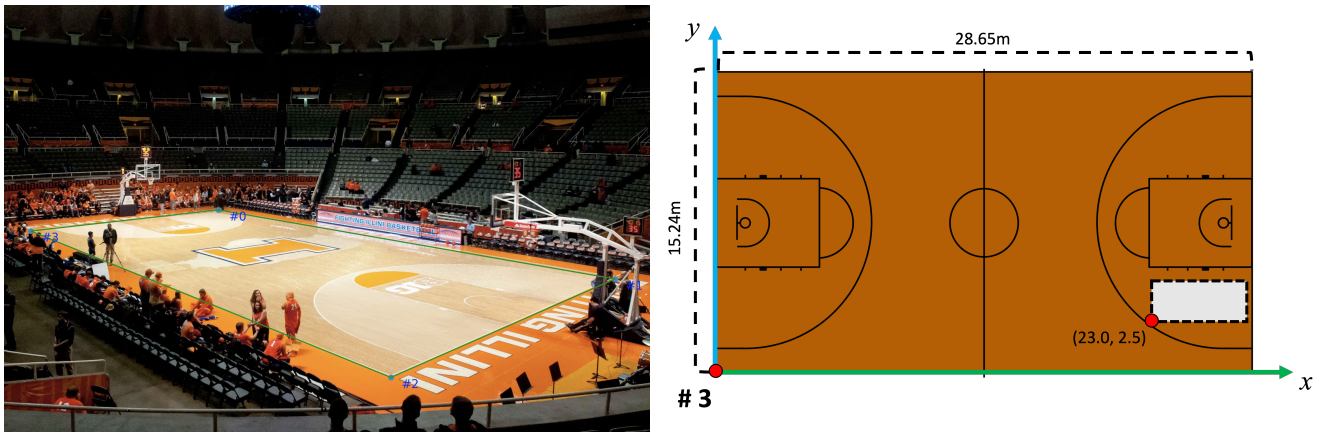


Figure 1: Left: target image with keypoints; Right: court dimensions and coordinate convention.

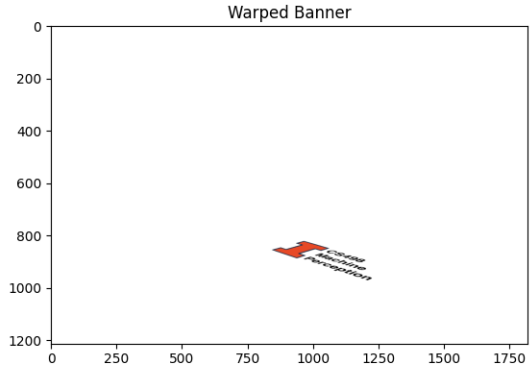
```python
def findHomography(pts_src, pts_dst):
    n = pts_src.shape[0]
    A = np.zeros((2 * n, 9))
    for i in range(n):
        Xi = pts_src[i, 0]
        Yi = pts_src[i, 1]
        xi = pts_dst[i, 0]
        yi = pts_dst[i, 1]
        A[2 * i + 0, :] = np.array([Xi, Yi, 1,  0,  0, 0, -1 * xi * Xi, -1 * xi * Yi, -1 * xi])
        A[2 * i + 1, :] = np.array([ 0,  0, 0, Xi, Yi, 1, -1 * yi * Xi, -1 * yi * Yi, -1 * yi])
    U, sigma, V = np.linalg.svd(A)
    H = V[-1] / V[-1, -1]
    return H.reshape((3, 3))
```

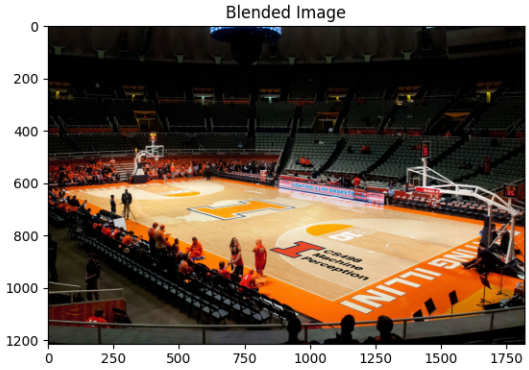
I used the matrix equation from the end of the Lecture 3 slides to construct the A matrix as shown inside the for loop in the code. I then did SVD on this matrix and took the last column of the V to find the best solution to the matrix equation. I reshaped this column vector into a 3x3 matrix to find the homography matrix.

