Project 1

AR Tag Homography and Projection

ENPM 673 – Perception for Autonomous Robots

Rene Jacques, Zachary Zimits

Image Processing Pipe Line

We started by loading a video file and reading it into our pipeline frame by frame. To switch video files being processed the file name can be changed at the end of the code. We began processing each frame by converting it to gray scale and initializing a simple blob detector to find the white pages. The blob detector returns an array of key points and radii that encompasses the blobs. In some cases, the blob detector will find the tag instead of the paper. To fix this problem we increased the radius by 60% to fully encompass the tag for the next step.

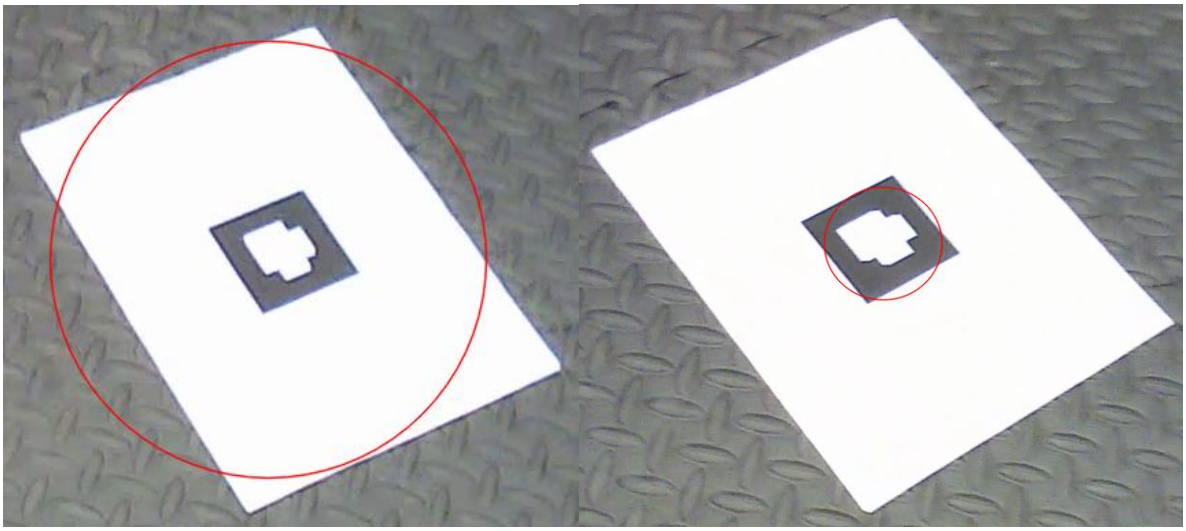

Figure 1: Blob Detection Full Page (Left), Blob Detection Tag (Right)

A for loop is used to go through the array of key points and process each blob on at a time. A bounding box is created from the point and radius and everything outside of the box is set to by black. Next *findContours()* and *approxPolyDP()* are used to find corner points for the different contours in the image. Working from the deepest nested contour out we look for the contour that contains the contour of tag. The contours found a visualized below in gray.

Figure 2: Post Threshold all Contours (Left), Selected Tag Contour (Right)

The points of this contour are passed into our Homography function to find the H and P matrices.

Homography

We next had to calculate the homography matrix in order to transform points from the world frame into the camera frame using the following equation:

$$x_k^c = \lambda H_c^w x_k^w$$

where H is a 3x3 matrix:

$$H_w^c = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

In order to calculate H we can use the following equation

$$Ah = 0_{8\times 1}$$

where h is H flattened into a 9x1 vector. To solve this, we need to solve 8 equations, one for each h in the homography matrix. This system of equations can be represented as the 8x8 matrix A from the equation above. We then define 4 world coordinate points and find 4 corresponding camera coordinates (the corners of the AR tag) which we plug in to find h:

$$\begin{pmatrix} x_1^w & y_1^w & 1 & 0 & 0 & 0 & -x_1^c x_1^w & -x_1^c y_1^w & -x_1^c \\ 0 & 0 & 0 & x_1^w & y_1^w & 1 & -y_1^c x_1^w & -y_1^c y_1^w & -y_1^c \\ x_2^w & y_2^w & 1 & 0 & 0 & 0 & -x_2^c x_2^w & -x_2^c y_2^w & -x_2^c \\ 0 & 0 & 0 & x_2^w & y_2^w & 1 & -y_2^c x_2^w & -y_2^c y_2^w & -y_2^c \\ x_3^w & y_3^w & 1 & 0 & 0 & 0 & -x_2^c x_2^w & -x_2^c y_2^w & -x_2^c \\ 0 & 0 & 0 & x_3^w & y_3^w & 1 & -y_3^c x_3^w & -y_3^c y_3^w & -y_3^c \\ x_4^w & y_4^w & 1 & 0 & 0 & 0 & -x_4^c x_4^w & -x_4^c y_4^w & -x_4^c \\ 0 & 0 & 0 & x_4^w & y_4^w & 1 & -y_4^c x_4^w & -y_4^c y_4^w & -y_4^c \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

To solve for h we can take the SVD of A to get the matrix of eigenvectors V:

$$A = UDV^T = np.linalg.svd(A)$$

$$V = \begin{bmatrix} v_{11} & \cdots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \cdots & v_{99} \end{bmatrix}^T$$

As the last column of V is the eigenvector that corresponds to the smallest eigenvalue of A, we save this eigenvector

$$V = V[8]$$

giving us:

$$V = (v_{91} \quad v_{92} \quad v_{93} \quad v_{94} \quad v_{95} \quad v_{96} \quad v_{98} \quad v_{99})$$

We can then normalize V by dividing it by its last value v_{99} to get h:

$$h = \frac{V}{V[8]} = \frac{V}{v_{99}}$$

All that is left then is to reshape h into the 3x3 matrix H:

$$H = np.reshape(h, (3,3))$$

which we can then use to transform pixels between the world and camera frames and back again.

Transform the Tag and Place Lena

After the homography matrix is returned we set all the pixels with a value of 200 in our gray image to be 201. Next we use *drawContours()* to set every pixel within the boundary of the tag to be 200. Using *np.where ()* we are able to get a list of all the pixel coordinates within the tag. We concatenate an array of ones to the end to make it a homogenous coordinate system and multiply it by the inverse of H to convert them to the world coordinate system.

$$Homogeneous\ Coordinates = C = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \\ x_1 & x_2 & \cdots & x_n \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

$$Projection\ Transform = T_p = H^{-1}C = np.matmul(np.linalg.ing(H), C)$$

$$Make\ C\ Homogeneous = C_h = \frac{T_p}{T_p[2,:]}$$

Using the new world coordinates we find their corresponding color values by taking them from the original image and save this to a new image tag.

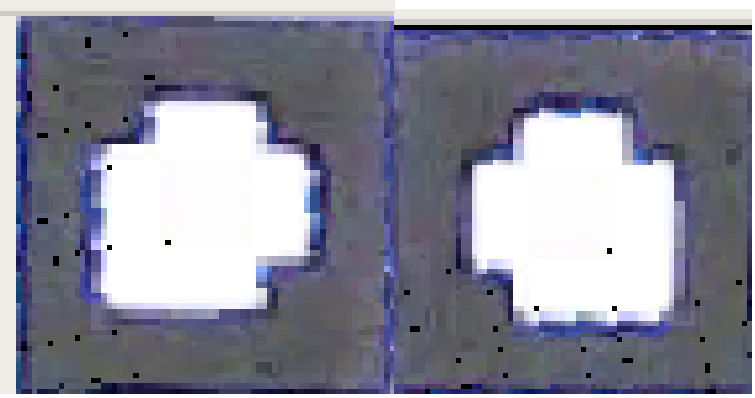

Figure 3:Pre-rotation and Post-rotation of the AR Tag

We then broke the image up into quadrants based on the reference image below

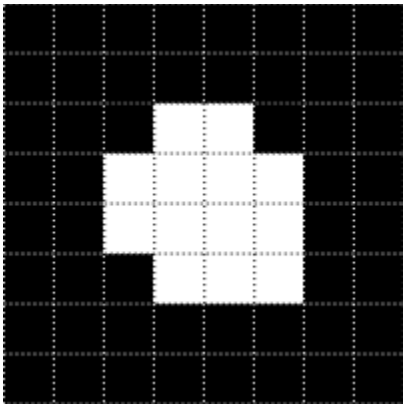

Figure 4: Reference Image with Grid

Looking at 12 quadrants that surround the data portion of the tag we found the nine quadrants with the highest average value and set them to be 255. With a known value for white we check the four corners to see which one is white and rotate the necessary amount to put it in the bottom right corner. We also record the smallest mean that is still white.

