Homework 4

Due: December 11, 2020

Imitation Learning is a form of Machine Learning where the model imitates the behavior of a user and recreates the steps or actions taken. One approach to imitation learning is training a classifier to predict action based on current observation. In this assignment, you will use Imitation Learning to make a robot follow white line. First, you will collect demonstration data by manually controlling the robot using keyboard. Then, you will train a Convolutional Neural Network as action classifier given the image observation. Training a deep neural network from scratch often requires large amount of data and is usually not available for tasks such as ours. You will use a technique called Transfer Learning where you fine-tune a pretrained network for your task.

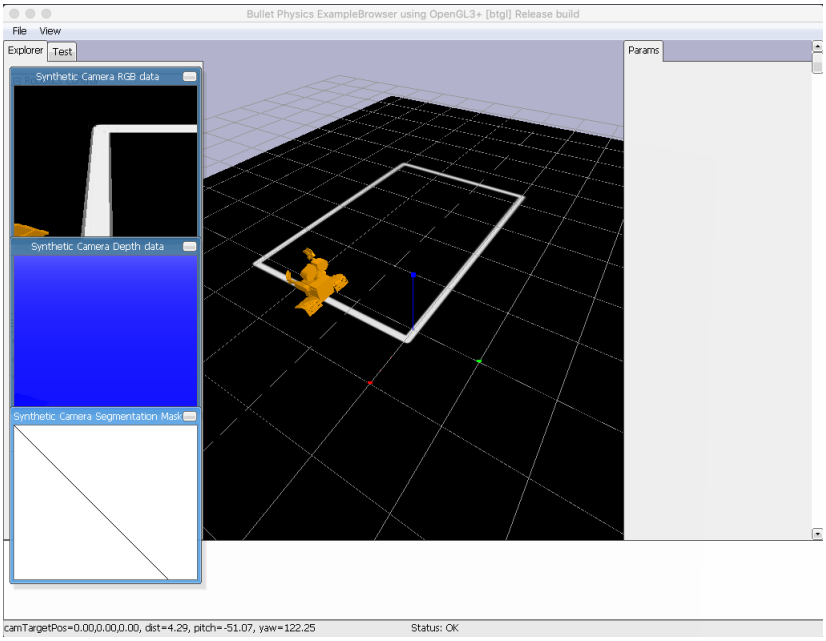
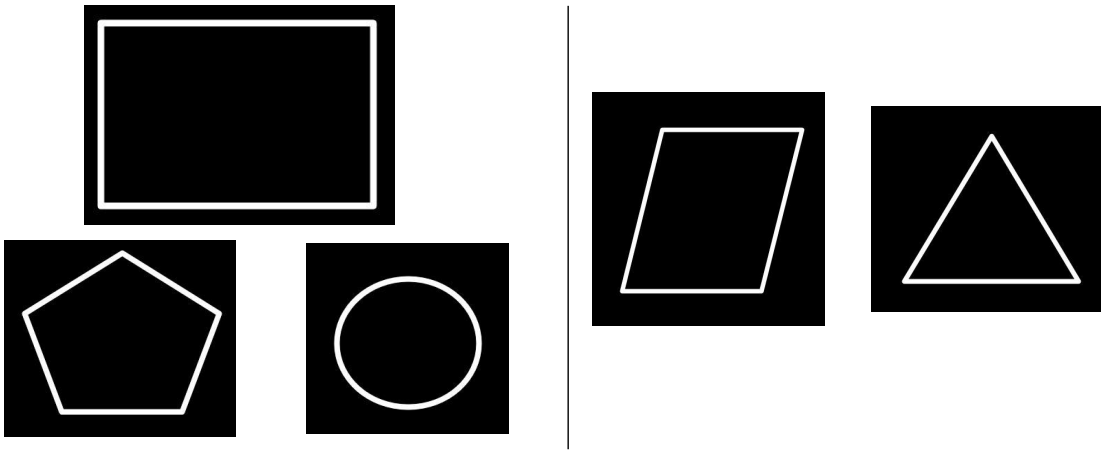


Figure 1: Simulation environment

You will also apply homogeneous transformations to move robot given action and implement a non-learning rule based line following algorithm. Figure 1 shown an image of the simulation environment. Figure 2 shows the training and testing maps.