Question 3 [4pts]: We want to promote our CS498 class across the university. One of the marketing idea we came up is to create an on-court ad in Illinois's State Farm Center. Now you are taking in charge of the virtual signage task – inserting the logo of our class (`logo.png`) electronically onto the basketball court (`court.png`). Specifically, the size of the logo needs to be 3x6 meters; we want to place the bottom left logo corner at $(23, 2)$ on the basketball court. In order to do so, we need two steps:

- **3.a [1pt]:** calculate the homography transform between the image coordinate of the two images `logo.png` and `court.png`. Hints: 1) could you compute the transform from `logo.png` to the basketball court 2) could you leverage the homography matrix you computed in Question 2?
- **3.b [2pt]:** complete the `warpImage` function and use the homography you computed in 3.a to warp the image `logo.png` from its image coordinate to the `court.png`'s image coordinate. Hints: 1) suppose $(x', y', 1)^T = \mathbf{H}(x, y, 1)^T$, we have $I_{\text{target}}(x', y') = I_{\text{source}}(x, y)$; 2) you might find `numpy.meshgrid` and `numpy.ravel_multi_index` to be useful.

- **3.c [1pt]:** alpha-blend the warped logo onto the court: $I = \alpha F + (1 - \alpha)B$





Perspective Projection [7pts]

Till now we have been working on transformations between 2D coordinate systems. Now let us to lift up to the 3D coordinate. Consider the same image of the state farm center, but this time, we annotate four additional points, corresponding to the four corners of the basketball backboard.

Question 4 [1pt]: The lower rim of the backboard is 2.745 meters above the ground; the backboard width is 1.83 meters and backboard height is 1.22 meters. The backboard protrudes 1.22 meters out from the baseline. Now let us consider the world frame. The world frame coordinate origin is at the point #3, where x-axis is along the long edge (sideline) and y-axis is along the short edge (baseline), and z-axis is upwards, perpendicular to the basketball court. Could you compute the 3D coordinate of all the eight keypoints?

```
side_to_board = (court_width - backboard_width) / 2
point4 = (board_to_baseline, side_to_board, lower_rim + backboard_height)
point5 = (board_to_baseline, side_to_board + backboard_width, lower_rim + backboard_height)
point6 = (board_to_baseline, side_to_board + backboard_width, lower_rim)
point7 = (board_to_baseline, side_to_board, lower_rim)
corners_3d = [(0, court_width, 0), (court_length, court_width, 0), (court_length, 0, 0), (0, 0, 0), point4, point5, point6, point7]
```

Question 5 [4pt]: Now we have established eight pairs of 2D and 3D keypoint correspondences. Consider this set of correspondence pairs in homogeneous coordinate $(\mathbf{x}_i, \mathbf{X}_i)$. Our goal is to compute a perspective projection matrix such that:

$$\mathbf{x}_i = \alpha \mathbf{P} \mathbf{X}_i = \alpha \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} \mathbf{X}_i = \alpha \begin{bmatrix} \mathbf{p}_1^T \mathbf{X}_i \\ \mathbf{p}_2^T \mathbf{X}_i \\ \mathbf{p}_3^T \mathbf{X}_i \end{bmatrix}$$

where $\mathbf{p}_i^T$ is ith row of $\mathbf{P}$; $\mathbf{x}_i = (x_i, y_i, 1)^T$; α is an arbitrary scalar.

