There are two essential factors to train the reinforcement learning algorithm: environment and agent. Through the agent consistently do practice in the environment, optimize the strategy policy, it can survive in the environment, and get the highest reward eventually.

For special reasons, it is impossible to apply the algorithm for the real-robot environment in this semester. In this project, we use python, the most popular programming language, to simulate an agent and an environment to apply it.

The most famous library in RL is the gym. According to the official website, the gym is a toolkit for developing and comparing reinforcement learning algorithms. This project implements the algorithms based on the already built structure of the gym and the additional methods.

The structure of the primary function

Thought the standard example from gym，we reconstructed and rebuilt it according to our simulated robot’s project.

if \_\_name\_\_ == "\_\_main\_\_":

    env = MainEnv()

    for \_ in range(MAX\_EPISODES):

        s = env.reset()

        while True:

            action = rl.choose\_action(s)

            for \_ in range(SAM\_STEP):

                s\_prime, reward, done = env.step(action)

                if DO\_PLOT:env.render()

                    s\_prime = discretization(s\_prime)

            rl.learn(s, action, reward, s\_prime)

            s = s\_prime

            if done:

                env.close()

                break

The \_\_init\_\_() of the env class and reset() functions can initialize the world and the agent when the environment class is instantiated.

The \_\_init\_\_() function is responsible for initializing variables, such as state initialization, GUI initialization, etc. but it does not assign values to any variables.

The reset() assigns the values to the state of the agent. The values could be a constant or a random number.

action = rl.choose\_action(s) provides an action based on the current state.

for \_ in range(SAM\_STEP):

      s\_prime, reward, done = env.step(action)

      if DO\_PLOT:env.render()

It is the most critical function. The input is the action (or a set with many actions), and the output is a state, reward, etc. Within the step(), the agent can read the input action and deduce the next state based on the previous state “self.state” from the env class. The process can be modeled in the physical world, i.e., gravity, friction, etc. A clear structure, the appropriate models, and parameters can help in the training of RL algorithms.

When the step function is called for each time, the simulated agent moves one step in the world. Then the state, reward, and done (the variable that determines whether this process ends). Since we are using the discrete RL method, the state’s output should not be exported of each moment.

Here, the for loop is added to simulate the process of sampling or overserving according to the global variable SAM\_STEP.

s\_prime = discretization(s\_prime)

Since the state of the agent output is continuous and the type of state is double, we need to use the discretization() function to convert the continuous state into a discrete state.

rl.learn(s, action, reward, s\_prime)

s = s\_prime

The pre-states, latest states, reward, and action are training dates to train and update the q-table. According to the RL algorithm, the state should be updated.

Render function

render() function is primarily responsible for rendering the world, the movement of the robot, and the robot’s arm.

The logic of the rendering is shown in the following figure:

If a new viewer is created, it runs as follows:

from gym.envs.classic\_control import rendering

self.viewer = rendering.Viewer(SCREEN\_W, SCREEN\_H)

l, r, t, b = -ROBOT\_W / 2, ROBOT\_W / 2, ROBOT\_H / 2, -ROBOT\_H / 2

robot = rendering.FilledPolygon([(l, b), (l, t), (r, t), (r, b)])

self.robot\_trans = rendering.Transform()

robot.add\_attr(self.robot\_trans)

self.viewer.add\_geom(robot)

First, import the rendering library provided by the gym. Through this library’s functions, we can quickly draw basic shapes such as rectangles, circles, etc. But it also has some drawbacks, the comparison of advantages and disadvantages are discussed in later paragraphs.

In the function rendering.Viewer(SCREEN\_W, SCREEN\_H), the size of the viewer is defined, and here we used 600 times 600 pixels.

l, r, t, b = -ROBOT\_W / 2, ROBOT\_W / 2, ROBOT\_H / 2, -ROBOT\_H / 2

robot = rendering.FilledPolygon([(l, b), (l, t), (r, t), (r, b)])

The function FilledPolygon(), which provided by rendering already can draw the robot. Also, we need to note that the robot’s anchor is the origin(0,0), and the shifting of anchor point is the position of the robot.

self.robot\_trans = rendering.Transform()

robot.add\_attr(self.robot\_trans)

self.viewer.add\_geom(robot)

This step allows us to perform displacement, and the updated displacement and position are implemented through self.robot\_trans. By add\_geom(), the new state can be loaded in the viewer.

After that, we can implement self.robot\_trans.set\_translation(x) to refresh the position of robot in the screen.

Compare rendering and pyglet modules

The rendering module is a module provided by the gym, while the piglet module is a more fundamental module, which means that the construction of rendering module is based on pyglet module. After implementing of rendering module, we found out its pros and cons.

The advantage of the rendering module is that the reference is apparent. The structure of rendering is also simple, and we could draw the robot and control its movement directly by using FilledPolygon() and Transform() functions and, at the same time, calling self.viewer.render refresh the screen.

The disadvantages are that the rendering module cannot change the background and insert elements such as pictures. In order to render the robot’s position, arm’s angle, and reward, text messages should also be displayed on the screen. But it isn’t easy to do simply by calling the rendering module.

The piglet is more powerful and flexible, representing elements in the world in pictures and support for text refresh. The disadvantage is that we need to write our entire Viewer class. Clearing the screen and drawing a new frame must be realized manually.

As a quick building and testing in the pre- and mid-term, it is beneficial to visualize most of the necessary information using the rendering module. But in the next step, it is necessary to replace it with the piglet module and integrate relevant textual information on the screen.

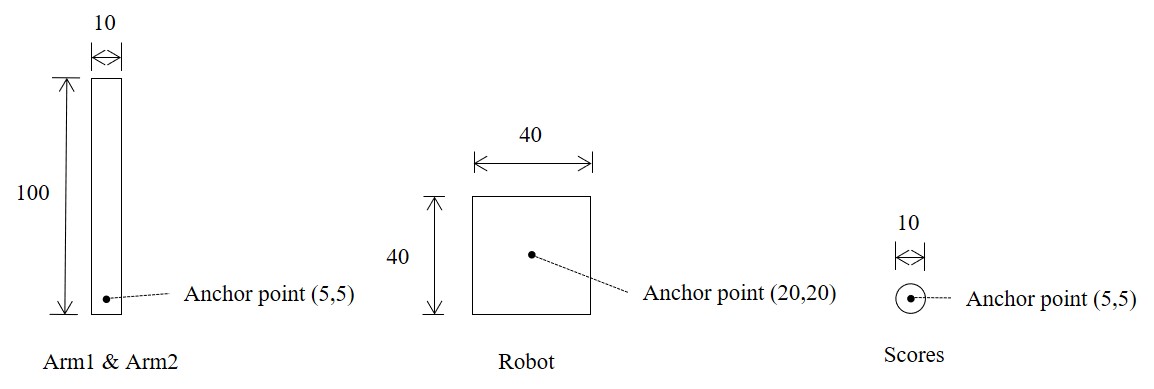
Drawing elements

In the section above, we discussed how to use the appropriate modules and code to refresh the screen, and in this section, we describe the elements that need to be drawn.

There are two different types of elements on the screen, static and dynamic.

The start and endpoints of the robot, as well as the trajectory of its movement, are the static type, which means that they do not change at all and can be displayed all the time.

Other elements are dynamic types. To draw these elements, we first need to find their anchor point, which is relative to themselves, so that we can determine each element’s movement. The robot moves as a base, and its anchor is in the geometric center. The two robot’s arm has the same anchor, which is shown below. Once the anchor has been determined, it is possible to place them in the world’s coordinate system and calculate the motion.



The position of the robot and the arms can be calculated using the following formula:

Position of robot: 

Position of arm1: 

Position of arm2:

