ENPM 673 : Perception for Autonomous Robots - Project 1

Akanksha Patel - 116951029

Aruna Baijal - 116130110

Sri Manika Makam - 116697415

26 February 2020

Abstract

In this project we focus on detecting and tracking a custom AR Tag in a video sequence. The detection part of the project focused on finding the location and identity of the AR Tag. The tracking part of the project involved superimposing an image and placing a 3D cube over the tag. For improving the estimated homograph, Kalman Filter was applied to keep a track of four corners of the AR Tag but our implementation of Kalman Filter did not produce the desired results.

1 Part 1 - Detection

In this part we need to locate the AR tag in the video, and return the location and identity of the tag. To identify the AR tag we make use of cv2 function findContours. We first convert the video frames to binary images to reduce noise and false contours. From the detected contours we need to identify the AR tag inside the white A4 sheet. To identify this, we filter out the detected contours and save the contours that are nested inside a parent contour and have a child contour of their own. We have this information from the hierarchy matrix that tells us the previous contour, next contour, first child contour and parent contour for each contour. For our AR tag contours, the parent contour is the white sheet, and the child contours is the inside white part of the AR tag. In addition to this filter, we make sure that the area of contour is greater than 150 pixels to battle noise.

To find the corners of our tag, we first ensure that the contour detected is a rectangle by using cv2 function arcLenth. To find the corners of this rectangle we use cv2 function approxPolyDP that returns the most probable corners of a polygon.

Now we need to find the identity of the AR tag. Using the corner coordinates of the AR tag as source coordinates, and an 8x8 square at origin as the destination coordinates, we calculate the homography using the equation Ax = 0, where:

$$A = \begin{bmatrix} -x1 & -y1 & -1 & 0 & 0 & 0 & x1*xp1 & y1*xp1 & xp1 \\ 0 & 0 & 0 & -x1 & -y1 & -1 & x1*yp1 & y1*yp1 & yp1 \\ -x2 & -y2 & -1 & 0 & 0 & 0 & x2*xp2 & y2*xp2 & xp2 \\ 0 & 0 & 0 & -x2 & -y2 & -1 & x2*yp2 & y2*yp2 & yp2 \\ -x3 & -y3 & -1 & 0 & 0 & 0 & x3*xp3 & y3*xp3 & xp3 \\ 0 & 0 & 0 & -x3 & -y3 & -1 & x3*yp3 & y3*yp3 & yp3 \\ -x4 & -y4 & -1 & 0 & 0 & 0 & x4*xp4 & y4*xp4 & xp4 \\ 0 & 0 & 0 & -x4 & -y4 & -1 & x4*yp4 & y4*yp4 & yp4 \end{bmatrix}$$

$$X = \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix}$$

We solve the equation Ax = 0 using Singular Value Decomposition (SVD) of A. SVD factors the matrix into a diagonal matrix D and two orthogonal matrices U and V, such that

$$A = UDV^T$$

The least-squares solution (x) to the equation is given by the column of V, which corresponds to the smallest eigen value of $A^T A$. From x, we get the elements of homography matrix is given by

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$$

We apply this homography on our detected AR tag to get a transformed tag as a 8x8 square. We read this transformed square to calculate the identity of the AR tag and to understand the orientation of the tag.

2 Part 2 - Tracking

2.1 Superimposing an image onto the tag

To place the Lena image on top of the tag, we recalculate Homography matrix by using Lena corners as source coordinates and the detected AR tag corners as destination coordinates. We apply this Homography on the Lena image to obtain transformed coordinates. To superimpose Lena on our original image, we overwrite the pixel values of our original image at the transformed pixels with the corresponding Lena pixel values.

The image frames from the given data, Tag0.mp4 and multipleTags.mp4, superimposed with Lena on the AR tag are shown in the figures 1 and **??** respectively. The AR tag ID is also been printed.

2.2 Placing a virtual cube on the tag

In this section we implement an augmented reality application of projecting a 3D shape onto a 2D image. For this, we need to determine how an arbitrary point from a planar marker, $x^w = [x^{(w)}; y^{(w)}; 0; 1]^T$ projects

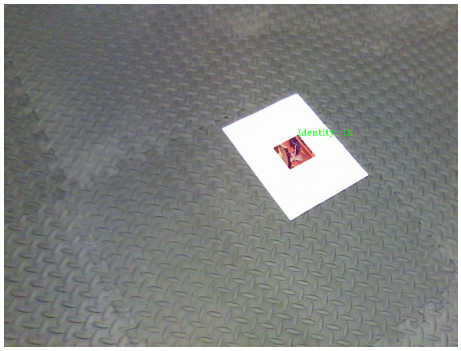

Figure 1: Superimposition of Lena on single AR Tag in a frame

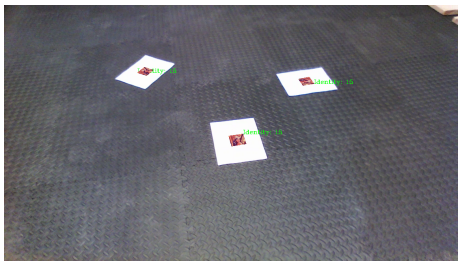

Figure 2: Superimposition of Lena on multiple AR Tags in a frame

to the point in the camera's image plane, $x^{(c)} = [x^{(c)}; y^{(c)}; 1]^T$. For this we use the equation, $x_{(c)} = Px^{(w)}$, where P is the projection matrix given by $P = K[R|t]$.

