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

Raman Kumar Jha (N13866145)

Submission Deadline (No late submission): NYC Time 11:00 AM, October 9, 2024
Submission URL (must use your NYU account): https://forms.gle/EPyThuLsYBopQQ3MA

1. Please submit the **.pdf** generated by this LaTex file. This .pdf file will be the main document for us to grade your homework. If you wrote any code, please zip all the **code** together and **submit a single .zip file**. Name the code scripts clearly or/and make explicit reference in your written answers. Do NOT submit very large data files along with your code!

2. You don't have to use AprilTag for this homework. You can use OpenCV's Aruco tag if you are more familiar with them.

3. You don't have to physically print out a tag. Put them on some screen like your phone or iPad would work most of the time. Make sure the background of the tag is white. In my experience a tag on a black background is harder to detect.

4. Please typeset your report in LaTex/Overleaf. Learn how to use LaTex/Overleaf before HW deadline, it is easy because we have created this template for you! **Do NOT submit a hand-written report!** If you do, it will be rejected from grading.

5. Do not forget to update the variables "yourName" and "yourNetID".

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`

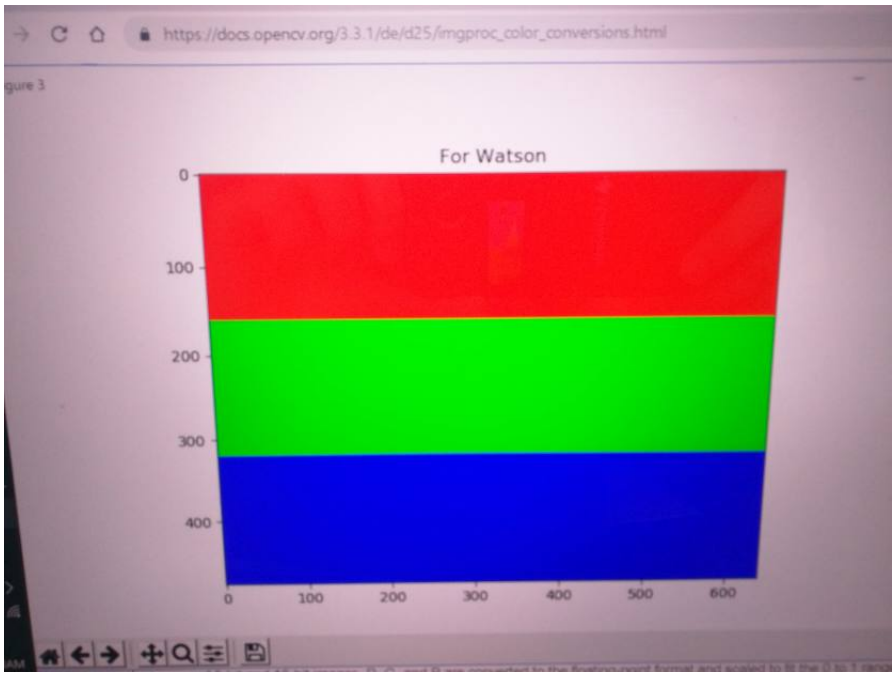
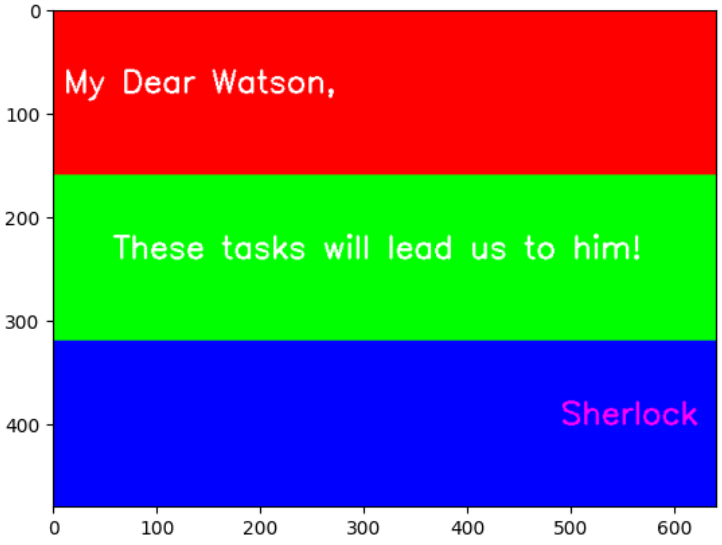


