Homework 4

Due: April 28, 2022

Introduction

Input and action representations often play a decisive role in the success of learning based robotic systems. In this homework, we are going to explore Visual Affordance (also called Spatial Action Map): a state-of-the-art action representation for robotic pack-and-place that played critical role for the MIT-Princeton team's victory in the 2017 Amazon Robotics Challenge.

In **Problem 1 and 2**, you will implement and train a Visual Affordance model with manually labeled supervision. In **Problem 3**, you will implement Action Regression, an alternative action representation and explore its difference with Visual Affordance. In **Problem 4**, you will improve the existing Visual Affordance pipeline with the method of your choice.

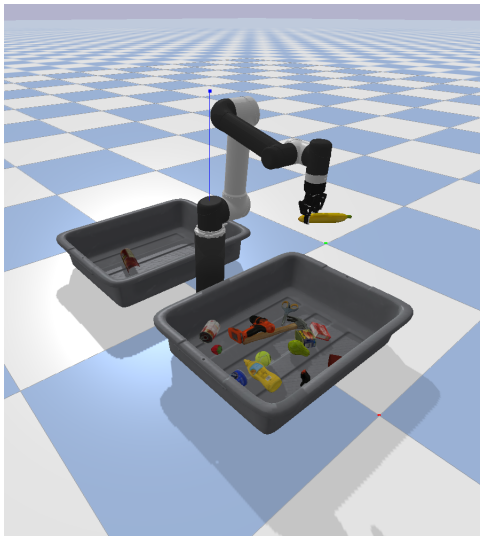

Figure 1: UR5 robot moving an object to another bin

You will be directly editing the provided python files. For verbal questions, compile your answers in hw4_report.pdf.

Apart from the code and report, you are also required to submit videos for all problems. These videos are screen recordings of your completed tasks. You can use any screen recording software as you desire. One choice is OBS or QuickTime Player for Mac users.

For more details on submission, please see Section Submission instructions.

Getting started

For this homework, we will install python interpreter and dependencies using miniforge (also works for Apple Silicon, comparing to miniconda). Please download the corresponding Mambaforge-OS-arch.sh/exe file and follow the installation instructions. You can skip this part if you already installed mamba in WH3.

After installation and initialization (i.e. conda init), launch a **new** terminal and run

```
mamba env create -f environment_gpu.yaml
```

if you have an Nvidia GPU on your computer, or

```
mamba env create -f environment_cpu.yaml
```

otherwise, inside the unzipped homework zip. This will create a new environment named "comsw4733_hw4", which can be activated by running

```
mamba activate comsw4733_hw4
```

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

This assignment uses PyBullet simulation engine extensively and hence, we recommend you to read the *Introduction* section in PyBullet API documentation to get some working knowledge of the engine. To interact with the simulation GUI, you can change the camera viewpoint by zooming in/out or pressing *ctrl* key and dragging the cursor at the same time. Pressing *g* key will toggle the image windows on left side.

Visual Affordance

In this homework, we are going to make two assumptions:

- 1. The image observations come from a top-down camera from where the entire workspace is visible.

- 2. The action is limited to top-down grasping, where the pose of the gripper is reduced to 3 degrees of freedom (2D translation and 1D rotation).

These assumptions allow us to easily align action with image observations (hence the name spatial-action map). In general, Visual Affordance is defined as a per-pixel value between 0 and 1 that represents whether the pixel (or the action directly mapped to this pixel) is graspable. Using this representation, the transnational degrees of freedom are naturally encoded in the 2D pixel coordinates. To encode the rotational degree of freedom, we rotate the image observation in the opposite direction of gripper rotation before passing it to the network, effectively simulating a wrist-mounted camera that rotates with the gripper.

Problem 1: Supervision data generation (5 points)

In this problem, you will get familiar with the data annotation pipeline and generate training data for the other problems.

