**custom\_dataset.py**: definition of dataset (is created during training/testing of model)

**graph\_loss.py**: just for printing loss of training model

**print\_occs.py**: you can edit parameters to view occupancy maps once dataset is saved

r**obot.py**: robot definition. Contains expert control, model control post-processing, sensor reading, VREP handles, etc.

**saver.py**: class to handle scene saving

**sceneplot.py**: functions for plotting robot positions, distances, etc post-episode

**scene.py**: definition of scene, contains rendering functions, simulation function to propagate scene, reset robot positions, VREP handles for initialization and robot movement, adjacency matrix information, etc.

**simConst.py**: constants for VREP handling

**sim.py**: simulation library

**single\_position\_graph**: metric graphing

**state.py**: definition of robot state (x,y,theta, transforms, etc)

**suhaas\_agent.py**: class that contains model. Contains hyperparameters, test/train loops, saving functions

**test5\_more\_robots.py**: main runner class. Contains total simulation loop, post simulation saving, etc.

**graphs/models/suhaas\_model.py**: contains model definition, forward/backward propagation, etc. To get deeper into GNN definition, go to **utils/graphUtils/graphML.py**.

Notes:

1. To run test5\_more\_robots.py, make sure VREP is running and the proper scene is loaded (scene/scene\_three.ttt)

2. There are some plots that occur after every episode (distance, position, and control plots). These are stored in the figs/ folder. I save the entire position list over all episodes run in a simulation at end of test5\_more\_robots.py, and then use this saved information in single\_position\_graph.py.

3. Since we are appending to a dataset after every episode (Dagger), a new custom\_dataset is created every time ‘train’ is called in suhaas\_agent.py.

4. The input to the neural network is the occupancy maps, the GSO, and the reference distance. The output is the control for each wheel.