ENPM 673 - Perception of Autonomous Robots
Project 1

Pradeep Gopal
Sahana Anbazhagan
Srikumar Muralidharan

February 8, 2020

Introduction

This project will focus on detecting a custom AR Tag, that is used for obtaining a point of reference in the real world, such as in augmented reality applications. There are two aspects to using an AR Tag that is implemented in this project, detection and tracking. With the intrinsic camera parameters given, based on the orientation and position of the tag, the image processing was done.

Working and Execution of the project

Part 1

We first take the input from the user as to whether they want to choose to detect a single tag or multiple tags. We obtain the files from the location they are being stored in. Following the input from the user, the detection, processing and tracking is performed respectively.

We obtain the contours by writing a function called contour_create(). Based on the values obtained we remove the outer contours leaving us with only the tag shape and draw a contour around the required tag. Now using this we can obtain the code that we are interested in. We put constraints on the hierarchy of contours, hence eliminating the unnecessary contours, and then we detect the corners to obtain the bottom right, bottom left, top right and top left of the AR tag.

We have the coordinates of the AR tag and we also take the coordinates of a reference image. Using this information we compute the A matrix from where we obtain the H matrix.We have then written a function called homography() to obtain the H matrix. The H matrix is given as follows:

$$\begin{bmatrix}\mathbf{R} & T\end{bmatrix} \tag{1}$$

We get the U, S and V parameters and using the V matrix the Homograph matrix is obtained. We then reshape it to the required dimensions and return it. We find the H matrix by using:

$$\begin{bmatrix}x'\\y'\\1\end{bmatrix} = \begin{bmatrix}h_{00} & h_{01} & h_{02}\\h_{10} & h_{11} & h_{12}\\h_{20} & h_{21} & h_{22}\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{2}$$

Using the below equation with the calculated A matrix we get the H matrix:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

We wrote a function called warp(). In this warping function we first compute the inverse of the H matrix obtained. Then, the image is converted to black and white and we then check if the resolution of the obtained image is within the resized resolution of the image. After this we obtain the warped image and this is then decoded.

The decode function basically crops and resizes the image to obtain just the blocks that have our ID. Then we check if they are black or white by assigning black to 0 and white to 255. This function then returns the ID obtained.

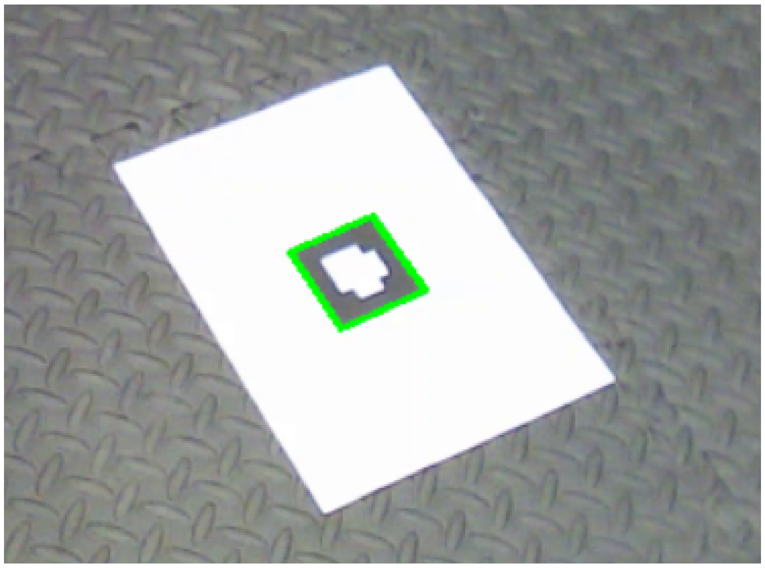


Figure 1: AR tag detection

Tracking:
We then performed tracking.Tacking is nothing but locating an object in the successive frames of a video. We

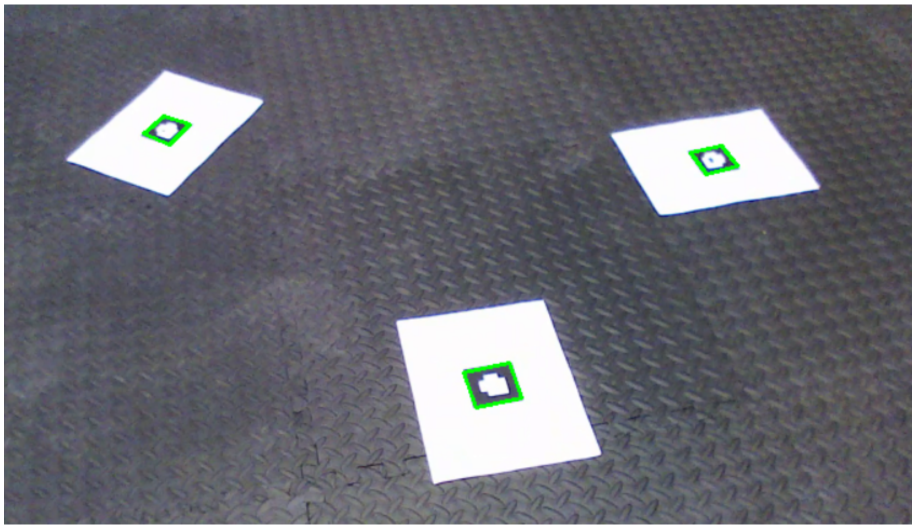


Figure 2: Multiple AR tags detection

have used Kernelized Correlation Filters (KCF) for the same. This is a fast and accurate tracker. Tracking is done by taking the tag along with its contour. We also take a few pixels around the tag and extract the features for them all. We take the extra space around the tag because there is a possibility that some features are lost when only a part of the tag is seen. We take the coordinates of this new rectangular area we have designed. Then the coordinates vary dynamically as the tag moves because the rectangular area also moves along with it.

