Autonomous Waypoints Planning and Trajectory Generation for Multi-rotor UAVs

Yilan Li*
Syracuse University
Syracuse, NY
yli41@syr.edu

Hossein Eslamiat*
Syracuse University
Syracuse, NY
heslamia@syr.edu

Ningshan Wang
Syracuse University
Syracuse, NY
nwang16@syr.edu

Ziyi Zhao
Syracuse University
Syracuse, NY
zzhao37@syr.edu

Amit K. Sanyal
Syracuse University
Syracuse, NY
aksanyal@syr.edu

Qinru Qiu
Syracuse University
Syracuse, NY
qiqiu@syr.edu

ABSTRACT

Autonomous trajectory generation in a complex environment is a challenging task for multi-rotor unmanned aerial vehicles (UAVs), which have high maneuverability in three-dimensional motion. Safe and effective operations for these UAVs demand obstacle avoidance strategies and advanced trajectory planning and control schemes for stability and energy efficiency. To solve those problems in one framework analytically is extremely challenging when the UAV needs to fly large distance in a complex environment. To address this challenge, a two-level optimization strategy is adopted. At the higher-level, a sequence of waypoints is selected that lead the UAV from its current position to the destination. At the lower-level, an optimal trajectory is generated between each pair of adjacent waypoints analytically. While the goal of trajectory generation is to maintain the stability of the UAV, the goal of the waypoints planning is to select waypoints with the lowest control thrust consumption throughout the entire trip while avoiding collisions with obstacles. The entire framework is implemented using deep reinforcement learning, which learns the highly complicated and non-linear interaction between those two levels, and the impact from the environment. A progressive learning strategy is investigated that not only reduces convergence time but also improves result quality. We further investigate and provide results regarding the tuning of gains in the optimal trajectory scheme using genetic algorithm. The experimental results demonstrate that our proposed approach is able to generate a list of obstacle-free waypoints with minimum control energy and develop an optimal trajectory with optimized platform velocity, acceleration, jerk and control thrust.

CCS CONCEPTS

• **Computing methodologies** → *Motion path planning*;

KEYWORDS

Deep Reinforcement Learning, Waypoints Planning, Optimal Trajectory Generation, Multi-rotor UAV

ACM Reference format:
Yilan Li, Hossein Eslamiat, Ningshan Wang, Ziyi Zhao, Amit K. Sanyal, and Qinru Qiu. 2019. Autonomous Waypoints Planning and Trajectory Generation for Multi-rotor UAVs. In *Proceedings of Design Automation for CPS and IoT, Montreal, QC, Canada, April 15, 2019 (DESTION '19),* 10 pages.
https://doi.org/10.1145/3313151.3313163

*These authors contributed equally to this work.

DESTION '19, April 15, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6699-1/19/04…$15.00
https://doi.org/10.1145/3313151.3313163

1 INTRODUCTION

Onboard real-time trajectory planning is particularly important in beyond visual line-of-sight (BVLOS) operations, and applications that require unmanned vehicles to move in cluttered and dynamic environments [6]. Such applications include indoor operations [4], package delivery in urban and suburban areas, monitoring of civilian infrastructures like bridges and highways, autonomous landing on moving platforms [22], and tracking wildlife in forested areas. Autonomous trajectory generation has received increased attention in the past decade [24][5][32], especially for autonomous systems such as unmanned aerial vehicles (UAVs). Safe and effective operations of these UAVs demand that we consider trajectory generation as a constrained optimization problem. While there are many different ways to formulate the objectives and constraints, the basic requirement is to consume minimum flight energy while avoiding obstacles and maintaining the UAV stability during the journey. To solve such problem analytically is extremely difficult if impossible when the UAV needs to fly large distance in a complex environment. Some of the classical approaches apply rapidly-exploring random trees [11] and voronoi graph [1]. In general, these approaches generate the feasible path by traversing the graph made by all possible paths and searching for possible links between path nodes. The convergence to the optimal path takes much time, and hence it cannot be applied during the real flight time. Furthermore, they either uses a large margin for obstacle avoidance, or adopt a trial-and-error based iterative approach, which will further increase the complexity. [27] implements a gradient decent approach which converges

This work is partially supported by the National Science Foundation under Grant CNS-1739748.

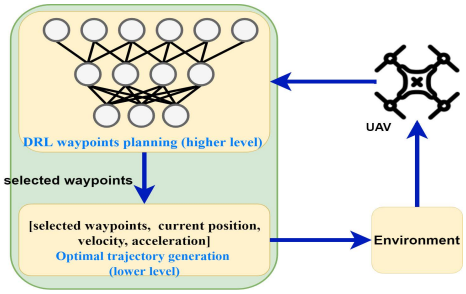


Figure 1: Overall framework of proposed autonomous waypoints planning and trajectory generation scheme.

more quickly without loss of robustness. However, non-smooth trajectory are produced that are difficult for UAVs to precisely follow. Neither does it consider other objective functions such as minimum control thrust.

Recently deep learning has drawn extensive attention in areas of robotics applications for its outstanding abilities to learn representations of complex environment. Such representation is essential for environment awareness for applications such as UAV trajectory generation. Among others, deep reinforcement learning is extremely suitable to solve goal-oriented robotics tasks that has close interaction with environment dynamics [8]. Such interaction provides feedback, which is useful to improve the performance of the task being learned. Reinforcement learning has been used for robot path planning in some previous works. Real-time model-based reinforcement learning framework [7] as well as Q-learning [2] are adopted to find path in 2D surface. These models consider only the robot (or UAV) status as the system state. The environment information (e.g. the location of the obstacles) is not part of the system state. In other words, these models learn the environment instead of learning the relationship between optimal control and the surrounding environment. When the environment changes, the model needs to be trained again. Such blindness to the surrounding environment is not realistic in today's UAVs. With the availability of 3D map and sensors such as the image and depth camera, the UAVs will have partial information about the environment. Environment information is used as an input in [26] and [12], where deep Q network (DQN) and deep deterministic policy gradient network, are deployed. However, without a lower level optimization, they are not capable to maintain the stability of UAVs or minimize the flight energy. At lower-level, [21][25] have decoupled time and geometry, constructed a geometric trajectory and then parameterized it in time. Utilizing differential flatness of dynamics to generate a trajectory is also adopted in [30]. Furthermore, [16] develops a high level trajectory generator in conjunction with a motion primitive generator to choose an optimized trajectory among different motion primitives. However, planning scheme with consideration only of the actuation model of UAVs is not enough, especially when maneuvers that go beyond hovering or level flight are required. In situations where large maneuvers are required, fast online trajectory planning schemes like the one proposed in this work become necessary.

In this paper, we tackle the problem of UAV trajectory generation in a known 3D environment by solving it as a two-level optimization. Figure 1 shows the overall framework. At the upper-level, a sequence of waypoints is selected that lead the UAV from its current position to the destination, and at the lower-level, the optimal trajectory between each pair of adjacent waypoints is generated analytically. While the goal of trajectory generation is to maintain the UAV stability with minimum flight energy, the goal of the waypoints planning is obstacle avoidance with minimum control thrust in a (partially) known environment over the entire trip.

