Autonomous Navigation of UAV by Using Real-Time Model-Based Reinforcement Learning

Nursultan Imanberdiyev[1,2], Changhong Fu[1,2], Erdal Kayacan[1], and I-Ming Chen[1]
[1]School of Mechanical and Aerospace Engineering
[2]ST Engineering-NTU Corporate Laboratory
Nanyang Technological University, 50 Nanyang Avenue, 639798, Singapore
E-mail: iman0005@e.ntu.edu.sg, {changhongfu, erdal, michen}@ntu.edu.sg

***Abstract*—Autonomous navigation in an unknown or uncertain environment is one of the challenging tasks for unmanned aerial vehicles (UAVs). In order to address this challenge, it is necessary to have sophisticated high level control methods that can learn and adapt themselves to changing conditions. One of the most promising frameworks for such a purpose is reinforcement learning. In this paper, a novel model-based reinforcement learning algorithm, TEXPLORE, is developed as a high level control method for autonomous navigation of UAVs. The developed approach has been extensively tested with a quadcopter UAV in ROS-Gazebo environment. The experimental results show that our method is able to learn an efficient trajectory in a few iterations and perform actions in real-time. Moreover, we show that our approach significantly outperforms Q-learning based method. To the best of our knowledge, this is the first time that TEXPLORE has been developed to achieve autonomous navigation of UAVs.**

I. Introduction

In recent years, unmanned aerial vehicles (UAVs) have been extensively applied for different types of civilian tasks, e.g., object tracking [1], wildlife protection [2], disaster rescue [3], and 3D reconstruction [4]. Regardless of numerous applications, autonomous flight of UAVs is still a challenging area in the domain of control for many reasons. To achieve high autonomy in UAV tasks by using model-based controllers, the controller requires a precise mathematical UAV model. However, in practice, dynamics of UAVs are difficult to model, and they are usually sensitive to unforeseen internal or external uncertainties. Therefore, it is essential to provide a learning ability to UAVs in order to adapt their behavior to changing conditions. One of the most common frameworks for learning and adapting its behavior is reinforcement learning (RL). The RL methods enable an agent to learn right actions with little or no prior knowledge about its environment or dynamics, and it can be used to adapt to randomly changing environmental conditions [5]. Therefore, the RL methods have become a promising tool for improving autonomous behavior in different robotics applications, e.g., [6].

For having a fully autonomous UAV flight, it is necessary to have a sophisticated high level control that allows the UAV to make intelligent decisions to complete a mission. Because of its nature, RL can effectively be utilized in a high level decision making system to learn an efficient mission route for UAVs despite limited knowledge of the environment. Therefore, the main purpose of this paper is to demonstrate an implementation of the RL method as a high level control of UAV mission in an unknown or uncertain environment. In particular, this paper focuses on learning an efficient mission route using RL for quadcopter UAV in real-time.

The experiments conducted in this paper are arranged to represent one of the potential real-life applications of UAV that may benefit from RL. Finding an efficient route, when UAV is constrained in mission time or battery life, is very important to achieve a safe autonomous flight. For this reason, in our experiments, the quadcopter UAV monitors its position and battery level, and it learns to modify its route, if it is necessary for the intermediate recharging of batteries in order to safely achieve its objective.

In this paper, we use the model-based RL algorithm known as TEXPLORE [7]. TEXPLORE is designed to address the real-world tasks such as robotics problems. In contrast to many RL algorithms, e.g., Q-Learning [8], which ensures optimality by random and exhaustive exploration, TEXPLORE implements targeted exploration quickly on states that are both uncertain in the model and promising for the final policy. The RL algorithms, which perform thousands of actions to learn, are not practical for real robots, especially for UAVs, since these real-world actions may be dangerous and expensive. Therefore, TEXPLORE is the method that learns in very few episodes and it allows to take actions in real-time due to its parallel architecture [9]. These properties of TEXPLORE are promising to solve the necessary challenges to be successful in real-world problems. To the best of our knowledge, this is the first time in literature when TEXPLORE is developed to achieve autonomous navigation of UAVs.

To effectively elaborate our proposed approach, we use the robot operating system (ROS) and Gazebo simulator as the testing environment. The main reason for utilizing the combination of ROS and Gazebo is that it provides seamless interfaces with the real quadcopter UAV. In addition, the software implementation used in ROS-Gazebo can be directly implemented in real quadcopter flights.

The rest of this paper is organized as follows: Section II states quadcopter dynamics. Section III explains the model-based RL algorithm, i.e. TEXPLORE, in detail. Section IV describes the experimental setup. Section V provides simulation results that show the efficiency of our solutions, while Section VI presents the closing remark.

