Deep RL Arm Manipulator

Ge Yao, alexgecontrol@qq.com

Abstract—In this paper a DQN agent based on heuristic reward functions is designed to carry out two primary objectives: First, have any part of the robot arm touch the object of interest, with at least a 90% accuracy of at least 100 runs. Second, have only the gripper base of the robot arm touch the object, with at least a 80% accuracy of at least 100 runs.

Index Terms—DQN, LSTM, Robotic Arm, Kinematic.

1 Introduction

A Basic problem in robotics is the control of manipulators. One typical example is the actuation of robotic arm. Traditional approach relies on the analysis of kinematics. This approach can solve the problem in a straight forward way. However, it also requires a very good knowledge of geometry and the scope of solvable problems are limited. With the advent of deep reinforcement learning, we can solve the problem using the typical data driven patterns in an easy way.

In this paper the end-to-end work flow for building deep reinforcement learning agents for the two tasks will be illustrated.

2 Reward Function Design

The reward function consists of five terms. Two of them are rewarding terms and the rest three are penalty terms. The hyper-parameters of the reward function is shown in Fig. 1.

```
/*
/ Parameters for Reward Function
*/
#define REWARD_WIN                    +10.0f
#define REWARD_LOSS                   -10.0f
#define REWARD_APPROACHING            +2.0f

#define MIN_APPROACHING_VELOCITY      +0.025f

#define FACTOR_TOUCH_BY_GRIPPER_BASE  +1.0f
#define FACTOR_TIMEOUT                +1.0f
#define FACTOR_TOUCH_GROUND           +1.0f

#define ALPHA  0.4f
```

Fig. 1. Reward Function Hyperparameters

The agent can get reward from the following two actions:

- Touching object with gripper base as shown in Fig. 2.
- Approaching object at speed above a threshold as shown in Fig. 3.

The agent will get penalty from the following three actions:

- Touching object with parts other than gripper base as shown in Fig. 4.
- Time out as shown in Fig. 5.
- Touching the ground with gripper as shown in Fig. 6.

```
// Reward Term 1: touching object with gripper base
if (
    strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0
) {
    rewardHistory = REWARD_WIN;

    if (
        strcmp(contacts->contact(i).collision2().c_str(), COLLISION_GRIPPER_BASE) == 0
    ) {
        // case 1 -- touch by gripper:
        rewardHistory *= FACTOR_TOUCH_BY_GRIPPER_BASE;
    } else {
        // case others -- unwanted touch:
        rewardHistory = REWARD_LOSS;
    }

    newReward  = true;
    endEpisode = true;
}
```

Fig. 2. Reward Function, Rewarding Term 1

```
// moving average of approaching speed:
avgGoalDelta  = (avgGoalDelta * ALPHA) + (distDelta * (1.0 - ALPHA));

// Reward Term 2: approaching object at speed above a threshold
float rewardGripperGoalDelta = REWARD_APPROACHING * (avgGoalDelta - MIN_APPROACHING_VELOCITY);
```

Fig. 3. Reward Function, Rewarding Term 2

```
// Penalty Term 1: touching object with parts other than gripper base
if (
    strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0
) {
    rewardHistory = REWARD_WIN;

    if (
        strcmp(contacts->contact(i).collision2().c_str(), COLLISION_GRIPPER_BASE) == 0
    ) {
        // case 1 -- touch by arm:
        rewardHistory *= FACTOR_TOUCH_BY_GRIPPER_BASE;
    } else {
        // case others -- unwanted touch:
        rewardHistory = REWARD_LOSS;
    }

    newReward  = true;
    endEpisode = true;
}
```

Fig. 4. Reward Function, Penalty Term 2

```
// Penalty Term 2: time out
    if( maxEpisodeLength > 0 && episodeFrames > maxEpisodeLength )
    {
            printf("[EOE]: episode has exceeded %i frames\n", maxEpisodeLength);

            rewardHistory = FACTOR_TIMEOUT * REWARD_LOSS;

            newReward     = true;
            endEpisode    = true;
    }
```

Fig. 5. Reward Function, Penalty Term 2

```
// Penalty Term 3: touching the ground with gripper
const float goalDistance = BoxDistance(gripBBox, propBBox);

/*
/ TODO - set appropriate Reward for robot hitting the ground.
*/
if(isGroundContact)
{
    printf("[EOE]: GROUND CONTACT\n");

    rewardHistory = FACTOR_TOUCH_GROUND * REWARD_LOSS;

    newReward     = true;
    endEpisode    = true;
}
```

Fig. 6. Reward Function, Penalty Term 3

2.1 Approaching with Any Part of Arm

For this task, the constraint of term 1 of rewarding parts can be released to include link2, gripper middle, left and right. Besides, since this is a relatively easy task. Epsilon should be reduced to starting from 0.2.

2.2 Approaching with Gripper Only

For this task, the constraint of term 1 of rewarding parts should only include gripper base. Besides, since this is a relatively challenging task and the reward is sparse. Epsilon should be increased to starting from 0.7.

3 Control Strategy

The control strategy for the system is as follows. The agent generate the categorical output, one integer between$[0, 2 \times DoF)$, as control command. It will be mapped to increase and decrease of one control variable by delta for even and odd output respectively. Two type of realizations, namely velocity control and position control, are implemented. Position control is used for later experiment.

3.1 Velocity Control

The implementation of discrete velocity control is shown in Fig. 7.

```
// parse joint index:
int jointIndex = action / 2;

/*
/ Velocity Control - Increase or decrease the joint velocity based on whether the action is even or odd
/  If the action is even, increase the joint velocity by the delta parameter
/  If the action is odd,  decrease the joint velocity by the delta parameter
*/
float direction = (0 == action % 2 ? +1.0 : -1.0);
float velocity = vel[jointIndex] + direction * actionVelDelta;

if( velocity < VELOCITY_MIN )
        velocity = VELOCITY_MIN;

if( velocity > VELOCITY_MAX )
        velocity = VELOCITY_MAX;

vel[jointIndex] = velocity;

// actuation:
for( uint32_t n=0; n < DOF; n++ )
{
        ref[n] += vel[n];

        if( ref[n] < JOINT_MIN )
        {
                ref[n] = JOINT_MIN;
                vel[n] = 0.0f;
        }
        else if( ref[n] > JOINT_MAX )
        {
                ref[n] = JOINT_MAX;
                vel[n] = 0.0f;
        }
}
```

Fig. 7. Velocity Control

3.2 Position Control

The implementation of discrete position control is shown in Fig. 8.

```
// parse joint index:
int jointIndex = action / 2;

/*
/ Position Control - Increase or decrease the joint position based on whether the action is even or odd
/ if the action is even, increase the joint position by the delta parameter
/ if the action is odd,  decrease the joint position by the delta parameter
*/
float direction = (0 == action % 2 ? +1.0 : -1.0);
float joint = ref[jointIndex] + direction * actionJointDelta;

// limit the joint to the specified range
if( joint < JOINT_MIN )
        joint = JOINT_MIN;

if( joint > JOINT_MAX )
        joint = JOINT_MAX;

// actuation:
ref[jointIndex] = joint;
```

Fig. 8. Position Control

4 Hyperparameters

The hyper parameters of DQN agent are shown in Fig. 9.

```
/*
/ Hyperparameters for DQN agent
/ Raw Input Params:
/   W-H-C: 64-64-3
/ DOF:
/   3
*/
#define INPUT_WIDTH    64
#define INPUT_HEIGHT   64
#define INPUT_CHANNELS 3

#define NUM_ACTIONS ((ArmPlugin::DOF)*2)

#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 256

#define GAMMA 0.9f

#define EPS_START 0.7f
#define EPS_END 0.02f
#define EPS_DECAY 200

#define USE_LSTM true
#define LSTM_SIZE 256
#define ALLOW_RANDOM true
#define DEBUG_DQN false
```

Fig. 9. Hyperparameters of DQN Agent

- The input dimensions are determined from camera message medadata.
- The output dimensions, the number of actions, are set as the dimension of control. Since in this project only planar motion planning is used, hence the LOCKBASE is set to true and agent DOF is set as 2. So the dimension of output is set as 2 DOF which is 4.
- The optimizer is selected as Adam with learning rate 0.1 for LSTM out of Andrew Ng's best practice suggestion

5 Results

5.1 Approaching with Any Part of Arm

The results for arm approaching task is shown in Fig. 10. The accuracy is 90.35% which is above 90%.

5.2 Approaching with Gripper Only

The results for arm approaching task is shown in Fig. 11. The accuracy is 82.09% which is above 80%.

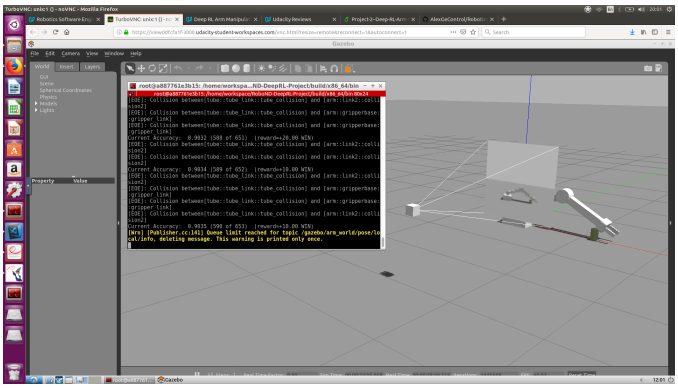

Fig. 10. Results for Arm Approaching

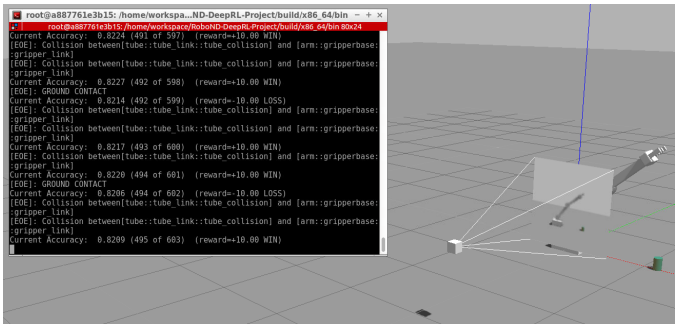

Fig. 11. Results for Gripper Grasping

6 Future Work

In this paper a DQN agent based on heuristic reward functions is designed to carry out the whole arm approaching and gripper grasping tasks. It can achieve the required accuracy with satisfactory performance.

In future work, the heuristics reward function should be refined to counter the sparse nature of the reward, which creates great challenge for task 2. Besides, auto-hyper parameter tuning could be tried to reduce the amount of manual work.

References