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

Harry Wang (cw4418)

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`

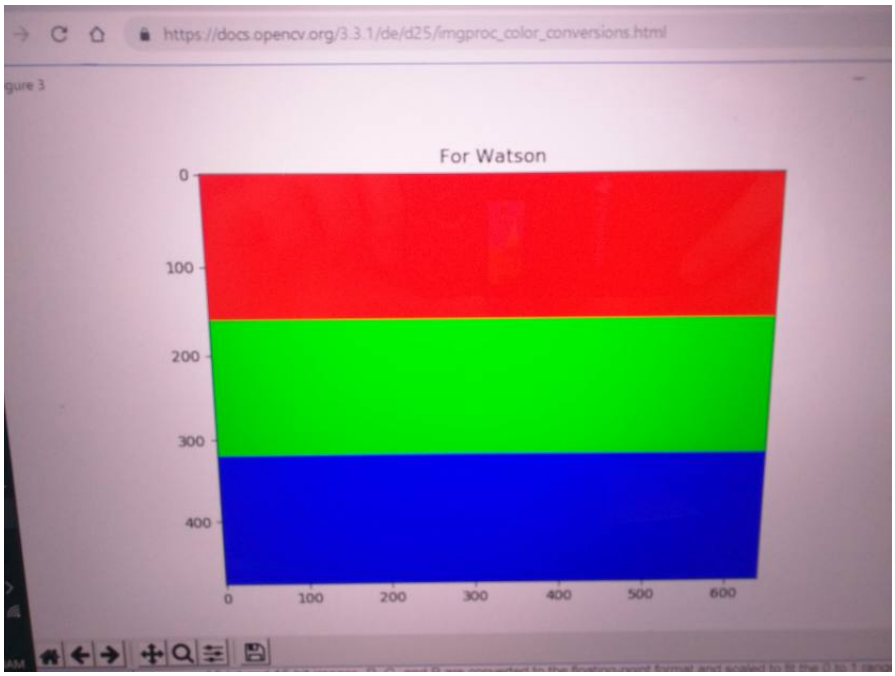
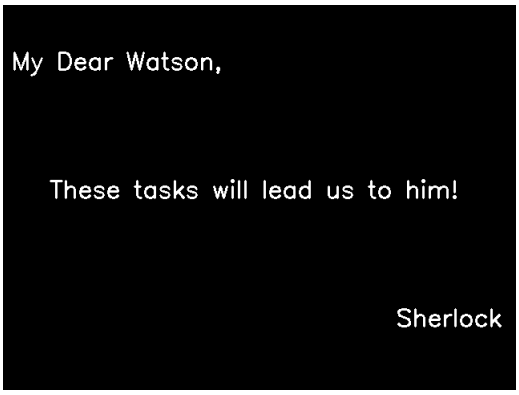


Figure 1: The Secret Message Left by Detective Sherlock

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:
I first converted the original image into gray scale, then divided the image into 3 sections with equal heights, where histogram equalization is applied to each section to enhance contrast and bring out the hidden message. The code can be found at (https://github.com/harrygugu/HwFiles.git). To replicate the result, run decode.py and the resulting image (grayscale_message.png) should be created in the data folder.

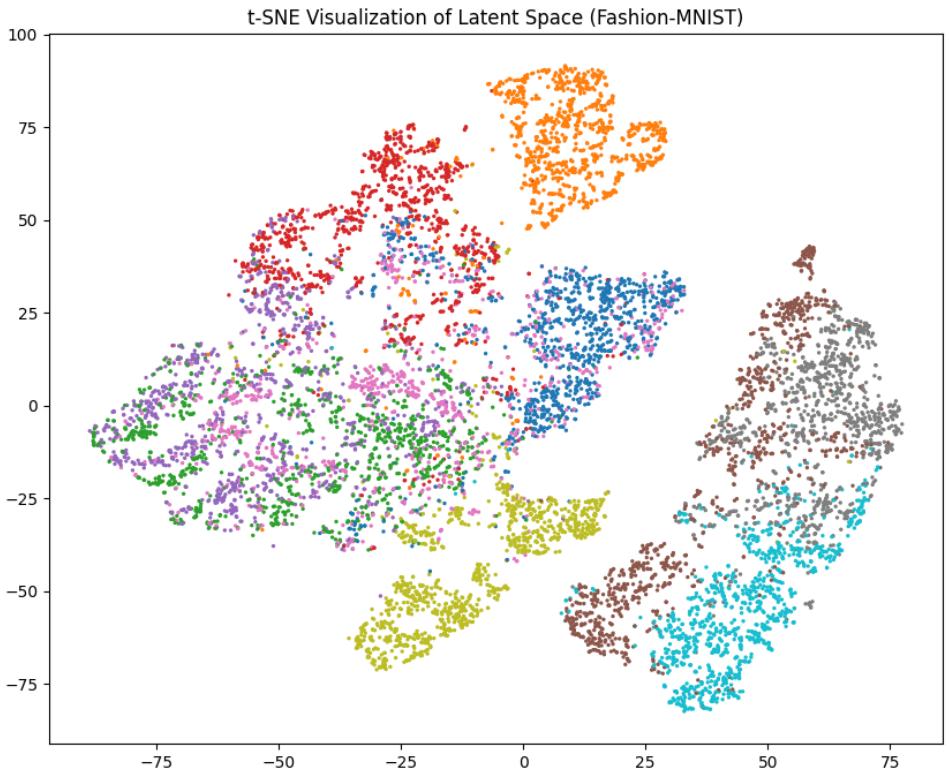
Task 2. Deep Learning with Fasion-MNIST (5pt)

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:

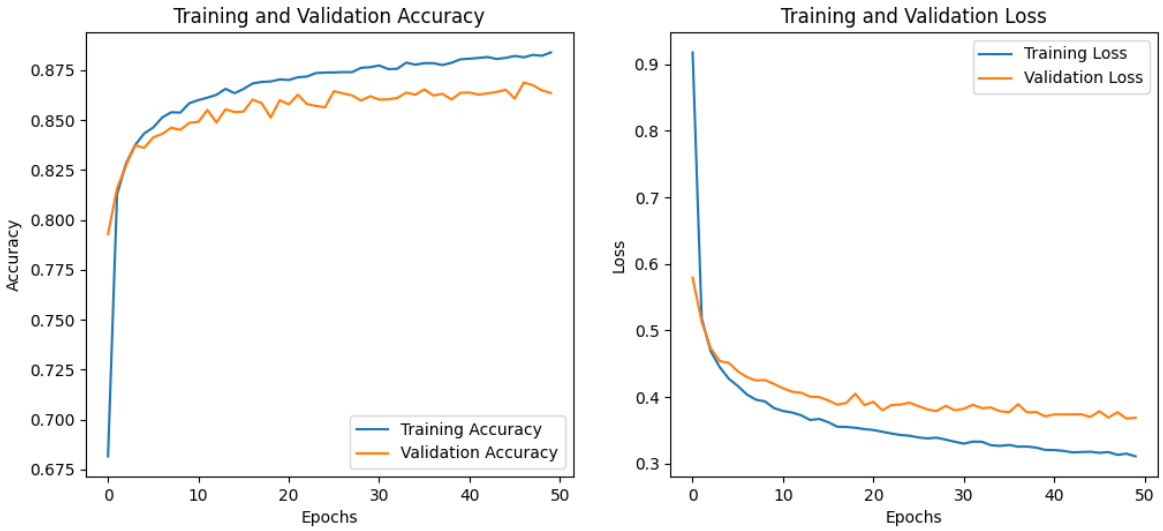


Part B (3pt)

