[Fall 2024] ROB-GY 6203 Robot Perception Homework 1

Jithin George (jg7688)

Submission Deadline (No late submission): NYC Time 11:00 AM, October 9, 2023
Submission URL (must use your NYU account): https://forms.gle/EPyThuLsYBopQQ3MA

Contents

Task 1 Sherlock's Message (2pt)

Detective Sherlock left a message for his assistant Dr. Watson while tracking his arch-enemy Professor Moriarty. Could you help Dr. Watson decode this message? The original image itself can be found in the data folder of the overleaf project (https://www.overleaf.com/read/vqxqpvbftyjf), named `for_watson.png`

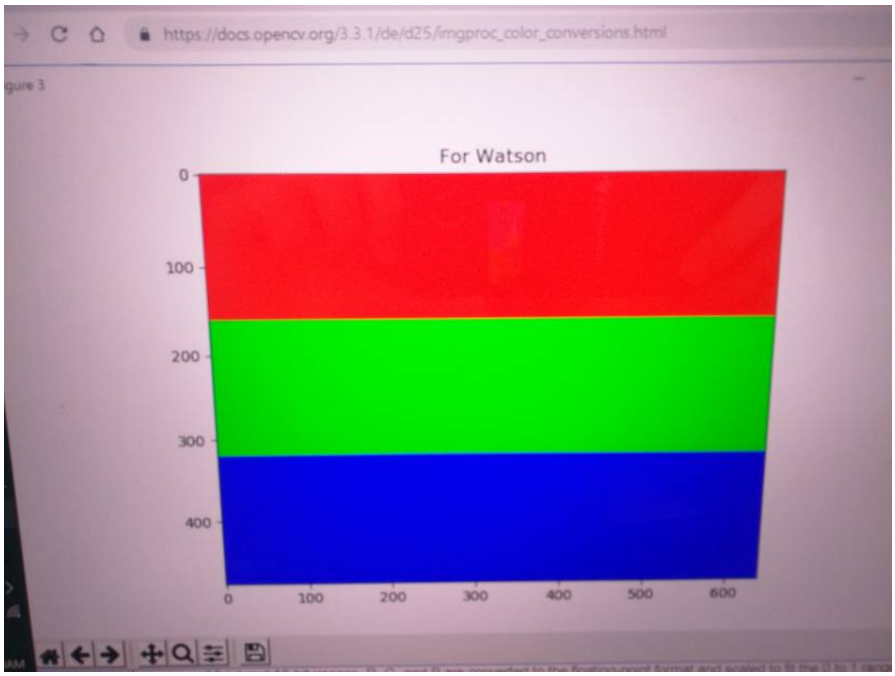

Figure 1: The Secret Message Left by Detective Sherlock

Part A (1pt)

Please submit the image(s) after decoding. The image(s) should have the secret message on it(them). Screenshots or images saved by OpenCV is fine.

Answers:
The following is the decoded message:

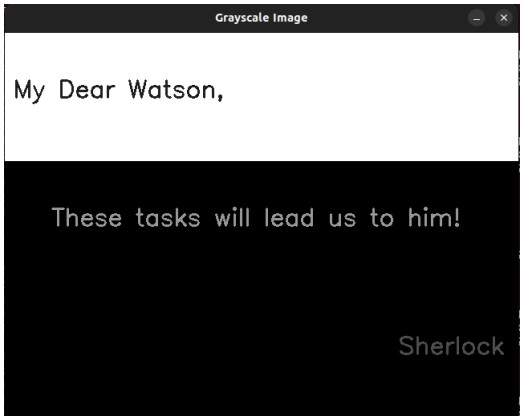

Figure 2: Decoded Message

Name: Jithin George, netID: jg7688

Part B (1pt)

Please describe what you did with the image with words, and tell us where to find the code you wrote for this question.

Answers:
The following code processes an image, applies thresholding, and modifies specific pixel intensities in the grayscale version of the image to reveal concealed patterns or text.

(Steps Taken):

1. Loaded the image using OpenCV.
2. Converted the image to grayscale to simplify processing.
3. Applied thresholding by setting specific pixel values to contrasting white (255) or black (0) to make the hidden message more visible.
4. Displayed the processed image using OpenCV.