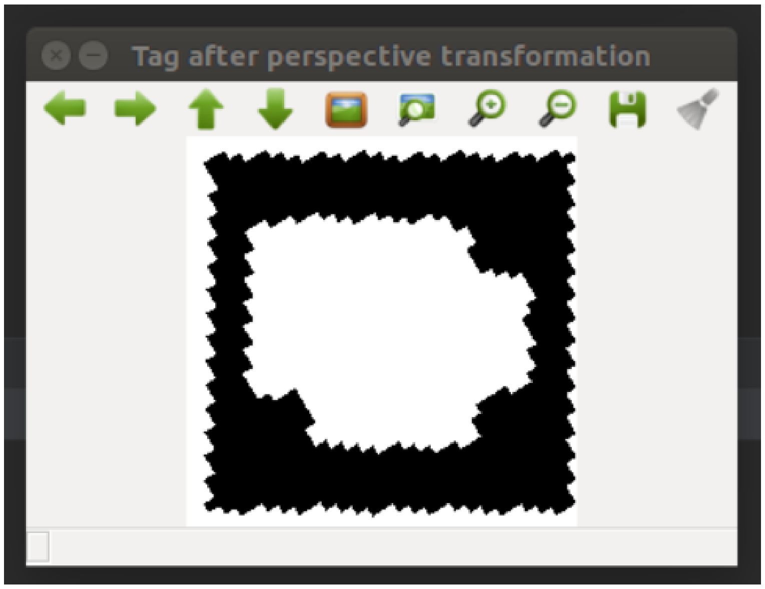


Figure 3: AR tag after Perspective Transformation

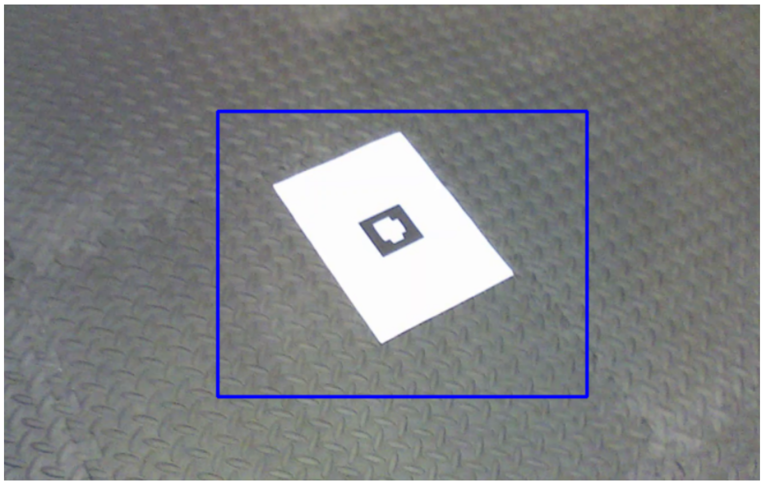


Figure 4: AR tag detection and tracking

Part 2

Imposing the Lena image on the detected code

We had to project Lena's image with the exact same orientation as that of the tag. We first obtain the Lena image that is to be super imposed on the tags. Then we calculate the coordinates of Lena's image. After this we obtain the homograph matrix and perform reverse warp here because we want the image to be the same orientation as the tag. We will now obtain the required result where Lena's image will be in the same orientation of the tag at all times.

Tracking was extended here as well. The tag was replaced with the Lena image superimposed. The same steps were followed here as well.

