ME5418 Machine Learning in Robotics

Autonomous Fruit-Picking Robot

Group 7 Final Report

AY24/25 Semester 1

Professor: Guillaume SARTORETTI

Student: GAO Yilin (A0303608L)

Teammates: LIN Haoyu, SUN Yining

National University of Singapore

1 Introduction

1.1 Background

With the continuous advancement of robotic technology, automated picking is gradually changing the production methods of modern agriculture. However, due to environmental dynamic changes and operating complexity, fruit picking has become a difficult task for traditional mechanization. This prompted us to think about how to design an intelligent fruit picking robot that can learn independently and flexibly respond to complex scenarios and complete this important task more efficiently and accurately.

This study explores the use of deep reinforcement learning (DRL) to train a fruit-picking robot so that it can continuously optimize decision-making strategies through continuous interaction with the environment. With DRL, robots can learn to complete picking tasks autonomously in a dynamic environment. This type of technology can not only provide innovative ways to solve the shortage of agricultural labor, but also inject new vitality into the development of smart agriculture and is expected to play a key role in future large-scale agricultural production.

1.2 Existing Conventional Approaches

There already exist various of classic path planning algorithms, such as A* and Dijkstra, which perform well in static and low-dimensional environments and can find optimal paths efficiently. However, these algorithms are inadequate when faced with dynamic environments and high-dimensional state spaces. Especially in complex orchard scenes, it is difficult to cope with complex non-path planning operations such as grabbing or extracting fruits by manipulators. Moreover, traditional reinforcement learning methods, such as Q-learning, can optimize its decision-making through state-action value tables. Although they achieve good results in simple tasks, it will increase significantly the storage requirements and computing costs in high-dimensional space. The common limitation of these methods above is that they lack sufficient flexibility and scalability to meet the complex needs of fruit-picking robot in high-dimensional and dynamic scenarios.

Therefore, when dealing with complex orchard picking tasks, we need not only deep learning (DL) models that can understand complex environments, but also reinforcement learning (RL) algorithms that can adapt to dynamic changes and handle multi-dimensional continuous actions. Fortunately, it is the advantage of DRL, which combines the capabilities of DL and RL to provide an ideal framework for solving complex problems in fruit-picking robots.

1.3 Project Statement

The environment model has been adjusted many times, and the following is the final version used for training and testing:
In order to simulate the orchard scene more realistically, we designed a 3D discrete grid space (10×10×10, before is 50x50x15) containing fruit trees, fruits, and a robot (Figure 1). The robot is located at a certain point (x, y, h) in the grid and aims to intelligently and efficiently complete the

task of picking fruits while avoiding collisions with trees and boundaries. Since fruits are relatively light in reality, we ignore the collision problem between the robotic arm and fruits. And fruits will return to their original positions after each collision. At the beginning of each episode, ripe and unripe fruits (x, y, z, ripeness) are randomly distributed at different heights in the tree, and their positions remain unchanged during this episode. But once a fruit is picked, it will disappear and lead to a dynamic environment.

We also simplified the robot model (Figure 2) for relatively easy training and rendering display (The new URDF file has been updated). Our fruit-picking robot is generated in two parts: the base and the robotic arm. Its goal is to pick 3 ripe fruits as soon as possible (compared to the previous goal of 10), while avoiding going out of bounds and colliding with the tree trunk. In addition, the robot's base (3x3) allows it to move forward and backward and turn left and right on the ground plane through the grid. The first link of the robot arm can be extended and retracted vertically, while the second link is fixed, and the end effector can pick fruits by extraction action. During the task, the robot dynamically adjusts its strategy based on global field of view, including the location and ripeness of fruits and distribution of tree trunk. The episode ends when the robot successfully picks 3 ripe fruits or the number of steps exceeds the maximum limit (for example, 2000 steps). It should be noted that if the task is not completed within the allowed number of steps, the robot will not receive additional rewards.

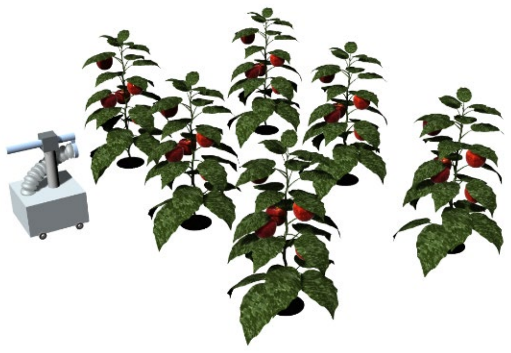


Figure 1 Orchard Environment

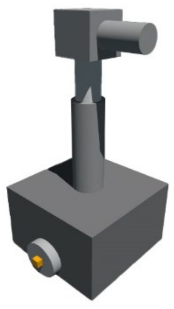


Figure 2 Simplified Robot Model

2 Implementation of Fruit-picking Robot

2.1 RL Cast

The RL framework is also different from the previous version, we have modified it more suitable for subsequent training. The changes mainly focus on reducing the size of state space and keeping independent actions, as well as reward adjustments.

• **State Space:**

The state space includes descriptions of the states of the robot, fruits, and fruit trees, and stores these state variables in three maps [obs_map, pos_map, goal_map] respectively.

Robot State (x, y, h): x and y are the center of robot base position, h is the height of manipulator. Those data are saved in the map pos_map.

Goal State (x, y, z, ripeness): x, y and z are the 3D position of fruit, ripeness represents if the fruit is ripe (1) or unripe (0), which are all saved in the map goal_map.

Obstacles State (x, y, z): x, y and z show the location of tree trunks, which can be calculated for collision penalty. The information stored in the map obs_map.

• **Action Space:**

Now changed to 7actions (before 9 actions), deleted the horizontal extension and retraction.
1) 4 Robot moving actions: move Forward (0), move Backward (1), turn Left (2), turn right (3).
2) 3 Manipulator actions: increase height (4), decrease height (5), extract fruit (6).

• **Reward Structure:**

ACTION_COST = -1 (every movement penalty)
PICK_NOTHING_PENALTY = -3 (execute extract action but grab nothing)
PICK_UNRIPE_PENALTY = -5 (pick an unripe fruit)
COLLISION_PENALTY = -5 (collide with tree trunks)
OUT_OF_BOUNDS_PENALTY = -5 (move out of the boundaries)
MANIPULATOR_EXTEND_LIMITATION = -5 (robotic arm extends beyond its limitation)
PICK_RIPE_REWARD = 15 (pick a ripe fruit)
GOAL_REWARD = 50 (pick 3 ripe fruits and end this episode)

2.2 OpenAI Gym and Render

In order to test the influence of model complexity on convergence, we designed two different versions of the gym environment, one relatively simple and the other relatively complex, so that we can compare and adjust hyperparameters more specifically. The difference between two versions is the presence or absence of unripe fruits and the nonlinear transformation action of extracting fruits. But in general, the logic of the two versions is the same, and the implementation explanation is as follows:

When implementing the gym environment, the logic is mainly designed by writing a custom class inherited from *gym.Env*. Firstly, the basic properties of the environment are initialized through the *__init__()* method, including the action space *self.action_space* and the state space *self.observation_space*, to ensure that they match the task requirements of the agent. This method also can load the initial parameters of the orchard scene, such as the position, maturity of fruits and distribution of trees, and calls the *reset()* method to set the initial state. In the *reset()* method, the environment resets the state of agent and fruits, and returns the current observation value as the initial input of DRL model. Next, in the *step(action)* method, the environment updates its state based on the incoming action, including adjusting the movement of the agent or the extension of the robot arm, detecting collisions and boundaries, or fruit picking results. At the same time, the reward value is calculated based on the task performance, judging whether the task is completed, and returning new observations, rewards, and completion mark (finished this episode).

To support debugging and visualization, the *render()* method uses *Pybullet* to render the three-dimensional environment, intuitively showing the interaction process between the agent and the fruit. Through the cooperation of these methods, the logic of the entire environment is fully implemented, and the interface design is clear and easy to use.

2.3 Neural Network Structure

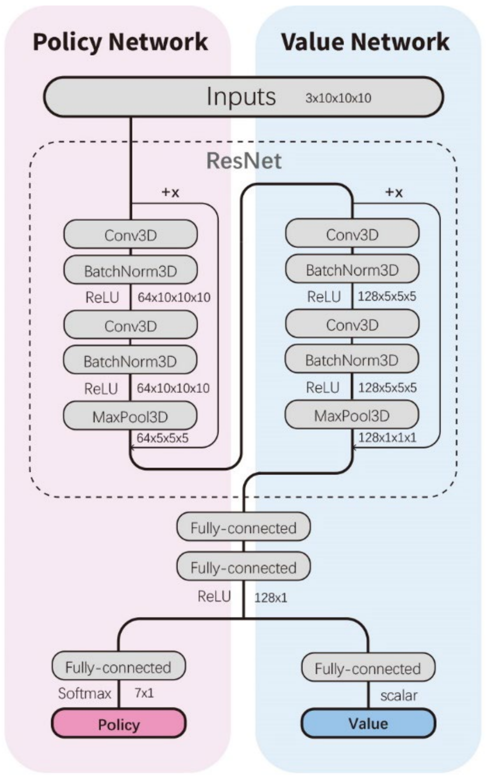


Figure 3 Neural Network Structure

For the part of neural network design (Figure 3), we also made some changes to it, mainly removing the LSTM structure and retaining only the ResNet residual network, followed by three fully connected layers to output policy and value respectively (the others are similar with the previous version, please check the 2nd submission in report for NN). The reason for removing the LSTM is that the learning agent already includes a memory module (old actor agent), which makes the LSTM somewhat redundant in this scenario and it may affect the training efficiency. This memory module will be discussed in detail in the next section.

In addition, we adjusted the middle channels of the network to make data processing more reasonable. Specifically, we increased the number of channels in the early layers of the network to more fully extract the features of the input data, while the number of channels in the later layers was appropriately reduced, which can effectively aggregate information and reduce computational complexity. The ResNet structure is retained because its advantage is to introduce residual connections, which can effectively alleviate the gradient vanishing problem in deep networks, accelerate training convergence, thereby improving overall performance.

2.4 Learning Agent

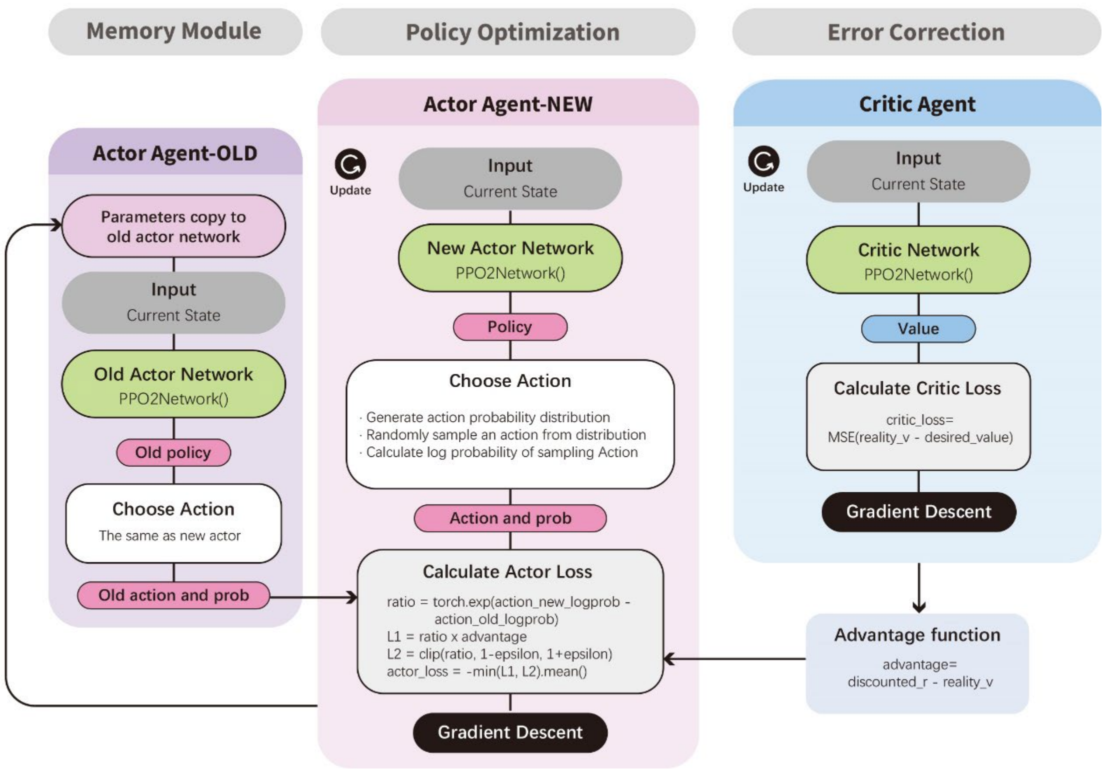


Figure 4 PPO2 Agent Structure

As for the PPO2 agent, we have made almost no changes compare to third submission and still use the previous version of the code. For details, please refer to the third report submitted. However, in the final report, I reorganized the logic of the PPO2 method and presented it in a more clarified diagram (as shown above in Figure 4). This PPO2 agent inherits the structure of Actor-Critic, but it also has differences. It adds a memory module, named old actor agent, and uses a clip mechanism to prevent the policy update from being too large to ensure stable updates.

Overall, the agent is mainly composed of three modules: the new actor agent for policy optimization, the old actor agent for storing memory, and the critic module for error correction. Although these three modules have different functions, their inputs are all the current state, and they are processed by the common neural network *PPO2Network* to output the required policy probability and value scalar values respectively. (All formulas in this process can be referred to the last report.) Finally, by calculating the actor-loss and critic loss, as well as performing backpropagation, the neural network is updated to optimize the strategy.

The learning agent, as the core module connecting gym environments and neural network, plays a vital role. It is not only responsible for coordinating and managing the flow of information during the training process, but also ensures that the model can adjust the strategy in time according to the feedback from the environment in each step of training to achieve the purpose of optimization.

3 Runing Results

3.1 Training Process

We trained several different versions of parameters and models, and all results can be viewed in the "results" folder of our submission. Next, I will show four representative results and discuss them in detail. The table below shows the hyperparameters of these four policies and for each parameter's meaning, you can check the code comments through 'ppo2agent.py'.

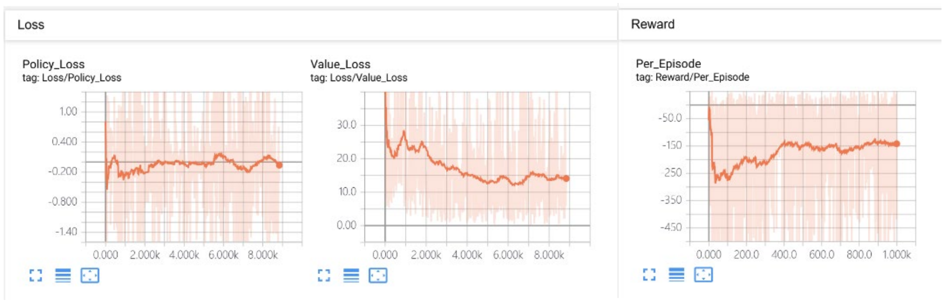
Table - PPO2 Agent Hyperparameters

PPO2 Agent Hyperparameters	**Simply Model**	**Complex Model**		
	Policy1.1	Policy 2.1	Policy 2.2	Policy 2.3
EP_MAX	1000	1000	1000	1000
EP_LEN	2000	2000	2000	2000
GAMMA	0.8	0.8	0.8	0.6
A_LR	0.0001	0.0005	0.0001	0.0001
C_LR	0.0003	0.0008	0.0003	0.0003
BATCH	128	128	128	128
A_UPDATE_STEPS	10	10	10	10
C_UPDATE_STEPS	10	10	10	10
kl_target	0.01	0.01	0.01	0.01
lam	0.5	0.5	0.5	0.5
epsilon	0.2	0.2	0.2	0.2

3.1.1 Simply Environment without Extract Action

For the simple gym environment, we set all fruits are ripeness and the robot only executes 6 actions (index 0 to 5, without extracting fruit action). The state space is relatively smaller comparing to the complex one and we assume that it will result in more positive results and can converge faster, which will guide us to master the approach to adjust the parameters for complex gym model. (Please check the video "*Video_PPO2_easy.mp4*" referring to Policy 1.1.)

- **Policy 1.1:**

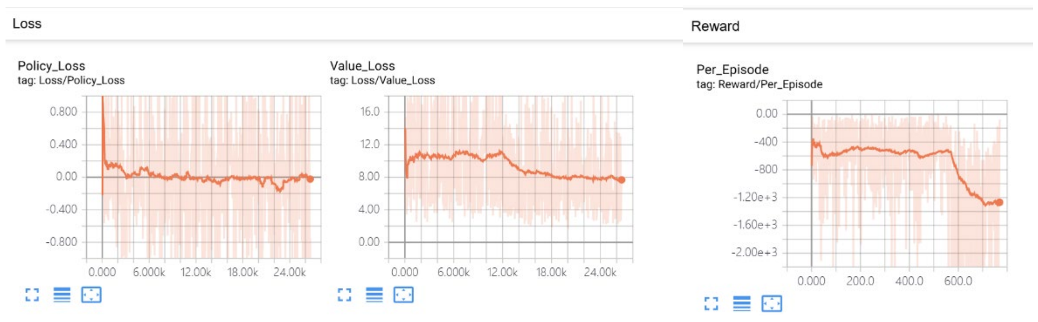


3.1.2 Complex Environment with Extract Action

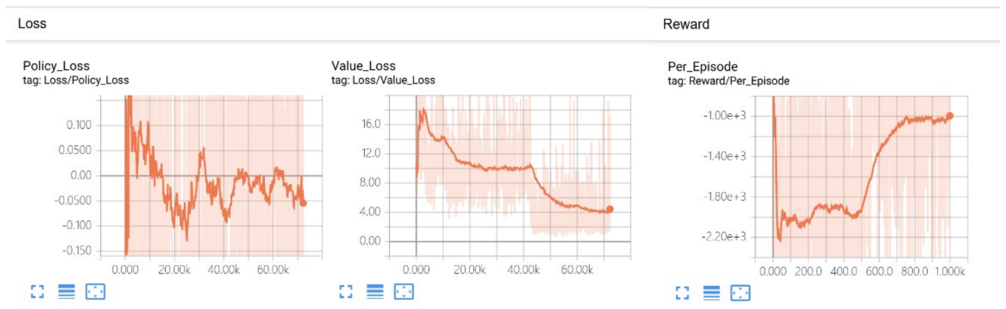
For the complex gym environment, we have both ripe and unripe fruit and all 7 actions of our fruit-picking (index 0 to 6, includes extracting fruit action). Due to the larger scale of state space and non-linear action, the convergence is relatively harder and it need more time to train a better solution. In addition, this model is very sensitive to the adjustment of hyperparameters. Therefore, we have experimented many times and got the most satisfactory result (Policy 2.3) so far.

(Please check the video "*Video_PPO2.mp4*" referring to our best solution Policy 2.3.)

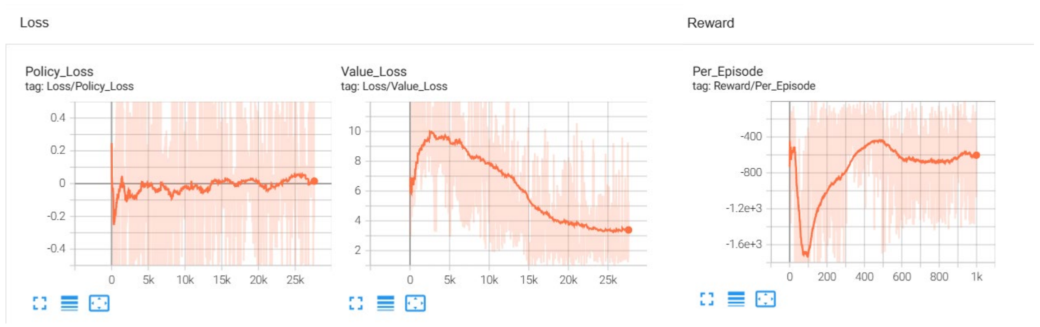
- **Policy 2.1:**



- **Policy 2.2:**



- **Policy 2.3:**



3.1.3 Hyperparameter Adjustment Analysis

From the results of Policy 1.1 and 2.2, it is clear to see that when the learning rate for actor and critic agent is 0.0001 and 0.0003, it can converge after around 600 episodes but loss fluctuates violently. However, it is even worse if it changes to 0.0005 and 0.0008 (Policy 2.1), the reward comes to the local optimal solution. Therefore, I keep the better learning rate and set the GAMMA value smaller (say 0.6) to Policy 2.3, which considers more for long-term returns. As the diagrams shown, actor loss is gradually stabilizing and critic loss goes down smoothly, while rewards increase smoothly as well. All those values are getting better from 200 episodes and stabilize at around 500 episodes.

3.2 Testing Process

The testing process utilized the most satisfactory policy – Policy 2.3, which has the best performance in both loss convergence and higher total reward. For every 100-episode test, the average total rewards waves from -720 to -810, which almost matches the reward value in the second half of training (around -790). And within one minute rendering, the robots can pick 3 ripe fruits and end this episode successfully. But for Policy 2.1 and 2.2, they are not able to get all goals as fast as Policy 2.3, and robot can even not pick any fruit within 5 mins by Policy 2.1.

4 Discussion

4.1 Advantages and Limitations of Our Approach

The PPO2 method has shown significant advantages in our autonomous fruit-picking robot project. As a DRL algorithm based on Actor-Critic, PPO2 effectively improves the stability of learning through the combination of policy optimization and value function estimation. Especially in our project, it requires complex decision-making and long-term training in high-dimensional discrete state spaces, but PPO2 can achieve efficient exploration and utilization through adaptive policy updates, thus improving the performance of robots in different environments.

However, PPO2 also has certain limitations. Since it relies on continuous policy updates, larger amount of computation and storage are needed, and the training time is longer with large training data. In addition, although PPO2 uses the clip mechanism to limit excessive policy updates for complex environments and action spaces, it may still face the problem of slow convergence or falling into a local optimal solution, which requires more tuning to improve adaptability.

4.2 Comparison with State-of-the-art Conventional Alternative

To have a better understanding of the difference between DRL and conventional method, we also implement our autonomous fruit-picking robot by A* heuristic algorithm. This algorithm is mainly used for shortest path planning, and its core logic is based on the heuristic function:

$$f(n) = g(n) + h(n)$$

Where g(n) is the cost of moving from the starting point to the specified grid, while h(n) is the estimated cost of moving from the specified grid to the end point.

We adapted the A* algorithm to the gym interface and made some fine-tuning on the action logic. (Please refer to the code and video in the folder '***a_star_agent.py***'.) Specifically, the framework of the gym environment *class FruitPickingEnv* is retained, while the picking logic is simplified: remove the fruit extracting action at the end effector (end effector now moved to the robot center), and ignore unripe fruits (only generate ripe fruits). When running, the rendering video shows that the robot can quickly plan a path to the specified fruit. However, A* method will recalculate the path every time a fruit is picked, and then executing the simple retraction logic of the robot arm, so it cannot achieve long-term planning and overall strategy learning like PPO2.

4.3 Challenges Encountered and Lessons Learned

In the process of implementing fruit-picking robot, we encountered several technical challenges. First, the custom design of gym environment and the definition of state space is really a complex process. To ensure that the robot can perform tasks effectively, we need to accurately simulate the interaction between the robot and the fruit in the environment, especially to model the robot's actions in a high-dimensional discrete space. Second, when using DRL to train the learning agent, we faced the problem of slow policy updates and convergence, especially in complex environments. Although the PPO2 method provides relatively stable policy updates, the training process is still slow due to the complexity of the action space (the action of extracting fruits) and the high-dimensionality of state space. Through this process, we learned how to improve training efficiency by adjusting hyperparameters, optimizing neural network architectures, and refining reward mechanisms. In general, by solving these problems, we not only deepened our understanding of PPO2 algorithms, robot control, and environmental modeling, but also improved our ability to deal with complex problems in practical applications.

4.4 Potential Future Works

In the future, we can improve the robot's environmental perception ability by accessing sensor data processing, in order to have more realistic interaction. At the same time, we can explore continuous multi-task learning methods based on DRL, such as picking fruits and go back to the start point, so that the robot can adaptively adjust its strategy in different environments and achieve more flexible and efficient fruit picking operations.

4.6 Use of Existing Code

The code of PPO2 agent partially refers to the tutorial on the Internet [4]. The main logic follows the blogger's structure, but we have made many modifications. The original project is suitable for a simple continuous environment, while our task is to deal with a complex high-dimensional discrete environment, so we have adjusted the input tensor and the calculation method of the loss equation to meet our needs. The traditional A* algorithm logic also borrowed from the tutorial on GitHub [5] and learned its framework, but we made a lot of customized changes to better adapt to our specific problems and environment.

5 Conclusion

This project is centered around an autonomous fruit-picking robot, and aims to achieve autonomous decision-making and fruit picking in complex environments through deep reinforcement learning method (PPO2) and traditional path planning algorithms (A*) respectively. We trained both PPO2 agent and A*, and tested their learning ability and planning effect. During the testing phase, PPO2 showed strong global planning and dynamic adaptability, while A* showed superiority in short-term path planning with its intuitive and efficient logic. In the future, we may combine DRL with state-of-the-art conventional algorithm to achieve a stable and more intelligent learning agent. In addition, our next efforts also include continuously optimizing the real-time and efficiency of the model.

6 References

[1] Alex Krizhevsky et al, ‘Imagenet classification with deep convolutional neural networks’, NeurIPS 2012.

[2] Laith Alzubaidi, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”, Journal of Big Data, 2021

[3] John Schulman, 'Proximal Policy Optimization Algorithms', arXiv:1707.06347

[4] PPO2 link: https://zhuanlan.zhihu.com/p/538486008

[5] A* link: https://github.com/while-TuRe/A-star-ShortestPath