We then calculate the mean of the four quadrants in the center of the tag and compare them to the minimum value calculated earlier. If it is smaller, we mark that bit as a zero and add it to our data variable. Once all four quadrants have been calculated the value is returned and drawn next to the tag on the output image.

Next, we resized *Lena.png* to the dimensions of the tag and set the tag equal to it. The tag is then rotated back to its original orientation. We then reversed the process mentioned earlier to find the values of the tag.

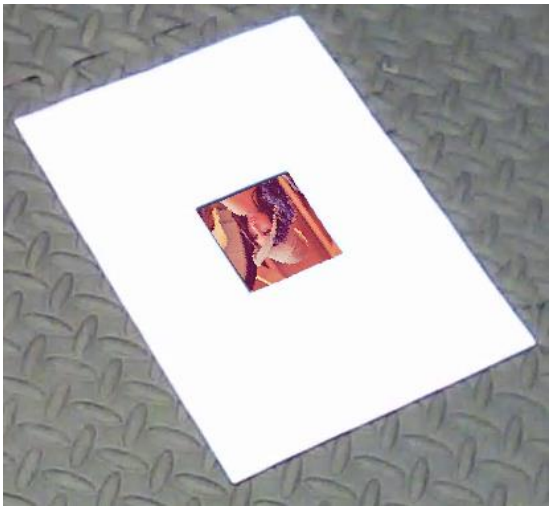

Figure 5: Page With Lena Replacing AR Tag

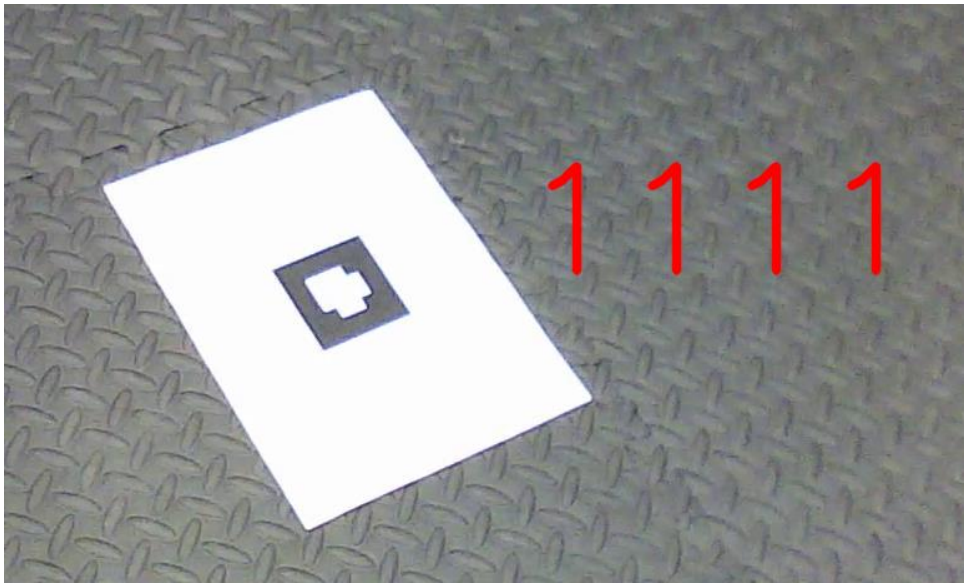

Figure 6: Tag Data Output

Projection Matrix

In order to be able to transform 3D objects into the camera frame from the world frame we need to calculate the projection matrix P:

$$P = K(R|t)$$

which is equal to the camera calibration matrix times the transformation matrix between the world and camera frames (the transformation matrix is a 3x4 matrix consisting of the rotation and translation matrices). To calculate P we first define $B = [r_1, r_2, t]$ and define the following relationships:

$$x^v = Px^w$$

$$x^c = K[r_1, r_2, r_3, t][x^w, y^w, 0, 1]^T$$

$$x^c = K[r_1, r_2, t][x^w, y^w, 1]^T$$

$$x^c = H[x^w, y^w, t]^T$$

where $H = K[r_1, r_2, t]$. Since $B = [r_1, r_2, t]$ we can solve using H and K:

$$B = K^{-1}H = np.dot(np.linalg.inv(K), H)$$

$$b1, b2, b3 = columns\ of\ B$$

The first column of B is proportional to the first column of the rotation matrix (r_1), the second column is proportional to the second column of the rotation matrix (r_2), and the third column is proportional to the translation vector (t). Since we are calculating a rotation matrix, and the last column of a rotation matrix (r_3) is perpendicular to the first two, we can r_3 by finding the cross product of $r1$ and r_2. The columns of the rotation matrix should be unit vectors but since we are estimating them, they may not actually have a unit length. Therefore, we must estimate the scale of the columns of the rotation matrix which we will call λ. We can estimate λ by finding the lengths of b_1 and b_2, averaging them, and then inverting the average so that we will be normalizing B when we multiply it by λ:

$$\lambda = \left(\frac{||b1|| + ||b2||}{2}\right)^{-1} = \left(\frac{np.linalg.norm(b1) + np.linalg.norm(b2)}{2}\right)^{-1}$$

With λ we can now calculate our final B by first defining $\tilde{B}$ as:

$$\tilde{B} = \lambda K^{-1}H$$

We then determine the determinant of $\tilde{B}$, because the determinant of B equals the determinant of $\tilde{B}$, and it the determinant of B is negative then the object we are transforming is located behind the camera and if the determinant if positive then the object is in front of the camera. Therefore, if the determinant of $\tilde{B}$ is negative we multiply $\tilde{B}$ by -1 to correct the object transformation and multiply $\tilde{B}$ by λ to get B:

$$B = \lambda\tilde{B}(-1)^{|\tilde{B}|<0}$$

We now have the corrected r_1, r_2, and t vectors:

$$r1, r2, t = columns\ of\ B$$

and can use r_1 and r_2 to calculate r_3:

$$r3 = r1 \times r2 = np.cross(r1, r2)$$

Putting all the rotation vectors and the translation vector into one matrix T:

$$T = [r1 \quad r2 \quad r3 \quad t]$$

we can finally solve for our projection matrix P by multiplying K (the camera calibration matrix) by T^T (T is transposed to make it a 4x3):

$$P = KT^T = np.dot(K, T.T)$$

Cube Placement

We pick the eight points of the cubed based off the world coordinates for the tag and set the in the following format.

$$p_1^w = \begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

We multiply this point by the projection matrix P and normalize it to find the corresponding camera coordinate.

$$p_1^c = Pp_1^w = np.dot(P, p_1^w)$$

$$p_1^c = \frac{p_1^c}{p_1^c[2]}$$

After all eight points are found we used the *line()* function to connect the points