- **5.a [1pt]:** Please prove that:

$$\mathbf{A}_i \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} = \mathbf{0} \text{ where } \mathbf{A}_i = \begin{bmatrix} \mathbf{0} & \mathbf{X}_i & -y_i \mathbf{X}_i \\ \mathbf{X}_i & \mathbf{0} & -x_i \mathbf{X}_i \end{bmatrix}$$

Hints: consider using cross product between two similar vectors.

We know that:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \alpha \begin{bmatrix} \mathbf{p}_1^T \mathbf{X}_i \\ \mathbf{p}_2^T \mathbf{X}_i \\ \mathbf{p}_3^T \mathbf{X}_i \end{bmatrix}$$

It follows that $1 = \alpha \mathbf{p}_3^T \mathbf{X}_i$, and hence:

$$\alpha = \frac{1}{\mathbf{p}_3^T \mathbf{X}_i}$$

So, if we rearrange the top two rows using this value, we find:

$$\mathbf{p}_2^T \mathbf{X}_i - y_i \mathbf{p}_3^T \mathbf{X}_i = 0 \Rightarrow 0\mathbf{p}_1^T + \mathbf{p}_2^T \mathbf{X}_i - y_i \mathbf{p}_3^T \mathbf{X}_i = 0$$
$$\mathbf{p}_1^T \mathbf{X}_i - x_i \mathbf{p}_3^T \mathbf{X}_i = 0 \Rightarrow \mathbf{p}_1^T \mathbf{X}_i + 0\mathbf{p}_2^T - x_i \mathbf{p}_3^T \mathbf{X}_i = 0$$

Converting this to matrix form, we find that:

$$\begin{bmatrix} \mathbf{0} & \mathbf{X}_i & -y_i \mathbf{X}_i \\ \mathbf{X}_i & \mathbf{0} & -x_i \mathbf{X}_i \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} = \mathbf{0}$$

Hence:

$$\mathbf{A}_i \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} = \mathbf{0} \text{ where } \mathbf{A}_i = \begin{bmatrix} \mathbf{0} & \mathbf{X}_i & -y_i \mathbf{X}_i \\ \mathbf{X}_i & \mathbf{0} & -x_i \mathbf{X}_i \end{bmatrix}$$

- **5.b [3pt]:** complete the `findProjection` function and use this function to recover the perspective camera projection matrix, from the 3D world coordinate to the 2D image coordinate. Hints: 1) establish a linear system based on your derivation 2) given the eight keypoint correspondences, how many equations and how many unknown variables? 3) consider using `numpy.linalg.svd` to solve the linear system. 4) you might find `numpy.concatenate` and `numpy.reshape` to be handy.