The matrix K is called intrinsic camera matrix which represents the transformation of a 3-D point from the camera's coordinate system to the 2-D point in the camera's image plane. In this project, we were given the intrinsic parameters. Matrix [R — t] consists of rotation matrix R and translation vector t. These are the camera's extrinsic parameters, which describe the transformation of a 3-D point from the world coordinate system to the 3-D point in the camera's coordinate system.

In part 1, as we already detected the corners of AR Tag, we now use the calculated projection matrix P to get the corners of the cube in the camera coordinate frame. Once we have the value of corners, we draw the cube by drawing the edges between the corresponding corners ($cv2.line$).

The image frames from the given data, Tag0.mp4 and multipleTags.mp4, superimposed with cube on the AR tag are shown in the figures 3 and 4 respectively.

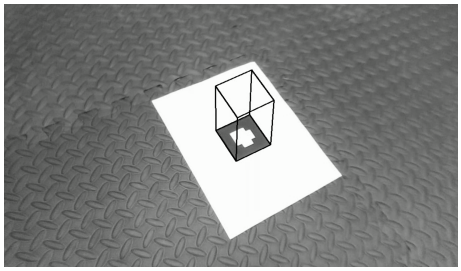

Figure 3: Superimposition of cube on single AR Tag in a frame

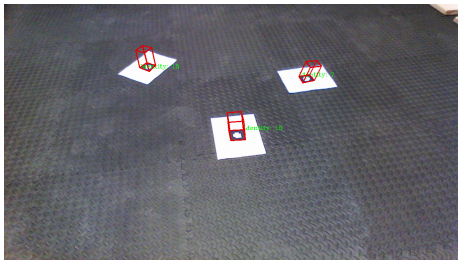

Figure 4: Superimposition of cube on multiple AR Tags in a frame

2.3 Tracking using the Kalman Filter

We observed the cv2.findContours isn't always able to detect the AR Tag, especially in case of sudden camera motion. So, the location of ARTag is sometime lost in the video. We tried to overcome this problem by applying Kalman filter to estimate the location of the tag. However, the estimated location had large errors associated with it. We attribute these errors to the poor tuning of the filter parameters like the values of process and measurement noise covariance matrices. Due to time constraints, we could not tune the parameters properly.

The implementation details of the Kalman Filter are given below. We assumed a constant velocity model and defined the state as follows:

$$X = \begin{bmatrix} x_1 \\ y_1 \\ v_{x1} \\ v_{y1} \\ x_2 \\ y_2 \\ v_{x2} \\ v_{y2} \\ x_3 \\ y_3 \\ v_{x3} \\ v_{y3} \\ x_4 \\ y_4 \\ v_{x4} \\ v_{y4} \end{bmatrix} \qquad Z = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix}$$

where x_i and y_i denote the position of four corners and v_{xi} and v_{yi} denote the velocity with which the position of four corners is changing.

The state transition matrix is given by:

$$State\_Transition\_Matrix = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The measurement matrix is given by:

$$Measurement\_Matrix = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Process and measurement noise covariance matrices is given by:

$$R = 0.1 * I(8) \qquad Q = 10^{-5} * I(16)$$

where $I(n)$ represents an identity matrix of n elements.

3 Problems Encountered

We encountered a couple of problems while working on this project:

1. When generating counters and filtering with child and parent, we get extra contours when the AR tag identity is not 15 as it detects the inner contours as well. To overcome this we added an area threshold as well to remove the inner contours. 2. We initially used cv2.approxPolyDP function to directly get the corners of the detected AR Tag. But we noticed that the function sometimes returns more than 4 corners even if the shape is a quadrilateral. We overcame this problem by taking the extreme four corners of the detected polygon instead of the corners returned by the function. 3. The projection of cube on the image is not stable. This is because of the errors in homography estimation (which is almost indistinguishable in case of an image projection.) To reduce these errors, we have used Kalman Filter. 4. The output of Kalman Filter is supposed to be better than the output without any filtering techniques. But in our experiments, the estimates of Kalman Filter contains large errors, which we attribute to the poor tuning of the filter parameters.

Note: The video outputs can be found in this link: https://drive.google.com/drive/folders/1heKJ2yg9p0px7vm4Vs-o50CZtKI$_k Ld9?usp = sharing$