The two level approach is optimized together and it effectively reduces complexity of lower level optimization, where detailed aerodynamic model of UAVs is applied to generate a short trajectory in a localized free space without worrying about obstacles. The upper-level optimizer (i.e. the waypoints planner) ensures that a global optimal solution can be obtained by connecting the sequence of locally optimized short trajectories. This is a combinatorial optimization problem with exponential search space, and its results are heavily affected by the lower level optimization. In this work, we use deep reinforcement learning (DRL) for waypoints planning. The DRL framework not only learns the highly complicated and nonlinear interaction between the upper and lower level optimizers, but also learns and generalizes the impact from the environment (e.g. the location of obstacles) to the waypoints selection. While the DRL model also relies on trial-and-error approach and detailed aerodynamic model for training, this is done offline. During the mission, the waypoints are selected only based on the surrounding environment, the target location, and the flight status of the UAV.

We formulate the DRL as a Deep Q-Network (DQN), which approximates the optimal selection of actions in time based on the instantaneous configuration of the environment. With awareness of this configuration and the feedback at every time step, the UAV modifies its behavior, i.e. the selection of the next waypoint. The learning of the DRL is carried out in a controlled environment in a progressive manner so that the UAV can first discover its own dynamics and then learn how to cope with the external environment. After the generation of waypoints, an optimal trajectory is calculated, where the control inputs actuate the three degrees of rotational motion and one degree of translational motion in a body-fixed coordinate frame. The translational motion is controlled by a single thrust along a body-fixed direction vector, which can be controlled by the attitude of the vehicle. This actuation model covers a wide range of unmanned vehicles like fixed-wing and quadcopter UAVs, unmanned underwater vehicles, and spacecraft.

The rest of paper is organized as follows. Section 2 provides details about structure of our proposed model, the learning process and different ways to select training data for learning and waypoints generation. Section 3 provides details about continuous and discretized dynamics model. Trajectory generation through waypoints and gain selection are given in section 4. Experimental results are given in section 5 and section 6 summarizes the work.

2 DRL-BASED WAYPOINTS PLANNING

2.1 Deep Q Network

The applicability of traditional reinforcement learning is limited to tasks with low-dimensional state space. To address this problem, DQN [15] has been introduced recently to approximate the relationship between the immediate actions and their gains (i.e. *reward*) by predicting the Q-value of action-state pairs $Q(state, action)$ that

has very high-dimensions. The learner (i.e. *agent*) develops the knowledge of environment by accumulating its experience through interaction with environment and makes decision (i.e. *action*) to maximize the reward. The environment transforms the instantaneous configuration (i.e. *state* and *action*) into next state and a reward, and the agent transforms the new state and reward into next decision. DRL composes an optimization process throughout the whole state space in order to maximize the accumulated reward. For a specific policy, the $Q(state, action)$ value is the expectation of the accumulated discounted reward of each state action pair as shown in Equation (1a), where $\mathbb{R}(state_i, action_i)$ is the reward achieved in decision epoch i and i starts from current epoch t, and $\lambda \in (0, 1]$ is the discount factor.

DQN is adopted when the total number of possible actions is finite. At i_{th} of an execution sequence, the agent uses deep neural network (DNN) to estimate the $Q(state_i, action_i)$ and then select action $action_i$ either with random choice or by choosing the one that has the highest estimated $Q(state_i, action_i)$ value. At the end of each epoch, the agent performs mini-batch updating [23] that updates the DNN using new target estimations. The target value y_{target} is given in Equation (1b), where $state'$ and $action'$ are state and action at next decision epoch while taking action in current state. In order to reduce the instability caused by the correlations between state-action value $Q(state, action)$ and target value y_{target} [15], a target network with weights θ' is maintained and is updated periodically by Equation (1c). In other words, the weights of target model θ' is the exponential moving average of the weights θ of Q network.

$$Q(state_t, action_t) = \mathbf{E}(\Sigma_{i=t}^{\infty} \lambda^{i-t} \mathbb{R}(state_i, action_i)) \tag{1a}$$

$$y_{target} = \mathbb{R}(state, action) + \gamma \max_{action'} Q(state', action') \tag{1b}$$

$$\theta' = \tau\theta + (1-\tau)\theta', \qquad 0 \le \tau \le 1 \tag{1c}$$

2.2 Problem Formulation

In this work, we focus on trajectory generation for multi-rotor UAVs. These UAVs have fixed plane of rotors that actuate the vehicle in three-dimensional transnational and rotational motion, hence they have the property of under-actuation. Given a closed environment, the UAV takes off from an arbitrary position and reaches a target position which is preassigned, without colliding with obstacles. As stated before, the first step is to select waypoints based on the environment. The entire 3D environment is divided into $N \times N \times N$ grids. The environment is described by a function $\mathbf{M}()$ maps a grid (x, y, z) to a real value $\mathbf{M} : (x, y, z) \rightarrow \mathbf{R}$. A grid, g, that contains obstacle will be mapped to -10, $\mathbf{M}(g) = -10$. The destination grid that the agent needs to reach is mapped to 10, and the grid where the UAV is currently located is mapped to 1. All other grids are mapped to zeros in the discretized environment block. Let $W_0, W_1, \cdots, W_{N-1}$ be the sequence of generated waypoints, where each one is a $3D$ vector corresponding to a grid in the environment. Let $f(W_i, W_j)$ denote the control thrust for the UAV to follow the trajectory between waypoints W_i and W_j generated by the lower level optimizer. Also, let $\mathbf{G}(W_i, W_j)$ denote the set of grids that the generated trajectory between W_i and W_j will pass through. The total thrust cost along the trajectory is denoted as $\mathbf{F}$. The problem of waypoints generation can be formulated as the following:

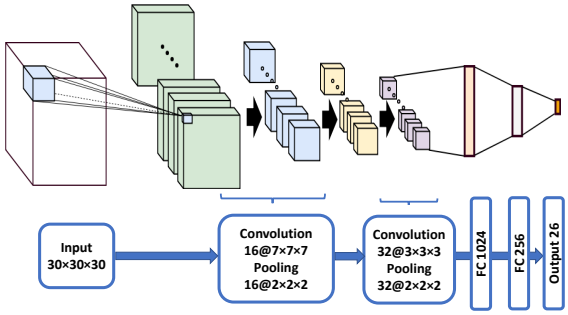


Figure 2: Network structure of proposed Deep Q Network.

Problem 1 (Optimal obstacle avoidance waypoints planning). *Minimize*

$$\mathbf{F} = \Sigma_{i=0}^{N-2} f(W_i, W_{i+1}) \tag{2}$$

subject to

(1) *reaching the target position from current position,*

$$\mathbf{M}(W_0) = 1, \quad \mathbf{M}(W_{N-1}) = 10, \tag{3}$$

(2) *reaching the target position without colliding with obstacle,*

$$\mathbf{M}(g) \neq -10, \quad g \in \mathbf{G}(W_i, W_{i+1}), \quad 0 \le i \le N-2, \tag{4}$$

To find the set of W_i, $0 \le i \le N-1$ is a combinatorial problem. The goal is to achieve minimum control thrust without obstacle collision if the UAV flies along the waypoints and trajectory. A large reward will be received at the end of the flight if the UAV reaches the destination. While this is the problem formulation of the upper level optimizer, the functions $f(W_i, W_j)$ and $\mathbf{G}(W_i, W_j)$ are determined by the lower level optimizer. Reinforcement learning provides a way to solve such constrained optimization problem with delayed reward. Incorporating with deep neural network, an optimal policy is learned to guide the UAV to the next selected actions (i.e. waypoints) that can lead to maximum future rewards.