Figure 1: The Secret Message Left by Detective Sherlock

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:



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:
In this problem to decode the Sherlock's message, I have used the cv2.imread function to read the image, and in the same function, I utilized cv2.IMREADunderscoreCOLOR so that the colors of the images will also be read.
After this, I used the cv2.threshold method for the original image to threshold it in the binary format. For thresholding it in the binary format, I used cv2.THRESHunderscoreBINARY function, and set the minimum threshold value as 0, and the maximum threshold value as 255.
I have added the decoded message in the code snippet above.
I have included the code for this task in the Zip file as well with the name HW1task1.ipynb, and the decoded image as decodedunderscoremessage.png.

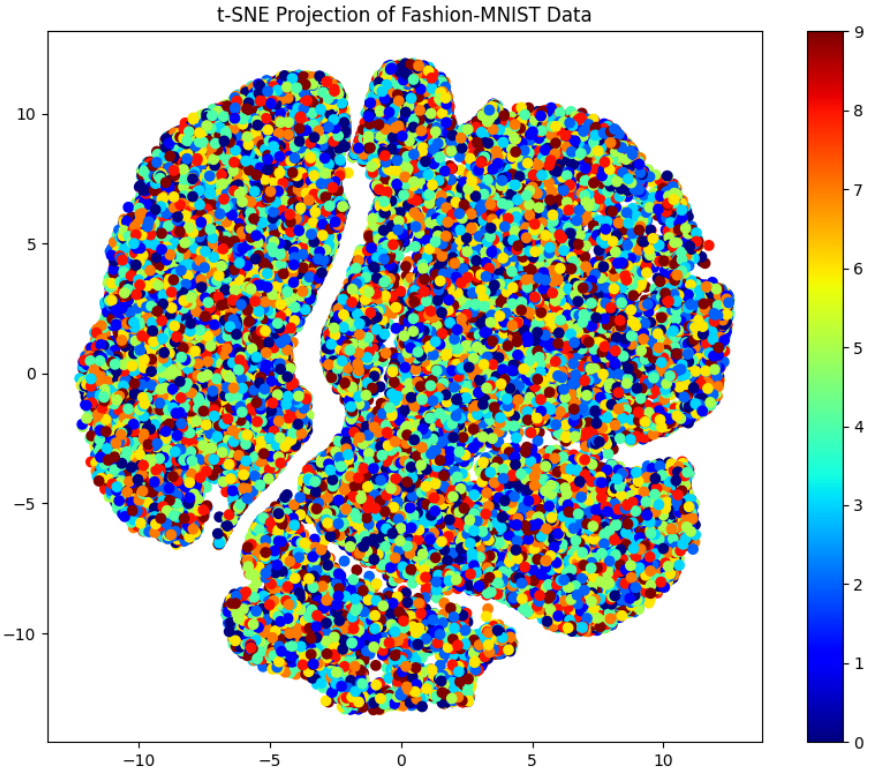
Task 2. Deep Learning with Fasion-MNIST (5pt)

Given the Fasion-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:
I trained a Variational Autoencoder (VAE) using PyTorch and obtained latent space representations of the input images. To visualize these high-dimensional representations, I applied t-SNE for dimensionality reduction to two dimensions. This produced a 2D plot of all 10,000 images, revealing their structure and distribution in the learned latent space.



The code can be found in the zip file with the name HW1Task2PartA.ipynb. The dataset is downloaded and transformed into a tensor for processing. A DataLoader is created with a batch size of 64 and shuffling enabled. The autoencoder consists of an encoder and a decoder.

In the encoder, images pass through a 2D convolutional layer with a kernel size of 3, stride of 2, and padding of 1, reducing their size from 28x28 to 14x14 while preserving detail. This is followed by a ReLU activation and another convolutional layer, further reducing the size to 7x7, followed by another ReLU and flattening to prepare for the decoder.

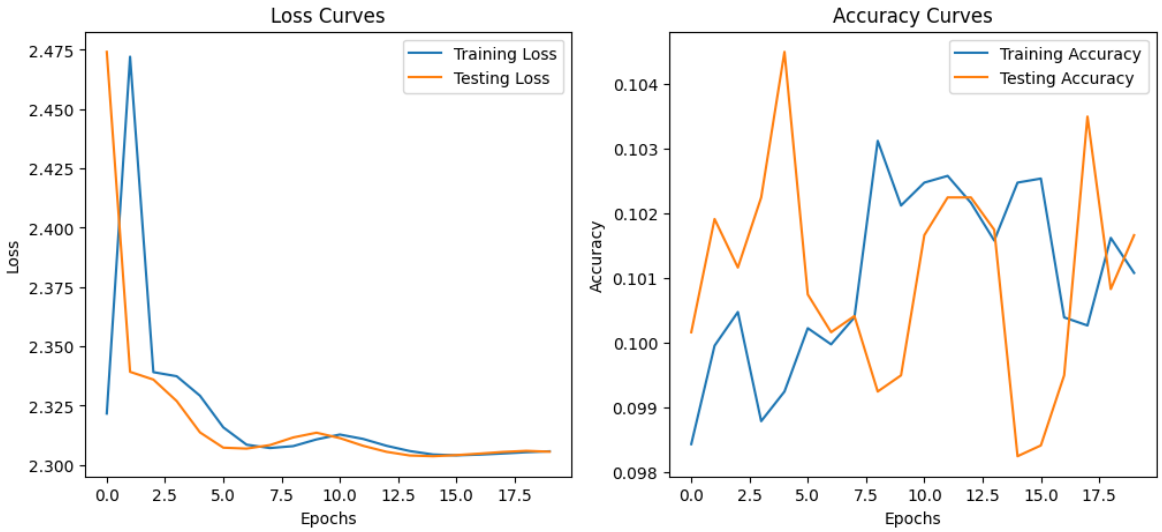
The decoder begins by unflattening the encoded representation back to 7x7. It then uses a transposed convolutional layer with similar parameters to upscale the image to 14x14, followed by a ReLU activation and another transposed convolutional layer that restores the image to its original 28x28 size, concluding with a sigmoid activation for output.

The network was trained using MSE loss and the Adam optimizer for 10 epochs, finishing with a loss of 0.0006. For t-SNE visualization, I extracted the lower-dimensional encodings by feeding the images through the autoencoder and collecting only the encoded outputs. I applied t-SNE with 2 components for 2D visualization, using a perplexity of 30 and 300 iterations. The resulting visualization illustrates the structure of the latent space, as shown above. Further details can be found in the linked notebook.

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.

