Instruction for Project 2

Behavior Cloning for RC car racing

1. Introduction

In this assignment, you will develop proficiency in using a behavior cloning (BC) algorithm known as Gaussian Process Regression (GPR) to control an RC car. GPR requires gathering expert demonstrations for training purposes. You should collect these demonstrations using the controller developed in the previous project (Project 1). Afterward, you need to train the GPR model you've defined with these expert demonstrations.

2. Algorithm

Figure 2 provides a brief overview of BC for a general configuration space, assuming the environment as a Markov Decision Process (MDP). This includes a set of states S, a set of actions A, and a transition model $P(s,a)$ that describes the environment. The notation τ represents the agent's policy, while $\tau^* = (s_0^*, a_0^*, s_1^*, a_1^*, \dots)$ depicts the trajectories generated through expert demonstrations. With τ^* provided, we treat the state-action pairs as independent and identically distributed (iid) and apply GPR to model these relationships.

1. Collect demonstrations (τ^* trajectories) from expert
2. Treat the demonstrations as i.i.d. state-action pairs: $(s_0^*, a_0^*), (s_1^*, a_1^*), \dots$
3. Learn π_θ policy using supervised learning by minimizing the loss function $L\big(a^*, \pi_\theta(s)\big)$

Figure 2. Pseudo code of BC

Followings are the steps of the BC algorithm that you need to implement in Project 2.

Step 1. Collect demonstration observations and actions using the pure pursuit algorithm from Project 1.

Step 2. Load expert's demonstration observations and actions. Normalize this data by calculating their mean and standard deviation, and shuffle the data to minimize correlation. We recommend using the scikit-learn library for coding these processes. **Don't forget to pre-process the data for better performance!**

Step 3. Fit the GPR model using the normalized and shuffled demonstration data as targets.

Step 4. Start racing! Continue predicting actions from the fitted GPR model until the environment terminates.

3. Explanation of Code

[TEAM_NAME]_project1.py

We have specified the directory (IS_[TEAM_NAME]/project/trajectory) where your demonstration observations and actions should be stored. Please save the scan values as observations and the steering and speed values as actions. The scan values consist of 720-dimensional lidar sensor readings that correspond to angles from -90° to 90°.

[TEAM_NAME]_project2.py

You can select the model by using the '--model_name' argument and decide whether to train a new model or load a previously trained one using the '--mode' argument. The paths for the demonstration data and the trained model are specified as 'IS_[TEAM_NAME]/project/trajectory' and 'IS_[TEAM_NAME]/project/model,' respectively. The 'GaussianProcess' class encapsulates the behavior cloning model along with functions to either train or load it.

GaussianProcess.__init__()

Initialize the GPR model. You have the option to customize it by modifying the RBF kernel or adjusting other parameters.

GaussianProcess.train()

Load the expert demonstration data and fit into the defined GPR model. Calculate the mean and standard deviation of both demo observations and actions to normalize those values. To break the

correlation among data sequences, shuffle the orders. Save the trained model and configuration for pre/post-processing.

GaussianProcess.load()

Load trained model and configurations of pre/post-process to be used for get_action().

GaussianProcess.get_action()

Predict the action from current observation using the GPR model specified by "model_name". Don't forget to pre-process the current observation and post-process the action output.

4. Running codes

[TEAM_NAME]_project1.py

For Project 2, you have to save trajectories from the pure pursuit demonstrations. You can save trajectory by using "--save" argument. Also, you can run without rendering by using "--no_render" for faster process.

```
ros2 run rccar_bringup RLLAB_project1 --save --no_render
# Replace RLLAB with your team name
```

[TEAM_NAME]_project2.py

When you want to train new model and evaluate, you can run project2 code with "--mode train" argument.

```
ros2 run rccar_bringup RLLAB_project2 --mode train
# Replace RLLAB with your team name
```

When you want to load trained model without training, you can just run without "--mode" argument since its default value is "val".

```
ros2 run rccar_bringup RLLAB_project2
# Replace RLLAB with your team name
```

Caution

Before you run the codes, you must add a new entry point for RLLAB_project2 in your setup.py file as follows.

```
'RLLAB_project2 = rccar_bringup.project.IS_RLLAB.project.RLLAB_project2:main'
# Replace RLLAB with your team name
```

Manually publishing map topic

For each project code, you can publish "/query" topic manually using following command in another terminal.

```
ros2 topic pub --once /query message/msg/Query "{id: '0', team: 'RLLAB', map: 'map1', trial: 0, exit: false}"
# Replace RLLAB with your team name and you can use other maps we provide in 'maps' directory
```

Note that you can publish the topic once with "--once" argument.

5. Submission format

To evaluate your code in a local PC, follow the instructions described in the class repository(https://github.com/rllab-snu/Intelligent-Systems-2024).

Commit and push your codes on github and make a query on our project web server.

Project 2 is due on **11/18(Mon) at 23:59 KST**, and the late query will not be submitted.