2.3 Network Structure

The detailed structure of the DQN is shown in Figure 2. The input of DQN is the state, which represents current known knowledge of the surroundings and the status of the UAV. The state is a $3D$ matrix with size $N \times N \times N$. Each entry (x, y, z) of the matrix is the mapped value $\mathbf{M}(x, y, z)$ of the corresponding grid in the $3D$ environment previously discussed. The state has the information about the relative position between agent and obstacles. A UAV can choose any of the $3 \times 3 \times 3$ grids around its current location as the waypoint. Therefore, there are 26 possible actions. This input state is fed into two $3D$ convolutional layers and each is followed by a pooling layer. The intermediate output of the second pooling layer is fed into two fully-connected layers with the size 1024 and 256 respectively. The output is a fully-connected layer with the size 26. Each output neuron estimates the $Q(state, action)$ values for one of the 26 actions at the given state.

Our goal is to generate trajectory for the UAV with minimum control thrust under the premise of reaching target position without hitting any obstacle. Therefore, our reward function is defined as

the combination of position reward and control reward as following:

$$\mathbb{R}(state, action) = \alpha \mathbb{R}_p(state, action) + \beta \mathbb{R}_c(state, action) \quad (5)$$

where α and β are the coefficients of position reward and control reward respectively, $\alpha = \beta = 0.5$ in our experiment. $\mathbb{R}_p(state, action)$ is the position reward of taking action in current state. It is defined as following:

$$\mathbb{R}_p(state, action) = \begin{cases} 10 & reach\ target\ position \\ -10 & collide\ with\ obstacles \\ 0 & others \end{cases}$$

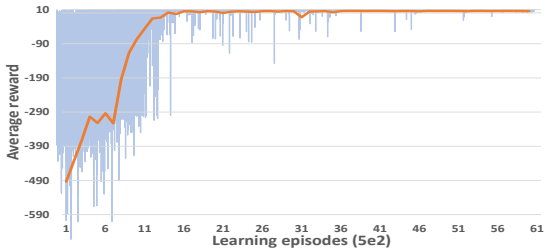
And $\mathbb{R}_c(state, action)$ indicates the control reward. It is calculated as the negative $L1$ norm of thrust cost, which is calculated by Equation 22 in optimal trajectory generation scheme proposed in section 4.

2.4 Learning of DQN

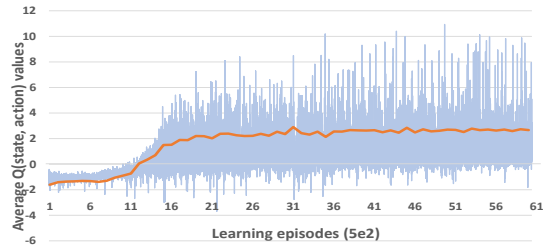
Since the problem complexity, i.e. the total number of state action pairs, is $O(26 \times N^3)$ which is relatively larger than many other existing problems [33][31][20], it is crucial to maximize exploration at the beginning of learning. Therefore $\epsilon-$greedy [14] is applied during the learning. Based on $\epsilon-$greedy, more random actions (i.e. exploration) are taken at the beginning of learning and more actions with maximum $Q(state, action)$ values (i.e. exploitation) are chosen as learning progresses.

To improve the learning, we also decrease the learning rate lr gradually because it becomes harder to improve performance with large learning rate as the gradient reaches plateau. In our approach, there are 30, 000 learning episodes in total. Instead of using a fixed learning rate, the learning rate starts from $1e-4$ and decreases every 5, 000 episodes based on $lr_{new} = (lr - \frac{lr}{epoch})_{epoch=i}$, $i \in \{5e+3, 1e+4, 1.5e+4, 2e+4, 2.5e+4\}$ and i is the i_{th} learning episode. This helps to prevent the learning from over correct after 5,000 iterations, which allows to maximize the exploitation.

Instead of randomly initializing model weights based on a uniform distribution, we initialize the weights of model based on a normal distribution. It initializes weights with relatively small values, and prevents outputs of the model from being either too large or too small. Batch normalization is used before the second convolutional layer and the first fully-connected layer to reshape the input of those hidden layers. In order to bound the training time, for every single training episode, the maximum steps W_{max} that the UAV can take is fixed. If the UAV has taken W_{max} steps but has not reached the target position, it will be forced to start a new learning episode. The start and target positions are randomly selected. So do the locations of obstacles. In this way the environment configuration of every episode is different, therefore each learning episode is independent. During learning, an experience replay is used to save last thousand times of performance and a randomly sampled mini-batch of size 32 is used to train the network. At each time step within an episode, the UAV takes an action and receives a +10 position reward if it reaches the target position and -10 if colliding with obstacles. Otherwise the position reward is 0. The control reward is determined based on the current and next position, velocity and acceleration of the UAV. The weighted sum of these two rewards and corresponding UAV state and action is saved in experience replay buffer.



(a) Average reward agent received in each learning episode.



(b) Average predicted $Q(state, action)$ values of selected actions

Figure 3: Learning results using progressive learning in a controlled environment.

2.5 Progressive Learning in a Controlled Environment

The waypoints planner is trained from scratch together with lower-level trajectory generation scheme. At the beginning of the training, the optimizer does not only have no idea about the optimal route to the target, it also does not know how the trajectory generation layer (i.e. the lower layer) will react to different waypoints selection, and if the UAV can keep its stability. It even does not know that the goal is to reach the target while avoiding obstacles. All of these must be taught globally using reward and trial. Before a baby learns walking, we put her in a small area free of obstacles, where she can roll and crawl and discover how to coordinate muscle movements. Then she can learn how to reach target and avoid obstacles. We believe that the same should be applied to teach a UAV fly and we refer this as ***progressive learning in a controlled environment***. It consists of two measures, "*progressive learning*" and "*controlled environment*". The progressive learning requires us to start learning with very low UAV mobility and gradually increase it as the learning progresses. By only allow the UAV to travel short distance, it can get quick feedback. Although it won't be able to travel long enough to reach the target and get the large position reward, based on the received control reward it learns how to coordinate with the lower level controller. Based on this learned knowledge, it will then learn how to reach target when the mobility increases. The controlled environment means to start the learning with a free space and gradually increase the number of obstacles. In this way, it can reach the target sooner with less failure. The received position reward helps the UAV to gain the knowledge of its goal.

In our original learning approach, the maximum number of steps W_{max} that the UAV can take is set to 1000. Using the progressive learning technique, the W_{max} is initialized to be 100 at the beginning of learning, and every 5,000 iterations, its value increases 50%. The new value is calculated as $(W_{max})_{new} = round(1.5W_{max})$. With such short travel distance, if the UAV is able to reach the target, it will gain the knowledge and increase the value of locations close to the target. If the UAV is not able to reach the target, it will still gain the knowledge about the control cost. Using the controlled environment technique, the number of obstacles is set to 0 at the beginning of the learning. The number increases by 5 every 5000 iterations. With the help of controlled environment, the UAV can reach target much quicker at the beginning of learning. Again, this helps it to learn the inherent relation and interaction between the upper and lower layer optimizer faster and better. Later in the learning process, when the UAV has more knowledge of its capability, it will learn how to avoid obstacles more quickly. Figure 3a gives the average reward the agent received in each learning episode during the entire learning process. And Figure 3b gives the average predicted $Q(state, action)$ values of those selected actions during the learning process. These figures indicate that as the learning goes on, the UAV receives more and more rewards and selects better and better actions. The learning converges at around 7,500 iterations. Experimental results also show that using progressive learning in controlled environment not only reduces learning time but also improves learning quality. Details can be found in Section 5.

3 DYNAMICS MODEL OF MULTI-ROTOR UAV

3.1 Continuous Time Dynamics

The rigid body model considered in this work has four control inputs for the six degrees of freedom. These control inputs include a torque for the three degrees of freedom of rotational motion and one thrust along a body-fixed thrust vector. This model is identical to that used in [28][29]. It can be applied to several unmanned vehicles, and the particular case of a quadrotor UAV is considered in section 5 for numerical results.

In this work, $b \in \mathbb{R}^3$ denotes the rigid body's position vector expressed in an inertial coordinate frame and $\mathcal{R} \in \mathrm{SO}(3)$ is the rigid body's attitude expressed as the rotation matrix from inertial frame to body-fixed frame. Without loss of generality, it is assumed that the thrust vector is along the third body-fixed coordinate frame axis. The translational dynamics motion equation is:

$$m\dot{v} = mge_3 - fr_3, \tag{6}$$

where g is gravitational acceleration, $v \in \mathbb{R}^3$ is the translational velocity in inertial frame, $e_3 = [0,\ 0,\ 1]^T$, $u = fr_3 \in \mathbb{R}^3$ is the control thrust vector of magnitude f acting on the body, and r_3 is the unit vector along the third axis of the body-fixed coordinate frame, expressed in the inertial frame. Note that r_3 is also the third column of the rotation matrix $\mathcal{R}$. Equation (6) can be rewritten as:

$$\dot{v} = ge_3 - \frac{1}{m} fr_3. \tag{7}$$

The velocity kinematics for the translational motion expressed in inertial coordinate frame is simply $\dot{b} = v$. Consider a "triple integrator" dynamics model for position trajectory generation, given by

$$\dot{b}(t) = v(t), \tag{8}$$

$$\dot{v}(t) = a(t), \tag{9}$$

$$\dot{a}(t) = u(t), \tag{10}$$

where the vectors b, v, a, $u \in \mathbb{R}^3$ represent position, velocity, acceleration, and jerk respectively. Let $x \in \mathbb{R}^9$ denote the state vector, i.e., $x = \begin{bmatrix} b^{\mathrm{T}} & v^{\mathrm{T}} & a^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$. The resulting system can be compactly expressed as follows:

$$\frac{dx}{dt} = Ax + Bu, \tag{11}$$

$$y = Cx. \tag{12}$$

where

$$A = \begin{bmatrix} \mathbf{0}_{3\times3} & I_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}, B = \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} \\ I_{3\times3} \end{bmatrix},$$

$$C = \begin{bmatrix} I_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix},$$

where $I_{3\times3}$ is the 3×3 identity matrix. A trajectory is to be generated for this system to pass through a given set of k waypoints in position, where $k \geq 1$. The set of waypoints consisting of positions in $\mathbb{R}^3$ with respect to an inertial frame, are generated by the method described in previous section. To facilitate numerical computation of the system, the dynamics expressed in (11)-(12) is discretized in the next subsection.

3.2 Discretization of Dynamics

Consider a fixed step size in time, h, and a fixed time interval $[0, T]$ over which the trajectory is to be generated in discrete time. Without loss of generality for the system, the initial time is assumed to be 0. Thus time is discretized as $t_n = nh$ with $T = m_k h$, so that m_k is a positive integer that corresponds to the final time at which the generated trajectory passes through the final waypoint. Let the discrete-time state variable be given by $x_n = x(nh)$, where $n \in \mathcal{N}$ and $\mathcal{N} = \{0, 1, \ldots, m_k\}$. Denote the discrete time instants at which the trajectory passes through the given position waypoints by m_i, $i = \{1, \ldots, k\}$, with $\{m_1, \ldots, m_k\} \subset \mathcal{N}$. The discrete system representation of (11)-(12) can be obtained as:

$$x_{n+1} = A_d x_n + B_d u_n, \tag{13}$$

$$y_n = C_d x_n, \tag{14}$$

where

$$A_d = e^{Ah}, \quad B_d = \int_0^h e^{A\sigma} B d\sigma, \quad C_d = C,$$

Due to the nilpotent nature of A, only the first three terms of the exponential series are needed to calculate e^{Ah} exactly. Therefore, the above discretization leads to an *exact discretization* of the continuous time system (11)-(12). The optimal control problem is formulated and its solution is presented in the next section.

4 OPTIMAL POSITION TRAJECTORY GENERATION AND GAIN SELECTION

4.1 Position Trajectory through Waypoints

The problem of trajectory generation amounts to constructing a feasible discrete-time desired trajectory through the given set of

k waypoints generated by DRL-based algorithm. Let the set of k waypoints be given by tuples $(y^w_{m_1}, m_1), (y^w_{m_2}, m_2), \cdots (y^w_{m_k}, m_k)$, where the time instants corresponding to these waypoints are denoted by the subscript $m_i \in \mathcal{N}$, with $i = 1, \cdots k$. We construct a discrete optimal control problem such that the output y_n passes through the given waypoints in specified time instants, i.e. $y_{m_i} = y^w_{m_i}$, for $i = 1, \cdots k$. Let the initial state be given by $x(0) = x_{init}$. The boundary condition at the end point is determined by the last waypoint, $y^w_{m_k}$. The optimal control problem can be formulated as follows:

Problem 2 (Discrete-time Optimal Trajectory Generation). *Minimize*

$$\mathcal{J}^d = h \sum_{i=0}^{m_k} \frac{1}{2}(x_i^T Q x_i + u_i^T R u_i) + \frac{1}{2} \sum_{j=1}^{k} \left(C_d x_{m_j} - y^w_{m_j}\right)^T S \left(C_d x_{m_j} - y^w_{m_j}\right), \quad (15)$$

subject to

(1) *satisfying the dynamical model,*

$$x_{i+1} = A_d x_i + B_d u_i, \quad (16)$$

(2) *and the boundary conditions given by,*

$$x_0 = x_{init}, \quad (17)$$

$$C_d x_{m_k} = y^w_{m_k}. \quad (18)$$

Here $Q \in \mathbb{R}^{9\times9} \geqslant 0$, $R \in \mathbb{R}^{3\times3} > 0$ and $S \in \mathbb{R}^{3\times3} \geqslant 0$ are square, symmetric matrices.

In problem 2, high values of the position, velocity, acceleration and the derivative of acceleration (also known as "jerk"), are penalized. Additionally, at the time instances corresponding to the waypoints, the error between actual position and the desired position waypoint is penalized. The problem 2 can be approached from the first principles of optimal control. Let the augmented performance index be written as,

$$\mathcal{J}_a^d = \mathcal{J}^d + \sum_{i=0}^{m_k-1} \lambda_{i+1}^{\mathrm{T}} (A_d x_i + B_d u_i - x_{i+1}), \quad (19)$$

here $\lambda_i \in \mathbb{R}^9$ is a vector of co-states. The optimal control input is found to be (details are removed for brevity):

$$u_i = -[R + (B_d)^{\mathrm{T}} P_{i+1} B_d]^{-1} (B_d)^{\mathrm{T}} (P_{i+1} A_d x_i + \eta_{i+1}). \quad (20)$$

This control input generates an optimal, smooth trajectory between waypoints.

Remark 1. *Let $K_i = [R + (B_d)^T P_{i+1} B_d]^{-1} (B_d)^T$, then the optimal control can be written as*

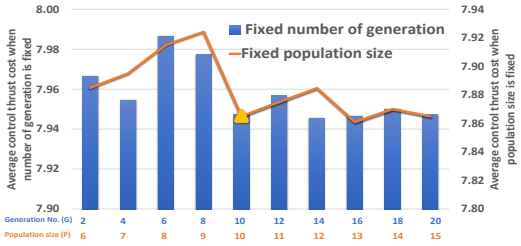
$$u_i = -K_i \left((P_{i+1} A_d x_i + \eta_{i+1})\right)$$

After applying the optimal control, the dynamics of the discrete system given in (13) *becomes,*

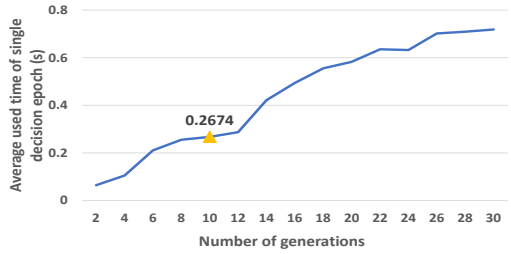
$$\begin{aligned} x_{i+1} &= A_d x_i - B_d K_i (P_{i+1} A_d x_i + \eta_{i+1}), \\ &= \left(A_d - B_d K_i P_{i+1} A_d\right) x_i - B_d K_i \eta_{i+1}. \end{aligned} \quad (21)$$

Remark 2. *Throughout all steps, thrust force can be calculated by:*

$$f_i = m \| a_i - g e_3 \|. \quad (22)$$



(a) Trade-off among average control thrust cost, number of generations and population size.



(b) Relation between number of generations and time consumption for single decision epoch.

Figure 4: Genetic algorithm utilization analysis.

4.2 Gain Selection through Genetic Algorithm

During the trajectory generation, the system gives the acceleration of the UAV while solving Problem 2. Based on Equation 10 and 7, we derived Equation 22 to calculate the control thrust, which is the feedback reward for higher level waypoints planner during learning. It is necessary to mention that in the lower level of our module, optimal trajectory generation scheme, Q, R and S are three positive definite gain matrices. Each of these gain matrices penalize different aspects while generating the trajectory as indicated in Equation 15. Q is a matrix penalizing high values of position, velocity and acceleration. The higher the values in Q, the harder these parameters are penalized. The input jerk is penalized when R matrix has large eigenvalues and how much the waypoints will affect the trajectory is weighted by S matrix. From Equation 15 we can see that these three gain matrices have significant impact on the performance index $\mathcal{J}^d$ of the trajectory. Their values need to be tuned in order to minimize the control thrust f. In our experiment, f is calculated from the lower level scheme, while it will be measured by sensors in real field learning. We apply genetic algorithm (GA) [3] to optimize the gain matrices. The best set of (R, Q, S) satisfies $\arg\min_{R,Q,S} \sigma f(W_i, W_{end}, v_i, a_i)$ where W_i, v_i, a_i are initial position, velocity and acceleration of the UAV and W_{end} are the destination for UAV.

Without loss of generality, we let the gain matrices be identity matrices scaled by different factors. Therefore, each chromosome (i.e. solution) in the population has 3 genes, one for each gain matrix. For our problem, we randomly select 10 chromosomes based on a uniform distribution at the beginning. The negated L1 norm of thrust cost is used as the fitness of each solution. Based on the fitness value, we select the best set of (R, Q, S) within the current population as parents for mating. Next step is to apply GA variants (i.e. crossover and mutation) to produce the offspring, creating new

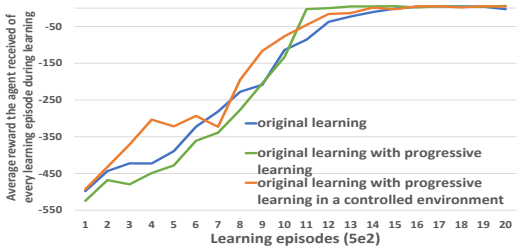
population by appending parents and offspring. In our approach, one point crossover and uniform mutation are adopted. Repeating these steps for several iterations, the returned optimal set of (R, Q, S) results in minimum control thrust when UAV flying from the W_i to W_{end}. As a stochastic optimization algorithm, the more generations and larger population evaluated by the GA, the better solution can be found. Figure 4a shows how the cost (i.e. inverse of fitness) reduces as the generation (blue labels) and population size (orange labels) increases. As we can see that the quality of the solution saturated when the size of population and the number of generations are both beyond 10. Figure 4b shows that the runtime of the GA is almost a linear function of the number of generations. Based on the above analysis we set the population size to 10 and maximum generations also to 10 indicated as yellow triangle in Figure 4.

5 EXPERIMENTS

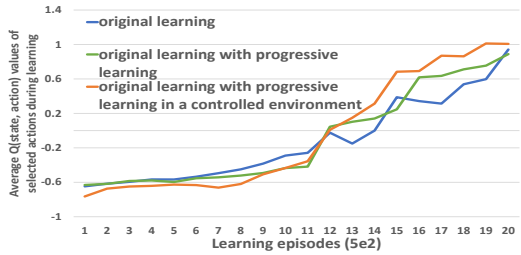
In this section, we demonstrate the performance of our proposed model. The training and testing were done on NVIDIA TitanX (Pascal). In the experiments, the environment is divided into $10 \times 10 \times 10$ and the unit distance δ_d is 10 meters. Within each testing scenario, the number of obstacles is randomly generated and obstacles are placed randomly within the environment boundary. Besides, the start and target positions are also randomly selected. We report the results of two aspects: (1) the improvements achieved by using progressive learning in a controlled environment; (2) the results compared with some existing approaches.

5.1 Impacts of Progressive Learning in a Controlled Environment

Figure 5a and 5b compare the change of reward and Q values of learning with and without gradually increased UAV mobility (i.e. progressive learning) and environment complexity (i.e. controlled



(a) Average reward the agent received.



(b) Average $Q(state, action)$ values of selected actions.

Figure 5: Learning results of first 10,000 episodes comparisons before and after using improved learning.

Table 1: Average selected waypoints number comparison over different distances.

Normalized distance from start to destination	(0,4]	(4,8]	(8,12]	(12,16]
traditional learning	3.43	6.49	12.11	13.52
improved learning	3.40	6.43	11.88	13.42

Table 2: Average control thrust cost comparison over different distances.

Normalized distance from start to destination	(0,4]	(4,8]	(8,12]	(12,16]
traditional learning	1.4321	3.5951	6.2375	6.2459
improved learning	1.3479	3.4859	6.0139	6.1138

environment). The results of traditional learning, which adopts none of those two improvements, are represented by blue curves and the results after adopting only the progressive learning are represented by green curves in the figure. The orange curves show the results of applying both progressive learning and controlled environment techniques. To make it clear to see, we show the result of first 10,000 episodes with average of every 500 learning episodes. With the help of progressive learning, the learning converges after 7,500 episodes. It is 16.67% less compared to the traditional learning that converges after 9,000 episodes. Because of the reduced mobility, an episode at the beginning of progressive learning is much shorter than an episode in the traditional learning. Therefore, the reduction in computing time is even more. It uses around 10 hours total learning time with both improvement techniques which has 47.4% reduction in learning time. From Figure 5b we can see that the progressive learning and controlled environment not only speed up the convergence, the quality of learning is also better because the Q values are more stable than that of traditional learning.

To evaluate the quality of learned model, we generated thousands different testing scenarios and divide the flight of the UAV into four groups: (1) the UAV collides into obstacles without reaching target; (2) the UAV collides into obstacles but eventually reaches the target; (3) the agent does not collide into obstacles neither does it reach the target; (4) the UAV does not collide into obstacles and it successfully reaches the target. If the UAV travels 100 steps without reaching the target, it is regarded as a failure. Only type (4) flights are considered as successes. We refer both type (4) and type (2) as achieved as

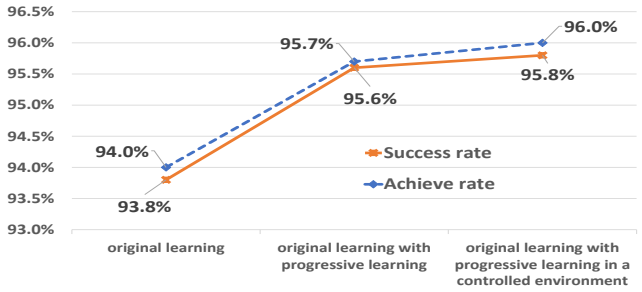
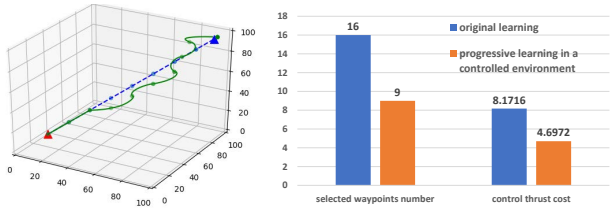


Figure 6: Success rate and achieve rate improvements using progressive learning in a controlled environment techniques. Situation (4): the agent reaches the target position and does not collide into obstacles.

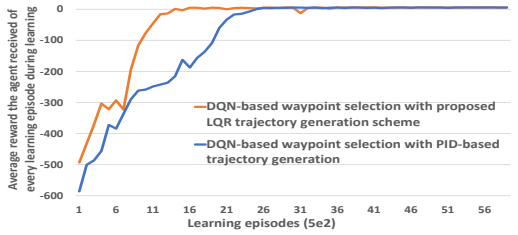


(a) (b)

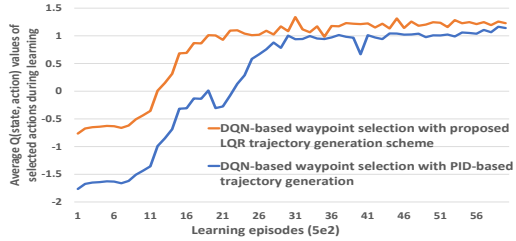
Figure 7: (a) Example of trajectory generation through waypoints selected by traditional learning (*green*) and improved learning (*blue*) respectively. (b) Comparisons tracking number of selected waypoints and thrust cost during generation.

they both achieved the goal, i.e. reaching the target. Figure 6 shows how the success rate and achieve rate improved after using the progressive learning and controlled environment. As indicated in the figure, the success rate increases from 93.8% to 95.8% and the achieve rate increases from 94.0% to 96.0% after optimization.

With respect to the number of steps taken and control thrust consumption during the flight, progressive learning in a controlled environment can also lead to improvement. Table 1 compares the number of waypoints generated by the traditional learning and the improved learning over different distances. The less number of waypoints means the fewer steps for the UAV to fly to the target. In general the UAVs learned using the improved learning need less number of steps. The reduction of waypoints usage becomes larger as the distance between start and target increases. Table 2 compares the average thrust cost if the UAV follows the trajectory, To illustrate it clearer, an example of a scenario in an obstacle-free environment is shown in Figure 7, which compares results of model trained in traditional way and one trained with controlled environment technique. In the example, the environment has the same configuration, i.e. same start (i.e. red triangle) and target position (i.e. blue triangle). The green curve indicates the trajectory along waypoints generated by traditional learning, and the blue curve shows the trajectory along waypoints generated by improved learning.



(a) Average reward the agent received every learning episode.

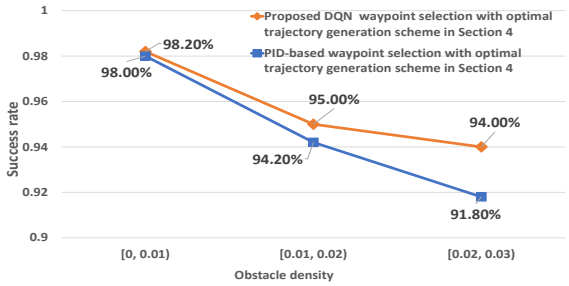


(b) Average $Q(state, action)$ value of selected actions.

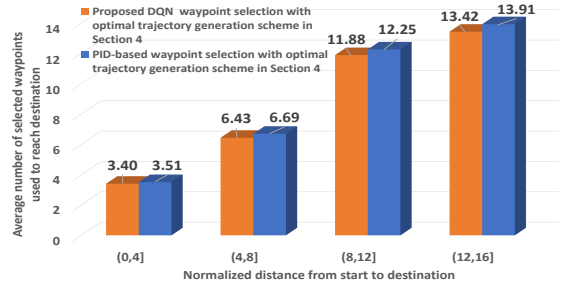
Figure 8: Learning results of proposed DQN waypoints selection together with proposed optimal trajectory generation scheme and PID-based trajectory generation baseline respectively.

5.2 Results Comparison

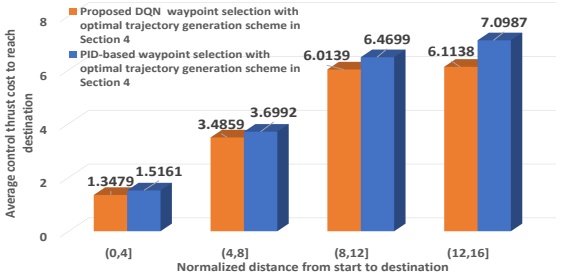
As a baseline of reference, we also implement a control scheme based on PID theory [19] to estimate the control thrust consumption along waypoints, and use it to replace optimal trajectory generation of Section 4 scheme as the lower level optimizer. We define position as measurable variables and velocities when reaching each waypoints are regarded as controllable variables. Every time the position of the agent is updated by the feedback of the environment, and the feedback is calculated by the environment simulation based on Kinematic theory [13]. Figure 8 shows the learning results of



(a) Comparison tracking success rate of proposed DRL with proposed optimal trajectory generation scheme and with PID baseline



(b) Average selected waypoints number to reach destination.



(c) Average control thrust cost along selected waypoints.

Figure 9: Comparisons tracking results of different trajectory generation schemes, i.e. proposed optimal trajectory generation scheme (orange) and PID baseline (blue), with the same proposed DQN waypoints selection process.

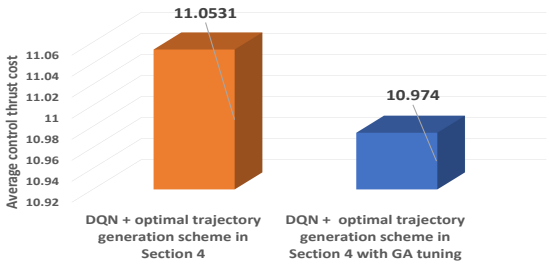
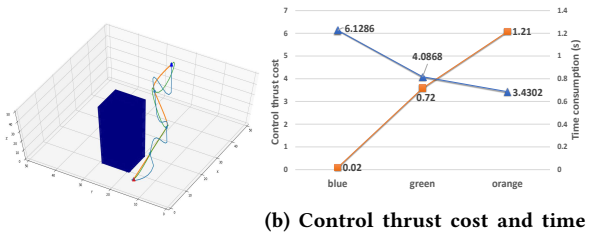


Figure 10: Improvements achieved with tuning gain matrices of optimal trajectory generation scheme in Section 4 using genetic algorithm

DQN when our purposed trajectory generation scheme or PID are used in the lower level. As indicated in the figure, the learning of our scheme converges 44% faster than using PID, because the proposed optimal trajectory generation scheme will not over correct the trajectory and there is no control latency, hence its performance is more stable and predictable. Also the DQN with optimal trajectory control achieves higher reward, i.e. it consumes less control thrust, because of higher fidelity of the proposed optimal trajectory generation scheme in Section 4. Figure 9 (a)~(c) reports the comparisons of success rate, the average number of selected waypoints to reach the target and the average control thrust cost to go through these waypoints. The comparisons are divided into four groups based on the Euclidean distance between start and target position.

Since the critical roles of gain matrix in proposed proposed optimal trajectory generation scheme in Section 4, it is crucial to improve the performance by tuning gain matrices. The control thrust of UAVs with and without optimized grain matrices is compared in Figure 10. The average thrust cost of using genetic algorithm is indicated as blue labels, while orange indicates the result of manually selecting gain matrices. As shown in the figure, the average control thrust consumption decreases from 11.0531 to 10.9740 in a $30 \times 30 \times 30$ discretized environment block. Figure 11a gives an example of trajectories without gain optimization (blue), with medium optimization (green) and with heavy optimization (orange). The corresponding control thrust and optimization time is given in Figure 11b. In the example, four waypoints (blue dots) are selected to reach the target. The start and target points are shown in red and blue triangles respectively and the cylinder represents obstacles.



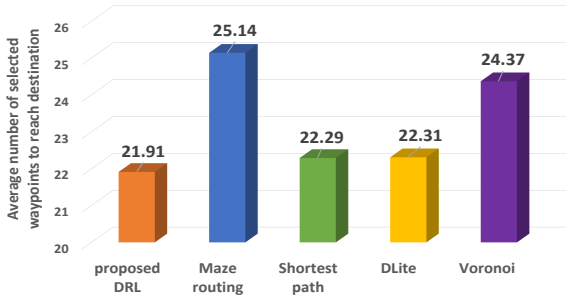
(a) Example trajectories.

(b) Control thrust cost and time consumption comparison.

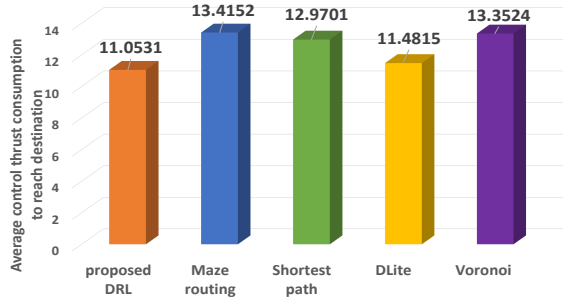
Figure 11: Example of trajectory generation without and with gain matrices tuning by genetic algorithm. Blue: trajectory without gain optimization; Green: trajectory with medium gain optimization; Orange: trajectory with heavy gain optimization.

The blue curve shows the trajectory generated with fixed R, Q and S, which has large overshoot and some sharp curves. The control thrust for the blue trajectory is 6.1286 force cost as indicated in Figure 11b. If we apply GA between current waypoints W_i and next waypoint W_{i+1} to select the optimal set of R, Q and S just for each segment, the control thrust cost reduces to 4.0868 (i.e. green curve), after 7 generations of GA search. The control thrust cost further reduces to 3.4302 after 10 generations of GA search (i.e. orange curve). And the orange line in Figure 11b gives time cost for the GA optimization.

Finally, we compare the DRL based waypoints selection with four traditional waypoints selection approaches in aspects of average number of steps needed to reach target, and the average control thrust cost along the trajectory. These four approaches include maze routing [17], shortest path [18], DLite algorithm [9] and voronoi path [10]. To make it more convincing, the size of discretized environment is set as $30 \times 30 \times 30$. We generated 1000 different test scenarios by randomly select different start positions, destination positions, types and locations of obstacles. Figure 12 compares the average number of selected waypoints to reach destination and the average control thrust cost for the UAV to go through these waypoints. For all waypoints selection approaches,optimal trajectory generation scheme in Section 4 is used for trajectory generation. In Figure 12a, the results show that our approach only needs an average 21.91 waypoints which is 6.6% less than other approaches. In Figure 12b, the comparison of average control thrust consumption is reported. As indicated in the figure, our approach



(a) Average number of selected waypoints to reach destination.



(b) Average control thrust cost along trajectory generated through waypoints.

Figure 12: Results comparison between proposed DQN scheme, routing, shortest path, DLite and Voronoi

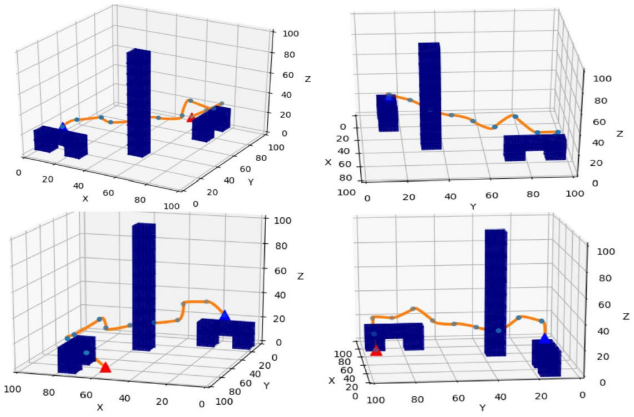


Figure 13: An example of autonomous waypoints planning and trajectory generation using proposed framework.

use least control thrust which has 13.33% reduction than other approaches. According to these results, our purposed scheme with fewer selected waypoints is less possible to be over constrained and less times of lower level scheme invocation is needed.

A more realistic scenario is given in Figure 13. In this scenario, a door is set as the start position which is indicated by a red triangle in the figure. The UAV takes off from the door, and then land on the center of the table near the door first. After that, it takes off again to reach the final destination which is indicated by blue triangle in the figure. As shown in the figure, blue dots are selected waypoints provided by our proposed DQN scheme. And the smooth orange curve shows the trajectory generated by optimal trajectory generation scheme purposed in Section 4. The UAV does not collide into obstacles during the flight. In order to display the trajectory clearly, we show four different views of the 3D trajectory plot.

6 CONCLUSION

A two-level framework to generate navigation trajectory for UAVs to follow in a complex environment is introduced. The framework's construction, processing and analysis are presented. The proposed waypoints planning and trajectory generation framework effectively avoids obstacles in complex indoor environment and reduces the control thrust consumption during flight. Also, it is general enough to be applied in other robotics tasks such as parcel delivery and conflicting routing of high-density UAVs.

REFERENCES

[1] Scott A Bortoff. 2000. Path planning for UAVs. In *American Control Conference. Proceedings of the 2000*, Vol. 1. IEEE.

[2] Pradipta K Das, SC Mandhata, HS Behera, and SN Patro. 2012. An improved Q-learning algorithm for path-planning of a mobile robot. *International Journal of Computer Applications* 51, 9 (2012).

[3] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[4] M. Hehn and R. DAndrea. 2015. Real-time trajectory generation for quadrocopters. *Robotics, IEEE Transactions on* 31, 4 (2015), 877–892.

[5] Markus Hehn and Raffaello DâĂŹAndrea. 2011. Quadrocopter trajectory generation and control. In *IFAC world congress*, Vol. 18. 1485–1491.

[6] M. Hoy, A. S. Matveev, and A. V. Savkin. 2015. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica* 34 (2015), 467–497.

[7] Nursultan Imanberdiyev, Changhong Fu, Erdal Kayacan, and I-Ming Chen. 2016. Autonomous navigation of UAV by using real-time model-based reinforcement learning. In *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*. IEEE, 1–6.

[8] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[9] Sven Koenig and Maxim Likhachev. 2002. D^* Lite. *Aaai/iaai* 15 (2002).

[10] Boris Lau, Christoph Sprunk, and Wolfram Burgard. 2010. Improved updating of Euclidean distance maps and Voronoi diagrams. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 281–286.

[11] Steven M LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. (1998).

[12] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[13] Sandeep Kumar Malu and Jharna Majumdar. 2014. Kinematics, localization and control of differential drive mobile robot. *Global Journal of Research In Engineering* (2014).

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[16] M. W. Mueller, M. Hehn, and R. D'Andrea. 2015. A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation. *IEEE Transactions on Robotics* 31, 6 (Dec 2015), 1294–1310. https://doi.org/10.1109/TRO.2015.2479878

[17] John A Nestor. 2002. A new look at hardware maze routing. In *Proceedings of the 12th ACM Great Lakes symposium on VLSI*. ACM, 142–147.

[18] Rosli bin Omar. 2012. *Path planning for unmanned aerial vehicles using visibility line-based methods*. Ph.D. Dissertation. University of Leicester.

[19] Lluis Pacheco and Ningsu Luo. 2015. Testing PID and MPC performance for mobile robot local path-following. *International Journal of Advanced Robotic Systems* 12, 11 (2015), 155.

[20] Riccardo Polvara, Massimiliano Patacchiola, Sanjay Sharma, Jian Wan, Andrew Manning, Robert Sutton, and Angelo Cangelosi. 2018. Toward End-to-End Control for UAV Autonomous Landing via Deep Reinforcement Learning. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 115–123.

[21] Charles Richter, Adam Bry, and Nicholas Roy. 2016. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*. Springer, 649–666.

[22] J. Sanchez-Lopez, S. Saripalli, P. Campoy, J. Pestana, and C. Fu. 2013. Toward visual autonomous ship board landing of a VTOL UAV. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*. IEEE, 779–788.

[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[24] Leena Singh and James Fuller. 2001. Trajectory generation for a UAV in urban terrain, using nonlinear MPC. In *American Control Conference, 2001. Proceedings of the 2001*, Vol. 3. IEEE, 2301–2308.

[25] Florin Stoican and Dan Popescu. 2016. *Trajectory Generation with Way-Point Constraints for UAV Systems*. Springer International Publishing.

[26] Lei Tai and Ming Liu. 2016. Towards cognitive exploration through deep reinforcement learning for mobile robots. *arXiv preprint arXiv:1610.01733* (2016).

[27] John Tisdale, ZuWhan Kim, and J Karl Hedrick. 2009. Autonomous UAV path planning and estimation. *IEEE Robotics & Automation Magazine* 16, 2 (2009).

[28] Sasi Prabhakaran Viswanathan, Amit K Sanyal, and Maziar Izadi. 2017. Integrated Guidance and Nonlinear Feedback Control of Underactuated Unmanned Aerial Vehicles in SE(3). In *AIAA Guidance, Navigation, and Control Conference*. Gaylord, TX. https://doi.org/10.2514/6.2017-1044

[29] S. P. Viswanathan, A. K. Sanyal, and R. Warier. 2017. Finite-Time Stable Tracking Control for a Class of Underactuated Aerial Vehicles in SE(3). Seattle, WA, to appear.

[30] Michael P. Vitus, Wei Zhang, and Claire J. Tomlin. 2012. A hierarchical method for stochastic motion planning in uncertain environments. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), 2263–2268.

[31] Juan Wu, Seabyuk Shin, Cheong-Gil Kim, and Shin-Dug Kim. 2017. Effective lazy training method for deep q-network in obstacle avoidance and path planning. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*. IEEE, 1799–1804.

[32] Wenda Xu, Junqing Wei, John M Dolan, Huijing Zhao, and Hongbin Zha. 2012. A real-time motion planner with trajectory optimization for autonomous vehicles. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE.

[33] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 3357–3364.