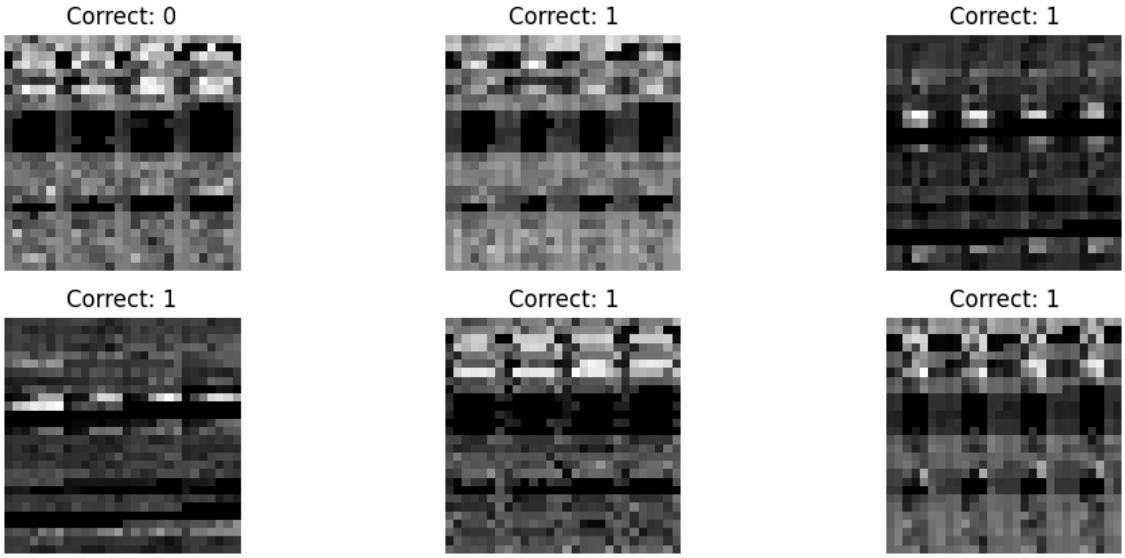
Answers:
In this task, we visualize the performance of a supervised classifier trained on the lower-dimensional latent representations from the Variational Autoencoder. As this is a continuation task of PartA, so the code for this will also be available in the same notebook, i.e. HW1Task2.ipynb