(a) Complete the class GraspLabeler (2 points)

Inside *pick_labeler.py*, update the pixel coordinate *self.coord* in the mouse callback *GraspLabeler.callback*, as well as update the gripper orientation *self.angle* in *GraspLabeler.__call__*. Note that in this homework, we follow the OpenCV convention for pixel coordinate and rotation angle (Fig. 2).

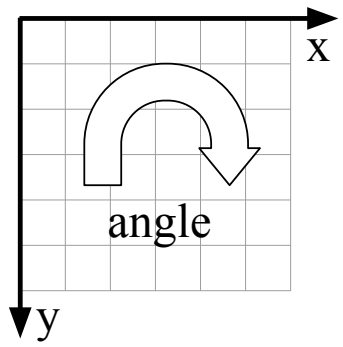

Figure 2: OpenCV pixel coordinate and rotation angle convention

(b) Label training data (3 points)

On your **local compute**, run:

```
python3 pick_labeler.py
```

Figure 3: Labeling GUI. The window might look slightly different across operating systems.

If your *GrapsLabeler* is implemented correctly, the script will launch a PyBullet GUI, then load the robot and objects. A OpenCV window will pop up (Fig. 3), where you can click left mouse button to select grasping location and use A and D keys to rotate the grasping orientation. To confirm the grasp pose, press Q or *Enter*. Note: if the label GUI does not show up, it might be hidden behind other windows (e.g. you might find a blank icon on your MacOS Dock). After confirmation, the robot will try to pick the object using the labeled grasping pose. If the grasping failed, you will be prompted to try again. Any discrepancy between the robot's behavior and the labeling GUI's visualization might indicate that there's a bug in your code.

You will label 5 training objects with 12 attempts each. This usually take around 5 minutes.

Problem 2: Implementing Visual Affordance model training (50 points)

In this problem, you will implement training related features for the Visual Affordance model. In comparison to input and action representations, the neural network architecture usually plays a less important role (comparing to fields like Computer Vision). Therefore, the *AffordanceModel* directly takes the *MiniUNet* architecture from Homework 2 and 3 (except the last layer). However, its training process will be quite different.

(a) AffordanceDataset (10 points)

What does the data & label look like? Any visualizations there?

Read and understand *RGBDataset* in *train.py*. Then, complete *AffordanceDataset* in *affordance_model.py*. In your report, **answer** why do we use *get_gaussian_scoremap* to generate the affordance target, instead of a one-hot pixel image. Hint: if you are not sure, try to train without it after you finished implementating this problem.

(b) Data augmentation (5 points)

Rotations? Or are rotations done in the training part?

Complete *AugumentedDataset* class and *get_center_angle* function in *train.py*. In your report, **explain** in English what transformations are done in *self.aug_pipeline*.

(c) Training loss (5 points)

In *affordance_model.py*, read method *predict* and *get_criterion*. In your report, **answer** what does *nn.BCEWithLogitsLoss* do and what is its advantage comparing to *nn.BCELoss*.

(d) Training (5 points)

If you have a Google Cloud GPU instance, upload your repo (including the data directory) and execute the following command:

```
python3 train.py --model affordance --augmentation
```

However, training on your CPU should also be fine.

If your training pipeline is implemented correctly, this script should train the AffordanceModel for 101 epochs until convergence. On a decent GPU (faster than Nvidia GTX 1070), it should finish in around 1 minute. In your report, **report** your training loss and test(validation) loss. Include a image from data/affordance/training_vis. It should look like Fig. 4.

(e) Grasp prediction (10 points)

In *affordance_model.py* implement AffordanceModel.predict_grasp for both prediction and visualization. It should look like Fig. 5 but it doesn't have to be pixel perfect.