Code Explanation

The code was written in Python using OpenCV and NumPy libraries.

```
Load the image
path = '/home/jithin/Desktop/Perception/HW1/forwatson.png'
matrixedimg = cv2.imread(path)
Convert to grayscale
grayscaled = cv2.cvtColor(matrixedimg, cv2.COLOR_BGR2GRAY)
Thresholding / modifying pixel values
grayscaled[grayscaled == 29] = 255
grayscaled[grayscaled == 150] = 0
grayscaled[grayscaled == 76] = 0
Display the processed image
cv2.imshow('Grayscale Image', grayscaled)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

(Code Location):
Code for decoding the message

Task 2. Deep Learning with Fashion-MNIST (5pt)

Given the Fashion-MNIST dataset, perform the following task:

Part A (2pt)

Train an unsupervised learning neural network that gives you a lower-dimensional representation of the images, after which you could easily use t-SNE from `Scikit-Learn` to bring the dimension down to **Visualize** the results of all 10000 images in one single visualization.

Answers:
PART A Code

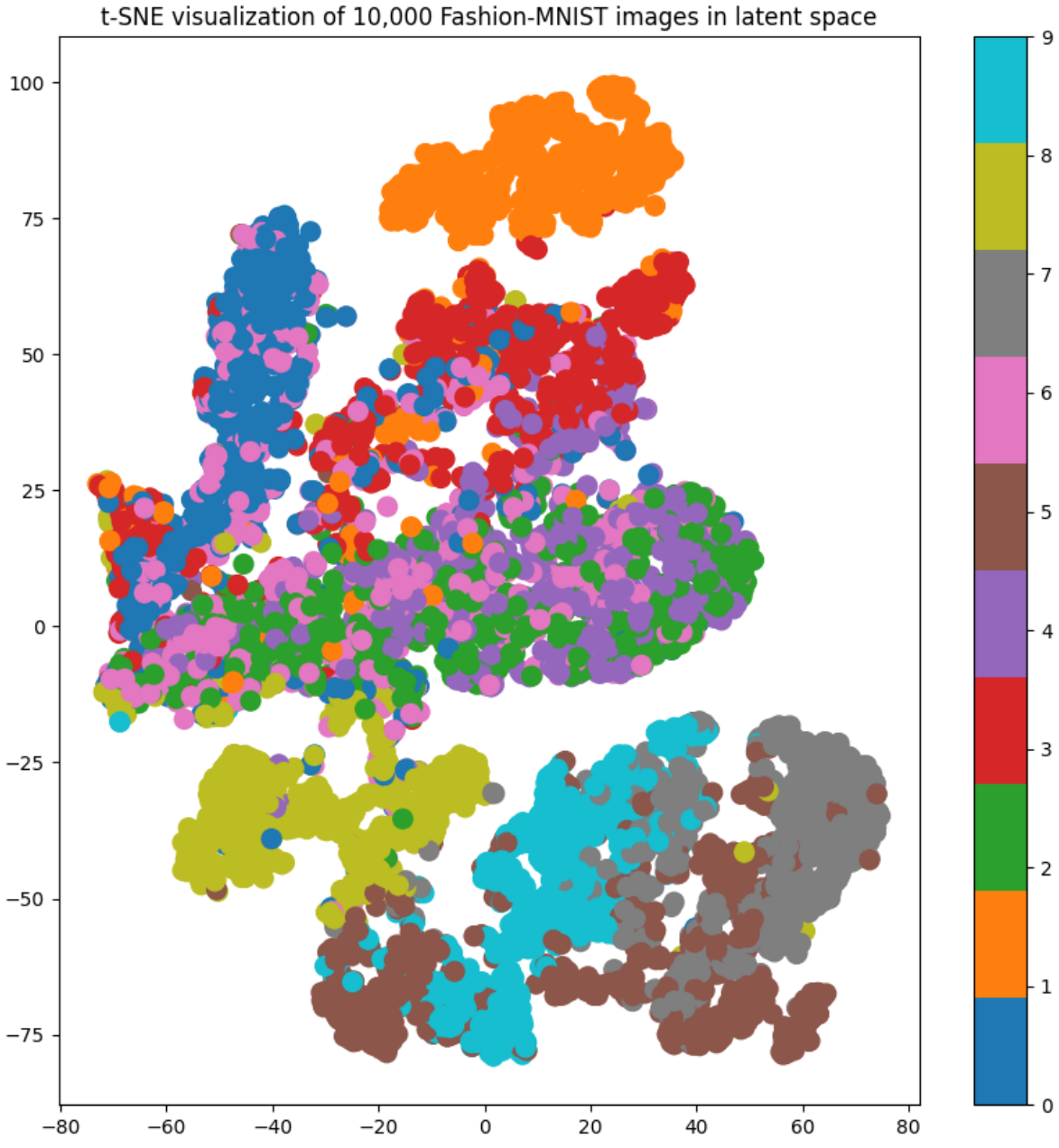

Figure 3: Tensor Flow Visualisation

Name: Jithin George, netID: jg7688

Part B (3pt)