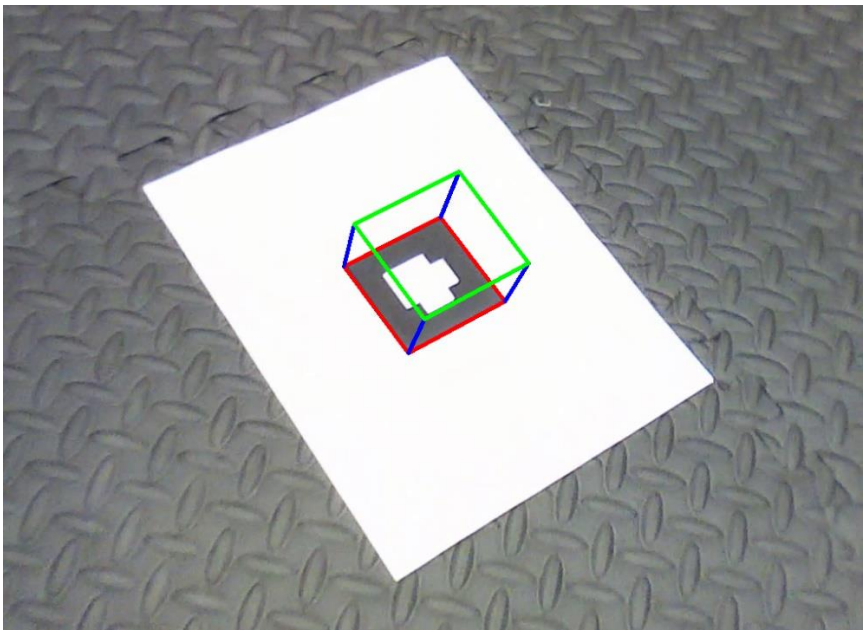

Figure 7: Cube Project View 1

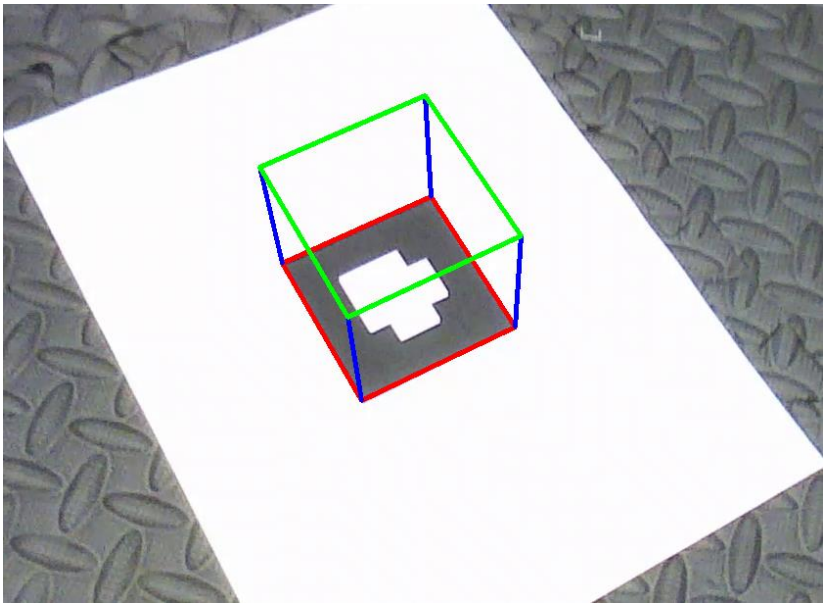

Figure 8: Cube Projection View 2

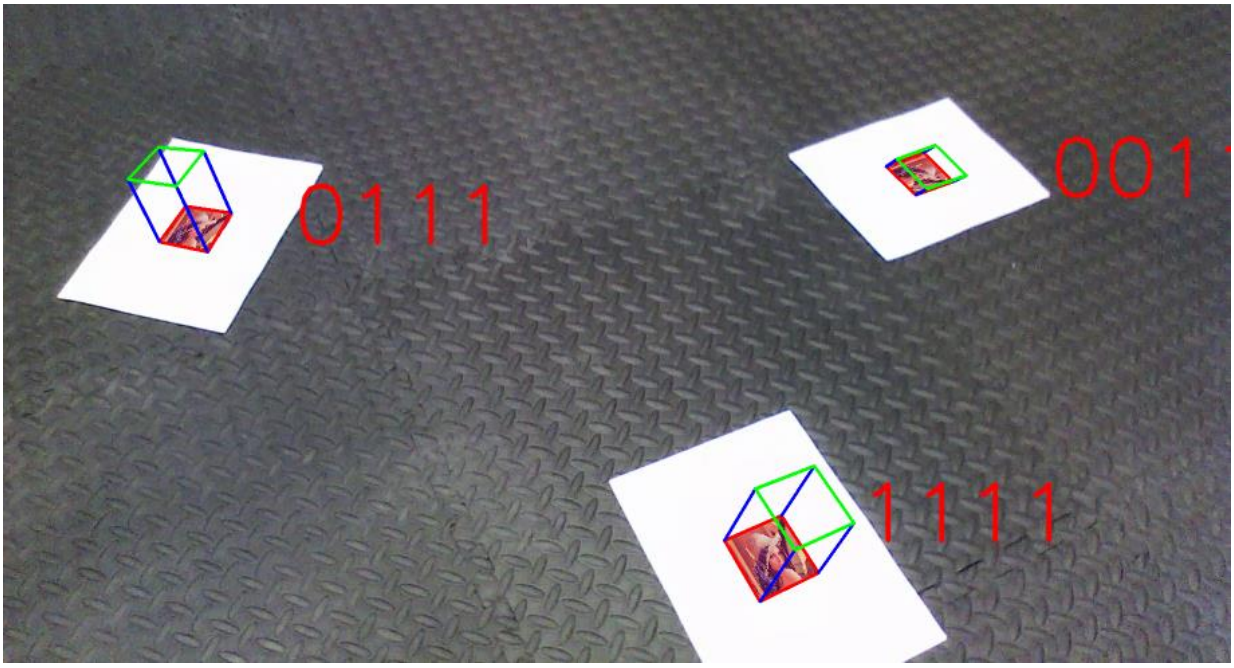

Figure 9: Multiple Tags with Full Implementation