Robotics:Perception Assignment 3
Image Projection

1 Introduction

In this programming assignment, we will use tracking and pose estimation to implement a simple augmented reality application. There will be two steps to this process. First, we will use a KLT tracker to get the position of corners across different frames. Then we will use homography estimation to compute the 3D pose of a set of 4 points in the world and, instead of simply overlaying a logo like in the previous assignment, render a 3D object in the frame. For simplicity we will be rendering a cube, but in principle, any object could be used. Below are a few example images of the results in 1. The goal will be to from a set of 4 projected points on your image with known coordinates (the AprilTag or Soccer goal corners), find the necessary position and orientation to draw the cube over the points.

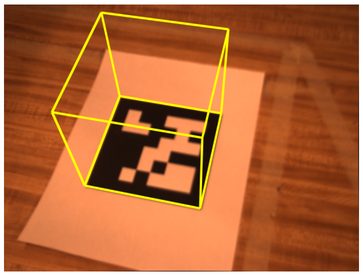

Figure 1: Example of 3D rendering

2 Technical Details

The functions you must implement are **track_corners** and **ar_cube.m**. First you will use MATLAB's point tracker to implement KLT in **track_corners**, to find where the corners are for each frame. The output of **track_corners** will be an array of the points you have tracked accross the images.

Then, using pixel coordinates of the corners you have tracked, the world coordinates of the points you have to render, and the calibration matrix. The desired output is the position and orientation (represented as a vector and a rotation matrix respectively). Then using that pose, you will generate the projected points (in pixel coordinates) of a virtual object. You will be given a set of generic points to project, but for the visualization will use a cube that will lie on top of the points.

You can visualize the results of these functions with the **play_video** script. (More details below)

Robotics:Perception Assignment 3
Image Projection

2.1 KLT Tracking

In **track_corners** you will be given skeleton code, where you have to fill in where the comments indicate. You are given the images where you will track the points, and location of the points in the first image where you should start the tracking. You do not need to implement KLT by yourself, you should use the MATLAB Computer Vision libraries. Search online for the relevant functions. You will store all the points in a 3D array. To get draw the virutal objects in the scene, these corners will be passed to **ar_cube**.

2.2 Homography Estimation

Now that you know where the points are in each image, we want to know the virtual object should be drawn. First to draw the pose, you need to estimate the pose of the camera. To estimate the pose from the corners, we will need to once again estimate the homography as in Assignment 2. You can just copy in your **est_homography** from Assignment 2 into the folder, and the test and visualization code will call that to estimate the homography.

2.3 Pose Estimation

So the computed homography will be passed into **ar_cube**, where we will extract the rotation and translation from it. As described in the lectures, we assume the homography is of the form:

$$Hp_w \sim \; p_c = Kp_{im}$$

$$H = \; \lambda \begin{pmatrix} | & | & | \\ r_1 & r_2 & t \\ | & | & | \end{pmatrix}$$

We give you K, p_{im}, and p_w and thus you can solve for H, then extract the rotation R and translation t from this homography, enfocing that the z component of t to be positive (more in the appendix). Please note it is highly encouraged for you create your own test cases for this - this can be done by generating a random rotation and translation and scale in MATLAB then passing that through **ar_cube**. You can search online to learn how to generate a random rotation.

2.4 Projecting Points

Once R and T are computed, we can use this to place virtual objects into the world. This is simply the projection of points done in Assignment 1. Given a set of points X

we simply project them onto the image:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = X_c = K(RX + t)$$

$$X_{im} = X_c / z_c$$

3 Visualizing Results

To check the results of only your tracker, run the **generate_klt_images** script. To visualize the full projected AR as a video, run **generate_ar_images** script. Then call the MATLAB command:
play_video(generated_imgs)

To save the projected images to file, call:
save_images(generated_imgs, im_name)
Where **im_name** is the prefix for the image files.

These will be run on a sequence with images of an april tag. You can also generate your own video with a set of points and edit **generate_klt_images** or **generate_ar_images** if you would like to play with your own data.

4 Submitting

To submit your results, run the **submit** script, which will test your **est_homography.m** and **ar_cube.m** functions by passing it some sample points. The Barcal Real goal image sequence will be used for testing the your corner tracking. The submit script will generate a mat file called RoboticsPerceptionWeek3Submission.mat. Upload this file onto the assignment page, and you should receive your score immediately.