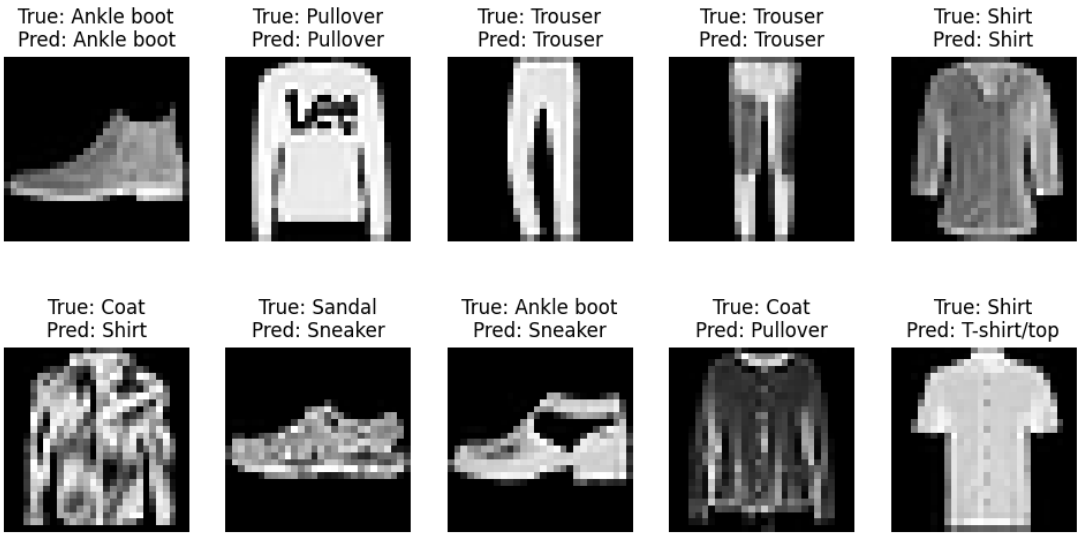
Take the lower-dimensional latent representation produced in Part A and **train** a supervised classifier using these features. **Visualize** the loss and accuracy curves during the training process for both the training and testing datasets. Discuss your observations on the behavior of both curves. 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:

Training both the autoencoder and the classifier 50 epochs, the model achieves a 86.36% accuracy overall, whereas the training curve performs slightly better than the validation curve as expected. Examples of the deployment results on the testing dataset is shown below, where the upper row shows 5 correct predictions while the bot-



tom row shows 5 incorrect predictions. To replicate the result, run fashion_classifier.py and the resulting image (grayscale_message.png) should be created in the data folder.



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:

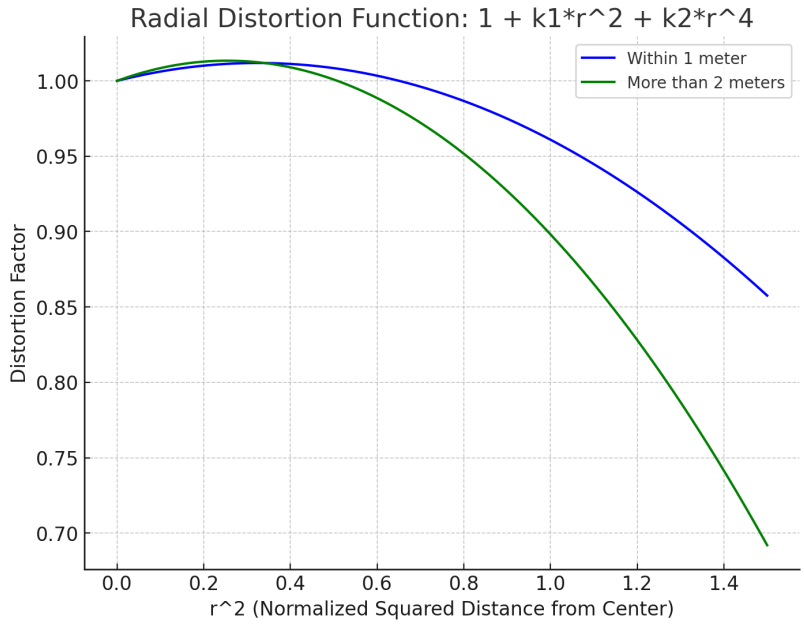
For the first set, the intrinsic parameters (K matrix) is: $\begin{bmatrix} 2.7238 \times 10^3 & 0 & 2.0155 \times 10^3 \\ 0 & 2.7244 \times 10^3 & 1.4770 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$

The distortion coefficients (k1 and k2) are: $[0.0728, -0.1118]$

For the second set, the intrinsic parameters (K matrix) is: $\begin{bmatrix} 2.6698 \times 10^3 & 0 & 1.9983 \times 10^3 \\ 0 & 2.6596 \times 10^3 & 1.4346 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$

The distortion coefficients (k1 and k2) are: $[0.1052, -0.2071]$

The focal length of the first set is slightly larger than the second set possibly because the calibration pattern occupies more space within the image the closer it gets, and thus can be more prone to lens distortion. As can be seen from the graph below, by plotting the distortion curve ($D(r^2) = 1 + k_1 r^2 + k_2 r^4$) for both sets, it can be observed that the first set has a slightly larger distortion effect, as it is closer.