(a) Training maps (b) Testing maps

Figure 2: Training and testing maps to be used. Train maps can be found in *maps/train* directory. Test maps can be found in *maps/test* directory

Getting Started

After unzipping the homework zip, you will install a python virtual environment inside homework directory. If your default python interpreter is not python3, use the command `python3` and `pip3` instead of `python` and `pip`. To install dependencies, first create a python virtual environment:

```
python -m venv env
```

Then, activate your environment by running:

```
source env/bin/activate // Linux or OSX
env\Scripts\activate.bat // Windows
```

Once you have activated your environment, install dependencies by running the following command:

```
pip install -r requirements.txt
```

Please do not introduce any other dependencies/packages without taking prior permission from TAs.

Problem 1: Moving robot using keyboard (20 points)

In this problem, you will write code to move the robot given an action. We have provided four different actions: *FORWARD*, *BACKWARD*, *LEFT* and *RIGHT* (see *sim.Action*). *FORWARD* or *BACKWARD* should move the robot forward or backward by *posDelta* amount (see *WallESim.move_robot* method inside *sim.py* file). *RIGHT* or *LEFT* should rotate the robot clockwise or anticlockwise by *rotDelta* amount. Please complete *WallESim.move_robot* method inside *sim.py* file. You should be able to test your code by running *python sim.py*. To trigger each action, you can use keyboard arrow keys, i.e. UP arrow key for *FORWARD*, DOWN arrow key for *BACKWARD*, LEFT arrow key for *LEFT* and RIGHT arrow key for *RIGHT*.

Problem 2: Rule based line following method (20 points)

To follow white line, the robot only needs to figure whether the white line is on left, front or right side of the robot. Following is a simple rule based line following method:

(a) Capture an image of the robot view

(b) Convert image to gray-scale

(c) Pick up a horizontal row of the image

(d, e) Threshold the intensity values of the selected row.

(f) The threshold-ed region denotes the area where white line is present. Find the mid point of this region.

We also visualize this entire algorithm in Figure 3. Once you know where the white line is present in the scene, you can decide what action to take. For example, in the Figure 3, since white line is present on the right side of the scene, robot should turn right.

Complete the class *ImgProcessingActionPredictor* inside *prediction.py* file. Currently the method *ImgProcessingActionPredictor.predict_action* always returns the FORWARD action. Test your implementation by running *python prediction.py −−map_path maps/test/map1*. In your **report**, briefly summarize the algorithm you used (if diffrent from above) and write down the performance of this algorithm (number of landmarks reached) on the two test maps (*maps/test/map1* and *maps/test/map2*).

Grading for this part will be done as following:

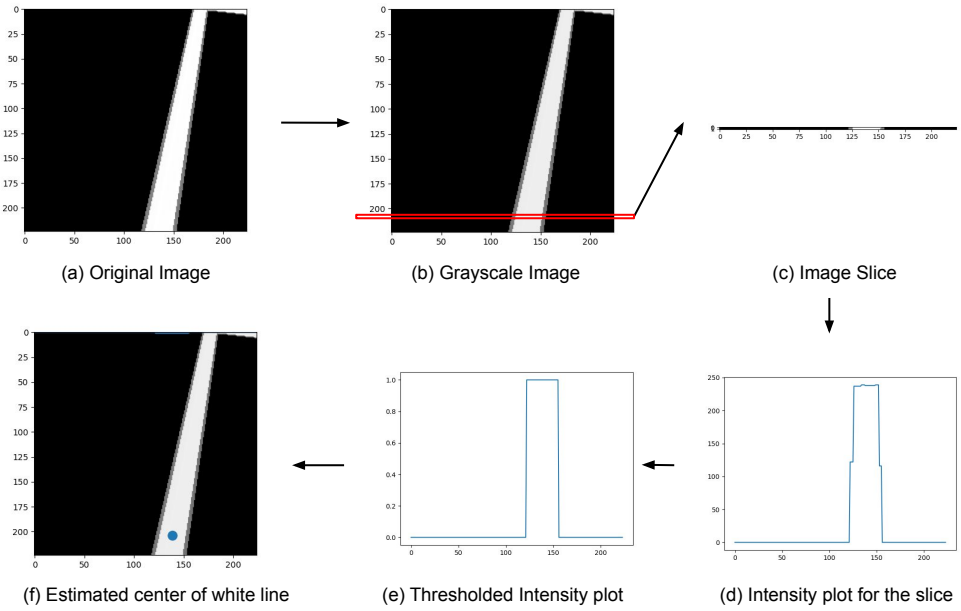


Figure 3: Rule based algorithm pipeline for line following bot. (b) The red rectangle denotes the chosen row of the image. (f) the blue dot denotes the estimated mid point of white region

- (10 points) Correct logic implementation
- (10 points) Algorithm performance on test maps. Score will be proportional to the number of landmarks reached on the following maps:
  - *maps/test/map1* and *maps/test/map2*
  - One unseen test map

Problem 3: Imitation learning based line following method (60 points)

Collect data (10 points)

In this problem, you will manually move the robot using keyboard such that it follows the line. While doing so, you will also collect this demonstration data so that it can later be used for training your classification model. Collect demonstration data by running *python collect_data.py −−map_path maps/train/map1*. This script will automatically start collecting data in *dataset/* folder as you move the robot around. See Figure 4 for more details on data collection.

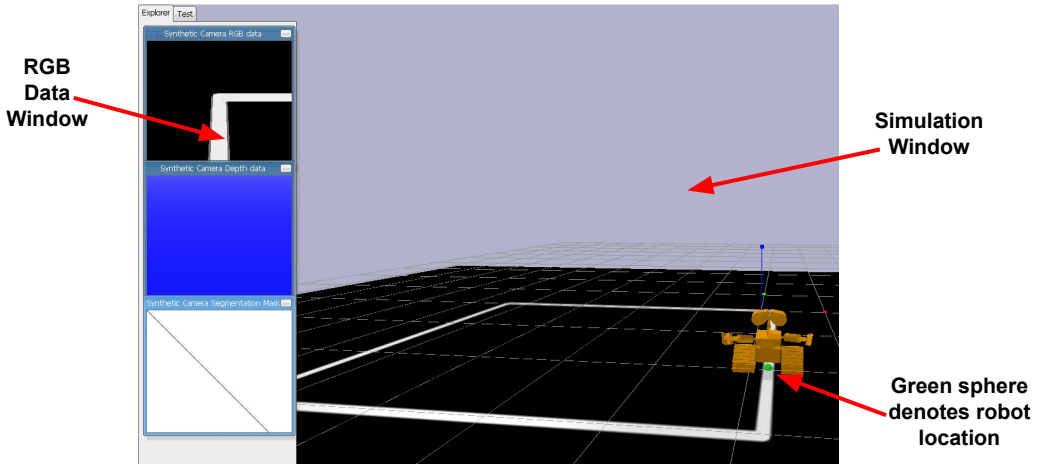


Figure 4: During data collection, avoid looking at the *simulation window*. Try to perform action just by looking at the *RGB Data Window*. This is necessary because the classification model will only have access to this image for predicting an action. The green sphere denotes the robot location and you should try to keep it on the white line.

As you will make the robot follow the line by manually pressing arrow keys on keyboard, image for a corresponding action will be saved in *dataset/<ACTION>* folder. For collecting data, you should only use the training maps - *maps/train*. For problem 3, you only need to collect data on *maps/train/map1*, *maps/train/map2* and *maps/train/map5*. **Do not use the testing maps** - as you will later use them for validating the performance of a trained model.

For grading, you will need to **submit the dataset folder**. Please see *Submission instructions and checklist* section for submission details.

Train an image classification model (20 points)

In this problem, you will train an image classification model where the model will predict an action given the image that the robot should take in order to follow the line. For training the model, you will use the collected demonstration data from the previous part. Training a deep learning classification model requires a lot more data than what a single person can possibly collect. To mitigate this problem, we will use transfer learning paradigm, where we fine-tune a pretrained model for a given task. In this problem, you will fine-tune a ResNet18 model pretrained on ImageNet dataset. Please have a look at pytorch tutorial on transfer learning, especially the part where they freeze the convolutional layers and add new new fully connected layers for fine-tuning.

Complete *train.py* file by adding code for

- Loading and splitting the dataset into training and validation set
- Loading the pretrained ResNet18 model
- Making appropriate changes to the model for fine-tuning, i.e. freeze the convolution layers, remove the fully connected layers and add new fully connected layers as per required number of output classes
- Prepare loss function and optimizer
- Train and validate the model for certain number of epochs

We will grade this part on correct code implementation for the bullet points above.

Predict actions to follow line (10 points)

