Weekly Report & Workpack- 08/05/2020

1. Matlab Code Analysis

The example Matlab script can be generalised to 4 parts:

1. Simulink System. The simulink system contains 5 main modules.

    1. **Generic Vehicle Simulation**. This module contains the kinetics of virtual vehicles. Given the body speed *bodySpd*, output the simulated pose of vehicle *pose*.
    2. **Lidar Sensor**. This module outputs the scan data *ranges* of lidar sensor, given the pose of vehicle and the map. It's followed by a **Add Lidar Noise** module, to add noise into the simulated lidar scan data.
    3. **Contact Check**. In this module, the terminal condition *t* is defined, and terminal signal *isDone* generated, so that the current training episode will be terminated. Also it will output the minimal range *minRange* in laser scan data.
    4. **Reward**. This module compute the *reward* given *minRange*, and *bodySpd* of last frame.
    5. **RL Agent**. This module is the main module of RL Agent. It takes *obeservation*, which is an array formed by *ranges* in original example, *reward*, *isDone*, output *action*, which will be demux to linear speed *v*, and angle speed *w*, which furthur composing *bodySpd*.

2. Initialization and Environment Interfaces Creation.

    This section of code first initialize the params, load map and simulink system,

    ```
    open_system(mdl) %open the simulink system
    rlMobileRobotParams; %run script which initialize params
    load(mapName) %load the occupancy map
    ```

    then build the simulation environment object.

    ```
    env = rlSimulinkEnv(mdl,agentBlk,obsInfo,actInfo);
    env.ResetFcn = @(in)rlMobileRobotResetFcn(in,scanAngles,maxRange,ma
    pName);
    ```

    in which, *mdl* is the simulink system, *agentBlk* specify which module represents the RL agent this script build below. Also the *rlMobileRobotResetFcn* is set here, which is the

reset function defining what to do to reset the environment when each episode is finished, e.g. generating new init poses for vehicl, obstacles and targets.

3. Build critic and actor network.

```
statePath = [
    imageInputLayer([numObservations 1 1], 'Normalization', 'none', '
    fullyConnectedLayer(50, 'Name', 'CriticStateFC1')
    reluLayer('Name', 'CriticRelu1')
    fullyConnectedLayer(25, 'Name', 'CriticStateFC2')];
actionPath = [
    imageInputLayer([numActions 1 1], 'Normalization', 'none', 'Name'
    fullyConnectedLayer(25, 'Name', 'CriticActionFC1')];
commonPath = [
    additionLayer(2,'Name', 'add')
    reluLayer('Name','CriticCommonRelu')
    fullyConnectedLayer(1, 'Name', 'CriticOutput')];
```

build the 3 paths of critic network above, then connect the paths, i.e. sub network.

```
criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,statePath);
criticNetwork = addLayers(criticNetwork,actionPath);
criticNetwork = addLayers(criticNetwork,commonPath);
criticNetwork = connectLayers(criticNetwork,'CriticStateFC2','add/i
n1');
criticNetwork = connectLayers(criticNetwork,'CriticActionFC1','add/
in2');
```

at last, define some options and create the critic object.

```
criticOpts = rlRepresentationOptions('LearnRate',1e-3,'L2Regulariza
tionFactor',1e-4,'GradientThreshold',1);
critic = rlRepresentation(criticNetwork,obsInfo,actInfo,'Observatio
n',{'State'},'Action',{'Action'},criticOpts);
```

The actor network is built in a similar way.

Finally, determine options for DDPG, and create the DDPG agent object.

```
agentOpts = rlDDPGAgentOptions(...
'SampleTime',sampleTime,...
'TargetSmoothFactor',1e-2,...
'DiscountFactor',0.995, ...
'MiniBatchSize',128, ...
'ExperienceBufferLength',1e6);
agentOpts.NoiseOptions.Variance = 0.1;
agentOpts.NoiseOptions.VarianceDecayRate = 1e-5;
agent = rlDDPGAgent(actor,critic,agentOpts);
```

4. Training.

   Set training options i.e. the stop training criteria, max episodes, etc.

```
maxSteps = ceil(Tfinal/sampleTime);
trainOpts = rlTrainingOptions(...
    'MaxEpisodes',maxEpisodes, ...
    'MaxStepsPerEpisode',maxSteps, ...
    'ScoreAveragingWindowLength',50, ...
    'StopTrainingCriteria','AverageReward', ...
    'StopTrainingValue',5, ...
    'Verbose', true, ...
    'Plots','training-progress');

trainingStats = train(agent,env,trainOpts);
```

5. Validate Trained Agent.

   Replay the training result.

2. Modification for random target reaching and obstacle avoidance

Based on the NTU dissertation, and training performance, I modified the system as following:

1. reward function to be a function of distance. Now it's shaped as a linear function, a better shape remains to be discovered.
2. Collision with an obstacle won't terminate the current training episode, but a severe penalty will be excuted on reward. And the robot will be stuck in the current positoion, until it is given a new command to move away from the obstacle, of which the code is not implemented yet.

My code is in the branch *lxy* of the github repo listed in the workpack last week. To clone a repo and change the branch, do as follow:

```
cd <your workspace folder>
git clone <some repo address> #clone the repo
cd <some repo address>
git branch -a #to show all branches
git checkout <branch name> #by checkout, your file will be changed to
the branch you selected.
```

3. Known Issue

1. In the origin toolbox example of obstacle avoidance, the origin position of the robot in each episode is generated randomly, and in the dissertation, the target point generated randomly, with the original position fixed. But in the training, I can't see a noticable reward increasing if either one is set randomly, so in my modified version, they are both fixed.
2. The reward increases very slow, when the robot can reach very close to the target point, i.e. distance $\approx$ 2, then it is very difficult to shorten the distance furthur to the threshold distance, 0.5 in this case. My thought is that it's because the closer to the target, the lower the derivative of distance to speed. So I tried to shape the reward function as a quadratic function of distance, which should at least have a similar performance with linear function theoretically, but it looks worse now, the parameters of quadratic function need to be tuned.

4. Tasks for next week

Notification

1. Since Peixuan has exams to take in the weeks future, her tasks will not be listed in the document for the next weeks.
2. From this week, all members shall have their weekly work documented in a weekly reeport, which shall be handed in before the next monday.

Tasks

The overall goal of next week is to build a multi-robot, formation controlled, obstacle avoidance system.

1. For *everyone*,

    - learn basic git operations.
    - learn to write a markdown file.

2. For *Yicheng*,

    - Build a simulink project that have:
        - 3 robots at least.
        - vehicle simulation and lidar modules for each robots.
        - a reward function module.

    - modules involving algorithm designing can be left blank or copied from example project, the modules or subsystem must be build in the project.
    - The project must be runnable.

3. For *Xiangyu*,

    - Review related work on formation control.
    - Design a demo of formation control using RL, which includes the work to choose a formation, design a collaborative reward function, etc.