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

Akanksha Murali (am14013)

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) **2**
Part A (1pt) . . . 2
Part B (1pt) . . . 3

Task 2. Deep Learning with Fasion-MNIST (5pt) **4**
Part A (2pt) . . . 4
Part B (3pt) . . . 4

Task 3 Camera Calibration (3pt) **6**

Task 4 Tag-based Augmented Reality (5pt) **7**

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`

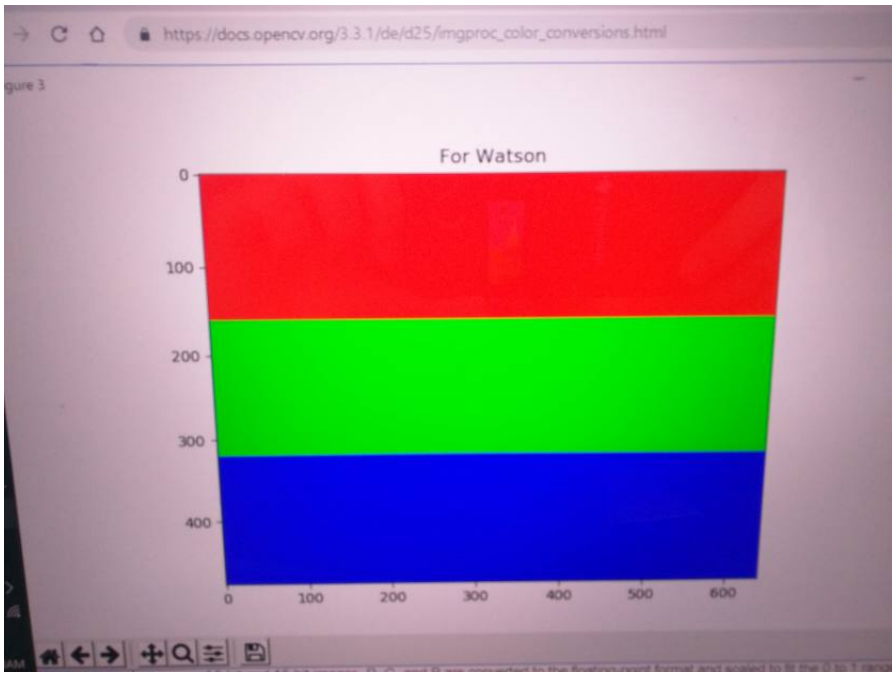


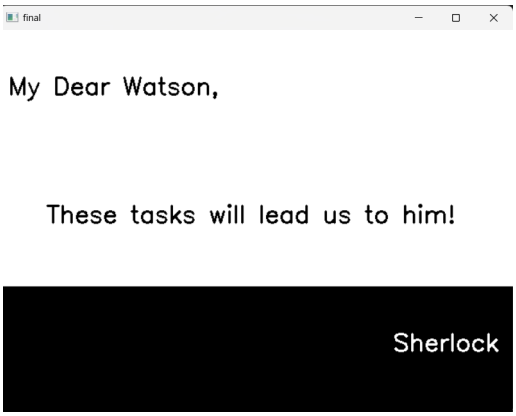
Figure 1: The Secret Message Left by Detective Sherlock

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:

You can use this code snippet to include a picture



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 code adds the image to itself ten times so that the following operations when performed results in the whole decoded message appearing at once. The code then converts the given image from BGR to HSV color space using BGR2HSV attribute and then, treats the resultant image as a BGR image and converts it to gray-scale using BGR2GRAY. The image is converted to HSV to obtain a uniform background, changing that to a gray-scale image helps in efficient threshold operation. The correct threshold value, in this case 230 results in the image being decoded.

In an alternate version of the code the same operations are done but the threshold operation is driven by a slider to ease the tuning making the hit and try method converge faster to decode the message.