Take the lower-dimensional latent representation produced in Part A and **train** a supervised classifier using these features. **Visualize** the loss and accuracy curves during the training process for both the training and testing datasets. Discuss your observations on the behavior of both curves. Evaluate the classifier's performance using accuracy or other appropriate metrics on the test set. **Report** your final accuracy, providing examples of correct and incorrect predictions.

Answers:

PART B Code

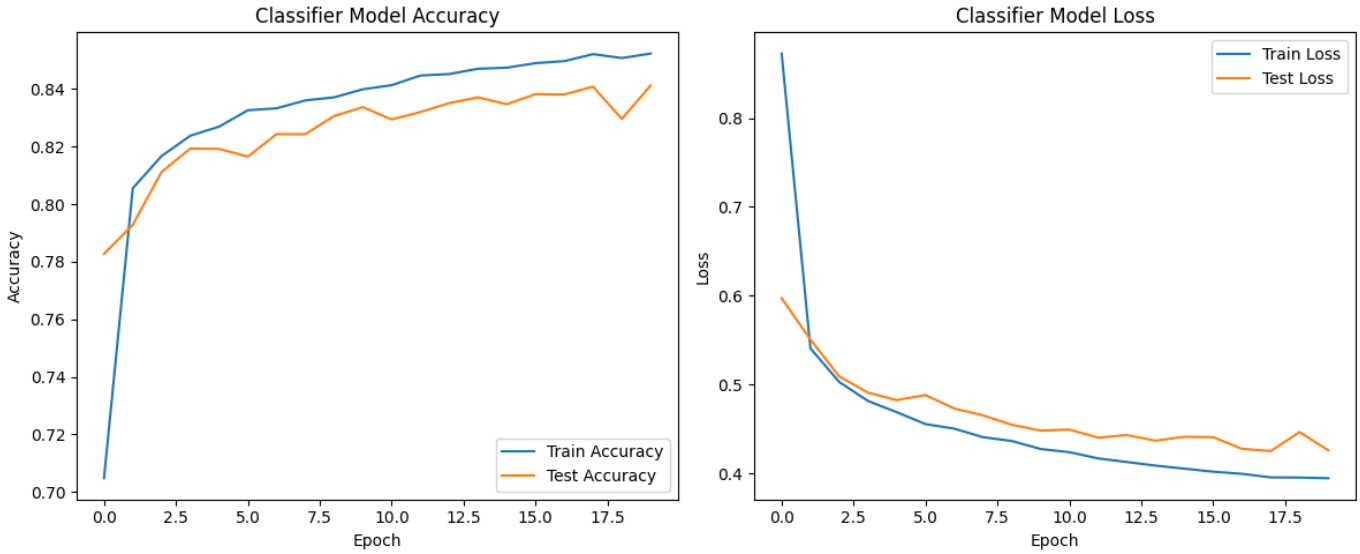

Figure 4: Classifier Model: Accuracy and Loss

Accuracy Curves

Training Accuracy (blue): The training accuracy increases rapidly during the initial epochs, stabilizing around 0.85. This indicates the model is effectively learning from the training data.
Test Accuracy (orange): The test accuracy follows a similar pattern, stabilizing around 0.82. The slight gap between the training and test accuracy suggests the model generalizes well, with minimal overfitting.

Loss Curves

Training Loss (blue): The training loss decreases sharply in the initial epochs and continues to decrease steadily, indicating effective training.
Test Loss (orange): The test loss mirrors the training loss but begins to fluctuate slightly after epoch 10, hinting at potential overfitting or instability.