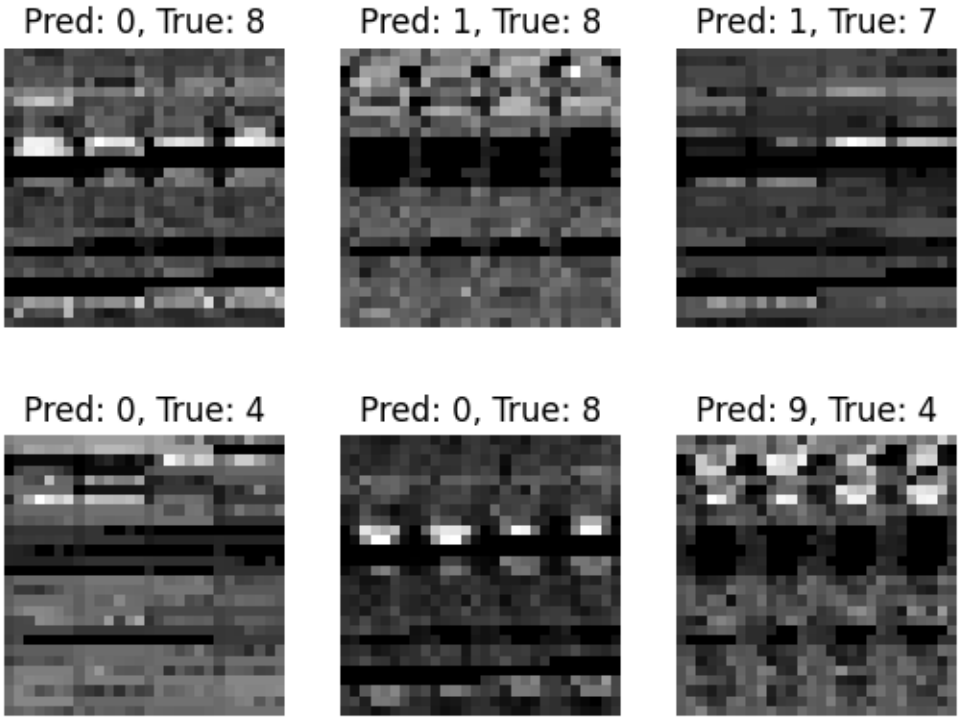


Looking at the loss and accuracy curves, I notice a few interesting patterns. In the loss curves on the left, both the training and testing loss start high and decrease rapidly within the first few epochs. This suggests that the model is learning quickly at the beginning. However, after about 3 epochs, the decrease in loss slows down, indicating that the model is approaching convergence. For the accuracy curves on the right, there's a lot of fluctuation, especially in testing accuracy. This variability suggests that the model's performance on unseen data is inconsistent, which might be due to overfitting or noise in the data. Training accuracy is more stable but also shows some variation. Overall, while the model initially learns well, the fluctuations in testing accuracy suggest potential issues with generalization. It might be beneficial to use techniques like regularization to improve the stability and performance of test data.

Final Test Accuracy: 0.1017

I begin by identifying correct and incorrect predictions using boolean indexing. For the correct predictions, we extract the indices and display six images from the test dataset, along with their true labels. Similarly, I do the same for the incorrect predictions, showing six images along with their predicted and actual labels. This visualization allow me to qualitatively assess the classifier's performance by examining both correct and incorrect classifications.


Correct: 0
Correct: 1
Correct: 1
Correct: 1
Correct: 1
Correct: 1


Pred: 0, True: 8
Pred: 1, True: 8
Pred: 1, True: 7
Pred: 0, True: 4
Pred: 0, True: 8
Pred: 9, True: 4

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:
I utilized the OpenCV camera calibration toolkit for this problem. The whole code is provided for both of the sets of images in the notebook named HW1Task3.ipynb

This is the values of the intrinsic matrix and Distortion Co-efficients, with the set of images kept within 1m between the camera and the calibration rig.

```
Intrinsic Matrix parameter(K):
[[2.26491561e+03 0.00000000e+00 1.28324475e+03]
 [0.00000000e+00 2.27353741e+03 8.90856990e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
Distortion Coefficients (k1, k2):
-0.008142203590221605 -1.3203650369970834
```

This is the values of the intrinsic matrix and Distortion Co-efficients, with the set of images kept between 2 to 3 meters between the camera and the calibration rig.

```
Intrinsic Matrix parameter(K):
[[2.37816139e+03 0.00000000e+00 1.34740699e+03]
 [0.00000000e+00 2.38721428e+03 9.35399840e+02]
 [0.00000000e+00 0.00000000e+00 1.05000000e+00]]
Distortion Coefficients (k1, k2):
-0.006676606943981716 -1.0826993303376082
```

$$K_1 = \begin{bmatrix} 2264.91561 & 0 & 1283.24475 \\ 0 & 2273.53741 & 890.85699 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K_2 = \begin{bmatrix} 2378.16139 & 0 & 1347.40699 \\ 0 & 2387.21428 & 935.39984 \\ 0 & 0 & 1.05 \end{bmatrix}$$

Here are some of the differences in the K Matrix, and the values of the k1, and k2:

1. The focal lengths in both the x and y directions are higher in the second set.
2. The principal point coordinates (cx, cy) are slightly shifted.
3. Both distortion coefficients are less negative in the second set, indicating reduced distortion effects.

Potential reasons for differences can be:
1. Perspective Variation: At different distances, the perspective distortion changes, affecting how the camera perceives depth and angles, which can impact calibration results.
2. Lens Behavior: Lenses may exhibit different characteristics at varying focal lengths and distances due to non-linearities, leading to changes in intrinsic parameters.
3. Calibration Rig Size: The relative size of the calibration rig in the image frame changes with distance, potentially affecting the accuracy of intrinsic parameter estimation.

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.

