1 Problem1- Detection

We need to encode the given AR tag which is represented in 8x8 grid by detecting the innermost 2x2 grid to get the tag id which should be affected by the camera rotation.

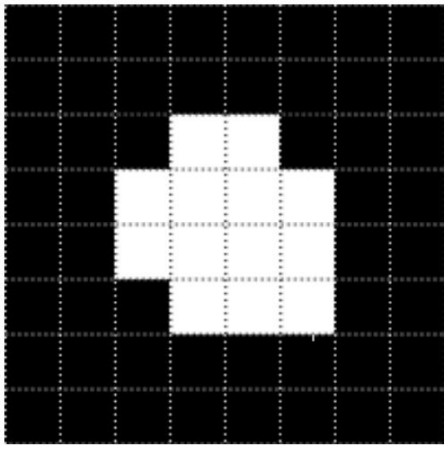

Figure 1: Grid view of AR tag

1.1 Procedure followed to solve Problem 1:

1. First we need the compute homography matrix of the corners of the AR tag to warp the image. Homography is done is using the

$$homography(worldcoordinates, pixelcoordinates) \tag{1}$$

function in the python file which uses the inbuild $svd()$ function to get the required vector. The n normalizing the obtained vector and reshaping it to 3x3 matrix to get the homography matrix. We have verified the coded homography matrix with in-build functions and got the same results.

```
Homography matrix by writing function
[[ 1.54896895e-01 -2.82544940e-02  4.59000000e+02]
 [ 2.64865793e-02  1.30296857e-01  2.82000000e+02]
 [ 3.06132448e-05 -2.37252644e-05  1.00000000e+00]]
----------------------------------------


Homography matrix from inbuilt function
[[ 1.54896895e-01 -2.82544940e-02  4.59000000e+02]
 [ 2.64865792e-02  1.30296857e-01  2.82000000e+02]
 [ 3.06132445e-05 -2.37252645e-05  1.00000000e+00]]
```

Figure 2: Homography matrix verification

2. After finding the homography matrix, we warped the AR tag image by multiplying every pixel with the homography matrix. We did not get perfect results in this part as there were white spots(holes) in the picture.

3. After warping, we encoded the tag for our id by performing operations on inner 2x2 grid. Given below are the comparisons of the output images got using running opencv function and our warping function. We get white spots(holes) as opposed to a uniform

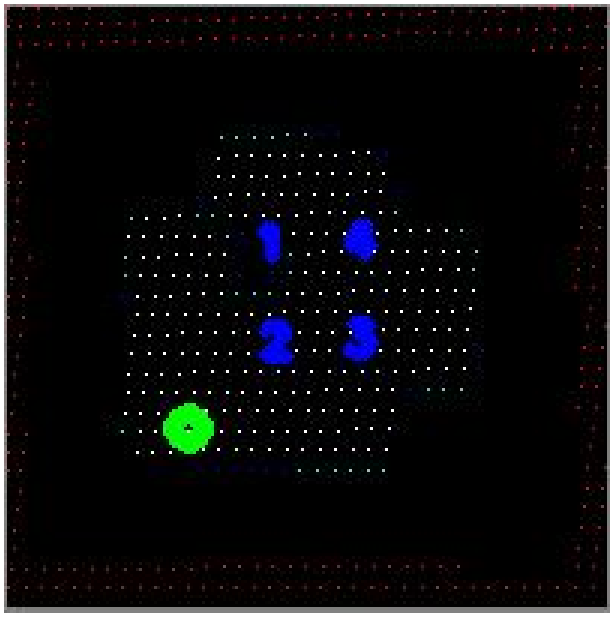

Figure 3: Output from developed warping function

color distribution when using the warping function developed. To improve this, we look at the neighborhood of neighboring pixels to determine the orientation of the AR tag. However, this is still not robust and as precise as the inbuilt function utilized in opencv, hence the orientation of AR tag shifts sometimes.