The model demonstrates promising generalization capabilities, as evidenced by the minimal discrepancy between training and testing metrics. However, there are subtle signs of overfitting emerging in the later stages of training, as indicated by the fluctuating test loss. To mitigate this issue, implementing regularization techniques or employing early stopping strategies could be beneficial. Overall, the model exhibits a stable learning pattern, and further fine-tuning could potentially enhance its test performance.

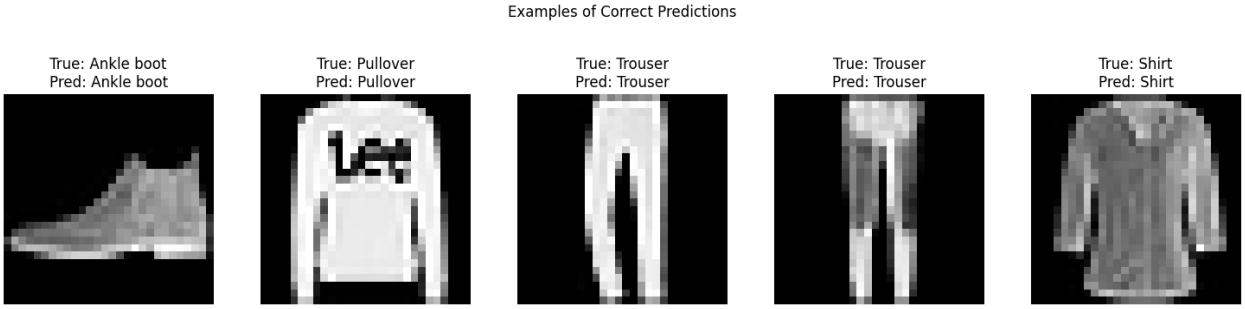

Figure 5: Correct Predictions

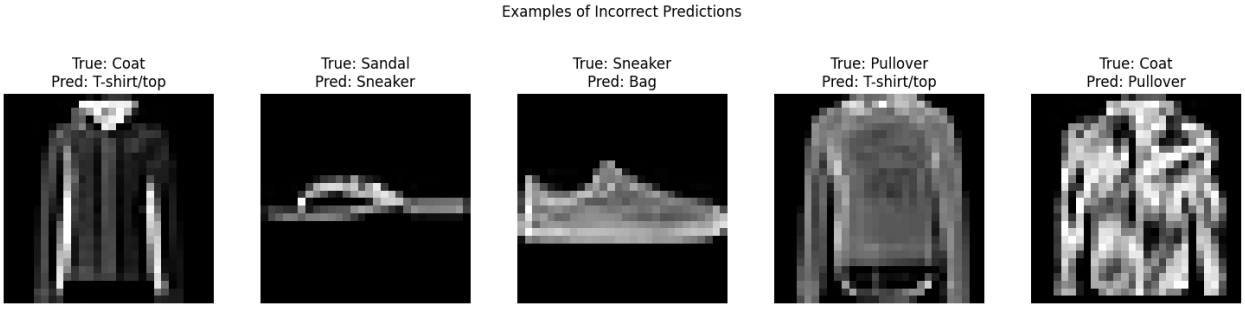

Figure 6: Incorrect Predictions

Task 3 Camera Calibration (3pt)

Compare and contrast the intrinsic parameters (K matrix) and distortion coefficients (k1 and k2) obtained from calibrating your camera using two different sets of images. For the first set, take images where the distance between the camera and the calibration rig is **within 1 meter**. For the second set, take images where the distance is **between 2 to 3 meters**. Use the provided pyAprilTag package or other available tools (such as OpenCV's camera calibration toolkit) to perform the calibration and analyze the differences between the two sets. Discuss potential reason(s) for the differences (A good discussion about these reasons could receive 1 bonus point).

Answers:
Code