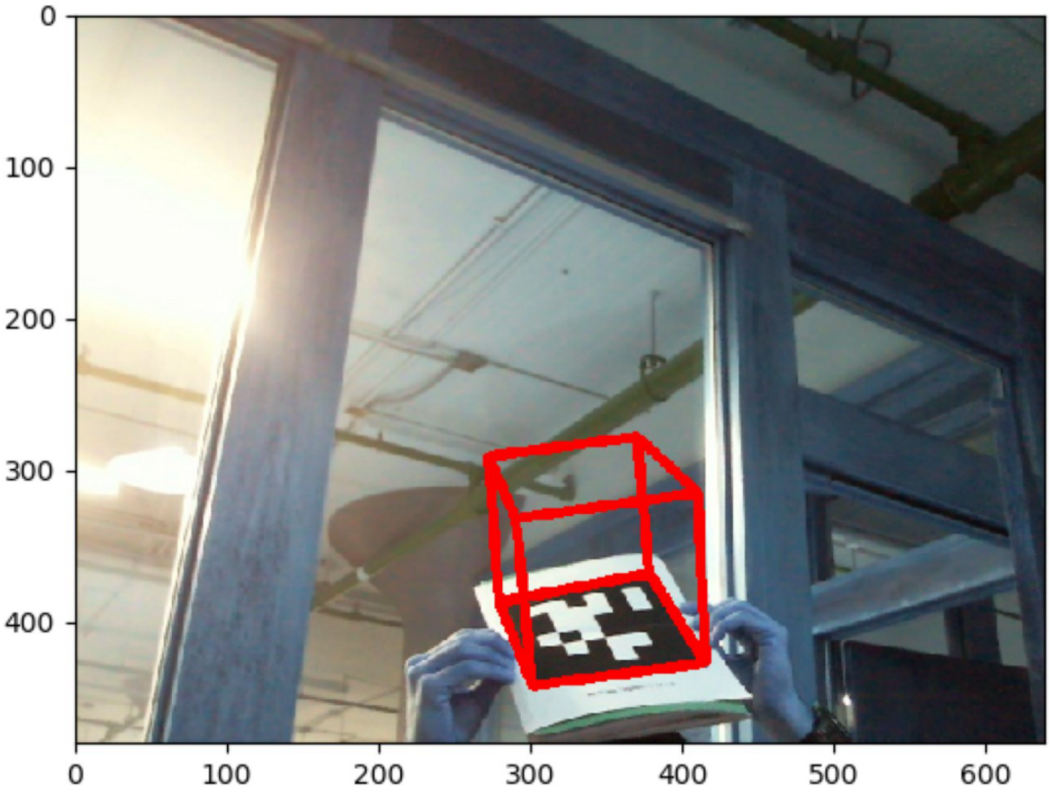


Figure 2: Projected Pyramid on checkerboard

Tips: There are many ways to do this, but you may find OpenCV's `projectPoints`, `drawContours`, `addWeighted` and `line` functions useful. You don't have to use all these functions.

Answers:

The code for this task is provided in the zip file with the name, HW1Task4.ipynb. It has the solution needed for generating the cube on the April tags.

In my implementation of the tag-based augmented reality task, I started by importing the necessary libraries: OpenCV for image processing and ArUco marker detection, NumPy for numerical operations, and Google Colab-specific modules for image display and file handling. I began by uploading a cropped image to Colab. I had to crop the original image as my face was visible, and I wanted to focus solely on the AprilTag that I had printed on paper.

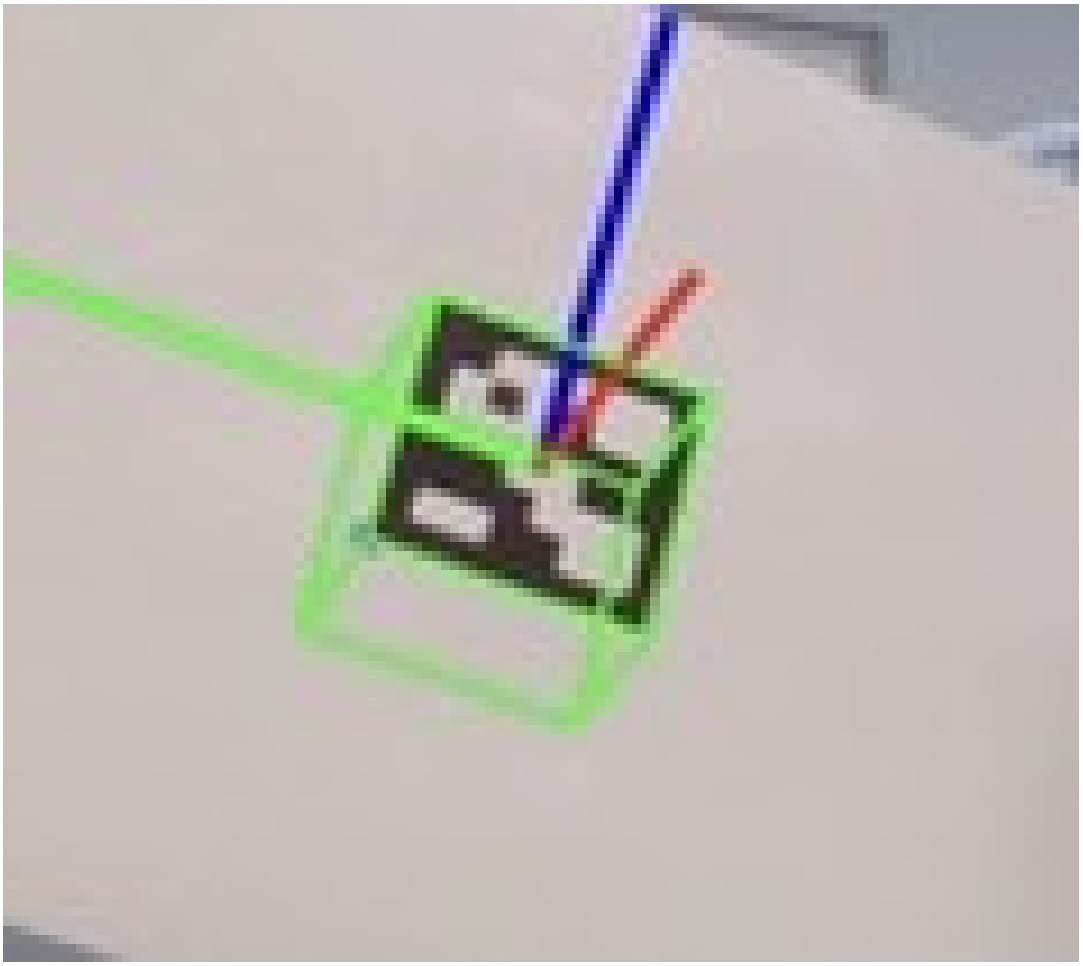


Figure 3: Projected Pyramid of April tag printed on white paper

This cropped image contained the AprilTag I had carefully printed and placed in the scene. Next, I defined a camera matrix K with estimated values. In a real-world scenario, I would have used precise calibration data, but for this demonstration, I used approximated values. I then created a function called drawunderscorecube to render a 3D cube on the detected marker. This function uses OpenCV's drawing functions to create the illusion of a three-dimensional object on the 2D image. The core of my implementation involved detecting the ArUco markers (which are similar to AprilTags) in the image. To ensure robust detection, I applied adaptive thresholding to the grayscale version of my image. This step helped to account for varying lighting conditions and improve marker visibility. Knowing that different ArUco dictionaries exist, and unsure which exactly matched my printed AprilTag, I implemented a loop to try multiple dictionaries. This approach increased the chances of successful detection. For each dictionary, I set up an ArUco detector with carefully tuned parameters. These parameters were adjusted to optimize marker detection, considering factors like marker size and image quality. Upon successful detection of a marker, I estimated its pose (position and orientation) relative to the camera. Using this pose information, I projected the 3D cube onto the 2D image plane, creating the augmented reality effect. I then drew the cube and coordinate axes on the image to visualize the augmented reality result. The cube appeared as if it were sitting on top of my printed AprilTag, demonstrating the successful implementation of the AR technique. Finally, I displayed the resulting image using Colab's cv2underscoreimwrite function, allowing me to save the augmented reality effect directly in the notebook. Throughout this process, I encountered and overcame several challenges. The need to crop the image to remove my face, ensuring the AprilTag was clearly visible, and fine-tuning the detection parameters were all crucial steps in achieving a successful result. This implementation demonstrated computer vision techniques, augmented reality principles, and the practical application of these concepts using OpenCV and Google Colab.