5 Appendix A: Computing Pose from a Homography

As we saw from the lectures, points on a plane take the form of a homography. Assuming all the world points lie on $Z = 0$:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = [R \quad t] \begin{pmatrix} x_w \\ y_w \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} | & | & | & | \\ r_1 & r_2 & r_3 & t \\ | & | & | & | \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ 0 \\ 1 \end{pmatrix}$$

$$\implies z_c \begin{pmatrix} x_c/z_c \\ y_c/z_c \\ 1 \end{pmatrix} = \begin{pmatrix} | & | & | \\ r_1 & r_2 & t \\ | & | & | \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix}$$

$$\implies \begin{pmatrix} x_{im} \\ y_{im} \\ 1 \end{pmatrix} = \underbrace{\frac{1}{z_c} \begin{pmatrix} | & | & | \\ r_1 & r_2 & t \\ | & | & | \end{pmatrix}}_{H} \begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix} = \begin{pmatrix} | & | & | \\ h_1 & h_2 & h_3 \\ | & | & | \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix}$$

Now that we have extracted H from the previous part using the AprilTag corners. We need to extract R and t from it. If there was no noise then we would have $h_1 = \lambda r_1$, $h_2 = \lambda r_2$, and $h_3 = \lambda t$, for some arbitrary scale factor λ. Since r_1 has norm 1, we could just say $r_1 = h_1/\|h_1\|$, $r_2 = h_2/\|h_2\|$, $r_3 = r_1 \times r_2$, and $t = h_3/\|h_1\|$. However, because of noise, h_1 and h_2 may not be orthogonal and thus what we computed would not be a rotation matrix. To fix this we use the SVD once again, giving our final estimate of R and t:

$$R' = \begin{pmatrix} | & | & | \\ h_1 & h_2 & (h_1 \times h_2) \\ | & | & | \end{pmatrix} = USV^T$$

$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(UV^T) \end{pmatrix} V^T$$

Thus we get our final equations to extract R and t from our estimated H:

$$R' = \begin{pmatrix} | & | & | \\ h_1 & h_2 & (h_1 \times h_2) \\ | & | & | \end{pmatrix} = USV^T$$

$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(UV^T) \end{pmatrix} V^T$$

$$t = h_3/\|h_1\|$$

IMPORTANT NOTE: The above deterimnes the basic algorithm, however, since a homography is only determined up to a scale, there is a sign ambiguity, as if you negate H you get the approximate rotation:

$$R' = \begin{pmatrix} | & | & | \\ -h_1 & -h_2 & (-h_1 \times -h_2) \\ | & | & | \end{pmatrix} = \begin{pmatrix} | & | & | \\ -h_1 & -h_2 & (h_1 \times h_2) \\ | & | & | \end{pmatrix}$$

This gives a flip of the x and y axes and thus we have two valid rotation matricies, corresponding to the translations t and $-t$ respectively. To disambiguate this, you must in your code enforce that the z component of t is positive (i.e. $t_3 = H_{3,3} > 0$) and multiply the H matrix by the appropriate sign to do this.

6 Appendix B: Review of Calculating Homographies

For clarity, we repeat from assigment 2 the estimation of a homography from points. As we learned in Lecture INSERT, 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:

$$\mathbf{x}' \sim H\mathbf{x} \tag{1}$$

$$\lambda \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2}$$

$$\lambda x' = h_{11}x + h_{12}y + h_{13} \tag{3}$$

$$\lambda y' = h_{21}x + h_{22}y + h_{23} \tag{4}$$

$$\lambda = h_{31}x + h_{32}y + h_{33} \tag{5}$$

In order to recover x' and y', we can divide equations (3) and (4) by (5):

$$x' = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}}{h_{31}x_1 + h_{32}x_2 + h_{33}} \tag{6}$$

$$y' = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}}{h_{31}x_1 + h_{32}x_2 + h_{33}} \tag{7}$$

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

$$-h_{11}x - h_{12}y - h_{13} + h_{31}xx' + h_{32}yx' + h_{33}x' = 0 \tag{8}$$

$$-h_{21}x - h_{22}y - h_{23} + h_{31}xy' + h_{32}yy' + h_{33}y' = 0 \tag{9}$$

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

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} h = 0 \tag{10}$$

Where:

$$a_x = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \end{pmatrix} \quad (11)$$
$$a_y = \begin{pmatrix} 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{pmatrix} \quad (12)$$
$$h = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{pmatrix}^T \quad (13)$$

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:

$$A = \begin{pmatrix} a_{x,1} \\ a_{y,1} \\ \vdots \\ a_{x,n} \\ a_{y,n} \end{pmatrix} \quad (14)$$

Our problem is now:

$$Ah = 0 \quad (15)$$

As A is a 9x8 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 $\vec{\epsilon}$:

$$Ah = \vec{\epsilon} \quad (16)$$

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.