```
Processing images from:  /home/jithin/Desktop/AprilTag/calibimg_1mtr
Case 1:  Camera matrix (K): [[3.31207676e+03 0.00000000e+00 1.99785473e+03] [0.00000000e+00
3.31227923e+03 1.51266554e+03] [0.00000000e+00 0.00000000e+00 1.00000000e+00]] Case 1:
Distortion coefficients (d):  [ 1.06953067e-02 1.21634329e+00 -4.18982616e-04 -1.03888267e-03
-6.09835313e+00]
Processing images from:
/home/jithin/Desktop/AprilTag/calibimg_12mtrNomarkersdetectedinimageIMG_0187.jpg
Case 2:  Camera matrix (K): [[3.28189532e+03 0.00000000e+00 1.96178639e+03] [0.00000000e+00
3.26287780e+03 1.54649719e+03] [0.00000000e+00 0.00000000e+00 1.00000000e+00]] Case 2:
Distortion coefficients (d):  [ 0.06594108 0.09081337 0.00118874 0.00112097 -0.89241043]
```

Case 1: Camera Matrix (K):

$$K_1 = \begin{bmatrix} 3.31207676 \times 10^3 & 0 & 1.99785473 \times 10^3 \\ 0 & 3.31227923 \times 10^3 & 1.51266554 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion Coefficients (d):

$$d_1 = \begin{bmatrix} 1.06953067 \times 10^{-2}, & 1.21634329 \end{bmatrix}$$

Case 2: Camera Matrix (K):

$$K_2 = \begin{bmatrix} 3.28189532 \times 10^3 & 0 & 1.96178639 \times 10^3 \\ 0 & 3.26287780 \times 10^3 & 1.54649719 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion Coefficients (d):

$$d_2 = \begin{bmatrix} 0.06594108, & 0.09081337 \end{bmatrix}$$

The radial distortion is more noticeable in the first case because the tags were captured within 1 meter. This is due to the fact that radial distortion becomes more pronounced as you move further away from the center of the image. Even though there is also radial distortion in the second set, it is less noticeable because the tags are captured at a greater distance from the center.

The focal length variations observed between the two cases could be attributed to the camera's autofocus mechanism adjusting to capture images at different distances. When the tags are closer (case 1), the camera may need to adjust its focal length to bring the subject into sharp focus. Conversely, when the tags are farther away (case 2), the camera might use a longer focal length to achieve the same focus.

This adjustment in focal length can introduce variations in the distortion coefficients, as different focal lengths can have slightly different optical characteristics.

Task 4 Tag-based Augmented Reality (5pt)

