Application of Q-Learning Neural Network in Quadrotor Path Planning with Limited Environment Perception

Yupeng Li YUPENGLI@SEAS.UPENN.EDU
Jiatong Sun JTSUN@SEAS.UPENN.EDU
Junfan Pan JFPAN@SEAS.UPENN.EDU

Abstract

For general obstacle avoidance trajectory and path planning for quadrotors, it is common to use A* and Dijkstra's algorithms for a given map. However, when it comes to an unknown environment and given limited perception of the environment, these two algorithms would not hold anymore. Thus we are trying to use reinforcement learning neural network to resolve this problem as proposed in (Bing-Qiang Huang, 2005)

1. Introduction

Obstacle avoidance is a key part for autonomous quadrotors. For environments already known, there have already been optimal and fast path generation algorithms such as A* and Dijkstra. These algorithms can help navigate the quadrotors when the map and obstacles are already revealed before planning the path. However in real world situations like search and rescue post disaster and archaeological excavation, we don't always have access to the whole map. Thus, a method which can navigate the quadrotors in complex and unknown environments is highly demanded.

Reinforcement learning can be applied in these situations where the quadrotors can then learn the desired actions like turning and forwarding as outputs providing current states including position, acceleration orientation, angular velocity and surrounding obstacles within visible range. For each pair of state x and action a, the quadrotor is trying to learn the state-action value $Q(x, a)$ which is the maximum discounted reward that is possible to be achieved at state x.

Since there are large number of state-action pairs in autonomous quadrotor obstacle avoidance problem, the Q values are stored using a trained neural network. The neural network here serves as a table where we can look up the expected Q values given state and action. Also, it allows the quadrotor to reasonably infer Q values for state-action pairs that has not been visited.

The neural network can be warmed up with existing state-action pairs which come from A* path planning on maps already given. Since the output path of A* algorithm using a suitable heuristic is optimal, we can consider all the actions taken in A* as the actions maximize $Q(x, a)$. Based on these facts, we generate random maps and use sandbox to simulate and output state-action pairs for all the points visited and train a 3-layer Q value neural network with these pairs.

The warmed up neural network can then be used to train the quadrotor in maps unknown and keeps updating until the Q network converges through the training process. The quadrotor will then navigate itself in unknown environments using the Q-value based decisions, get subsequent rewards from the decisions, update Q based on the rewards and train the neural network with the updated Q value.

2. Methods

2.1. Generate Random Maps

In order to initialize the neural network, a random map generator is required. This random map can generate a world map with randomly positioned blocks within a user-defined boundary, and generate random start and goal position for the quadrotor. Besides, for the training path to be valid, we must guarantee the existence of path. Therefore, several space constraints need to be satisfied among those randomly generated blocks and also between the start and goal point.

To create random obstacles, a class method called **random_block()** is created inside the class World. This class

University of Pennsylvania, CIS 419/519 Course Project.

method returns a world object describing boundary, random blocks volume and position, start and goal position parameterized by input arguments, which are lower bound and upper bound of the world, block width, block height, number of blocks, robot radius and safety margin. In addition, all space constraints are satisfied during generation process. For instance, the distance between two random blocks should be larger than two times of safety margin plus robot radius, blocks should never go beyond the boundary, start and goal position should be at least one margin plus robot radius from the closest point on the blocks, etc.. After that, all values of boundary, block, start point and goal point are stored in a list of dictionaries called world_data and then can be directly fed into Sandbox to generate flight simulation output, which are also our training data.

2.2. Obtain State-Action Pairs

Based on the randomly generated maps, A* algorithm is implemented with L2 norm as heuristic to generate an optimal trajectory. The trajectory then returns a Pandas Dataframe containing timestamp, positions, poses and commands.

The random generated map is fed into the sandbox. Then, based on the given control strategy, the function simulates how the quadrotor moves and saves all information with respect to the timestamp. Describing the information in detail, this function gives out 3D position, 3D position command, 3D velocity, 3D velocity command, quaternion, quaternion command, 3D angular velocity, rotation speed of each motor, 3D torque and thrust given by each motor. Finally, this function saves all the information as well as the corresponding timestamp into a pandas dataframe for further use.

2.3. Q-values, Rewards and Algorithm

Based on the state and action values from Section 2.2, we warm up the Q-value neural network with $Q(x,a) = 1$ for all the points in the planned trajectory, $Q(x,a) = -1$ for going into obstacles and walls and $Q(x,a) = 0$ for other state-action pairs.

Then we follow the algorithm 1 to learn Q and update the neural network in unknown environments until the quadrotor learns how to make an optimal decision. Note that in the algorithm, x stand for current state, a is the action taken, y is the actual next state,r is reward which will be covered below, $0 < \gamma < 1$ is the discounted factor and α is the learning rate.

Algorithm 1 Q-learning Algorithm

```
Input: Warmed-up Neural Network $M$
repeat
  Get current state $x$
  Determine optimal action st. $a^*(x)$=argmax$Q(x,a)$
  Quadrotor moves and obtains new state $y$
  if Collision Occurs then
    $r = -1$ and return to starting position
  end if
  Update $Q$ using
    $Q(x,a) = Q(x,a) + \alpha[r + \gamma max Q(y,a) - Q(x,a)]$
  Calculate the difference between updated $Q$ and original $Q$ to train the neural network
until $Q$ converge to $\epsilon$
```

The reward function is determined using the equation in (Richter C., 2016) as follows:

$$r = -\rho(\frac{d + D_n - D_c}{D_c}) - \gamma(dt - \frac{d}{V_{max}}) - \beta|\omega|$$

Where ρ, γ and ω are hyper parameters which respectively emphasize on the choice of locally optimal path, choice of the maximum possible velocity and choice of taking a straight path. d,D_c and D_n respectively stands for the distance travelled in the current time step by taking present action a, shortest distance to the destination before and after taking the specified action a.

References

Bing-Qiang Huang, Guang-Yi Cao, Min Guo. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. volume 1, pp. 85–89. IEEE, 2005. ISBN 0-7803-9091-1. doi: 10.1109/ICMLC.2005.1526924.

Richter C., Roy N., et al. Learning to plan for visibility in navigation of unknown environments. *2016 International Symposium on Experimental Robotics*, pp. 387–398, 2016. doi: https://doi.org/10.1007/978-3-319-50115-4_34.

3. Appendix

3.1. Group Members and Responsibilities

Yupeng Li Constructed the neural network reinforcement learning structure, algorithm and set up the reward function.

Jiatong Sun Wrote a script to generated output from quadrotor flight trajectory sandbox which can be used to train the neural network directly.

Junfan Pan Wrote a script which can generate random maps which can be the fed into the sandbox mentioned above for further reinforcement Q-learning