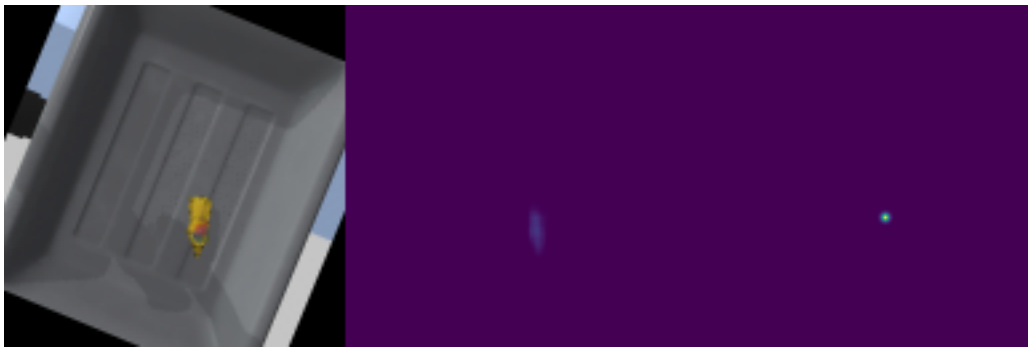

Figure 4: Example for training visualization. From left to right: input, prediction, target.

(f) Evaluation on the training set (10 points)

Download data/affordance/best.ckpt back to your *local computer* and execute the following command:

```
python3 eval.py --model affordance --task pick_training
```

If your prediction code is implemented correctly, it should have a success rate better than 70%. **Record** a video of the terminal and PyBullet GUI. In your report, **report** your success rate and include data/affordance/eval_pick_training_vis/YcbMustardBottle_2.png. It should look similiar to Fig. 5.

(g) Evaluation on all objects (5 points)

Execute:

```
python3 eval.py --model affordance --task empty_bin
```

The script will load 15 objects into a bin, and try to move all of them into another bin within 25 attempts. If your implementation is correct, there should be at most 3 objects left. **Record** a video of the terminal and PyBullet GUI. In your report, **report** how many objects are left. Also **explain** why is this method so sample efficient (can train from only 60 images).

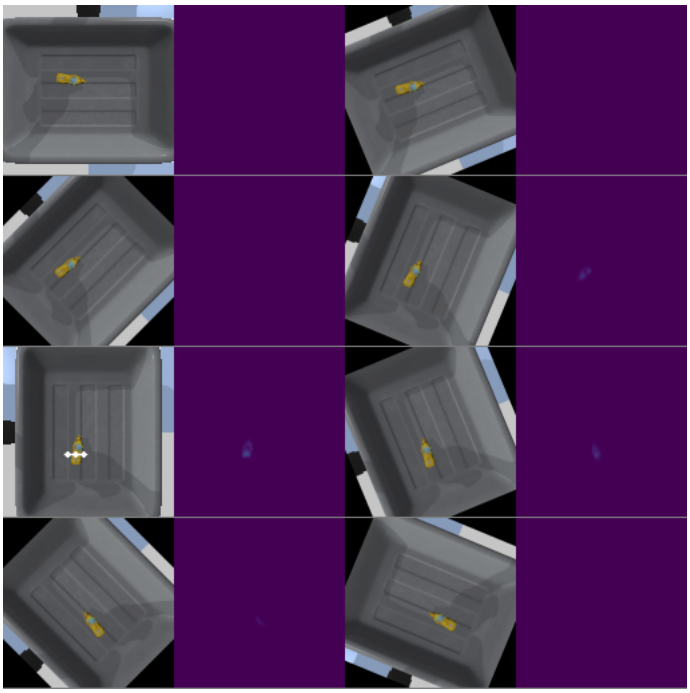

Figure 5: Example prediction visualization.

Problem 3: Alternative method: Action Regression (20 points)

What? Oh, not pixel-wise prediction, but one picture —> one vector?

Instead of using visually aligned representation where the distribution over the action space is discretized, we can also directly output the action as vector (regresion). This is the most common action representation prior to the popularization of visual affordance. More specifically for this homework, we will will predict the vector (x,y,angle) with each dimension normalized to between 0 and 1.

(a) Training (5 points)

In action_regression_model.py, complete *ActionRegressionDataset.__getitem__*, recover_action and *ActionRegressionModel.get_criterion*. Train your model using the command

```
python3 train.py --model action_regression --augmentation
```

In your report, **report** your training loss and test(validation) loss. Include a image from data/action_regression/training_vis.

(b) Evaluation on training objects (5 points)

In action_regression_model.py, complete *ActionRegressionModel.predict_grasp*. Execute:

```
python3 eval.py --model action_regression --task pick_training
```

Record a video of the terminal and PyBullet GUI. In your report, **report** your success rate and include data/action_regression/eval_pick_training_vis/YcbMustardBottle_2.png.

(c) Evaluation on all objects (10 points)

Execute:

```
python3 eval.py --model action_regression --task empty_bin
```

Record a video of the terminal and PyBullet GUI. In your report, **report** how many objects are left. Also **explain** why is this method perform worse comparing to Visual Affordance. Maybe it needs more data to power the model??

Problem 4: Open-ended improvement (25 points)

In this problem, you are asked to improve the performance of Visual Affordance model in anyway you want except:

- Changing the network architecture.
- Adding new input modality (such as depth or object mask).

More specifically, you will need to produce a checkpoint file in *data/affordance_improved/best.ckpt*. This checkpoint should be evaluated using the command:

```
python3 eval.py --model affordance_improved --task empty_bin
```

Here are some ideas (none of them are guaranteed to improve performance):

- Label more images per object.

- Change the loss.
- Adding more training objects (e.g., define primitive shapes directly using URDF).
- Add domain randomization by applying random colors or image transformations to the training data.
- Train the model via self-supervised trail-and-error. More specifically, using the existing model to collect more data (both success and failure). Add these data and label into your dataset. Train model using the new and old data, then repeat.

For the submission, you will need to submit a mini-research report that contains following component:

- **Describe** in your report what exactly modification has you made
- **Explain** your hypothesis: why you think this modification should help.
- **Show** results. What's the difference you observed before and after your made the modification. Does the modification help in improving the performance? – it is OK that the performance is not improved, but please describe why you think it is the case.
- **Record** a video of the terminal and PyBullet GUI for eval script.
- If you tried multiple methods, describe and discuss them **all**. Make the best performing model "affordance_improved". But also update others (3 maximum).

On top of that, you will need to report how many objects are left, a video of the evaluation process, the checkpoint file and all the code (ideally a single file called *train_improved.py*) you implemented for the improvement.

Evaluation. Your solution will be evaluated based on:

- (10 points) The clarity of your report, did you clearly describe what you are doing and why you think it should help.
- (10 points) The quality of your implementation. Did you implement what you proposed to do.
- (5 points) This 5 points will only give to solutions demonstrate one of the following properties: 1. Novelty. Good and creative ideas even if it doesn't help much on this task. 2. Effort. Implement multiple or non-trivial methods and did a through comparison. 3. Performance improvement: significantly improve the performance on challenging cases.

Submission instructions

- Generate separate URLs for each video containing solution to problem 2(f), 2(g), 3(b), 3(c) and 4 (see instructions for generating video URL at the bottom) in order. Place the URL in *url_main.txt* file.
- Compile all written answers and the video URL to *hw4_report.pdf*.
- Collect the report, all python code as well as the assets and data directory to a folder named **[Your-UNI]_hw4** and then upload the compressed directory with name **[Your-UNI]_hw4.zip**. For example, cc4617_hw4.zip.

Please make sure that you adhere to these submission instructions. **TAs can deduct up to 10 points for not following these instructions properly.**

Please note that for all students, we will look at both the code and the video for grading. If significant discrepancies are found between the submitted videos and the results from manually running the code, we will report the student to Academic Committee. **We will also randomly sample a fair proportion of students and ask them for a homework interview**.

Generating video URL

Upload video 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 video 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 6).