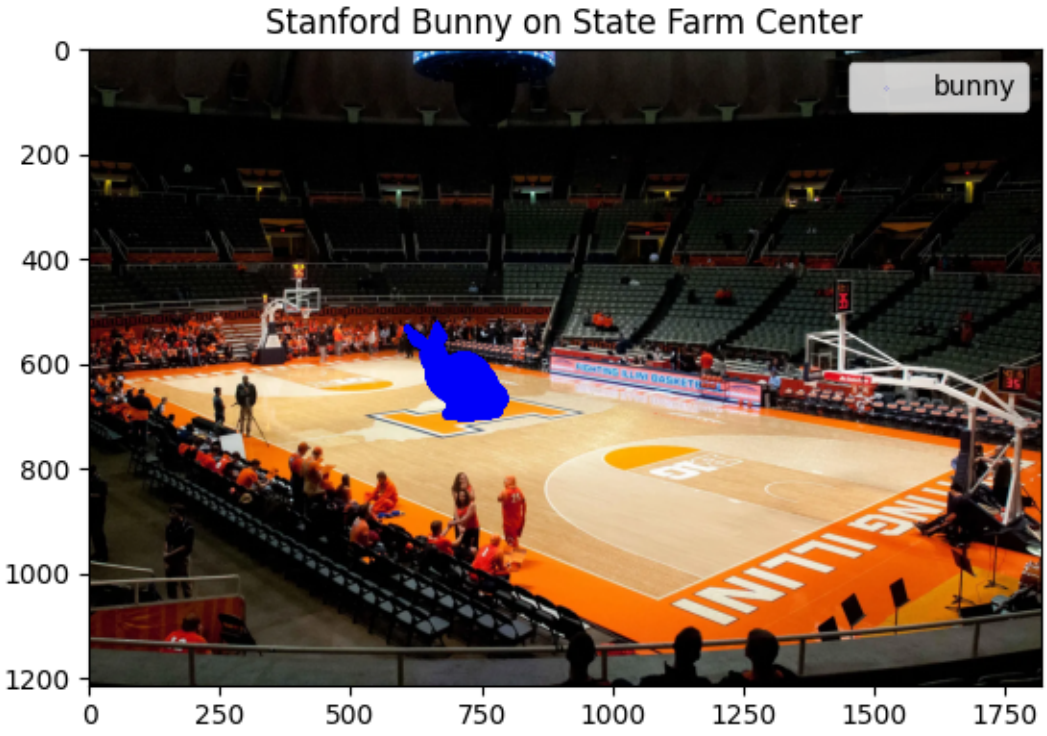
```
def findProjection(xyz, uv):
    n = xyz.shape[0]
    A = np.zeros((2 * n, 12))
    for i in range(n):
        Xi = xyz[i, 0]
        Yi = xyz[i, 1]
        Zi = xyz[i, 2]
        xi = uv[i, 0]
        yi = uv[i, 1]
        A[2 * i + 0, :] = np.array([Xi, Yi, Zi, 1, 0, 0, 0, 0, -1 * xi * Xi, -1 * xi * Yi, -1 * xi * Zi, -1 * xi])
        A[2 * i + 1, :] = np.array([0, 0, 0, 0, Xi, Yi, Zi, 1, -1 * yi * Xi, -1 * yi * Yi, -1 * yi * Zi, -1 * yi])
    U, sigma, V = np.linalg.svd(A)
    H = V[-1] / V[-1, -1]
    return H.reshape((3, 4))
```

- **5.c [1pt bonus]:** Try to answer the following questions: 1) assuming perfect correspondences, could we recover the projection matrix using even fewer points? Please indicate how many correspondences at least we need. 2) could we just use correspondences on the ground plane to recover this projection matrix? Please explain why or why not.

  We could recover a projection matrix using a minimum of 6 points. This is because we need to determine 11 independent values in order to come up with the entire projection matrix. Each correspondence point generates 2 rows the A matrix. Hence, if the A matrix has fewer than 11 rows, it will be impossible to determine a unique solution, i.e. the solution space will have multiple independent matrices. So, we need 6 points to get the A matrix to have enough information to uniquely determine the projection matrix.

  We cannot use correspondences just on the ground plane because this will not provide enough information about how the z direction is scaled. In other words, we need to have unique points in 3D space in order to have enough information to determine the projection matrix.

Question 6 [2pt]: Using our estimated projection matrix in Q5, we could insert any 3D object in the world coordinate frame onto the 2D image. In this question, let's consider the a simple case where we want to project a 3D point cloud onto the image plane and plot them. Suppose we want to put a giant stanford bunny statue in the center of the basketball court, could you compute the 2D coordinate of every vertex of this bunny using the projection matrix? Hints: 1) you might need to transform the bunny to the court center first 2) make sure the bunny is above the ground.



Question 7 [2pt bonus]: Try to use an off-the-shelf renderer to render the 3D mesh in a more realistic manner (e.g. pyrender, moderngl, blender, mitsuba2). You could consider the following perspectives that might influence the realism: 1) shading 2) shadows 3) occlusion reasoning 4) lighting environments.