Akash Atharv, Rishabh Choudhary, Sanchit Gupta

Project-1 Report
ENPM-673

Introduction

The project was focused on detecting a custom AR tag in a frame of video to generate a frame of reference. Also, the ID represented by the frame was to be read to get the correct orientation.

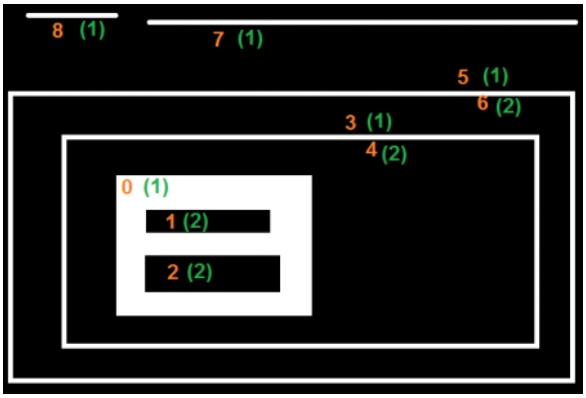
How to run

The final code is present in the file named "___", when the user runs this program, they will be prompted to select the Tag-video for which the code should run. As soon as they select by pressing digits [1-4], the program starts running and displays the information in new windows.

Implementation

We started this project by reading the video into our program, frame by frame and carrying out the necessary operations on it.

Tag Detection:
For tag detection, we used the findContours function of the OpenCV. Here we found the contours with hierarchy so that we always know which contour is inside which one. An example of the same is shown below:




```
[[[ 6 -1  1 -1]
  [-1 -1  2  0]
  [-1 -1  3  1]
  [-1 -1  4  2]
  [-1 -1  5  3]
  [-1 -1 -1  4]
  [11  0  7 -1]
  [-1 -1  8  6]
  [-1 -1  9  7]
  [-1 -1 10  8]
  [-1 -1 -1  9]
  [15  6 12 -1]
  [-1 -1 13 11]
  [-1 -1 14 12]
  [-1 -1 -1 13]
  [16 11 -1 -1]
  [17 15 -1 -1]
  [18 16 -1 -1]
  [19 17 -1 -1]
  [-1 18 -1 -1]]]
```

Hierarchy list returned by findContours function

By using this function, we generate a hierarchy list which has parent information at the 3rd indexed column of the list. From here we filter out all the contours which do not have a parent, as these would constitute the outermost contours which we don't need.

After this initial filtering we found those contours which are formed by more than 4 edges, this will almost at all times constitute for the inner tag shape that we have, we detect these contours and then take the information of its parent which will be the contour that we are interested in.

To capture the right contour, we put 2 other checks, the first one is that the desired contour must be composed of 4 edges and the second is that the contour must have an area less than 2700. We came down to this value after experimentation.

Once we find a contour which satisfies all these constraints, we use a custom created order function which orders the coordinates of the contour in a particular manner so that they are stored clockwise always. After this we append a list with all the correct ordered contour corner points. At the end the list contains all the coordinates of the contours which satisfies the constraints.

For each frame, we draw these contours using green bounding boxes to display the area selected or detected in that frame.

Now we have selected the coordinates of the tag in the world frame to be (0,0), (200,0), (200,200), (0,200). Using this information and the corners of the detected tag we calculate the Homography matrix between the world and the pixel plane.

The H matrix has the following format: $\begin{pmatrix} R1 & R2 & T \end{pmatrix}$

The following format is there because the Tag is present in a 2D plane in the world frame hence we do not need a rotation R3 for this.

Calculations for H =

The homography is calculated by first defining a A matrix as :

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

After constructing the **A** matrix, we decompose it via Singular Value Decomposition to construct separate **U,S** and **V** matrices.

The **V** matrix is then used to calculate the **"h"** vector

$$\boldsymbol{h} = \frac{[v_{19}, \ldots, v_{99}]^T}{v_{99}}.$$

The obtained **h** matrix is then resized to generate the required Homography matrix.

Once we have calculated the Homography matrix, we use the warpPerspective() function to transform the contours from the image plane to the world frame, we also set the max dimension of this to 200. We can then call the id_decode function to detect the correct orientation of the tag.

The id_decode function takes the world frame image of the tag and detects the correct orientation by checking where the white box is present in outer layer of the ID of the tag. It reads the ID of the image in accordance with the presence of the white block in the bottom right corner. This function returns the detected ID and the string giving the location of the tag.

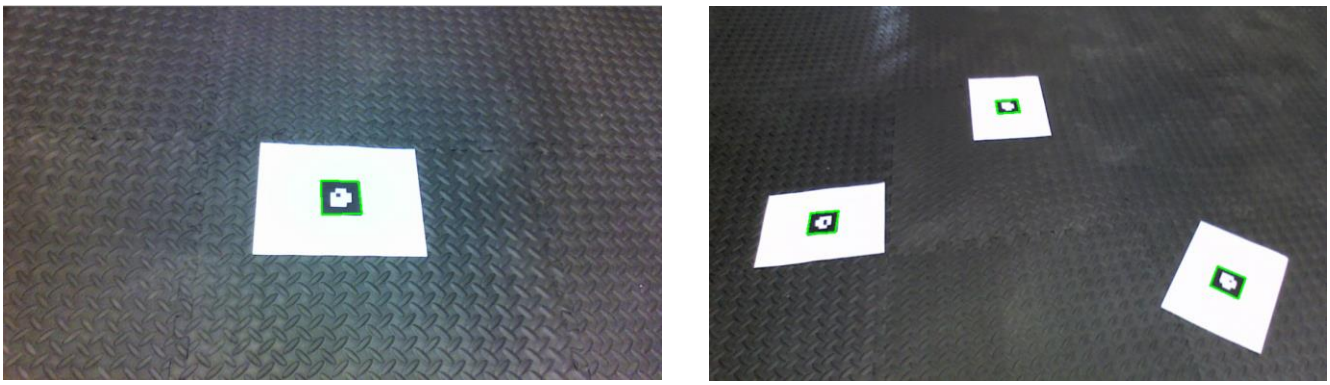
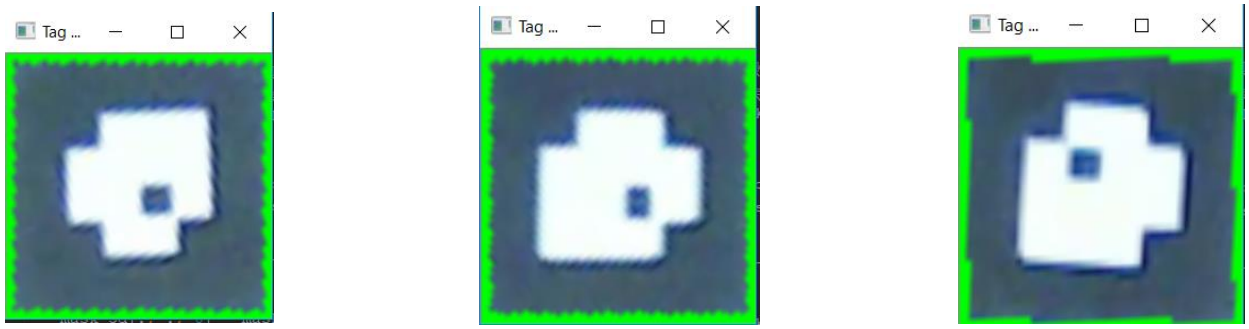


Fig: Detected Tags




```
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
ID detected [0, 1, 1, 1]
```

Detected ID example for an example video

For part 2 of this project we had to project Lena's image on the detected tags on the frame and that too in a correct orientation. We got the orientation from the id_decode() function and then used the reorient() function to calculate the correct order of Lena's image coordinates. The reorient function returns a list of points. The list of Lena's points and the corners of the tags are again fed to calculate the homography between the space. This time, the order of the points is in reverse because we want to project Lena back to the frame. The new image formed after projecting Lena on the frame return an image with Lena on a black background.

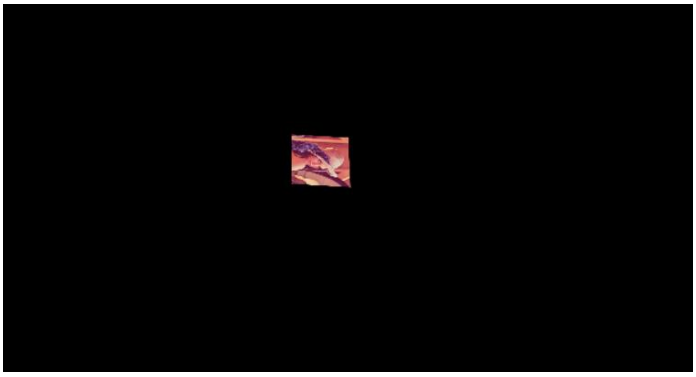


Fig: example of an image formed after projecting Lena on Tag on a black background

Now for showing the rest of the frame with Lena's image in the place of the tag, we used the bitwiseand() and bitwisenot() functions. In the image of the frame, we created a black hole in the place of where the tag was there and then added the formed Lena's image and the frame with a hole together to get the final image. Example of the same is shown below:

Fig: Lena placed on the tag with the frame background

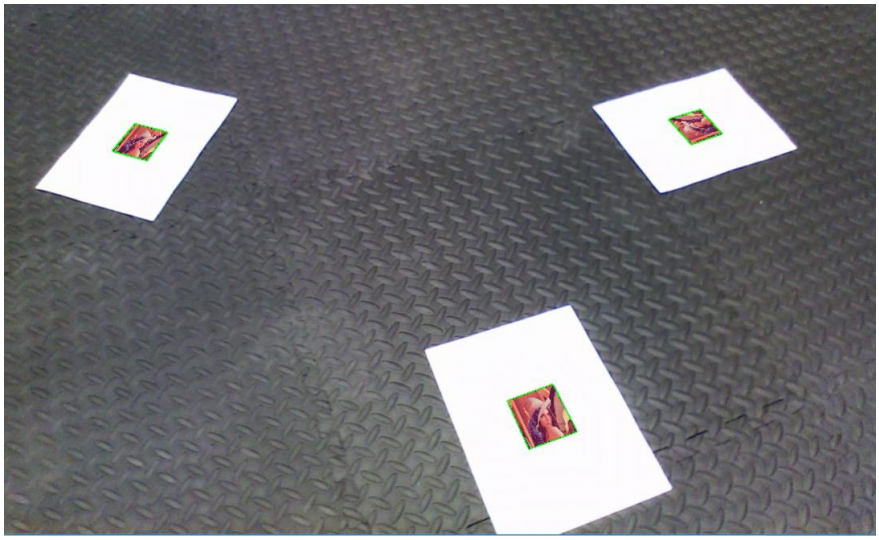


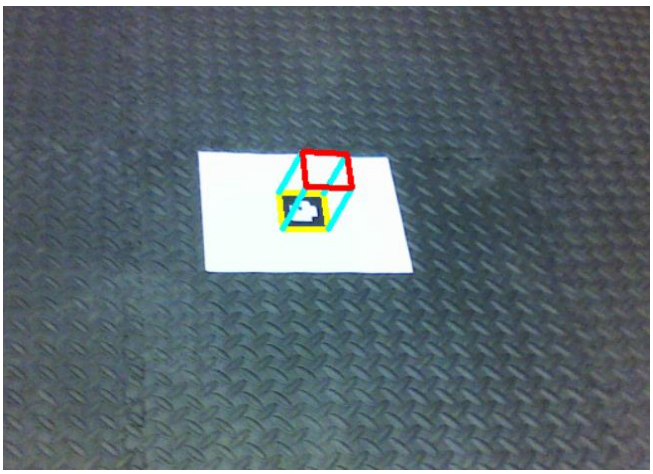
Fig: Lena's image on all 3 tags with background

Part 3: Cube placement on Tags

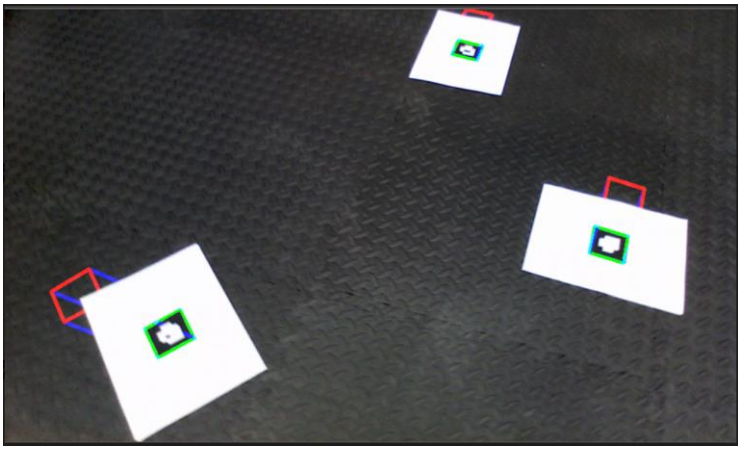
For placing a cube on the tag we first define the cube coordinates in the world coordinate frame and then calculate the transformation matrix which comprises of a rotation and translation matrix. The first step is to calculate a matrix which is the product of the homography matrix and camera matrix. The obtained matrix's first and second columns represent the first two columns of the rotation matrix (**r1** and **r2**). We then calculate the **r3** column of the rotation matrix through the cross product of **r1** and **r2**. The last column calculated from the afore-mentioned product is the required translation matrix. We then calculate a scaling factor "lambda" which is used for scaling the transformation matrix. Lambda is calculated as per the following equation :

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

The obtained data is used calculate the projected point coordinates which are then fed into a function which is then used to draw the cube on the tag. The projected point coordinates are calculated through the cv2.projectPoints() function and cv2.drawContours() and cv2.line() functions are used to draw the separate faces and then the pillars between the two faces respectively.



Example of Cube placement on the tag



Challenges faced during implementation

- **Corner Detection:**
  We tried various corner detection algorithms like Harris and GoodFeaturesToTrack to find the correct order of the tags but could not get the good coordinates from these. After which we found the contour detection which worked for us and gave us the information of contour parent.
- **Lena's image on the frame:**
  We could place the Lena's image on the black background using the homography matrix calculated but to add the her on the frame we had to perform bitwiseand and bitwiseor operation. This created a lot of instances where Lena's image was coming but tag was not omitting out. We could do it in the end after many iterations
- **Placing Cube on the Tag:**
  For placing a cube on the tag, we were facing a lot of challenges. We could find the projection matrix with 3x4 matrix and using that we generated 8 points to display on the pixel plane, but to draw them on the frame took us a very long time.

Collaborations

For Project 1, We collaborated with Aaradhana and Mrinali to discuss steps and for implementation of the code.

References

To find corners

- https://docs.opencv.org/3.4.2/d4/d73/tutorial_py_contours_begin.html
- https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html
- https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html

To place Lena in the frame

- https://docs.opencv.org/3.2.0/d0/d86/tutorial_py_image_arithmetics.html
- https://docs.opencv.org/2.4/doc/tutorials/core/adding_images/adding_images.html
- https://stackoverflow.com/questions/7589012/combining-two-images-with-opencv
- https://www.geeksforgeeks.org/addition-blending-images-using-opencv-python/

To place the cube on the frame:

- https://stackoverflow.com/questions/22180923/how-to-place-object-in-video-with-opencv
- https://www.youtube.com/watch?v=wVPcXGGsVlM
- https://www.geeksforgeeks.org/draw-geometric-shapes-images-using-opencv/

The supplementary given with the problem statement to find the homography matrix and to understand the concepts.