Code: Task1.py

Alertnate: Task1alternate.py

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

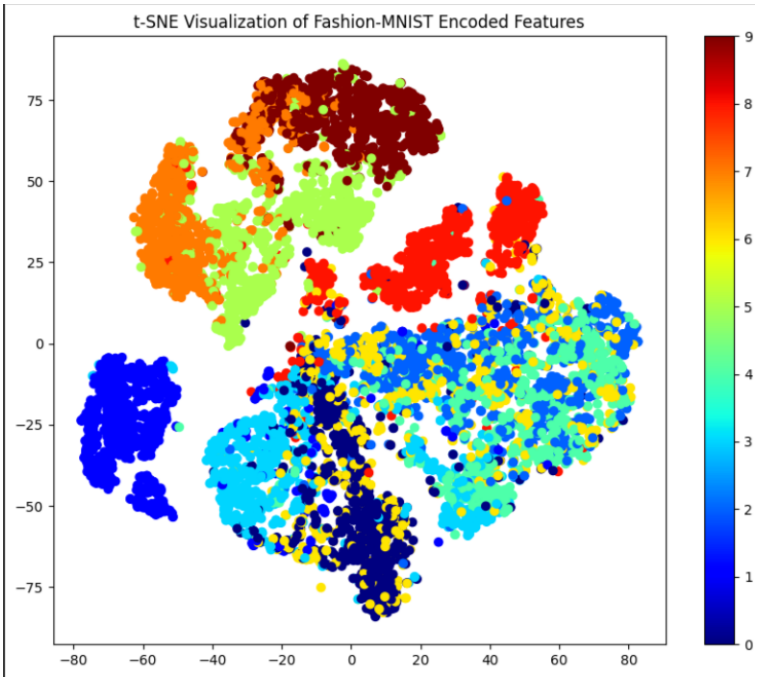
Given the Fasion-MNIST dataset, perform the following task:

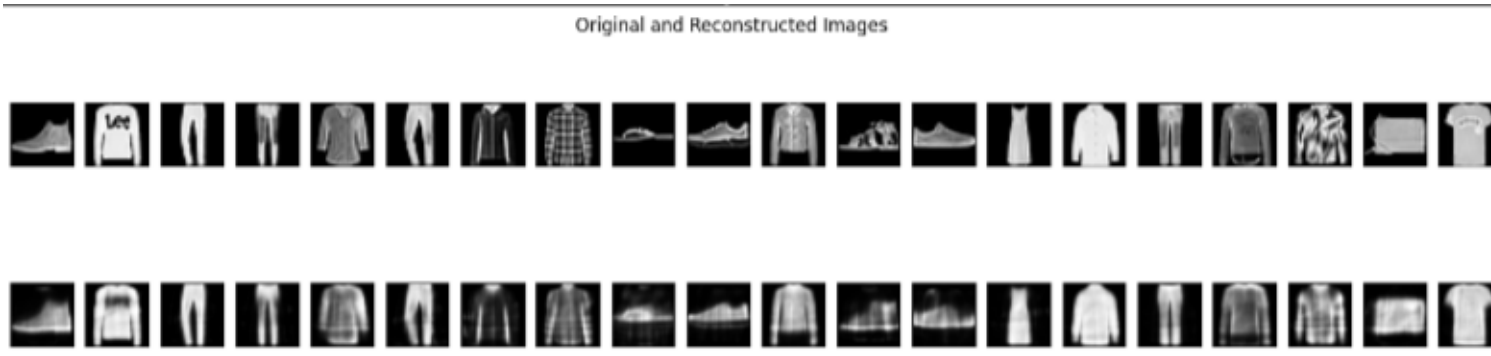
Part A (2pt)

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:
Code: Task2.ipynb

An autoencoder was trained on the database. The below image compares 20 input images at least 1 from each class with the compressed image output from the autoencoder. The encoder compressed the images. The compressed images are taken and its dimension is reduced to 2 with the help of t-SNE and distribution is visualized.



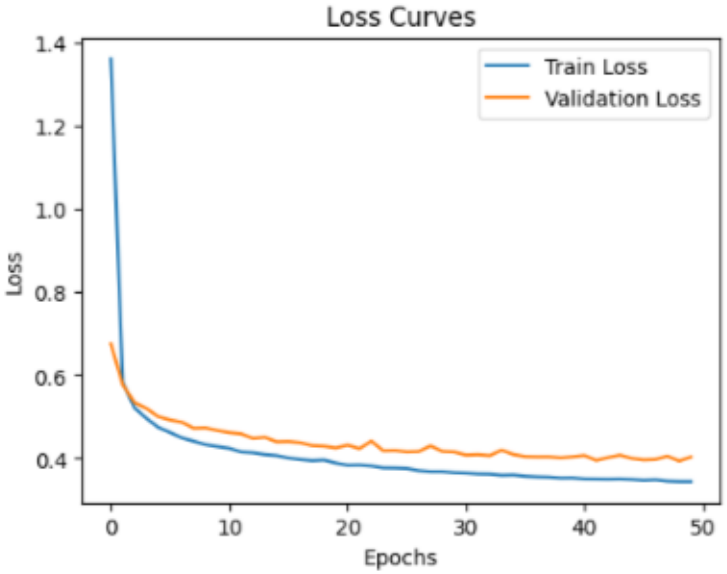


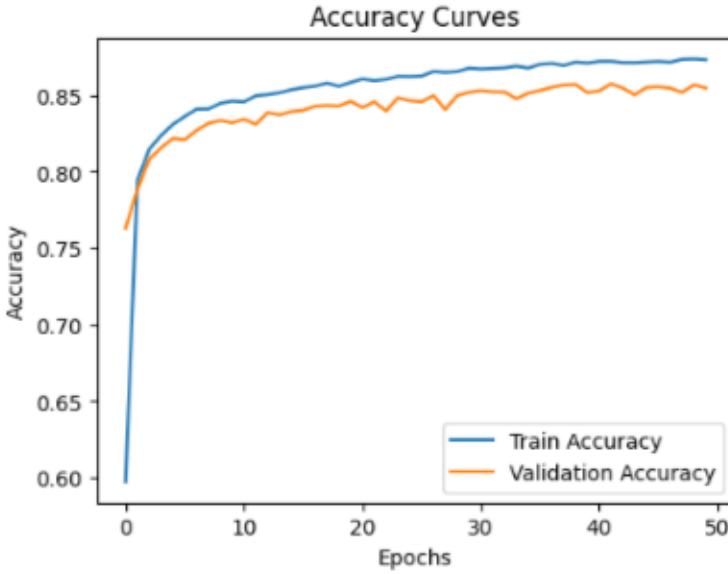
Part B (3pt)

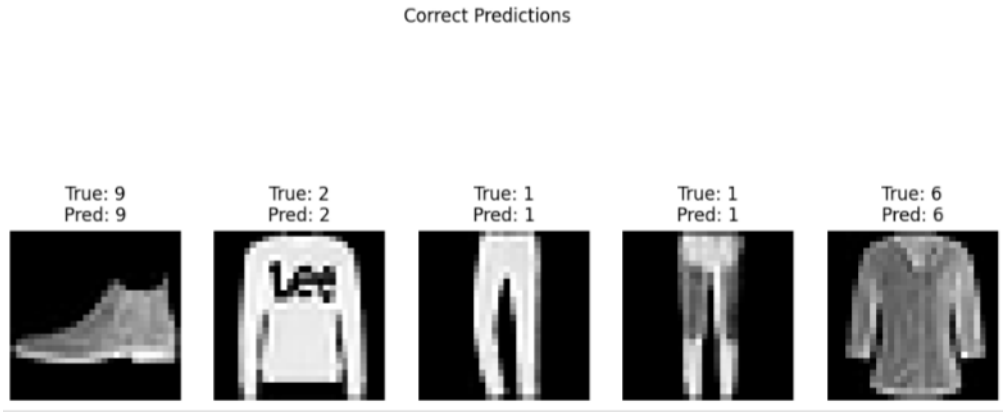
Take the lower-dimensional latent representation produced in Part A and **train** a supervised classifier using these features. **Visualize** the loss and accuracy curves during the training process for both the training and testing datasets. Discuss your observations on the behavior of both curves. 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:
The training loss decreases as the model minimizes the error in predictions. The validation loss also decreases indicating the model is learning effectively. The training accuracy starts lower in the early epochs and increases as the classifier learns patterns. The validation accuracy follows a similar pattern but fluctuates more than the training accuracy. Both the training and validation accuracy seem to be converging toward a higher value indicating that the model is generalizing well. The images below show 5 correct and 5 incorrect predictions. The test accuracy for this run is 85.48







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:
The OpenCV camera calibration was utilized. The calibration used a chessboard pattern. The edges were extracted and used as points from the image coordinate and the grid pattern was leveraged to get the world coordinates. Once we have the image and the world coordinates, we can find the matrix and decompose it to obtain the Camera's intrinsic and distortion coefficients. Reasons for differences between 1m and 2-3m images:

- As the distance between the camera and the chessboard increases the field of view captured by the camera changes This affects the pixel locations of the calibration points in the image, which affects the K matrix.
- Geometric scaling affects the accuracy of corner detection and reprojection error which influences the K matric and the distortion coefficients.
- Optical aberrations can vary with the focal distance of the camera which affects the K matrix.
- Distortion is larger for the close distance calibration, as it is more pronounced at close range.
- Pixel distribution affects the K matrix. At 1m distance, there are fewer pixels covering each calibration square, at 2-3m distance there are more pixels covering each calibration square.

The following are the matrices and one output image for 1m and 2-3m

Code: Calibration-images-close>Task3-close.py

Calibration-images-far>Task3-far.py

```
Camera matrix :

[[649.4555071    0.         364.01713423]
 [  0.         649.84535812 371.88444178]
 [  0.           0.           1.        ]]


dist :

[[ 0.12435873 -0.28029712  0.002117   -0.00041291  0.2161495 ]]
```

```
Camera matrix :

[[865.86615578   0.         385.9513426 ]
 [  0.         869.41603111 366.91819799]
 [  0.           0.           1.        ]]


dist :

[[ 0.27093504 -0.89640043  0.0020681   0.00327642  1.3435655 ]]
```

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.

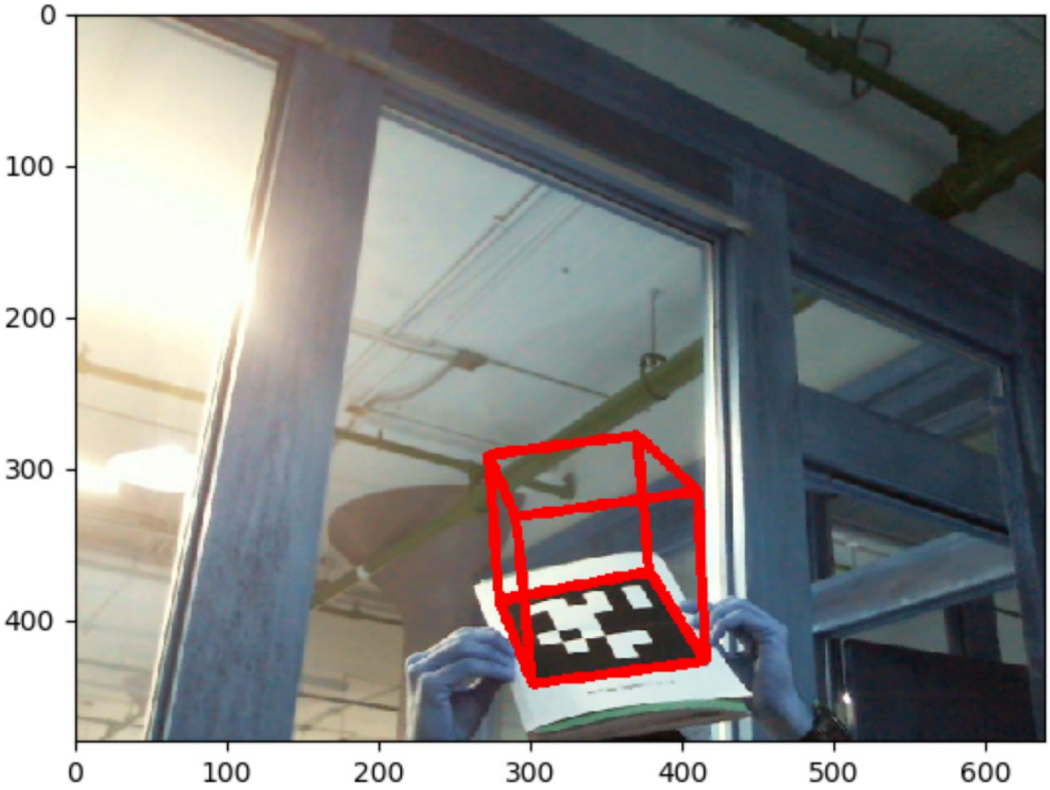


Figure 2: Projected Pyramid on checkerboard

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 detects ArUco markers in an image, estimates their 3D pose, and overlays a 3D cube on the detected marker. The first step is loading and resizing the image, converting it to grayscale for easier marker detection. The ArUco marker is detected using OpenCV's predefined dictionary. If markers are found, the code defines the camera's intrinsic parameters and distortion coefficients, which are used for pose estimation. Using solvePnP, the rotation and translation vectors of the marker are computed, representing its orientation and position in 3D space. A 3D cube is projected onto the marker using cv2.projectPoints and drawn on the image with colored lines. The result is displayed, showing the marker with a 3D cube overlay. Code: Task4.py