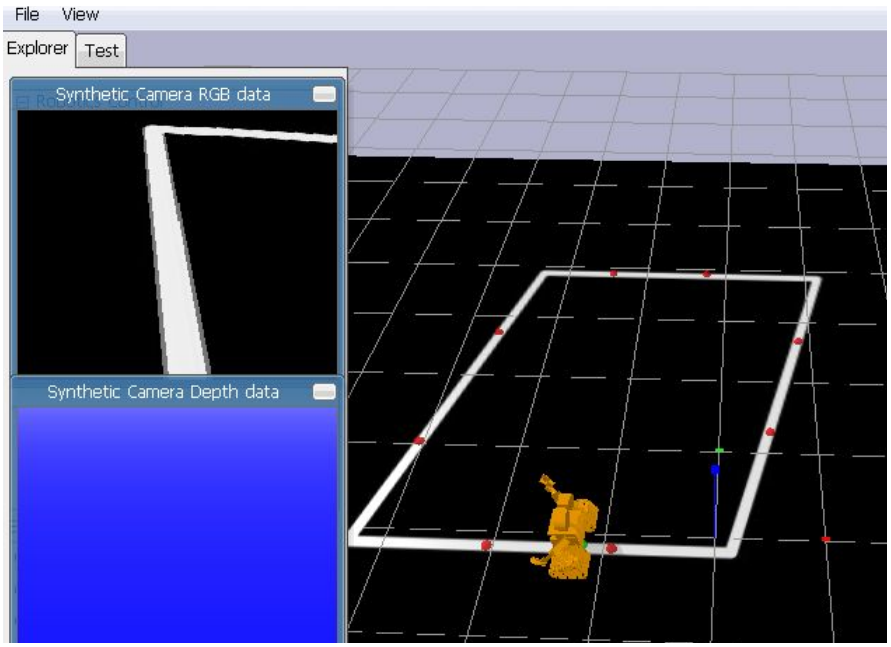


Figure 5: Landmark visualization using red spheres

Once your model is trained, complete the class *ImitationLearningActionPredictor* inside *prediction.py* which has two methods. Inside *ImitationLearningActionPredictor.__init__* method, you should load your trained model and initialize any transforms that you might have used during training. Inside *ImitationLearningActionPredictor.predict_action* method, use the input image and model to predict action. Once

completed, you can test your implementation by running the command: *python prediction.py −−use_imitation_learning −−map_path maps/test/map1 −−model_path following_model.pth*. The script will use predicted actions to move the robot. For each test map, we have provided landmarks, which are visualized as red flickering spheres (see Figure 5). Once a robot is close enough to a landmark, we mark the landmark as reached. The script will stop once the robot reaches all the landmarks.

Note that the flickering of red landmark spheres is an intended behavior and is implemented so that the input image to the model does not have these spheres.

We will grade this part on code implementation for *ImitationLearningActionPredictor* class.

Model Performance (20 points)

We will evaluate the model performance by testing your trained model on the following maps:

- *maps/test/map1*, *maps/test/map2*
- one unseen map

Your score will be proportional to number of landmarks reached on the three maps above. You will need to **submit the trained model checkpoint** for this part. Please see *Submission instructions and checklist* section for submission details.

Extra Credits: Improve your model (10 points)

All maps in Problem 3 only had two actions, i.e., RIGHT and FORWARD. As extra credits, you will make your robot follow more difficult maps, i.e., maps with both LEFT and RIGHT actions in addition to the original maps. Figure 6 shows the training and testing maps to be used for this part.

Grading of this part will be based on the performance of your trained model on the following maps:

- *maps/test/map1*, *maps/test/map2*
- one unseen map from Problem 3
- one more unseen map with both left and right turns.

