Path Planning Method Using Dyna-Q Algorithm under Complex Urban Environment

1st Jingyi Huang
Sino-French Engineer School
Beihang University
Beijing, China
pierre1231@buaa.edu.cn

2nd Qingke Tan
China Academy of Launch Vehicle Technology
Beijing, China
tanqingke@126.com

3rd Jiming Ma*
Sino-French Engineer School
Beihang University
Beijing, China
jiming.ma@buaa.edu.cn

4th Liang Han
Beihang Hangzhou Innovation Institute Yuhang, Beihang University
Hangzhou, China
liang_han@buaa.edu.cn

***Abstract*—Path planning and obstacle avoidance problems are now the focus of robotics research. This paper uses the Dyna-Q reinforcement learning algorithm to implement an obstacle avoidance and a path planning algorithm for unmanned ground vehicle(UGV) under urban environment. Using the reinforcement learning algorithm, we calculate the waypoints of the unmanned vehicle and achieve obstacle avoidance tasks and path planning using a vector field. Finally, we use a PID controller on unmanned aerial vehicle (UAV) to realize the air-ground collaboration task. The algorithms and the agents' modeling in this paper are implemented in the lab's simulation platform.**

***Index Terms*—Dyna-Q, path planning, UGV, simulation platform**

I. Introduction

The intelligent robot gets more popular as technology develops rapidly. All kinds of intelligent robots are used to improve productivity and reduce costs. Obstacle avoidance and path planning tasks are the foundation of other complex mission.

Reinforcement learning algorithms have been developed for nearly half a century and have demonstrated powerful capabilities in the robotics domain. Reinforcement learning algorithms can be applied to all aspects of intelligent robotics, including robot control, motion planning, decision-making, and perception. Huy X. Pham [3] used Q-learning to perform simple UAV decision-making and planning while using PID for UAV control and an optical motion capture device for locating the actual UAV. Roland Siegwart's team [8] used DQN algorithm for end-to-end control of the UAV via MPC control. Guan-Ting Tu [4] used the Q-learning algorithm with a physical UAV to perform a real-world simulation from perception to control on Microsoft's AirSim simulation platform. Zhangjie Cao et al. [11] collaborated with Toyota and utilized a hierarchical reinforcement learning approach to complete a vehicle obstacle avoidance task in a Near-Accident Driving scenario.

This work was supported by the Zhejiang Provincial Natural Science Foundation of China under Grant LGG22F030025 and the Fundamental Research Funds for the Central Universities.

The urban road scenario is a typical scenario for UAV and UGV to perform air-ground collaboration tasks. The vehicle must shuttle through all kinds of buildings to avoid obstacles effectively. The UAV must follow the vehicle to give air support.

Many researchers chose reinforcement learning for decision-making and planning [6] [7], followed by using a PID controller. In this paper, we first complete the dynamic modeling of the UGV and UAV. We simplify the complex urban environment into a grid world. Then we use the Dyna-Q reinforcement learning algorithm to make decisions for the UGV, calculate the waypoints, and use the vector field [2] [5] [9] for path planning. Finally, we use a PID controller for throttle and steering angle control. A UAV is then added to perform the air-ground collaboration task using PID controller.

The contribution of this paper is as follows :

- Complete the development and testing of UGV dynamics with a self-developed simulation platform named Potato
- Propose decision-making and path planning algorithm using Dyna-Q method and vector field
- Complete the UGV and UAV air-ground collaboration task in an urban road environment

The paper is organized as follows: Section II introduces the system model and dynamic models of UAV/UGV. Section III presents the Dyna-Q algorithm deployed in the self-developed simulation platform in Section IV. Simulation and results analysis are presented in Section V, and we make a conclusion in Section VI.

II. Modeling

A. System Modeling

In this paper, the map is reduced to an $M*M \in Z^2$ grid world with sides of 100 meters. Each cell have two attributes:

a building with a height of 100 meters (obstacle) and flat land where a vehicle can pass. We calculate the optimal path by Dyna-Q algorithm to give the specified waypoints.And the waypoints will only appear in the center of the cell.

We defined the action of the UGV as going four directions(north, south, west, and east). Each step of the UGV would be going into the center of the next adjacent cell. The UGV used in the experiments is Ackermann steering vehicle, which has more practical significance than Mecanum wheel vehicle which is commonly used in the laboratory.