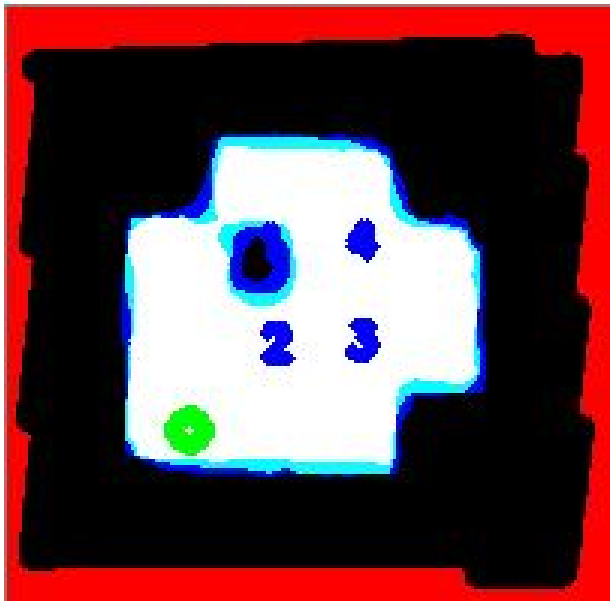

Figure 4: Output from inbuilt opencv function

1 represents the least significant bit
4 represents the most significant bit
Tag id is calculated in clockwise direction starting from 1 to 4

4. The snippets of the video is given below. The snippet shows the tag id in blue color.

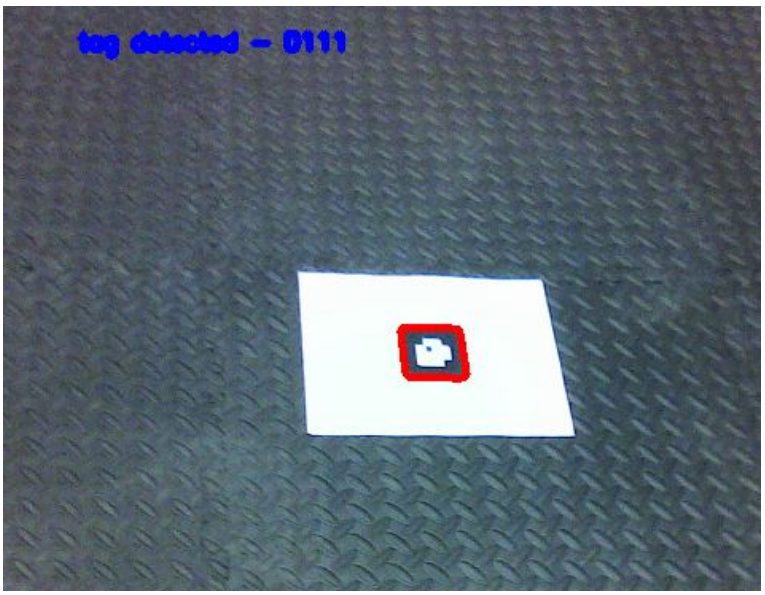

Figure 5: Tag id detection for video Tag1

1.2 Difficulties Faced:

1. The main difficult part was building the custom warping function. Even though we could warp the image, there were many holes present which made the AR tag id decoding difficult.

1.3 Contributions:

1. The warping function was written by Akwasi.A.Obeng
2. The sub-functions were written by Akwasi.A.Obeng, Eashwar Sathyamurthy
3. The report was prepared by Achal. P. Vyas

2 Problem2 - Tracking

2.1 Problem 2a - Superimposing an image onto the tag

2.2 Procedure for solving problem

1. First from problem 1, we obtained four corners of the AR tag. We used those AR tag corner coordinates and performed homography between corners of the Lena image and AR tag corners.
2. $homography(worldcoordinates, pixelcoodinates)$ in detection.py python file is used to compute the homography. The function takes 2 list as arguments and returns a 3x3 homography matrix.The Lena image corners were obtained $shape()$ function in opencv which gives the $[height, width]$ of the image.
3. After performing homography, we developed a $warping()$ function which warps all the pixels of Lena image onto the ARtag by multiplying each pixel with the homography matrix. First, the image is placed in correct orientation of the tag and then warped.

