ENPM 673: Perception for Autonomous Robots **Released: Feb-11.**

Report

Name: Arpit Aggarwal and Shantam Bajpai

Problem 1 - Detection

The goal here was to detect the corners of the AR Tag and return the ID and orientation of the tag with respect to the original orientation of the reference AR Tag shown as follows:

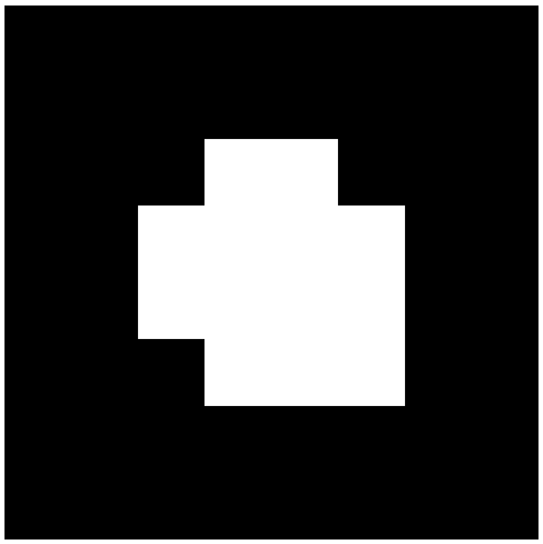

Figure 1: Reference AR Tag

We divide problem 1 into three steps which are explained as follows:

Detecting AR Tag corners

The AR tag corners were detected using "findContours" function of OpenCV. The "findContours" function returned the list of contours and the hierarchy relating the parent and child contours. Further, "approxPolyDP" function of OpenCV was used for estimating the polygon of the contour.

Using conditions on hierarchy, number of sides of the estimated polygon of the contour and the contour area, we were able to detect the AR Tag as shown below:

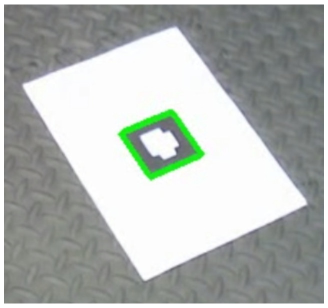

Figure 2: Detecting AR Tag

Next step was finding the corners of the detected AR Tag. The ”approxPolyDP” function as mentioned earlier gave a set of coordinates required to estimate a polygon for the contour. However, the function doesn’t give the coordinates in order. We used the article[1] to give the coordinates in ordered form: [(top-left), (top-right), (bottom-right), (bottom-left)].

Finding the Homography Matrix

Next step was finding the homography matrix between the detected corner of the AR Tag and the world coordinates to get the warped form of the AR Tag which was used to find the ID and orientation of the AR Tag. The world coordinates were defined as [(0, 0), (200, 0), (200, 200), (0, 200)] where the first coordinate is the column index and second is the row index.

After finding the four source points (that is the AR Tag corner coordinates) and the four destination points (that is the world coordinates), we constructed the A matrix shown as follows:

$$A = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & u_1 * x_1 & u_1 * y_1 & u_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & v_1 * x_1 & v_1 * y_1 & v_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & u_2 * x_2 & u_2 * y_2 & u_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & v_2 * x_2 & v_2 * y_2 & v_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & u_3 * x_3 & u_3 * y_3 & u_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & v_3 * x_3 & v_3 * y_3 & v_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & u_4 * x_4 & u_4 * y_4 & u_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & v_4 * x_4 & v_4 * y_4 & v_4 \end{bmatrix}$$

where, (x, y) represent the source coordinates and (u, v) represent the destination coordinates.