To replicate the result, run calibration.py to compute and display K and the distortion coefficients.

Name: Harry Wang, netID: cw4418

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.

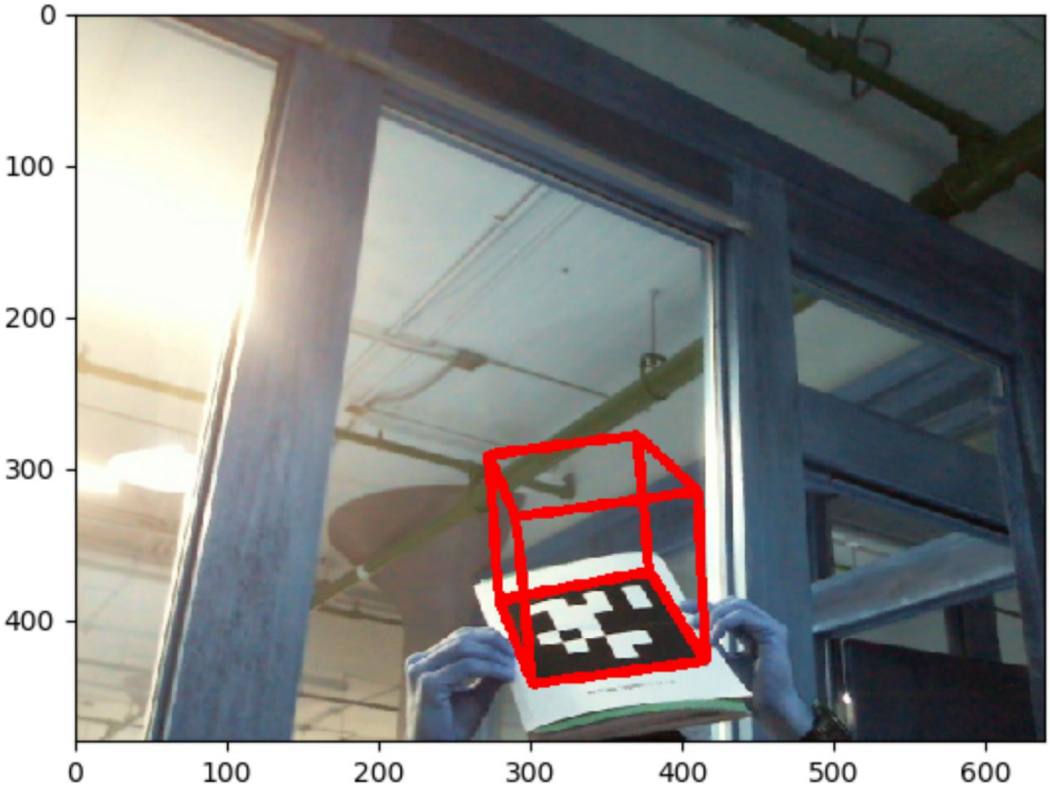


Figure 2: Projected Pyramid on checkerboard

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 logic of the code for the implementation of the method goes as follows: First, K and distortion coefficients are initialized with calibration results using the code from the previous section on a laptop camera. Second, an image containing AprilTags is loaded and converted to grayscale for processing, where the pyAprilTag library is used to detect AprilTags in the image. Third, for each detected AprilTag, its 3D pose (position and orientation) is calculated using the solvePnP method and projects a cube onto the detected tag in the image. Finally, the image with the projected cube drawn on the tag is displayed and saved. To replicate the result, run cube_generator.py to display cubes attached on top of the detected AprilTags. Two of the results are shown below: