Homework 5

Instructions

In the last homework, we developed our first pick-and-place system using pose estimation. One of the limitations of that system is that we need to know the objects beforehand. For example, we need the data to train the segmentation network and the object 3D model to perform ICP. In this homework, we will build a similar pick-and-place system that can pick up unseen objects. The method we are going to develop is based on Action Affordance (also called Spatial Action Map).

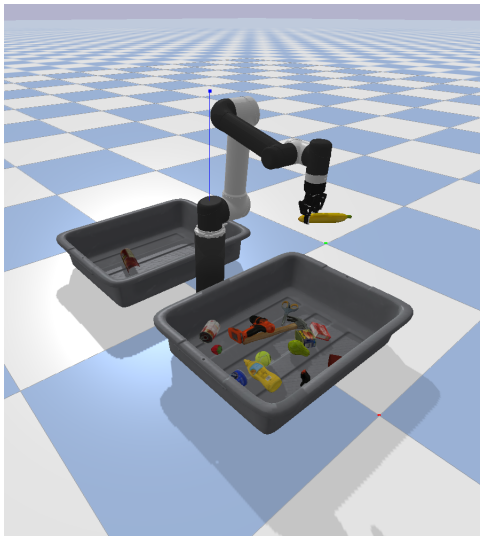

Figure 1: UR5 robot moving an object to another bin

In **Problem 1**, we will implement and train a Visual Affordance model with manually labeled data.

In **Problem 2,** you will write a report. If you take this course for 3 credits, write a short answer for the questions we provided (100 words max). If you take this course for 4 credits, you need to submit a 2-page report with the following two options: 1) Write a proposal about ideas to improve the system you have implemented. 2) Write a literature review on a topic we discussed in the lecture.

Problem 1 Concepts (10 points)

Overview. There are two key assumptions in this homework:

- The robot arm's image observations come from a top-down camera, and the entire workspace is visible.
- The robot performs only top-down grasping, where the pose of the gripper is reduced to 3 degrees of freedom (2D translation and 1D rotation).

Under these assumptions, we can easily align actions with image observations (hence the name spatial-action map). 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 translational 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 the gripper rotation before passing it to the network. In Problem 1 we will walk through an example to give you a better understanding.

1.1 Transforming grasp with respect to the object – 5 points

Given the following observation of a hammer, a good top-down grasp g is centered at $c = [0, 0]$ with a rotation angle $\theta = 0°$. Please answer the following questions and provide your answers in the written submission.

- If the object in the image I is shifted to the right $d = [10, 0]$ in pixel space, what is a good grasp for this object then?
- If the object in the image I is rotated by $a = 45°$, what is a good grasp for this object then?
- If we apply a general transformation with translation d and rotation a, what are c and θ?

Let's define the grasp function as $f_{\text{grasp}}(I) \rightarrow g$, where I is the input image and g is the output grasp parameter. We call this grasp detector function f_{grasp} is translational and rotational equivariant - translation and rotation applied on the image I should result in the same transformation on the output grasp parameter g.

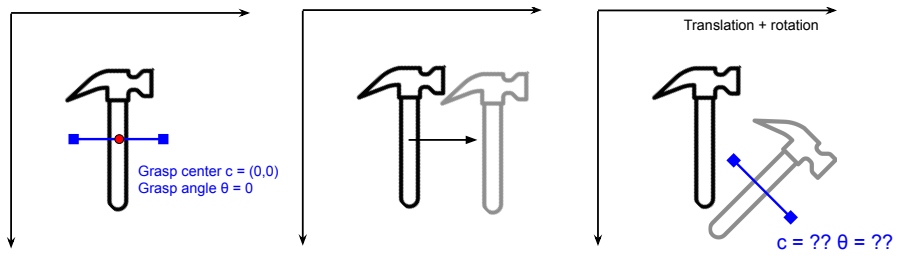

Figure 2: Transforming grasp with respect to object

1.2 Transforming object for grasp detection – 5 points

Now imagine you have a gasp detector f_{grasp} that can only detect horizontal grasp centered at the image center $c = [0, 0]$, $\theta = 0°$. If the current object orientation observed in the image is at 45°. How should we transform the image (i.e., what should be the rotation angle to apply) so that the detector can find the grasp? Please include your answer in the writeup.

In this assignment, we will train a grasp detector f_{grasp} represented by a neural network. By using a fully convolution network, the network naturally maintains translational equivariance. To encode the rotational degree of freedom, we rotate the image observation in the opposite direction of the gripper rotation before passing it to the network.

Problem 2 Coding (70 points)

Setup. Install required packages. If you have an Nvidia GPU on your computer run

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

otherwise, run

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

This will create a new environment named "hw5", which can be activated by running `mamba activate hw5`. Please do not introduce any other dependencies/packages without taking prior permission from TAs.

2.1 Generate Training Data (10 points)

In this problem, you will get familiar with the data annotation pipeline and generate training data for the subsequent problems. To start, run the following command on your **local compute**:

```
python3 pick_labeler.py
```

The script will launch a PyBullet GUI, then load the robot and objects. An 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). It might also be very small

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

at the top left corner of your screen. You can drag the screen to change the window size. After you confirm (Q or Enter), the robot will try to pick the object using the confirmed grasping pose. If the grasping fails, you will be prompted to try again.

Label 5 training objects with 12 attempts each. This usually takes around 5 minutes. You might notice one of the objects, namely YcbMediumClamp, shows up similar to a small dot and makes it hard to identify a grasping pose: try to click its center and position the gripper in any reasonable orientation.

Please read and follow these tips carefully:

- Pay attention to the grasp outcome. You can see the message on the terminal and robot action on the Pybullet UI. Can the robot grasp the object based on your annotation? We only store successful grasps.
- Pay attention to the angle of grasp use A and D keys to rotate the grasping orientation.
- You can enlarge the window to make the annotation interface bigger.
- **The quality of annotation you get in this step will have a significant impact on the later model performance.** You might want to come back and relabel some images after you see common failures in the system. To do that you need to delete existing data in the data folder for the program to ask for more annotation.
- Read the code in `pick_labeler.py` in the mouse callback `GraspLabeler.callback` to understand the annotation code.

2.2 Implement Visual Affordance Model (60 points)

We have included the folowing TODO list for your reference:

```
train.py
    __getitem__() -- 10 points
affordance_model.py
    AffordanceDataset.__getitem__() -- 10 points
    AffordanceModel.predict_grasp() -- 10 points
    suppress_failure_grasp() -- 10 points

Training visualization.

Evaluation:
    training - success rate, visualization
    testing - success rate, visualization
    empty_bin - success rate, visualization, video
    empty_bin with improvement - success rate, visualization, video
```

2.2.1 AugmentedDataset – `__getitem__` (10 points)

In `train.py`, we defined a new dataset class called `AugmentedDataset`. The task is to implement the `__getitem__` method. During training, we want to increase the number of training data without collecting more annotations. Since we know how transformations in the input image should result in changes in grasp parameters, we can augment the training data by applying random rotations to the image input and applying the same rotation in the grasp parameter (review example in sec 1.1).

To implement this idea we will use the library: `imgaug.augmenters` (example: https://imgaug.readthedocs.io/en/latest/source/examples_keypoints.html)

Please implement the following:

1. Use `get_finger_points` to get the left and right finger tip locations (provided).
2. Construct keypoints using `KeypointsOnImage`
3. Apply transformations on the image and keypoints using `self.aug_pipeline`
4. Compute the grasp center and angle using `get_center_angle` (provided).

2.2.2 AffordanceDataset – `__getitem__` (10 points)

In `affordance_model.py`, we defined a new dataset class called `AffordanceDataset`. The task is to implement the `__getitem__` method. In this function, we will prepare

the data before feeding it into the network. Given the observation image I (after the augmentation applied), and grasp parameters c_x, c_y, θ, prepare the following data:

1) '`input`' torch.Tensor (3,H,W), `torch.float32`, range [0,1]. It is the RGB image, after being rotated with a negative grasp angle (after binned into one of the 8 rotation angles `nn_angle`). Again use `KeypointsOnImage`, `iaa.Rotate`.

2) '`target`' torch.Tensor (1,H,W), `torch.float32`, range [0,1]. It is the ground truth image for the grasp center location, after being rotated with a negative grasp angle.

The simplest version of this target image is label 1 for the correct location and label 0 for everywhere else (i.e., one-hot encoding). However, that will result in a highly imbalanced training set and sparse supervision – too many 0s and too few 1s. **NOTE:** To provide a denser for training, we use `get_gaussian_scoremap(x,y)` (provided) to generate the affordance target (Gaussian distribution centered around the ground truth location), instead of training with the one-hot label. If we train with one-hot ground truth the network is likely to predict all zero values for the affordance.

2.2.3 Training (10 points)

After implementing these two functions, you should be able to train the network with labeled data. Training on your local CPU should be sufficient for this homework. If you want to use a Google Cloud GPU instance, upload your repo, including the data directory.

The model uses the `MiniUNet` architecture from Homework 3 we used for image segmentation. However, the last layer, training loss, ground truth, and training process will be different.

Execute the following command to start training your model:

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

If the 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, training on the CPU can take up to 60 mins. You don't necessarily need to train 101 epochs, something around 30 epochs should be enough. If the training is interrupted, you can resume the training, the model will try to load the best checkpoint from previous training if it exists.

Report your training loss and test (validation) loss in your writeup. Include an image from data/affordance/training_vis. It should look like Fig. 4.

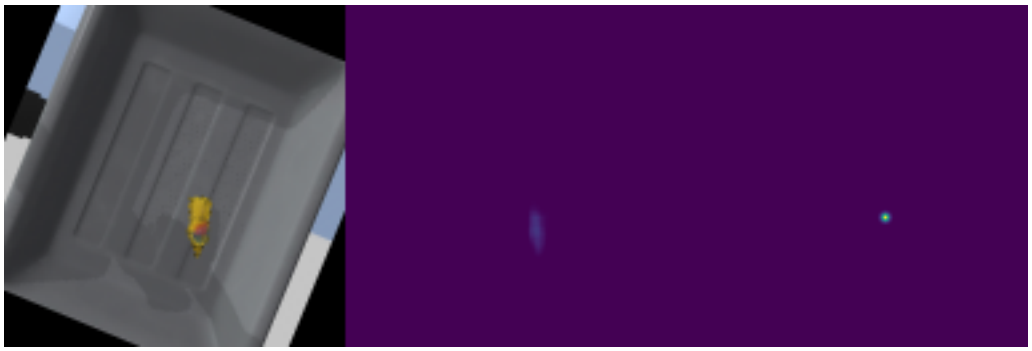

Figure 4: Example visualization. From left to right: input, prediction, target.

2.2.4 Grasp prediction – `AffordanceModel.predict_grasp` (10 points)

Now that you have trained the model that can infer grasp if the grasp is angle $\theta = 0$ in the image. However, we want to detect grasps from all angles!

You are going to implement the steps that prepare the input to the network. First, rotate the `rgb_obs` 8 times for each of the rotation angles. Use `rotators` constructed by `iaa.Rotate`. Note: torchvision's rotation is counterclockwise, while imgaug, OpenCV's rotation is clockwise. Then, stack the 8 rotated image as an input batch $(8 \times 3 \times H \times W)$ stored in rgb.

In `affordance_model.py` implement `AffordanceModel.predict_grasp`. It should look like Fig. 5.

2.2.5 Evaluation on the training set (2.5 points)

When your training is done, the model with the lowest loss should be saved as checkpoint data/affordance/best.ckpt (if using cloud compute, download data/affordance/best.ckpt back to your *local computer*).

Execute the following command to evaluate your model on the same set of objects used in the training dataset. If your prediction code is implemented correctly, it should have a success rate better than 70%.

```
python3 affordance_based_pnp.py --task train
```

Report your success rate, **include** data/affordance/eval_pick_training_vis/YcbMustardBottle_2.png. It should look similar to Fig. 5.

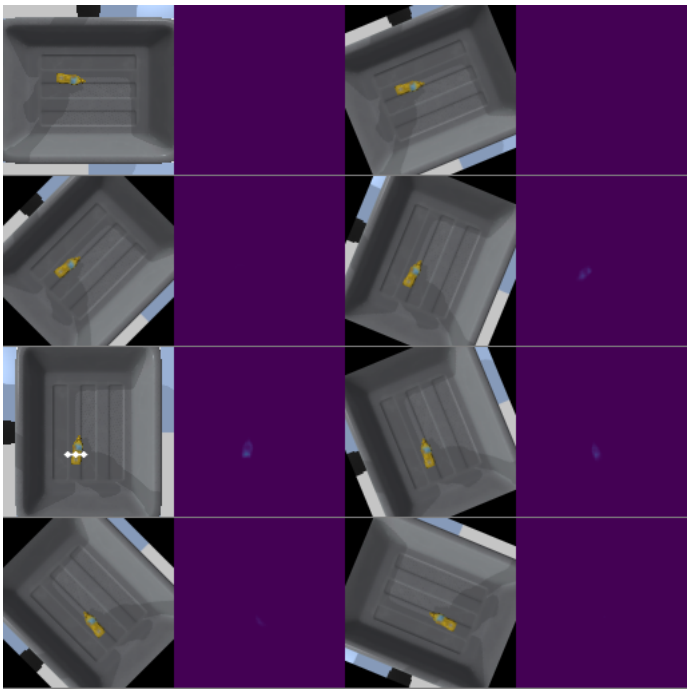

Figure 5: Example prediction visualization.

2.2.6 Evaluation on the held-out objects (2.5 points)

Execute the following command to evaluate your model on a novel set of objects that were **not** included in the training dataset:

```
python3 affordance_based_pnp.py --task test
```

Your model should do slightly worse than on the training set, expect around 50% success rate.

Report your success rate, **include** an image from data/affordance/eval_pick_testing_vis/.

2.2.7 Evaluation on mixed objects (5 points)

So far we have only tested grasping with a single object in a bin similar to the training scenario. Now we are going to test this model on cluttered environments with multiple objects. Will it work out of the box? Let's see!! Run the following command:

```
python3 affordance_based_pnp.py --task empty_bin --seed 2
```

This command will load 15 objects into a bin, and the model tries to move all of them into another bin within 25 attempts. If your implementation is correct, the model should be able to pick up at least 8 objects, with a few more left.

Record a video of the terminal and PyBullet GUI.
In your report, **report** how many objects are left. **include** a visualization.

2.2.8 Improve Performance – `suppress_failure_grasp` (10 points)

As you may have noticed, the trained model does not succeed 100% at test time. One of the most annoying failure cases is that the robot will keep failing on the same object with the same action. Can we give the robot some memory, and let it remember its past failures?

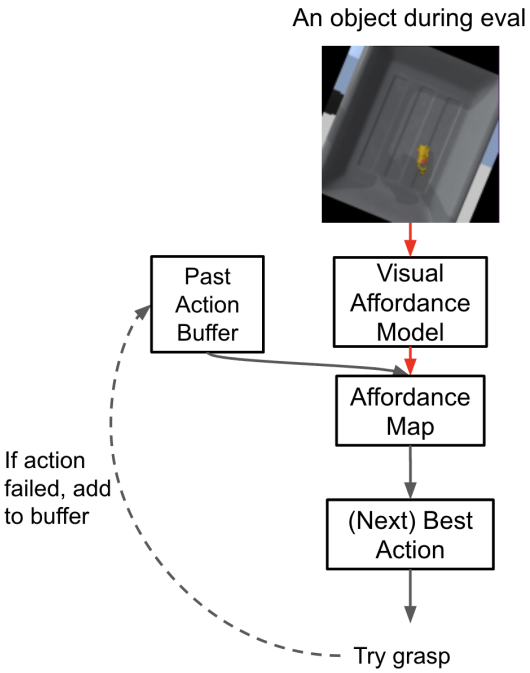

Figure 6: Illustration for the test-time improvement idea: add a buffer of past failed actions to the model, such that it selects the next-best actions when making a new attempt.

Recall that, in the visual affordance model, the spatial-action map formulation is essentially looking at an image observation and finding the best pixel to act on. So if an action turns out to fail after executing the grasp, the robot can try again and select the **next best** pixel/action. See Figure 6 for an illustration.

Implement this idea by editing `suppress_failure_grasp`.

```
update_affordance = current_affordance
For all coordinates stored in past_actions
       suppression_map = get_gaussian_scoremap(coordinates)
       update_affordance -= suppression_map
clip update_affordance between 0,1
return update_affordance
```

During evaluation, make the model attempt to grasp an object multiple times by setting the command line argument `--n_past_action 8` and keep track of the failed pixels' actions, such that for each new attempt, it avoids those before selecting a new action.

When you are done, evaluate the model again on training, testing, and mixed-object data like before, except now you perform multiple attempts when trying to grasp each object.

```
python3 affordance_based_pnp.py --task empty_bin --seed 2
```

Record a video of the terminal and PyBullet GUI.
Report how many objects are left and **include** a visualization.

Problem 3: Discussion - 20 points

3-credit students, answer the following questions:

Q1: Now that you have evaluated the model, in your report, try to **explain** why this method can perform well on both seen and unseen objects despite given only $\tilde{5}0$ images to train. (100-word max)

Q2: In your report, **discuss** how the performance from each evaluation run is different between and after you implement the test-time improvement. (200-word max)

4-credit students, submit a 2-page report literature review or survey on topics we discussed in the lecture. For example, a survey on "single-view depth estimation." In the survey, please categorize the related works into three categories. For each category, describe the general idea, the papers that belong to this category, and the general pros and cons for this category of approach.

Submission instructions

- Generate separate URLs for each video containing a solution to Problems 2.2.7 and 2.2.8.
- Compile all written answers and the video URL to `hw5_report.pdf`.
- Put the report, all python code, as well as the assets and data directory to a folder named `SUID_hw5` and then upload the compressed directory with the name `SUID_hw5.zip`. For example, shuran_hw5.zip.

Please make sure that you adhere to these submission instructions. TAs can deduct up to 10 points for not following these instructions properly. We will look at both the code and the video for grading. We will also randomly sample a fair proportion of students and ask them for a homework interview.