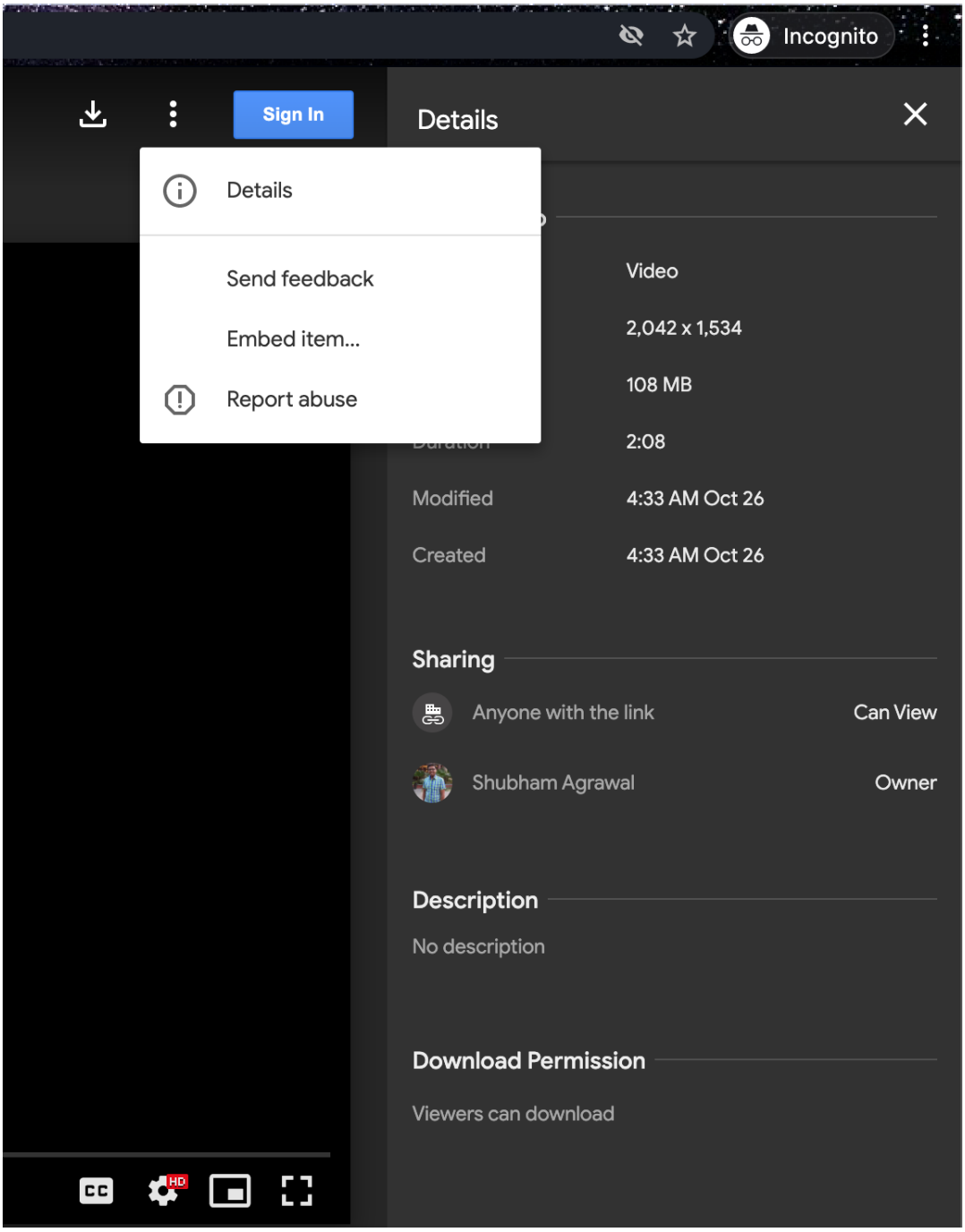

Figure 6: Video URL: should be accessible in Incognito / Private window. Modified date should be visible.