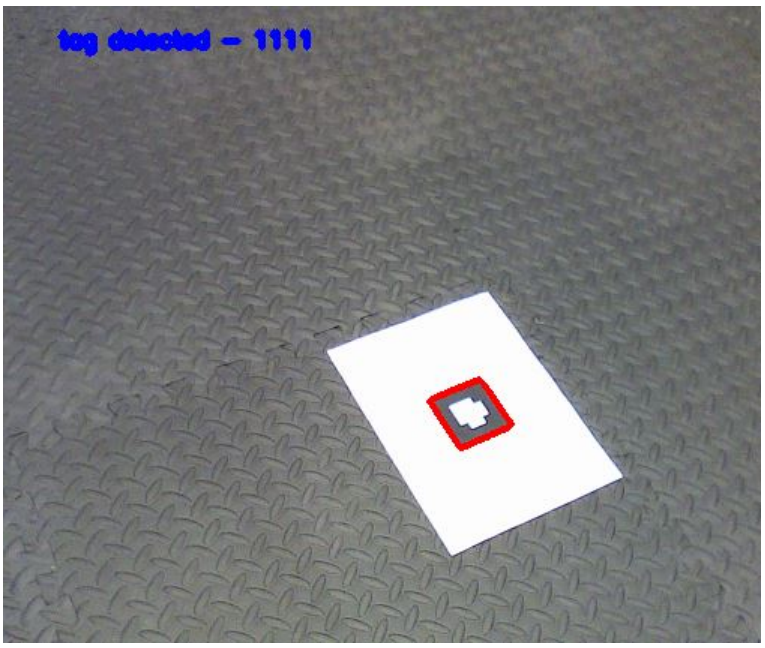

Figure 6: Tag id detection for video Tag0

4. The following are the output snippets of the tag being superimposed on AR tag on different videos.

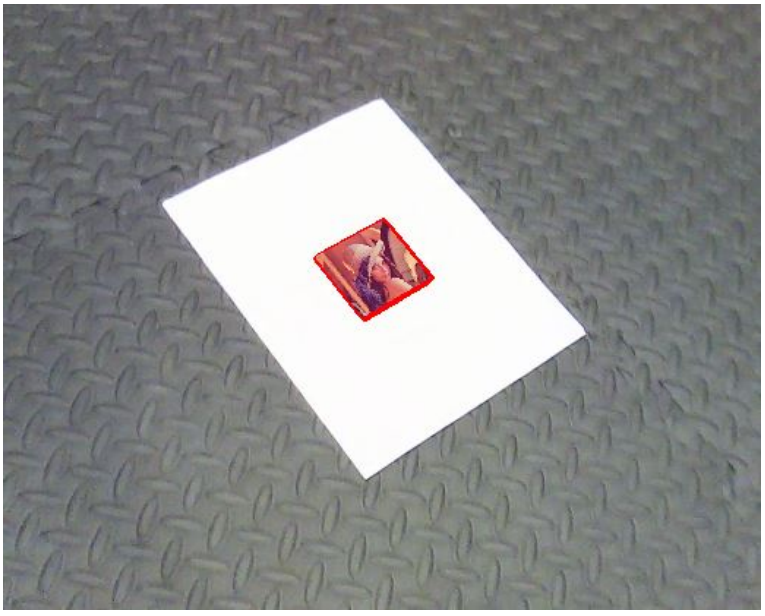

Figure 7: Superimposing Lena onto AR tag on Tag0 video

Figure 8: Superimposing Lena onto AR tag on Tag1 video

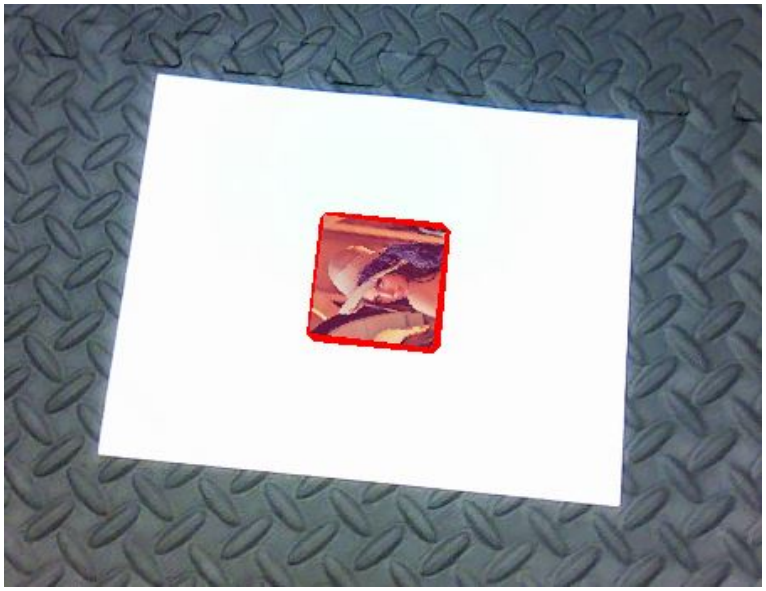

Figure 9: Superimposing Lena onto AR tag on Tag2 video

2.3 Difficulties Faced:

1. We faced a few problem in detecting edges. Sometimes the code detected the edges of the white paper and the image would get superimposed on the entire white paper instead of the ARtag.

2.4 Contributions:

1. The image superimposition function was written by Achal. P. Vyas
2. The sub-functions were written by Akwasi.A.Obeng, Achal. P. Vyas
3. The report was prepared by Eashwar Sathyamurthy

2.5 Problem 2(b) - Placing a virtual cube on the tag

2.6 Procedure for solving problem

1. First we computed the homography between the world coordinates and image plane of the AR tag.
2. Next we coded for obtaining the projection matrix from the homography matrix.
3. Then we wrapped the world coordinates of the AR tag onto the image plane.
4. Then we found out the projection matrix from the homography matrix.
5. Then we project the 8 3-D cube coordinates of the cube onto the image plane by multiplying with the projection matrix and converting it into the homogenous coordinates by dividing the 3-D point with its z-coordinate.
6. The following are the snippets of output from three videos.

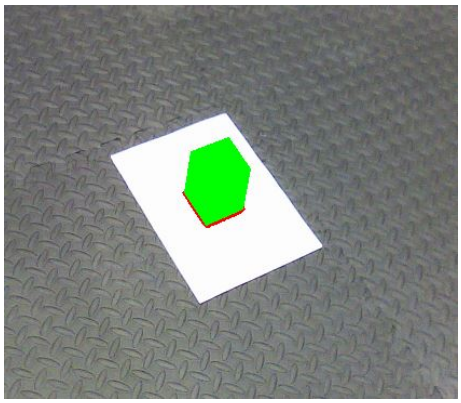

Figure 10: 3D cube superimpose on Tag0 video

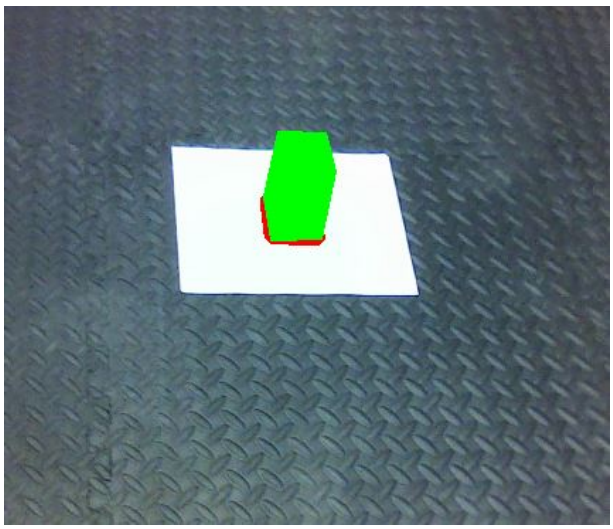

Figure 11: 3D cube superimpose on Tag1 video

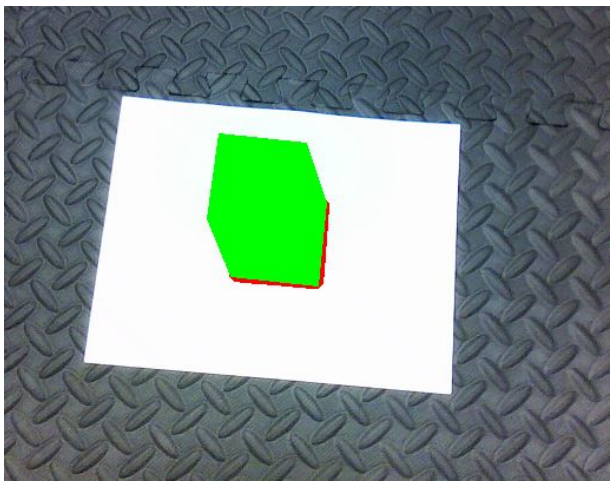

Figure 12: 3D cube superimpose on Tag2 video

2.7 Difficulties Faced:

1. We faced a lot for problem in the projection matrix and converting the 3-D coordinates of the cube to 8 2-D homogenous coordinates.

2.8 Contributions:

1. The cube superimposition function was written by Eashwar Sathyamurthy
2. The sub-functions were written by Akwasi.A.Obeng, Eashwar Sathyamurthy
3. The report was prepared by Eashwar Sathyamurthy, Achal. P. Vyas