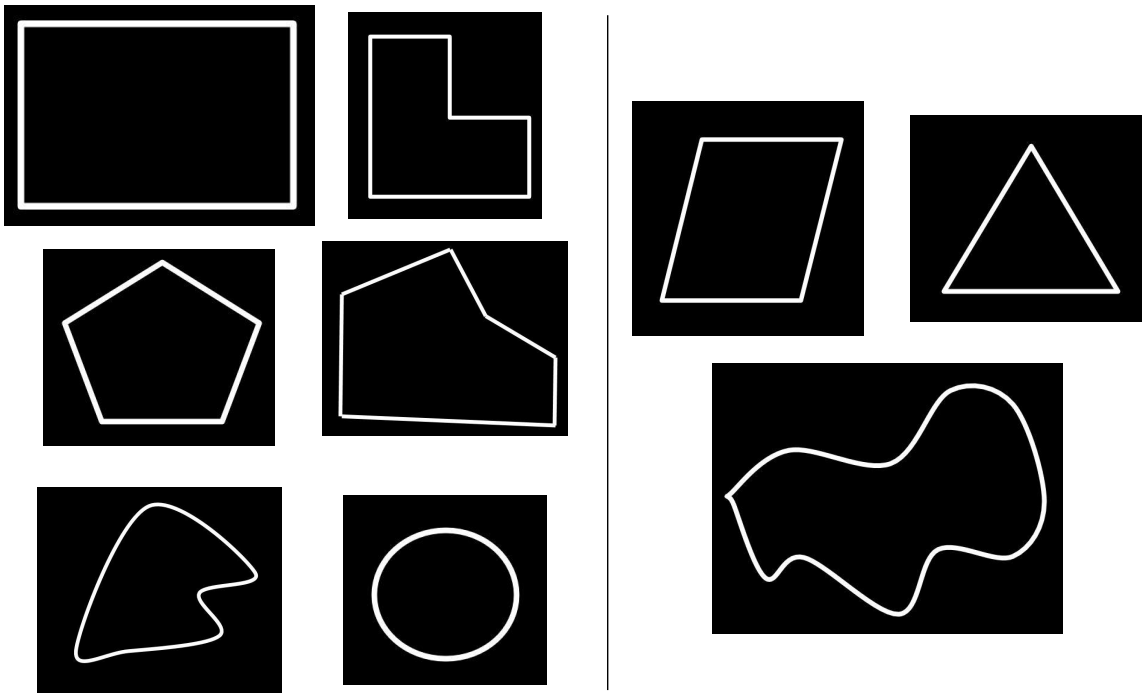


Figure 6: Training and testing maps to be used for extra credits.

Your score will be proportional to number of landmarks reached on the four maps above. You will need to **submit the trained model checkpoint** for this part. Also, **briefly summarize the steps required to make the model work in your report**. Please see *Submission instructions and checklist* section for submission details.

Submission instructions and checklist

For submission, you will need to create a zip containing your dataset folder and trained model checkpoints and upload this zip to google drive. You will also need to create a zip containing your code, report and a url.txt, and upload it to the courseworks.

Dataset and Model Checkpoint(s)

Create a folder with name *dataset_model_checkpoints*. Place your dataset folder inside it. Also place the model checkpoint from Problem 3 inside it as *following_model.pth*. Place the model checkpoint from extra credits inside it as *extra_following_model.pth*. If you have used the same model as Problem 3 for extra credits, make a copy of it and place it as *extra_following_model.pth* inside it. Your final folder should look like this:

```
dataset_model_checkpoints
|-- dataset/
|-- following_model.pth
|-- extra_following_model.pth
```

Create a zip of *dataset_model_checkpoints*. Upload the zip to your personal google drive account (**not Lionmail**). Generate shareable link such that **anyone with the link can view the file**. For testing, open the link in Incognito Mode / Private window. If done properly, you should be able to view the zip without logging in.

You should also make sure that the *Modified* date is visible. For this, click on burger button (three vertical dots) on top-right. Click on *Details*. You should be able to see the *Modified* date (see Figure 7).

Place the shareable link generated above inside url.txt file. Place this file inside your HW4 root.

Code and Report

- **Code**: Remove virtual environment folder *env*, *dataset* folder and any other folder that you might have introduced (for example: .git, .vscode, test python files etc).
- **Report**: Add solutions to problem 2 and extra credits in your report. Place it in hw4 root as uni_hw4_report.pdf.
- **url.txt**: Ensure that you have placed the correct url from above inside it. Place url.txt file at hw4 root.

Zip the hw4 root directory and upload it to courseworks as uni_hw4.zip

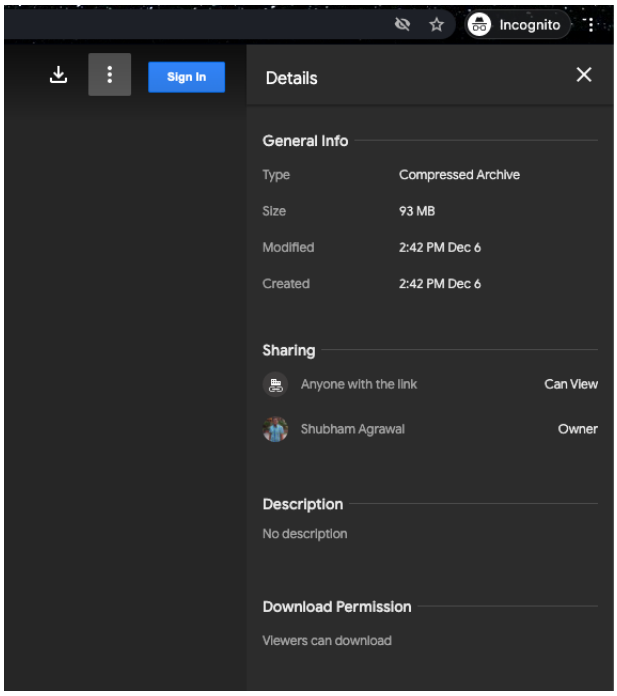


Figure 7: dataset_model_checkpoints.zip URL: should be accessible in Incognito / Private window. Modified date should be visible.