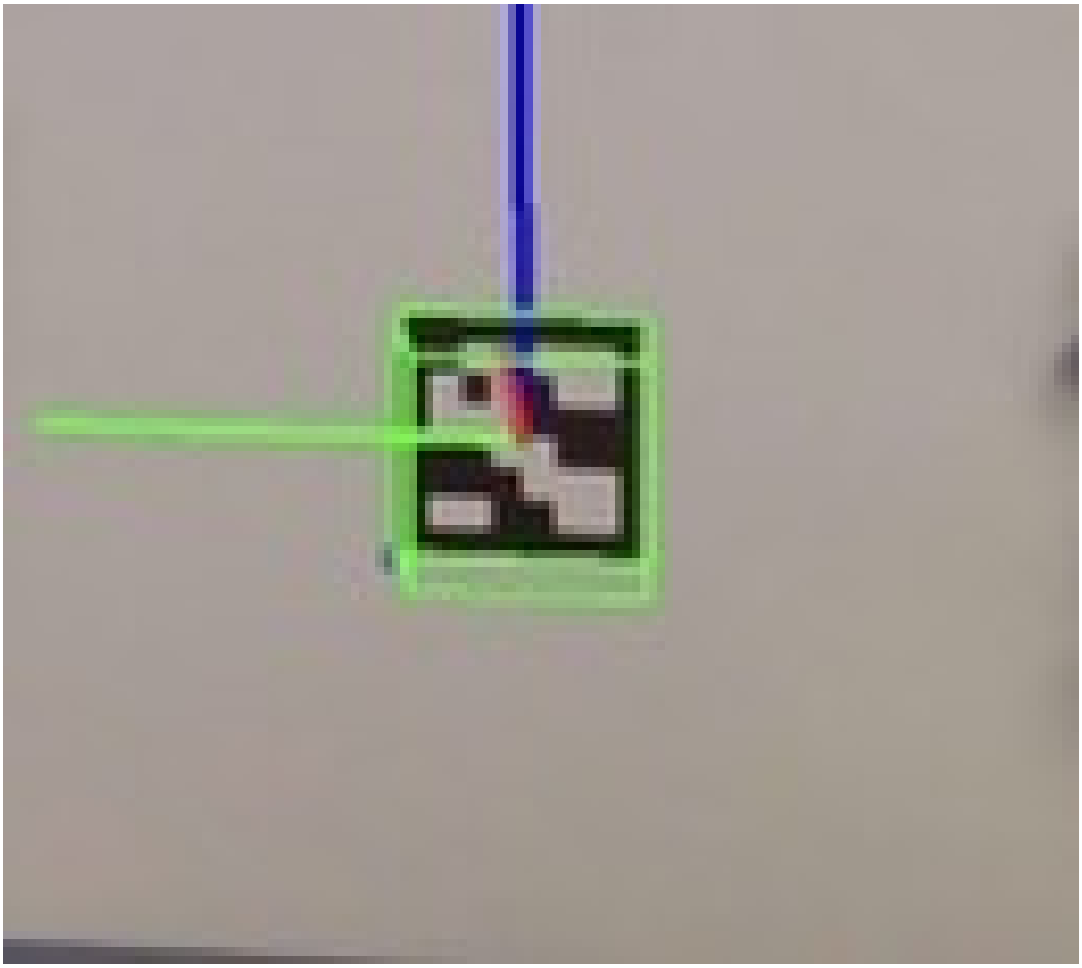


Figure 4: Projected Pyramid of April tag printed on white paper