Use the pyAprilTag package to detect an AprilTag in an image (or use OpenCV for an Aruco Tag), for which you should take a photo of a tag. Use the K matrix you obtained above, to draw a 3D cube of the same size of the tag on the image, as if this virtual pyramid really is on top of the tag. **Document** the methods you use, and **show** your AR results from at least two different perspectives.

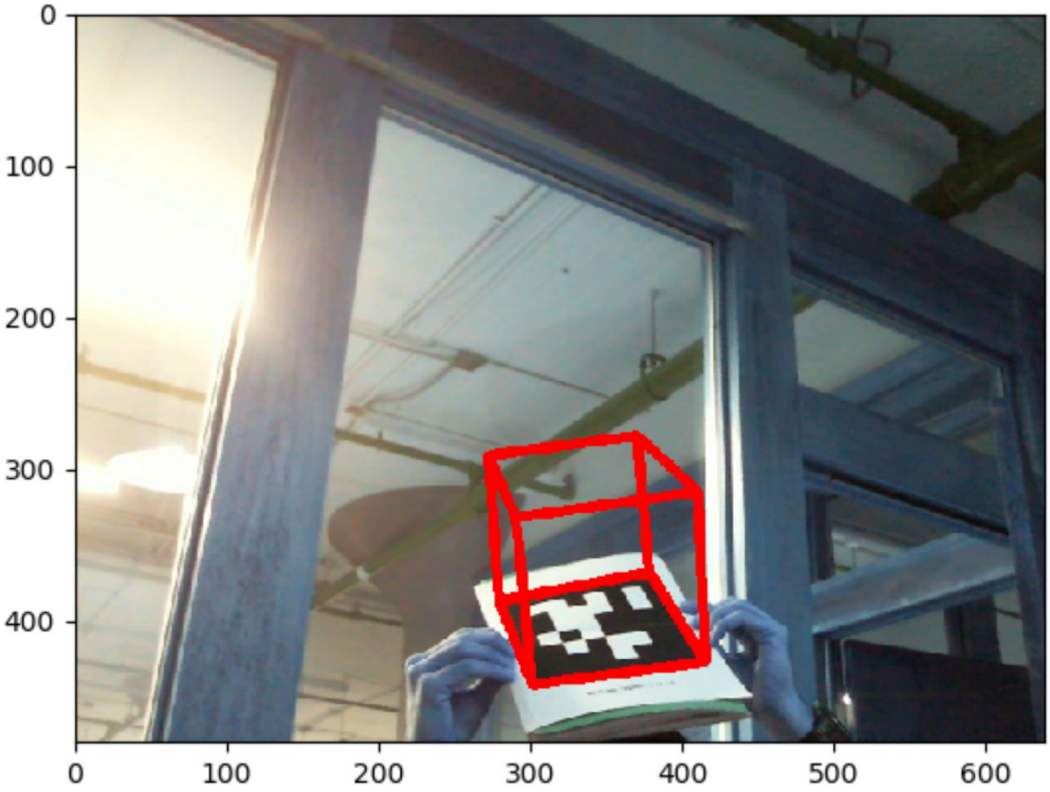

Figure 7: Projected Pyramid on checkerboard

Answers:
The code implements an augmented reality (AR) application that overlays a 3D cube onto a detected AprilTag or ArUco marker in an image. It uses the pyAprilTag or OpenCV ArUco library for marker detection and camera calibration data for accurate placement.

- A dictionary of AprilTags (`DICT_APRILTAG_36h11`) is created, specifying the size (`Tag_dim = 4.00 units`) and spacing of the marker. A grid board is generated, although only one marker is drawn (1x1 grid) for simplicity.
- The camera matrix K (obtained from above) contains the intrinsic parameters such as focal lengths and optical center coordinates.
- The vertices of the 3D cube are defined relative to the marker's center in a coordinate system, with the base lying on the $z = 0$ plane and the top at $z = 4$.
- The function `process_image()` reads an input image and converts it to grayscale. The `aruco.detectMarkers()` function detects the marker, returning the corners and IDs of any detected markers.
- The pose (rotation and translation) of the marker is estimated using the `aruco.estimatePoseSingleMarkers()` function, which computes the position of the marker relative to the camera.
- The cube's 3D points are projected onto the 2D image plane using `cv2.projectPoints()`, which uses the camera matrix, rotation, and translation vectors. The resulting 2D coordinates of the cube are used to overlay the cube on the marker.
- Using `cv2.drawContours()`, the bottom and top squares of the cube are drawn in green and red, respectively. Vertical lines connecting the base and top are drawn using `cv2.line()` to complete the 3D effect.

- The processed image, with the 3D cube overlay, is saved to an output file using `cv2.imwrite()`, and displayed on the screen using `cv2.imshow()`.
- The function `process_image()` is called twice for two different images, rendering the cube from different perspectives, and saving the final images with the cube overlay.
- This approach effectively detects an AprilTag in an image, estimates its pose, and overlays a 3D cube in real-time, providing a simple yet functional AR simulation.

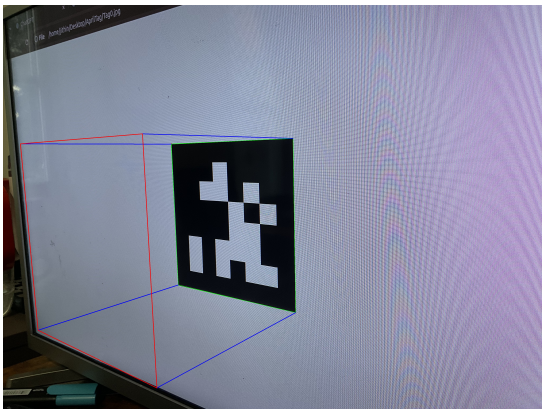

Figure 8: Right Perspective

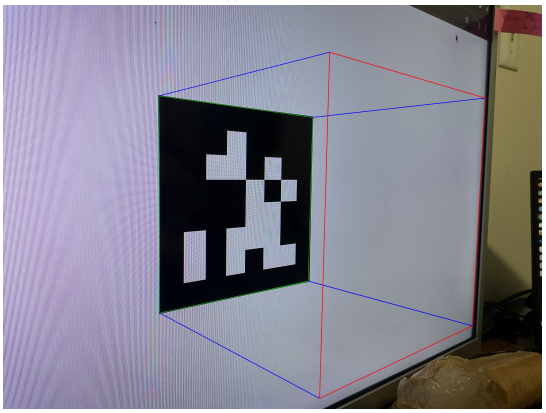

Figure 9: Left Perspective