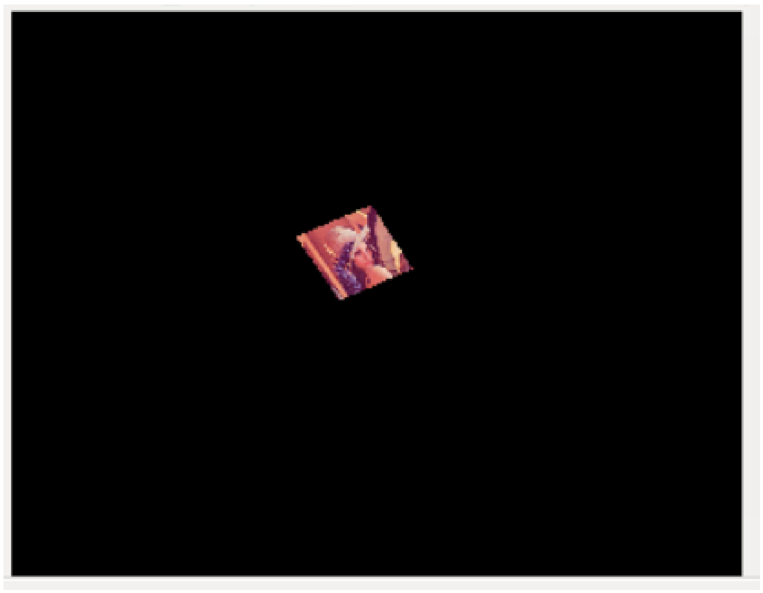


Figure 5: Lena image superimposed on the AR tag

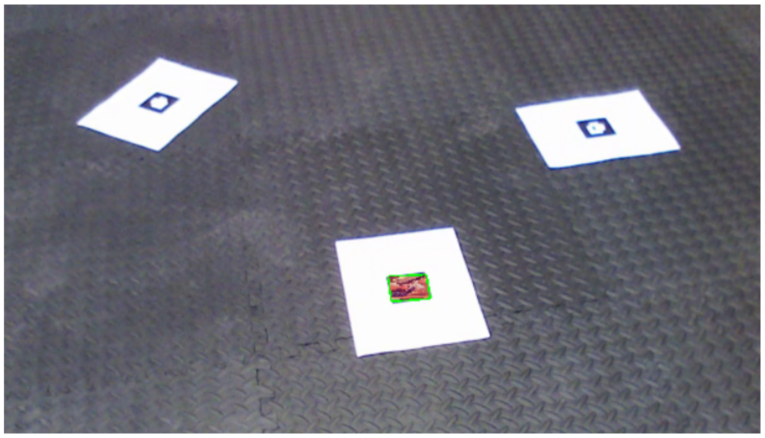


Figure 6: Multiple Lena image superimposition

Part 3

Placing a virtual cube on tag

For this part, we were asked to super impose a virtual cube on the tags identified. We are given with 3 videos which contain tags varying in number from 1 to 3. We offer a choice to user to choose which video to play and accordingly, we display the video of choice. As done in the previous case, we split the video into a series of images, frame by frame and read the image onto a variable called frame. We resize the frame into half its original dimensions and give the same dimensions for a black mask that we apply at a later stage to view the cube. This mask is initialised with zeros for the same shape as the frame. As mentioned clearly in the previous parts, we use the same method to get contour points that form the boundary for our tags. For each of these 4 contour points, we calculate seperate H matrices using our own Homography function. Now, we bring in the concept of Extrinsic parameters for a camera. Suppose our camera is observing a point in world coordinates, the coordinate system of camera 'C' is rotated and translated with respect to the world coordinate system as per the view that we achieve through the camera. In augmented reality, the relation between a point in the world frame and its equivalent point in the camera frame is determined by the projection matrix P, which is defined as follows:

The projection matrix can be written as the product of 2 matrices, the Intrinsic property of camera matrix K, and Extrinsic camera property matrix. K matrix is constant and is formulated during calibration. Extrinsic parameters keep varying as the camera angle changes.

To estimate the pose of camera, we relate the point in Camera coordinates to world coordinates using the Projection matrix.

$$X^C = PX^w \tag{3}$$

$$X^C = K[r1, r2, t]X^W \tag{4}$$

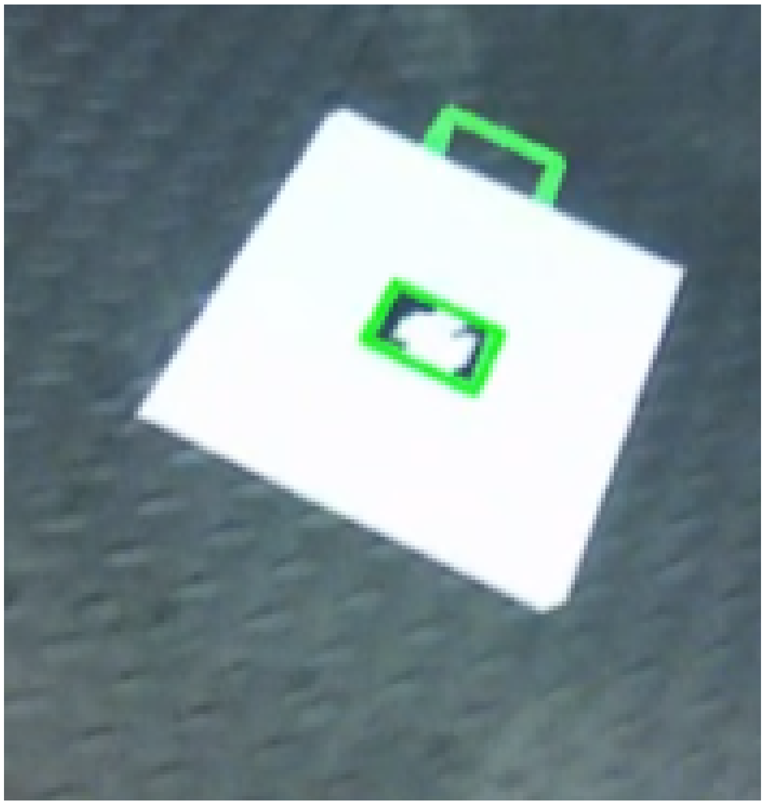
Here, we relate the extrinsic parameters to the homography matrix and deduce their matrices from it. Each column of the homography matrix is further form the corresponding columns of a new matrix called B matrix, which is further used to calculate the value of Lambda. They are related by the equation:

$$\lambda H = K * B \tag{5}$$

We get lambda value using the following equation:

$$\lambda = \left(\frac{||\boldsymbol{K}^{-1}\boldsymbol{h}_1|| + ||\boldsymbol{K}^{-1}\boldsymbol{h}_2||}{2} \right)^{-1}$$