Using Singular Value Decomposition (SVD) we can represent matrix A as $A = U\Sigma V^T$ where $U \in R^{8\times8}, \Sigma \in R^{8\times9}$ and $V \in R^{9\times9}$. The columns of the U vector are determined by finding out the eigenvalues of the AA^T matrix. We can determine the eigenvalues by solving the following equation:

$$\det \left| AA^T - \lambda I \right| = 0$$

Similarly, the columns of V vector are determined by finding out the eigenvalues of the A^TA matrix. We can determine the eigenvalues by solving the following equation:

$$\det \left| A^TA - \lambda I \right| = 0$$

Let $AA^T = M$ then the eigenvector for M can be computed by the following equation:

$$Mv = \lambda v$$

This can be further simplified to give us the eigen vector equation $(M - \lambda I)v = 0$. Here the eigen vector $v \in R^{8\times 8}$. Similarly we can perform the above steps to compute the eigenvectors for the matrix $A^T A$. As we saw above that the matrix A can be decomposed into 3 different singular matrices namely U, Σ and V. Now, to have only one unique solution the rank of matrix A should be $(n-1)$ where n is the number of columns of the matrix A. In this case the rank of the matrix is 8 hence the solution to the equation is $x = V_n$ (last column of V matrix corresponding to eigenvalue of 0). That is, the homography matrix is the last column of matrix V. The matrix was reshaped to form a 3x3 matrix.

After finding the homography matrix between the corner and the world coordinates, next step was to find the warped image of the AR Tag[2]. We wrote a function "warpPerspective" which took the source image as input, transposed it and then found the corresponding points between the source and destination images. The final image was transposed and the warped image was obtained. An example of warped AR Tag is shown below:

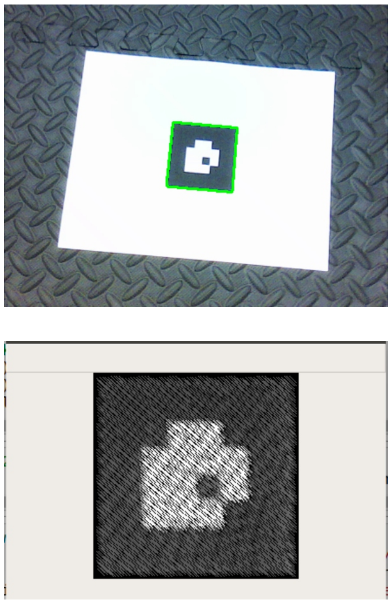

Figure 3: Warped AR Tag

Finding the ID and orientation of AR Tag

After obtaining the 200 x 200 warped AR Tag, the image was cropped to 100 x 100 size. The cropped image was used to find whether the tag was in bottom-right, top-left, bottom-left or top-right configuration[3]. Also, the ID was returned from the most significant bit to the least significant bit as mentioned in the problem statement.

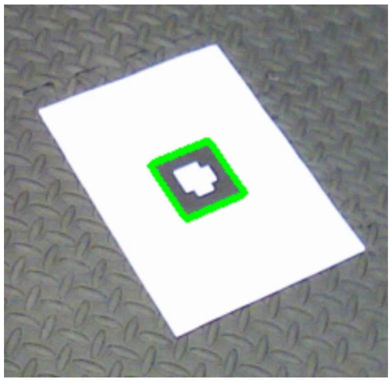

Figure 4: Orientation: Top-Left, ID: 15

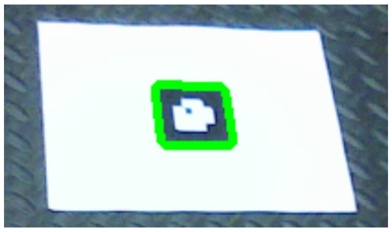

Figure 5: Orientation: Bottom-Left, ID: 7

Video Outputs for Problem 1

The following are the video output links for Problem 1:

Output for Tag0.mp4: https://drive.google.com/file/d/11YO94wywqBKN62c0E65OJwXRgwI1eZ_Y/view?usp=sharing
Output for Tag1.mp4: https://drive.google.com/file/d/1b7LpS0HH4l9tv6YXx47REg9rYkVGy1bL/view?usp=sharing
Output for Tag2.mp4: https://drive.google.com/file/d/1kpQB3M2mu3ojhSEC6SKd59kX_adKKkox/view?usp=sharing
Output for multipleTags.mp4: https://drive.google.com/file/d/12WFTkq0rqspGVlMzmaMjdsGEkiSSLePc/view?usp=sharing

Problem 2a - Superimposing an image onto the tag

The goal here was superimposing the lena image onto the AR Tag. First the lena image was resized to 200 x 200 size to match the world coordinates. Using the homography concept in Problem 1, the lena image was first superimposed on the AR Tag(taking into account the orientation of the

AR Tag) as shown below:

Figure 6: Lena Image

Figure 7: Superimposing Lena Image on the AR Tag

The following were the world coordinates, given the orientation of the AR Tag:

Bottom-Right (BR) configuration: [(0, 0), (199, 0), (199, 199), (0, 199)]
Bottom-Left (BL) configuration: [(0, 199), (0, 0), (199, 0), (199, 199)]
Top-Left (TL) configuration: [(199, 199), (0, 199), (0, 0), (199, 0)]

Top-Right (TR) configuration: [(199, 0), (199, 199), (0, 199), (0, 0)]

Using the above calculated world points and the corner points obtained in Problem 1, we calculated the homography matrix which was used for projecting the lena image onto the AR Tag.

Video Outputs for Problem 2a

The following are the video output links for Problem 2a:

Output for Tag0.mp4: https://drive.google.com/file/d/1nrU5DAQNK8B4u_3m6UVGBilAHh0UDNV_/view?usp=sharing
Output for Tag1.mp4: https://drive.google.com/file/d/1IEcXiP9KNthmUs_ZSsamFw9oGX2uLRFx/view?usp=sharing
Output for Tag2.mp4: https://drive.google.com/file/d/1Hlv3Ev0pegsXVgcIbRf4Y4sJNHQb0bPW/view?usp=sharing
Output for multipleTags.mp4: https://drive.google.com/file/d/12WFTkq0rqspGVlMzmaMjdsGEkiSSLePc/view?usp=sharing

Problem 2b - Placing a virtual cube onto a tag

The goal here was superimposing a virtual cube onto the AR Tag. For this, we first built the Projection matrix[4] from the Homography matrix and Camera matrix using the supplementary material given.

The next step was defining the 3D points which would be projected to 2D points using the projection matrix. The 3D points that were taken were: [[0, 0, 0], [0, 200, 0], [200, 200, 0], [200, 0, 0], [0, 0, -200], [0, 200, -200], [200, 200, -200], [200, 0, -200]]. We wrote the equivalent of "cv2.projectPoints()" function which was used to find the 2D point for a given 3D point, given the projection matrix. After calculating the projection matrix and the 2D representation of 3D points, the virtual cube was projected on the AR Tag.

Video Outputs for Problem 2b

The following are the video output links for Problem 2b:

Output for Tag0.mp4: https://drive.google.com/file/d/1dRk6aS0kljCm1NHKR-4Z2OXBVUpUPnLC/view?usp=sharing
Output for Tag1.mp4: https://drive.google.com/file/d/1djvfrd3XrP_0NHXGPMiAY1qfvOTVf5fa/view?usp=sharing
Output for Tag2.mp4: https://drive.google.com/file/d/1hoEnWk1q0rfmp36p4817aUwy8EFp0adm/view?usp=sharing
Output for multipleTags.mp4: https://drive.google.com/file/d/15QNMr6t-hxyGhuVeqLguC20cKE7MRswt/view?usp=sharing

Interesting problems encountered

The following were some interesting problems we encountered:

1. "cv2.findContours()" gave better results as compared to "cv2.cornerHarris()". For this problem, we found that using contour detection was the best way forward.
2. For obtaining a better warped image, we add a noise factor of 0.4 when obtaining the homogenous coordinates. Also, adding a window of (5 x 5) size helped in giving better results.

References

[1] https://www.pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/
[2] https://www.pyimagesearch.com/2014/05/05/building-pokedex-python-opencv-perspective-warping-step-5-6/
[3] https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/
[4] https://www.learnopencv.com/tag/projection-matrix/