Proceedings of the 37th Chinese Control Conference
July 25-27, 2018, Wuhan, China

Monocular Vision based Autonomous Landing of Quadrotor through Deep Reinforcement Learning

Yinbo Xu[1], Zhihong Liu[1], Xiangke Wang[1]

1. College of Mechatronic Engineering and Automation, National University of Defense Technology, Changsha 410073
E-mail: 1324261257@qq.com
zhihonglzh2005@163.com
xkwang@nudt.edu.cn

Abstract: An improved deep reinforcement learning (DRL) method is proposed to solve autonomous landing problem of quadrotor. Autonomous landing is a significant function for unmanned aerial vehicle (UAV) such as quadrotor. Previous solutions are mainly based on relative position calculation or the landmark detection, which either needs massive additional sensors or lacks intelligence. In this paper, we focus on realizing autonomous landing through DRL method. Whole landing process is implemented by an improved deep Q-learning network (DQN) based end-to-end control scheme. Only one down-looking camera is used to capture raw images directly as input states. An Aruco tag is placed at the landing region for feature extraction. Double network and the dueling architecture are applied to improve DQN algorithm. Besides, the reward function is well designed to fit the auto-landing scenario. The experiments show that the improved DQN can make the quadrotor land on the landmark successfully and achieve better performance while comparing to the original deep Q-learning solution.

Key Words: Unmanned Aerial Vehicle (UAV), Autonomous Landing, Deep Reinforcement Learning (DRL), DQN

1 Introduction

In the past few years, the autonomous landing problem of UAVs is of great importance, which plays an increasingly important role in geological survey, disaster rescue, intelligent logistics and daily entertainments etc [1]. In many applications, the autonomous landing is mainly relied on GPS. However, GPS is not a reliable service as its availability can be limited by urban canyons and is completely unavailable in indoor environments. Therefore, the use of monocular vision is very attractive to solve this problem because in places where the GPS is difficult to use, there are usually a lot of visual features.

Existing works on the vision-based auto-landing problems mostly focus on the position-based visual servo (PBVS) and the image-base visual servo (IBVS) [2]. The PBVS is required to get the relative position and attitude between the UAV and the ground land region through the visual system. Whileas the IBVS integrates the visual system into the design of the landing control law. The information of image pixel level is fully utilized so that the estimation of the relative position and attitude is avoided [3]. With the rise of artificial intelligence (AI), especially the deep learning technique, these applications are bound to step further in intelligent.

Deep learning catches people`s eyes by its remarkable performance in robotics and computer vision [5]. And the supervised deep-learning based quadrotor control scheme is becoming increasingly popular. Giusti [6], for example, has trained a deep neural network composing of several convolutional neural networks to enable a quadrotor to fly along an obstacle-fulfilled forest path. Nevertheless, in these work, enough labeled training data must be collected due to the characteristic of supervised learning, which is a time-consuming and labor-intensive work.

The reinforcement learning does not need the prepared labeled training data. It explores policies through trials. However, when facing a huge state space, the reinforcement learning is limited. So, the combination of the deep learning and the reinforcement learning, namely the deep reinforcement learning (DRL) [7], works. DRL has recently been shown to achieve outstanding performance on Atari series games [8]. Besides, DRL also has been applied to robotics. A D3QN model, for instance, has been trained to the obstacle avoidance problem successfully for a ground turtlebot indoor [9]. What`s more, the trained model from simulation can be seamlessly transferred to the real world environment. However, there are not so many cases where DRL is used on UAV platform especially in autonomous landing problem. Riccardo [10] tried to apply DRL to auto-landing, but the landing process is divided into two phases factitiously which makes the method be unpractical.

In this paper, we focus on using a DRL based end-to-end control scheme, a way being different from traditional methods, to realize the autonomous landing of the quadrotor. Our approach uses a monocular down-looking camera as the only sensor to capture raw images input to a hierarchy of Deep Q-Networks (DQNs). An Aruco marker is placed at the landing area for the sake of feature extraction of images. Directly the deep network outputs high-level speed commands which drive the quadrotor to the landing area. Inherited from Q-learning, DQN cannot overcome the inherent drawback, namely overestimation. Our scheme used Double DQN [11] (DDQN) to reduce overestimation. Dueling architecture is also applied to optimize DQN in the level of network architecture.

The main contribution is using an improved DQN to realize autonomous landing problem of quadrotor. More specifically, the quadrotor can be trained to autonomously land in complex unknown environment without any priori

*This work is supported by National Natural Science Foundation (NNSF) of China under Grant 61403406.

knowledge or labeled data. And the landing process can be finished in one phase from the departure position to landing area without hovering on the x-y plane to detect the land mark firstly.

2 Problem Definition

In this section, the auto-landing problem is modeled from the aspect of DRL. The autonomous landing of quadrotor can be considered as a decision-making process. As shown in Fig.1, the the only monocular camera interacting with environment, the input states are defined as the raw camera image. So deep neural network is used to extract features of such high dimensional state. The reward comes from the designed reward function which works according to the current state. The outputs of DRL model are the actions referring to the speed in quadrotor in x and y directions. Speed in z-direction is not considered in our model for which it is set to a fixed value. That is to say, the quadrotor is driven to the landing region in vertical direction no matter whether or not the quadrotor has aimed at the marker properly. This approach can narrow the action space, consequently making the problem a little bit easier. Besides, such setting does not conflict with the actual situation.

The quadrotor chooses an action $a_t \in \mathrm{A}$ according to state x_t at time $t \in [0, \mathrm{T}]$, observes a reward signal r_t and transits into next state x_{t+1}. The goal of the algorithm is to maximize the accumulative future reward $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$ in which γ is the discount factor.

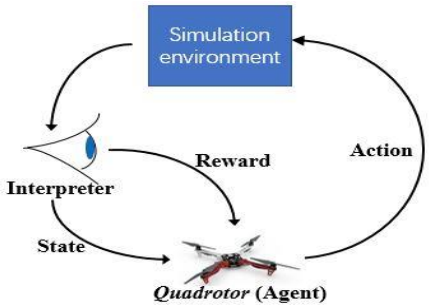

Fig. 1: The process of quadrotor interacting with environment

Given the policy $a_t = \pi(\mathrm{x}_t)$, the action-value (Q-value) is related to state-action pair $(\mathrm{x}_t, \mathrm{a}_t)$. So, Q-value can be defined as follows:

$$Q^{\pi}(\mathrm{x}_t, \mathrm{a}_t) = \mathrm{E}[\mathrm{R}_t, \mathrm{x}_t, \mathrm{a}_t, \pi] \tag{1}$$

The equation (1) can be rewritten as equation (2) by using Bellman equation

$$Q^{\pi}(\mathrm{x}_t, \mathrm{a}_t) = \mathrm{E}[\mathrm{r}_t + \gamma \mathrm{E}[Q^{\pi}(\mathrm{x}_{t+1}, \mathrm{a}_{t+1}), \mathrm{x}_t, \mathrm{a}_t, \pi]] \tag{2}$$

By choosing the optimal action each time where $Q^{*}(\mathrm{x}_t, \mathrm{a}_t) = \max_{\pi} \mathrm{E}[\mathrm{R}_t, \mathrm{x}_t, \mathrm{a}_t, \pi]$, the optimal Q-value function is obtained

$$Q^{*}(\mathrm{x}_t, \mathrm{a}_t) = \mathrm{E}_{x_{t+1}}[r + \gamma \max Q^{*}(\mathrm{x}_{t+1}, \mathrm{a}_{t+1}) \mid \mathrm{x}_t, \mathrm{a}_t] \tag{3}$$

which indicates that the optimal Q-value at time t is the current reward r_t plus the discounted optimal Q-value available at time t+1. Rather than computing the Q-value function directly over a large state space, the optimal policy can be obtained through approximating this Q-value function with deep neural network, which is the main principle behind DQN. The optimal function can be obtained through following update mode:

$$Q(\mathrm{S}_t, \mathrm{A}_t) \leftarrow \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t) + \alpha(\mathrm{R}_{t+1} + \lambda \max_{a} \mathrm{Q}(\mathrm{S}_{t+1}, \mathrm{a}) - \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t)) \tag{4}$$

It is worthy to point that that the target Q-value calculated by value iteration is not been given to new Q directly. A factor α is multiplied to the target Q-value before update with the aim of controlling the speed of approaching the target Q-value, which can minimize the impact brought by the error of estimation. When the optimal Q-value function is obtained, the optimal action-choosing policy comes as follows:

$$\pi(\mathrm{S}_{t+1}) = \arg\max_{a} Q(\mathrm{S}_{t+1,}\mathrm{a}) \tag{5}$$

3 Deep Q Learning for Monocular Vision Based Autonomous Landing

This part shows our end-to-end autonomous landing control framework and our improvements on DQN. In Fig. 2, the input state of control scheme is down-looking camera images. The scheme mainly consists of two parts. One is deep neural network with dueling architecture. The other is double DQN algorithm. Conclusively speaking, after four stacked raw images being input as current state, three CNNs are followed by two fully connected layers for tow streams of dueling architecture. The output of which is high-level speed commands in x and y directions. That is how a DRL based end-to-end control scheme is constructed.

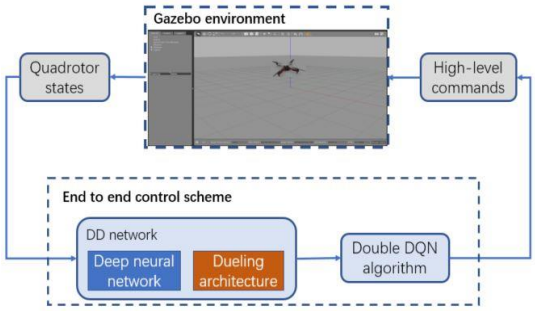

Fig. 2: The framework of DRL end-to-end control scheme

3.1 Double Deep Q-Network Architecture

In conventional methods, the network used to choose the optimal action a^{*} and estimate the target Q-value as well. This causes that there is correlation between data, and the training process will never be stable. To solve this problem, the prototype DQN in [11] introduces a target network alongside an online network. The online network, which is used to select actions, updates its trainable parameters through back-propagation at each training step; while the weights of the target network are fixed over a short period and then copied from the online network. As a duplication of the online network, the target network is responsible for estimating Q-value and then calculating the error. This two-network framework helps to eliminate the problem of

overestimation. The predicted Q-value can be described as follows:

$$Y_t^{DoubleQ} \equiv R_{t+1} + \lambda Q(\mathrm{S}_{t+1}, \mathrm{argmaxQ}(\mathrm{S}_{t+1}, \mathrm{a}; \theta_t^-); \theta_t) \quad (6)$$

in which the θ_t^- represents the parameters of online network while the θ_t belongs to target network. Fig. 3 has shown how the double Q network works.

Specifically speaking, the states are collected from a simulation environment, such as Gazebo. The states are put in memory pool to be sampled when needed. This trick can break the links between data which otherwise could cause an unstable and divergent training process of neural network. x_t is the current state while x_{t+1} is the resulting state. When given other elements including action a, reward r, and discounted factor λ, the training procedure works. $\oplus$, $\ominus$ and $\otimes$ represent the operations for addition, subtraction and multiplication.

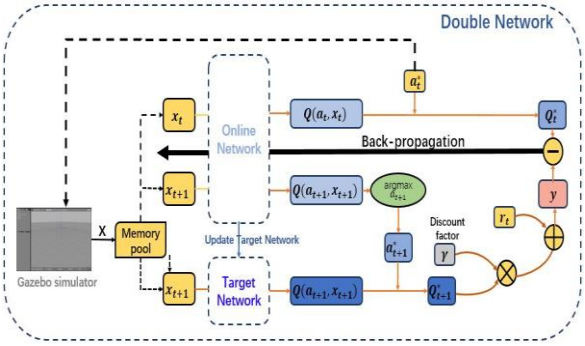

Fig. 3: The training procedure of Double Q Network

3.2 Dueling Network Structure

The tradition DQN has only one stream of fully connected layers which is constructed after the convolution layers to estimate the Q-value of each action-state pair. Considering the fact that for some states, the choice of action is useless to the development of training process and it is unnecessary to evaluate every action, Wang et al. [12] propose the dueling network. In the dueling architecture, two streams of fully connected layer are constructed to compute the state value function and the advantage function separately, namely

$$Q^{\pi}(\mathrm{s},\mathrm{a}) = \mathrm{V}^{\pi}(\mathrm{s}) + \mathrm{A}^{\pi}(\mathrm{s},\mathrm{a}) \quad (7)$$

Then these two are added together to compute Q-values. The comparison of the dueling architecture and the traditional network with only one stream is shown in Fig. 4 [12]. The top one is the traditional network and the down one is the dueling architecture.

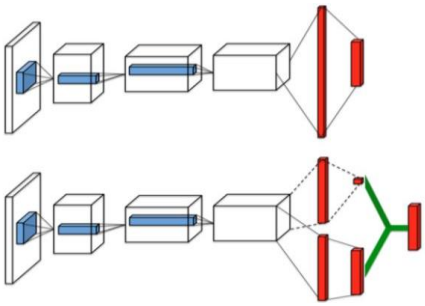

Fig. 4: The dueling architecture comparing with the traditional one.

The main neural network of the work is briefly shown in Fig. 5 and the dueling architecture is also included in the right part. It shows clearly how the dueling architecture is integrated into our end-to-end control scheme.

As shown in Fig. 5, four RGB images are stacked as the input state, which is followed by deep networks. Several convolutional neural networks (CNN) are constructed to extract features of current state. The output of CNN is divided into two streams of fully connected layers, making up a dueling network. So, the state value and advantage are calculated separately. Finally, the speed of x-direction and y-direction are predicted in parallel.

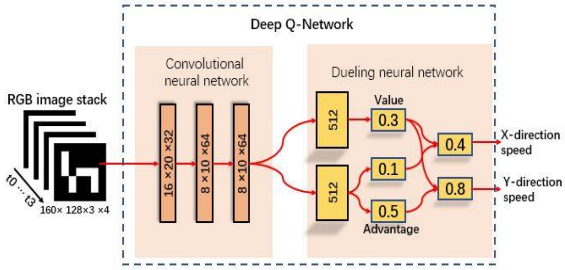

Fig. 5: The framework of deep Q-network

Dueling network has demonstrated a significant improvement on not only performance but also training speed in a number of Atari games. And the double network also enables AI to match professional human player while playing Atari games. With these two techniques, the improved DQN tends to be more data efficient and find a good policy faster.

3.3 Related Algorithm

The proposed algorithm is mainly the double DQN algorithm. One network is updated through training, responsible for calculating Q value. The other is updated by coping the former`s parameters, responsible for choosing the action. And the two networks are updated alternatively. Table 1 shows the double q-learning algorithm.

Table 1: Double Q-learning algorithm

Algorithm Double Q-learning

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

4 Experiment

4.1 Experimental Settings

The improved DQN is based on the double and dueling network. Their architectures are represented in Fig. 2 and Fig. 4 respectively.

As shown in Fig. 6, the training process is conducted in Gazebo, bringing us the convenience of respawning the quadrotor agent repeatedly. The movements of quadrotor are defined as "forward, stop and backward" in both x and y direction. There are obvious oscillatory effects during accelerations and decelerations that introduces a swinging behavior with consequent perspective distortion in the image acquired. Moreover, the inertia of quadrotor influences the motion when a new velocity command is given one by one. Too high speed will cause the network having no enough time to response to the fast change of state. While too low speed will consume too much time in one episode.

The reward function is elaborately designed. In the training phase, the position information of marker and the agent is used to construct the reward function. If the quadrotor is right up on the marker at the initial altitude, it will receive the maximal reward denoted as *MAX_R*. If the deviation from center position exists, the reward will be reduced. And the maximal reward changes when the altitude differs. The lower the height, the less the maximal reward. So, the positive reward function is defined as follows

$$r = MAX\_R \times (z/\ init\_z) \times \alpha - d \times \beta \tag{8}$$

where z refers to the current height of quadrotor while *init_z* is the initial altitude, and d is the distance to the center position. α and β are the adjustment coefficients which are empirical values coming from a considerable amount of experiments. If the view of down-looking camera lost the marker, the current trial is considered to be failed and current episode terminates immediately with an additional punishment of -10. Consequently, the movement of quadrotor is limited in an upside-down four-sides cone which is shown in Fig. 6 using red flying-zone.

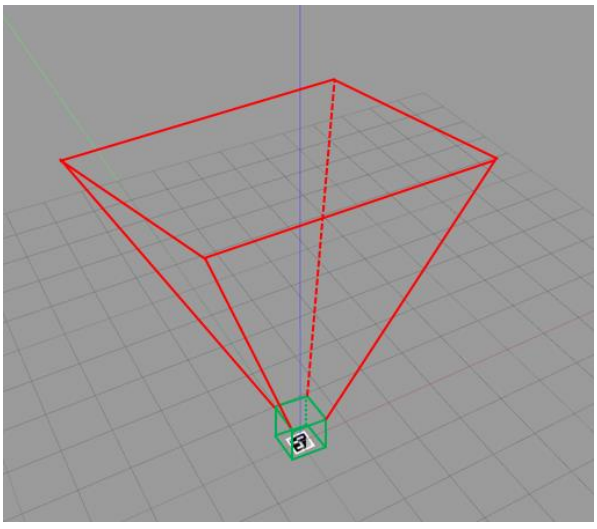

Fig. 6: The simulation environment and the flying zone

A height threshold is set to be 0.53 m for we found when the quadrotor is below a certain height, it easily deviates from the center position, even though the motion is slight. Besides, too low height makes it impossible for camera to capture a complete image of marker, which will confuse the agent to judge whether the marker is lost in the view or not. So, the reward function is slightly different when the quadrotor is below the height threshold. A small zone is designated below the threshold, which is shown in Fig. 5 using red cube. If the quadrotor is outside the cube, the current episode ends with the punishment of -5. Otherwise, the reward of each step still goes as formula (8).

Each episode will last until the agent reaches the maximum number of steps (650 steps in our setting) and ends with no punishment. What`s more, if the quadrotor is too low, the model will also be respawned and the episode comes to an end with no punishment. The reward of each step in one episode will be added to get the total episode reward.

The improved DQN model is trained in Gazebo simulation environment. And the original deep Q-learning model is baseline. The marker we use is the Aruco marker26. To train the network to generate feasible control policy, the alternative values of velocity commands are set to be -0.15m/s, 0m/s or + 1.5m/s which will produce nine kinds of behaviors. The initial altitude is set to be 2.8m. Correspondingly, the speed in z-direction is set to be 0.1m/s. The initial position in x-y plane is fixed. A laptop (Gazebo 7.9.0, ROS Jade) equipped with a NVIDIA GEFORCE GTX 960 GPU is used to train deep neural network.

4.2 Experimental results

The total accumulated reward of one episode is an important evaluating indicator. Tensorboard is a powerful tool which provides us a method of visualization of the data we care about. Fig. 7, which comes from Tensorboard, has shown the smoothed reward curves of the improved DQN model. We can see that the whole tendency of the curve is growing up. After about 200 episodes of training, the accumulated reward of one episode tends to be stable. The stable reward value is around 140. We also notice that the performance of the improved DQN is not always better than that in the previous episode. Because the vibrations of the curve do exist even in the stable phase where the accumulated reward tends to be maintained at 140. Nevertheless, the changing process of the curve, rising firstly and then tending toward stability, is quite obvious which indicates that the deep neutral network has learned how to control the quadrotor to fly and land on the Aruco landmark.

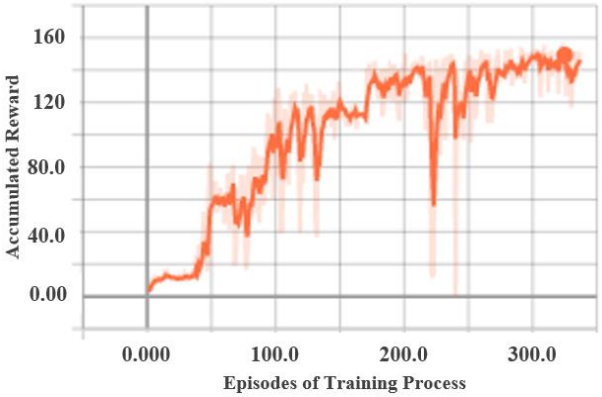

Fig. 7: Smoothed curve of reward with improved DQN

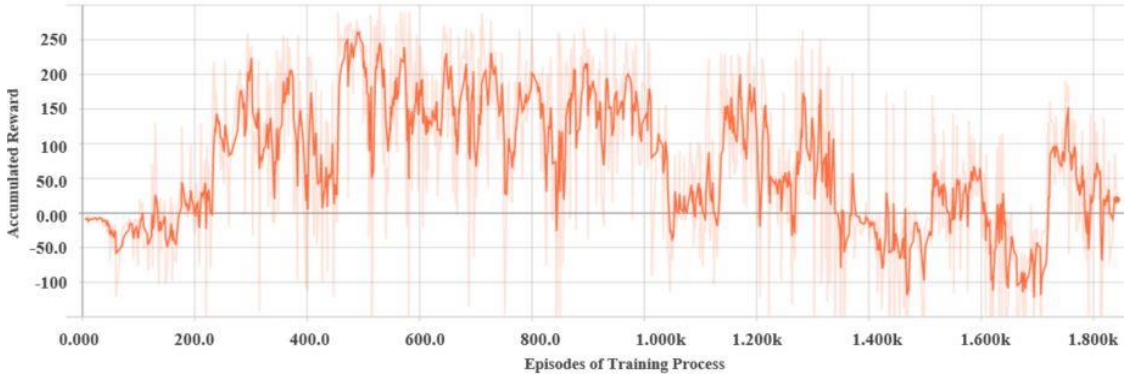

Fig. 8: Smoothed curve of reward with original deep Q-learning

As a contrast, the original deep Q-learning based experiments are also conducted. Fig. 8 has shown the result of accumulated reward curve. We can see that the curve ascends in the initial phase but oscillates violently. To observe the possible convergence and stability of the curve, we have conducted the experiments reaching up to more than 1800 episodes. However, the training process shown great instability. What`s worse is that the total reward tends to decline after around 600 episodes. All these phenomena indicate that the original deep Q-learning behaves unstably when applied to autonomous landing of quadrotor.

To furtherly demonstrate the results of our experiments and the comparison between the improved DQN and the original one, we take the td_error (time difference error) into consideration. Fig. 9 shows the td_error which is collected during the training process of the improved DQN. The td_error falls significantly with the training going on. After 300 episodes of training, it seems to reach a stable state where the value of the td_error is very close to zero.

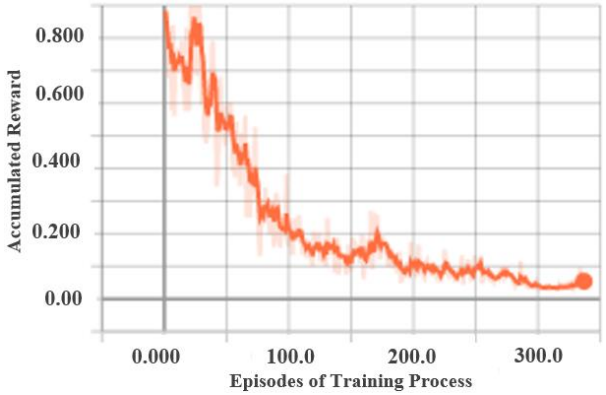

Fig. 9: Smoothed curve of td_error with improved DQN

As a contrast, Fig. 10 shows the td_error collected during training process of the original deep Q-learning model. The curve also descents in the first 200 episodes but fluctuates all the time. Such unstable performance cannot satisfy the requirements of landing the quadrotor on the landmark. And the central value of the fluctuation is around 0.200 which is far from the value in the improved DQN model. In the later period of training, the td_error jumps drastically, which means the original deep Q-learning model has diverged.

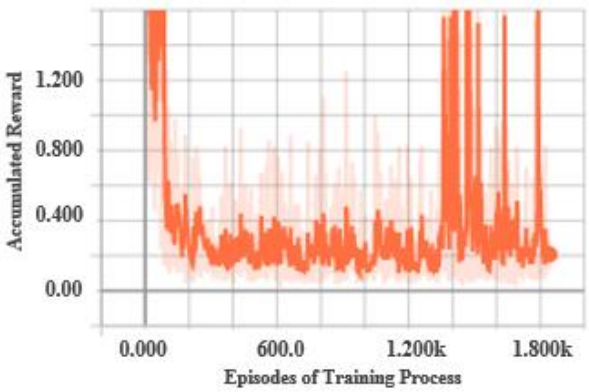

Fig. 10: Smoothed curve of td_error with original deep Q-learning

By comparison, indicators including accumulated reward and the td_error have reflected that the improved DQN can converge to a stable and acceptable state, and enables our quadrotor to acquire a stable accumulated reward

During the later training phase of the improved DQN model, we observed that the quadrotor can land on the landmark successfully. To further verify the effectiveness of the improved DQN when applied to quadrotor autonomous landing problem, we conducted separate confirmatory tests. In our tests, the well-trained network parameters are loaded at the beginning. And the train step is ignored which means network is no longer be updated. Fig. 11 is the stitching image which is made up of the consecutive images captured by down-looking camera in one successful auto-landing test.

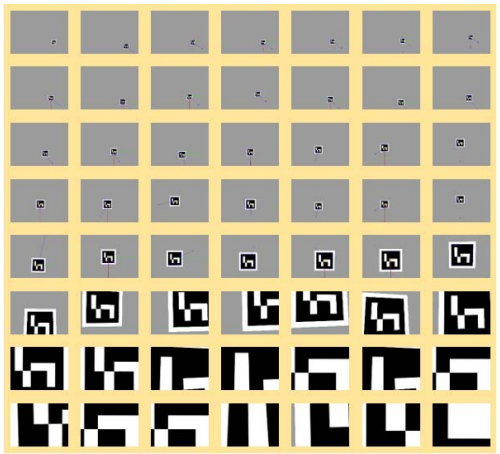

Fig. 11: Consecutive image in one successful test

We can see that the quadrotor always keeps the landmark in its field of view and eventually land on the landing region successfully. To make our result more persuasive, we have conducted more than one tests with a fixed initial position. Rviz helps us to visualize the trajectory of quadrotor. Fig. 12 is the Rviz screenshot which shows the trajectories of quadrotor in six tests with the same initial position. The trajectories are not exactly same with each other but they have the same landing destination.

Our neural network is trained with one initial position. To our satisfaction, the trained DQN still performs well when we change the initial position. Fig. 13 shows the trajectories of four tests with different typical initial position. The initial positions are located in completely opposite directions along x and y axes. However, our quadrotor can land on the landing region successfully with the improved DQN, which further demonstrates the effectiveness of our method when applied to autonomous landing problem.

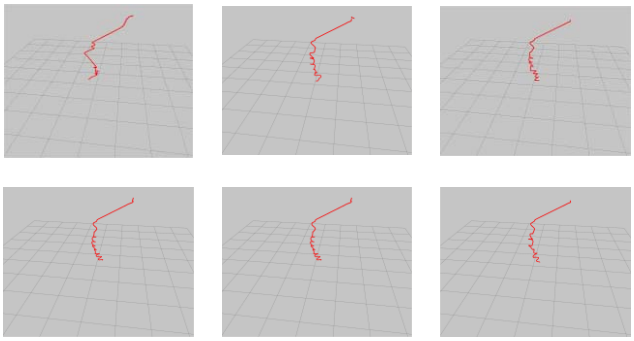

Fig. 12: Six tests with same initial position

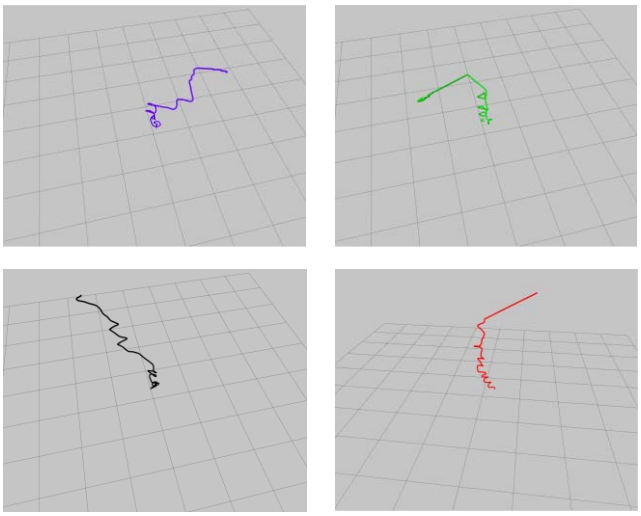

Fig. 13: Four tests with different initial position

5 Conclusions

In this paper, an improved deep reinforcement learning method (DQN) is proposed to solve autonomous landing problem of quadrotor. One down-looking camera image is taken as input of the agent state for the network. Our efforts on the improvements of DQN mainly focus on double and dueling network techniques. Besides, a reward function is properly designed to train our improved DQN model. Evaluating indicators including accumulated reward and td_error are considered, and the demonstration shows remarkable performance of the improved DQN method. A considerate amount tests with same and different initial positions are conducted and the results have shown the effectiveness of the improved DQN when applied to autonomous landing problem.

Our future attention will be paid to training the network in a more complex environment with more noise and transferring the well-trained network to the real-world tasks. Training the network with the labeled data to initial the parameters is also in our consideration.

References

[1] Adams S M, Friedland C J. A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management[C]//9th International Workshop on Remote Sensing for Disaster Response. 2011: 8.

[2] Chaumette F, Hutchinson S, Visual servo control. I. Basic approaches[J], IEEE Robotics & Automation Magazine, 2006, 13(4): 82-90.

[3] Zhang Y, Yu Y, Jia S, et al. Autonomous landing on ground target of UAV by using image-based visual servo control[C]//Control Conference (CCC), 2017 36th Chinese. IEEE, 2017: 11204-11209.

[4] Sen Wang, Ronald Clark, Hongkai Wen, Niki Trigoni, R Clark, S Wang, H Wen, N Trigoni, A Markham, A Markham, et al. Deepvo: towards end-to-end visual odometry with deep recurrent convolutional neural networks. In ICRA, 2017.

[5] Ronald Clark, Sen Wang, Niki Trigoni Andrew Markham, and Hongkai Wen. Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization. In CVPR, 2017.

[6] Giusti A, Guzzi J, Cireşan D C, et al. A machine learning approach to visual perception of forest trails for mobile robots[J]. IEEE Robotics and Automation Letters, 2016, 1(2): 661-667.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, and Alex et. al Graves. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[9] Xie L, Wang S, Markham A, et al. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning[J]. arXiv preprint arXiv:1706.09829, 2017.

[10] Polvara R, Patacchiola M, Sharma S, et al. Autonomous Quadrotor Landing using Deep Reinforcement Learning[J]. arXiv preprint arXiv:1709.03339, 2017.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, and Alex et. al Graves. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[12] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. CoRR, abs/1511.06581,2015.