Figure 7: Single cube drawn



using this value of lambda, we calculate the parameters of matrix B and further use them to calculate the column vectors for the Rotational and translational matrix that form the Projection matrix. These are the relations for Projection matrix.

$$R1 = \lambda * B1 \tag{6}$$

$$R2 = \lambda * B2 \tag{7}$$

$$R1 = Crossproduct(R1, R2) \tag{8}$$

$$T = \lambda * B3 \tag{9}$$

Where b1, b2 and b3 are the corresponding columns of B matrix. Finally, we create the rotational matrix with R1, R2 and R3 as their columns and also use T to create the projection matrix. Once we create the projection matrix, we use it calculate the cube points as mentioned in the Cube function and create the shape of a cube.

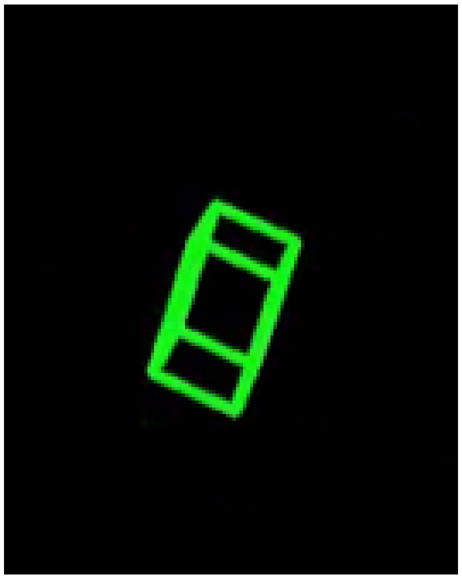


Figure 8: Masking for a single cube

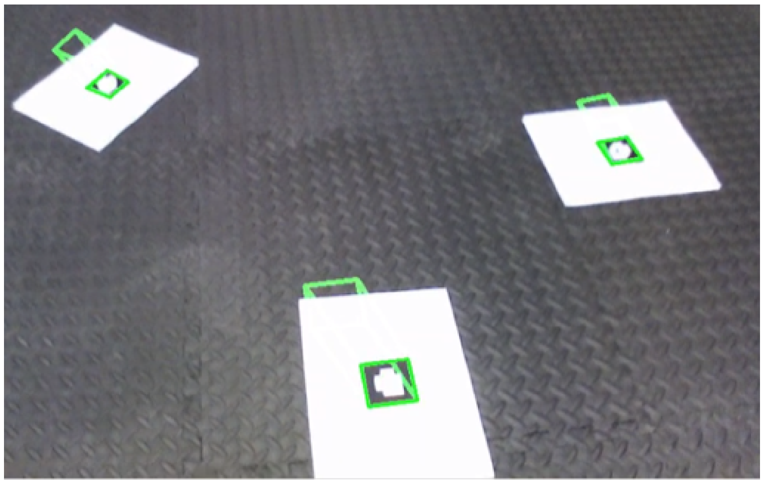


Figure 9: Multiple cubes drawn

Challenges faced and overcome

Unnecessary corners detected:
To avoid this issue, we used the cv2.contourArea function to measure the area of the closed polygon formed by contours. By setting a threshold on this area, we were able to eliminate all the unnecessary corners.

Noises while Detecting:
To eliminate the noises and to make the surrounding pixels smooth, we used a Gaussian and MedianBlur Filter. This reduced the noise by a significant amount.

Detection Failed sometimes:
Detection does not work all the time. So we used a Kernelized Correlation Filter as a Tracker to track the AR tag.

Problems with Extrinsic Matrix:
While calculating the Lambda value, we were unable to achieve the required result. The cube formed on top of the AR tag is highly fluctuating. We had trouble understanding the supplementary material provided. The formation of the cube on a Black screen seems to be accurate, but while super imposing we had issues. For better visualization of the results, refer the masking image.

Output link

`https://drive.google.com/drive/folders/18G7z9kykiiZ5c4xf_BHOSH91Pqu588nE?usp=sharing`