B. UAV Modeling

In this paper, we use a quadrotor UAV, and therefore we need to build the dynamic model of the quadrotor in our simulation. Quadrotors are widely used in many scenarios, but building a quadrotor model is difficult. It is tough to build a perfect dynamic model due to its complex aerodynamic characteristics. A quadrotor produces gyroscopic effects when the motors rotate at high speeds, and the airframe is subject to complex air drag at high speeds. However, the additional forces and moments mentioned above can be neglected due to the highly symmetrical nature of the quadrotor and its low-speed characteristics. Therefore, we can model the dynamics of quadrotor using the Newton-Euler equations by considering only the gravitational force of the quadrotor and the lifting force T_i and torque M_i generated by the rotation of the four rotors. The kinetic equations are as follows :

$$F = m\dot{V} \tag{1}$$

$$\tau = I\dot{\Omega} + \Omega \times I\Omega \tag{2}$$

where F is the total external force on the UAV, m is the mass of the UAV, V = $[v_x;\ v_y;\ v_z]^{\mathrm{T}}$ is the velocity. τ is the total external moment on the UAV, I is the inertia tensor matrix of the UAV, and Ω = $[p;\ q;\ r]^{\mathrm{T}}$ is the three-axis angular velocity of UAV in the UAV coordinate system B. Due to the symmetric layout of the quadrotor fuselage, the UAV's inertia tensor matrix can be approximated as a diagonal matrix, defined as I = diag($I_{xx};\ I_{yy};\ I_{zz}$)

Thus here, F and τ are :

$$F = R_b^n \begin{bmatrix} 0 \\ 0 \\ -\sum_{i=1}^{4} T_i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = R_b^n \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{3}$$

$$\tau = \begin{bmatrix} (T_2 - T_4)L \\ (T_1 - T_3)L \\ (M_1 + M_3 - M_2 - M_4) \end{bmatrix} = \begin{bmatrix} u_2 L \\ u_3 L \\ u_4 \end{bmatrix} \tag{4}$$

where the R_b^n represents the rotation matrix from the navigation coordinate system to the body coordinate frame.

The attitude angle is described by the ZYX Euler angle, denoted as Φ=$\{\phi, \theta, \psi\}^{\mathrm{T}}$, and the three components are called the roll angle, pitch angle, and yaw angle respectively. The transformation relationship between the Euler angular velocity

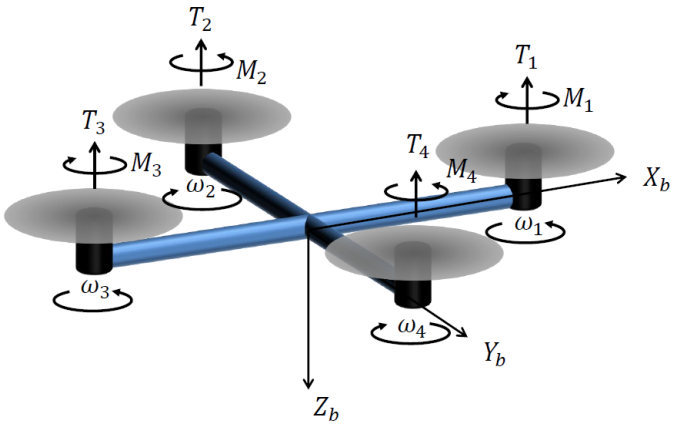

Fig. 1: UAV dynamic model

of the quadrotor and the three-axis angular velocity of UAV in the UAV coordinate system is :

$$\dot{\Phi} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \sin\phi\tan\theta \\ 0 & \cos\phi & \sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \Omega \tag{5}$$

When the pitch and roll angle are small enough, we can consider that $\dot{\Phi}$ = Ω, we can get the following formula :

$$\begin{cases} \ddot{\phi} = \dfrac{u_2 L}{I_{xx}} + \dot{\theta}\dot{\psi}\dfrac{I_{yy} - I_{zz}}{I_{xx}} \\ \ddot{\theta} = \dfrac{u_3 L}{I_{yy}} + \dot{\phi}\dot{\psi}\dfrac{I_{zz} - I_{xx}}{I_{yy}} \\ \ddot{\psi} = \dfrac{u_4}{I_{zz}} + \dot{\phi}\dot{\theta}\dfrac{I_{xx} - I_{yy}}{I_{zz}} \end{cases} \tag{6}$$

Assuming that the quadrotor flies near the hovering state, the dynamic model of the quadrotor can be simplified to the following form :

$$\begin{cases} \ddot{x} = (-\phi s\psi - \theta c\phi)g \\ \ddot{y} = (-\theta s\psi + \psi c\phi)g \\ \ddot{z} = -\frac{u_1}{m} + g = -\frac{\Delta u_1}{m} \end{cases} \tag{7}$$

The dynamic model of the quadrotor can therefore be reduced to a second-order system. Since all channels are decoupled, we can independently control the UAV's position in different directions in the simulation.

C. UGV Modeling

The dynamic model of the UGV has a very high degree of non-linearity, making it challenging to build an 'adequately accurate' dynamic model. Based on the dynamic model built by ETH for their racing car AMZ [1], we establish one simpler dynamic model of the UGV. It is based on the following assumptions: 1) The vehicle travels on flat ground. 2) The

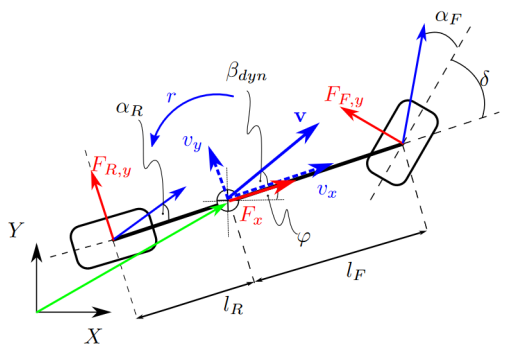

Fig. 2: Dynamic model built by ETH for their racing car AMZ.

body does not produce massive drift. 3) The longitudinal force of the body acts on the center of mass.

The dynamic model of the UGV are as follows :

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\phi} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos\phi - v_y \sin\phi \\ v_x \sin\phi + v_y \cos\phi \\ r \\ \frac{F_{R,x} - F_{F,y}\sin\delta}{m} \\ \frac{F_{R,y} + F_{F,y}\cos\delta}{m} \\ \frac{F_{F,y} l_F \cos\delta - F_{R,y} l_R + \tau_{TV}}{I_z} \end{bmatrix} \tag{8}$$

where X, Y are the position in world coordinate. ϕ denote the yaw angle, v_x and v_y represent the velocity decomposition on body frame, and r is angular acceleration. $F_{a,b}$ (a $\in \{F, R\}$ and b $\in \{x, y\}$) represents the force on the front/rear tire on direction x/y. l_d (d $\in \{F, R\}$) is defined as the distance between the front/rear tire and the center of mass. I_z is the moment of inertia on z. And τ_{TV} is an additional yaw torque generated by an underlying torque vectoring controller.

To calculate the forces between the tire and the ground, we need to build the tire model. Here we use the Pacejka tire model to calculate $F_{R,y}$ and $F_{F,y}$:

$$F_{R,y} = D_R \sin(C_R \arctan(B_R \alpha_R)) \tag{9}$$

$$F_{F,y} = D_F \sin(C_F \arctan(B_F \alpha_F)) \tag{10}$$

where B, C and D are empirical parameters, and α are defined as follows :

$$\alpha_R = \arctan(\frac{v_y - l_R r}{v_x}) \tag{11}$$

$$\alpha_F = \arctan(\frac{v_y + l_F r}{v_x}) - \delta \tag{12}$$

When it comes to the longitudinal forces, we use the drive-train model, F_x is defined as follow :

$$F_x = C_m D - C_{r2} v_x^2 \tag{13}$$

where $D \in$ [-1,1] represents the driver command of throttle. To control the car, we apply the steering angle δ and D as the input.In the ETH's article, they chose $\Delta\delta$ and ΔD as inputs. δ and D can only change continuously in real life, so the input design is quite close to reality. However, since we do not consider drift and the UGV in our experiment is always at a low speed, choosing δ and D as inputs is acceptable.

III. Reinforcement Learning and Dyna-Q

Reinforcement learning originated in the 1950s, flourished in the 1980s and 1990s, and produced another explosive breakthrough in the last decade. Reinforcement learning can be understood as studying a good strategy for intelligent agent. Agent continuously learns the optimal policy during its interaction with the environment until the decision sequence has the highest payoff. The process of reinforcement learning can be described as the Markov Decision Process (MDP). A tuple is defined to describe the learning process, which contains state space, action space, state transfer probability, reward, and discount factor. However, it is difficult for an agent to observe the entire state space completely for real-life scenarios, so the partially observable Markov decision processes (POMDP) are proposed as an ideal model for decision-making in uncertain environments. After the model is determined, the state value function and action value function are determined using the Bellman equation.

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s') \tag{14}$$

where V^π means the value using policy π, $R(s, \pi(s))$ is the reward got by taking step $\pi(s)$ in state s. s' denote the next state, and γ is the discount factor.

Based on the selection of the agent actions, we can classify the reinforcement learning algorithms into two categories: value-based algorithm and policy-based algorithm. The value-based approach calculates the value function and selects the policy corresponding to the maximum value function; the policy-based approach selects the action and updates the policy by maximizing the cumulative return.

Q-learning is a classical value-based reinforcement learning algorithm, and by improving it, we can obtain the Dyna-Q algorithm, which is a classical model-based reinforcement learning algorithm. Dyna-Q uses a method called Q-planning to generate simulated data based on a model and then uses the simulated data and actual data to improve the policy. Q-planning picks one state at a time that has been visited, takes an action that has been performed in that state, gets the transferred state and the reward by the model, and updates the action value function with Q-learning based on this simulated data.

IV. Simulation Platform Development

We designed a novel swarm simulation platform called Potato for swarm intelligence research. Most of the simulation platforms are now designed for agents under ten. Adding more agents becomes a catastrophe for users. So our lab decided to develop a simulation platform optimized for the swarm with over 1000 agents.

Algorithm 1: Dyna-Q algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

repeat

$S \leftarrow$ current (nonterminal) state

$A \leftarrow \epsilon$-greedy(S, Q)

Take action A; observe resultant reward R, and state S'

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

$Model(s, a) \leftarrow R, S'$

repeat

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S, A)$

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

until loop repeat n times

until end of the training

A. Platform Introduction

The Potato platform is a simulation platform developed by the Intelligent Swarm Laboratory from Beihang University. Potato is optimized for multi-agents simulation needs and can support simulation experiments of more than a thousand agents. The structure of Potato is developed by imitating the structure of ROS. This simulation platform is mainly divided into three parts: the display side, the algorithm side, and the central control side. The display side uses the cesium engine, which can visualize the algorithm on the world map. The agent can be operated, and tasks can be set using the mouse on a computer (windows). The agent can perform the attack, interception, resupply, monitoring, communication support, and other tasks. Developers can develop on the algorithm side and embed new algorithms for simulation and verification. The central control side contains mainly the dynamic model.

We rigorously defined concepts such as vision and communication support for large-scale simulation scenarios when designing this platform. We have also defined the agent's three basic capabilities, five modes, and three commands. The three basic capabilities are $hold$, $move$, and $follow$. Furthermore, we provide an offboard interface, through which developers can control the agent to validate their algorithms. Based on the three basic capabilities, we have designed a variety of modes and commands.And that will eventually lead the platform to be operated as an RTS game.

So far, our lab developers have completed scenario demonstrations of UGVs firing missiles against aerial targets and UGVs marching in formation, initially proving the effectiveness and reliability of the Potato platform. The developers can also manually turn on the agents' view field display on the display side. The battlefield situation can be easily accessed.

Each agent is given a serial number, and the first digit represents the type of agent. As can be seen in the figure: 3 is for UGV, and 5 is for missile. The last four digits of the serial number represent different individuals within the same

Fig. 3: Initial interface of the Potato platform.

Fig. 4: Scenario of UGV launching missile (The red/blue sphere in the bottom right corner shows the view field of other agents).

type of agent. At the bottom of the platform interface lies the timeline. Manual operation to adjust the timeline at the end of the simulation is available. We can replay the simulation results, rewind the operation and multiply the speed.

B. Platform Development

We are responsible for the development and testing of the UGV model. The development of UGV is divided into three parts: $dynamic\ model$, $autopilot$, and $path\ follower$.

We refer to the ETH team's modeling of their race car AMZ, and we modify the control quantities to steering angle and throttle. The drift term in the dynamic model is deleted in our modeling. So in the final dynamic model, UGV drives on flat ground and does not drift.

Autopilot is the control layer, where the target speed and heading angle are entered, and the control quantities can be obtained.

Path follower is the path planning layer, which can make UGV possess the three essential capabilities. In order to realize that, we need to program the UGV to be able to follow a given path and hold still at a certain point. Owing to the complexity of the trajectory planning algorithm, the author chose the path planning algorithm instead. Meanwhile, the path planning algorithm functions perfectly in all our scenarios. The path planning technique is to design a virtual vector field around the path. At each point of the space lies a vector indicating the desired heading angle and velocity. The vector field will

eventually lead UGV to follow the path. For a given path, the difference between UGV's heading angle and the path direction is defined as follows:

$$\chi_d(e_{py}) = -\chi^{\infty}\frac{2}{\pi}\tan^{-1}(k_{path}e_{py}) \tag{15}$$

where $\chi_d(e_{py})$ represents the desired heading angle when the distance between agent and path equals e_{py}. χ^{∞} is the desired heading angle when e_{py} equals infinity. And k_{path} is a positive constant that adjusts the changing speed of χ_d when e_{py} changes.

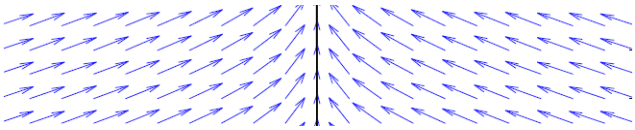

Fig. 5: Vector field

When the UGV is on the path, the desired heading angle is the same as the path direction. Here we set χ^{∞} equals $\frac{\pi}{2}$. When the UGV is very far away from the path, the desired heading angle of the UAV will be almost perpendicular to the path direction. The UGV should also be capable of stopping at a specified given location and holding still. Similarly, we design a vector field that points to where the UGV will eventually stop. Here we used a segmented approach where the UGV will stop throttle after being within 15m of the target point.

V. Simulation Experiment, Results and Analysis

We manually create a simple grid map using the technique mentioned in the previous section. The action space can be simplified to walking up/down/left/right. The reward is designed to be minus one for each step taken until the endpoint is reached. The Q-table is obtained by training with the Dyna-Q reinforcement learning algorithm. After initializing the location and the endpoint, waypoints are generated by the Q-table, and the path is obtained. After initializing the positions of the UAV and UGV, the simulation starts running. The task of the UGV is to avoid obstacles while reaching the endpoint. And the task of the UAV is to follow the UGV throughout the process.

Algorithm 2: Simulation Methodology

```
Initialize map and start/end point
Train using Dyan-Q and get waypoints
Set waypoints on the platform
Create path message p_i on the platform
Simulation starts:
while UGV do not reach the endpoint do
    UGV follows the present given path p_pre
    UAV follows UGV (without changing the altitude)
    if Distance UGV-Next_waypoint < 5 meters then
        Estabulish the vector field of the next path
        UGV follows the next path: p_pre ← p_{i+1}
    end
end
```

Fig. 6: Potato platform with Dyna-Q simulation.

Fig. 7: Potato platform with obstacle avoidance task.

The algorithm 2 shows the methodology.

The desired path is divided into multiple segments or subpaths; each subpath consists of a line from the previous waypoint to the following waypoint. When UGV gets within five meters of the next waypoint, it starts to take a turn(if it exists) and follow the next subpath. Due to the nature of the map, the angle of each turn(if it exists) is 90 degrees. Different speeds are set in different subpaths (20m/s for the first subpath and 30m/s for the rest) to simulate the actual situation and test the stability of the UGV's steering. The performance of a 120m fixed height UAV following UGV under speed-changing circumstances can be examined.

In the UGV path following, UGV follows the desired path well. It can be seen from the figure that UGV follows the next section of the path very quickly after turning. The perpendicular euclidean distance defines the error between the path and UGV. The path followed shifts at each turn, resulting in a sudden change in the error graph. Despite the slight overshoot, UGV succeeded in following the path at each turn. This proves the effectiveness of the reinforcement learning algorithm and path following algorithm used in this paper. Figure 9 shows the relative distance of the UAV to UGV in the XY plane. It demonstrates that the UAV moves quickly to the position of the UGV at the beginning and keeps a close distance during the following process. Even if UGV accelerates, the UAV follows very well after a quick adjustment.

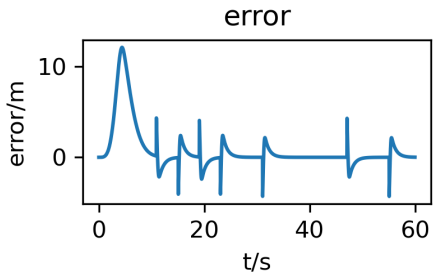

Fig. 8: Error between path and UGV.

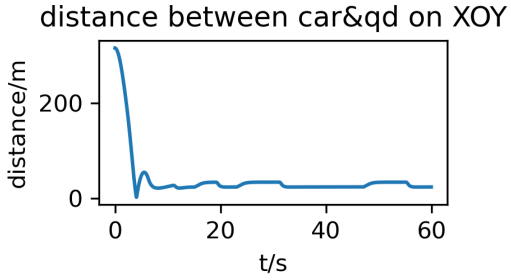

Fig. 9: Error between UAV and UGV.

VI. Conclusion

The experimental validation of the simulation platform shows that the Dyna-Q algorithm and the vector field based path planning method are adequate for the agent in path following and obstacle avoidance tasks. The simulation platform developed by our lab can complete the verification and simulation of algorithms at different levels. With regard to the simulation, some limitations need to be acknowledged. The dynamic model can be improved. In many large cities, skyscrapers will block the UAV's path. we will add taller buildings in the simulation to complete the UAV's following/obstacle-avoidance task.

References

[1] Kabzan J, Valls M I, Reijgwart V J F, et al. AMZ Driverless: The full autonomous racing system[J]. Journal of Field Robotics, 2020, 37(7): 1267–1294.

[2] Boroujeni Z, Mohammadi M, Neumann D, et al. Autonomous Car Navigation Using Vector Fields[A]. 2018 IEEE Intelligent Vehicles Symposium (IV)[C]. 2018: 794–799.

[3] Pham H X, La H M, Feil-Seifer D, et al. Autonomous UAV Navigation Using Reinforcement Learning[J]. arXiv:1801.05086 [cs], 2018.

[4] Tu G-T, Juang J-G. Path Planning and Obstacle Avoidance Based on Reinforcement Learning for UAV Application[A]. 2021 International Conference on System Science and Engineering (ICSSE)[C]. 2021: 352–355.

[5] Galceran E, Eustice R M, Olson E. Toward integrated motion planning and control using potential fields and torque-based steering actuation for autonomous driving[A]. 2015 IEEE Intelligent Vehicles Symposium (IV)[C]. 2015: 304–309.

[6] Naveed K B, Qiao Z, Dolan J M. Trajectory Planning for Autonomous Vehicles Using Hierarchical Reinforcement Learning[A]. 2021 IEEE International Intelligent Transportation Systems Conference (ITSC)[C]. 2021: 601–606.

[7] Hubmann C, Quetschlich N, Schulz J, et al. A POMDP Maneuver Planner For Occlusions in Urban Scenarios[A]. 2019 IEEE Intelligent Vehicles Symposium (IV)[C]. 2019: 2172–2179.

[8] Hwangbo J, Sa I, Siegwart R, et al. Control of a Quadrotor With Reinforcement Learning[J]. IEEE Robotics and Automation Letters, 2017, 2(4): 2096–2103.

[9] randybeard/uavbook: Repository for the textbook: Small Unmanned Aircraft: Theory and Practice, by Randy Beard and Tim McLain[EB/OL]. GitHub. /2022-08-15. https://github.com/randybeard/uavbook.

[10] ros/ros: Core ROS packages[EB/OL]. /2022-08-15. https://github.com/ros/ros.

[11] Cao Z, Biyik E, Wang W, et al. Reinforcement Learning based Control of Imitative Policies for Near-Accident Driving[A]. 2020, 16.