II. Quadcopter Dynamics

In this section, we introduce a necessary background on quadcopter dynamics. In general, a quadcopter can be considered as a rigid body to which torques and forces are applied from the four rotors located at the vertices of a square. We have the world-fixed inertial reference frame C_I and a body-fixed frame C_B attached to the quadcopter, where the center of mass of the quadcopter is the origin of the body frame C_B. The axes of the body-fixed frame C_B, $\vec{x}_B$ and $\vec{y}_B$, lie on the plane defined by the four rotors, while the axis $\vec{z}_B$ points downwards in the opposite direction of the total thrust. The quadcopter model with reference frames is shown in Fig. 1.

The locus (x, y and z) of the quadcopter is determined by the position of its center of mass with respect to the inertial frame C_I, while the attitude is described by the *ZYX* Euler angles (roll ϕ, pitch θ and yaw ψ). Therefore, the rotation matrix $\mathbf{R}$ from the body frame C_B to the inertial frame C_I can be defined using aforementioned Euler angles:

$$\mathbf{R} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\psi s_\phi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & c_\psi c_\phi + s_\phi s_\theta s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}, \quad (1)$$

where c_* is $cos(*)$ and s_* is $sin(*)$ for simplicity. Hence, the quadcopter dynamics can be written by the following equations [10]:

$$\dot{\mathbf{r}} = \mathbf{v} \quad (2)$$

$$\dot{\mathbf{v}} = \mathbf{g} + \frac{1}{m}\mathbf{R}\mathbf{f} \quad (3)$$

$$\dot{\boldsymbol{\omega}}_B = \boldsymbol{J}^{-1}\boldsymbol{\tau} - \boldsymbol{J}^{-1}\boldsymbol{\Omega}\boldsymbol{J}\boldsymbol{\omega}_B \quad (4)$$

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\Omega}, \quad (5)$$

where m is the total mass, $\mathbf{g} = [0, 0, g]^T$ is the vector of gravitational acceleration ($g = 9.81\mathrm{m/s^2}$), $\mathbf{v}$ is the velocity of the quadcopter in the inertial frame C_I and it is given by

$$\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T, \quad (6)$$

$\mathbf{f} = [0, 0, f_z]^T$ is the total thrust vector, where f_z is the sum of all forces generated by the four rotors which are aligned with $-\vec{z}_B$ axis and it is given by

$$f_z = f_1 + f_2 + f_3 + f_4. \quad (7)$$

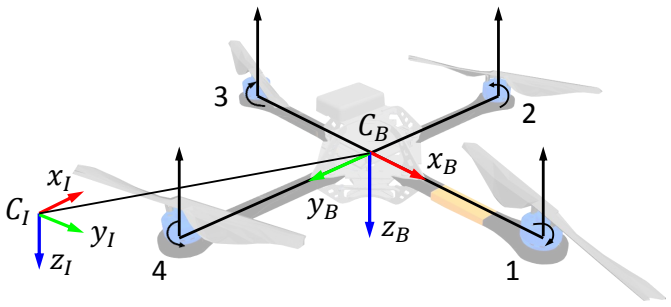


Fig. 1: Quadcopter model with reference frames.

Similarly, the quadcopter's angular velocity with respect to the body frame C_B is defined as

$$\boldsymbol{\omega_B} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T, \quad (8)$$

$\boldsymbol{\Omega}$ is the tensor form of $\boldsymbol{\omega_B}$ and it is given by

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \quad (9)$$

then, $\boldsymbol{\tau} = [\tau_x, \tau_y, \tau_z]^T$ is the torque vector on the quadcopter with respect to the body-fixed frame C_B. $\boldsymbol{J} = \mathrm{diag}(J_x, J_y, J_z)$ is the inertia matrix and $\mathbf{r} = [x, y, z]^T$ is the location of the quadcopter in the inertial frame C_I.

The quadcopter control input μ is described with three torques and the total thrust as follows:

$$\boldsymbol{\mu} = \begin{bmatrix} f_z & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \quad (10)$$

and the quadcopter's state ξ is

$$\boldsymbol{\xi} = \begin{bmatrix} x & y & z & v_x & v_y & v_z & \phi & \theta & \psi & \omega_x & \omega_y & \omega_z \end{bmatrix}^T. \quad (11)$$

The additional details regarding the quadcopter dynamics can be found in [10]. The numerical values of the quadcopter's parameters are the following: the total mass is equal to $0.71\mathrm{kg}$, the arm length is $0.17\mathrm{m}$ and $[J_x, J_y, J_z]$ = $[0.007, 0.007, 0.012]$ $\mathrm{kg \cdot m^2}$

III. Model-Based Reinforcement Learning Algorithm - TEXPLORE

Reinforcement learning is a machine learning method wherein an autonomous agent learns to find a (near)-optimal behavior through trial-and-error interactions with its environment [5]. This learning process is achieved by adopting Markov decision process (MDP) formalism where the agent performs learning through the sense-act-learn cycle. MDP is defined by a set of possible actions A, a set of possible states S, a reward function $R(s, a)$ and a state transition distribution $P(s'|s, a)$ or $T(s, a, s')$ which describes the dynamics of the system. In the standard RL scenario, the agent receives current state $s \in S$ represented as the sensory input from the environment. Depending on the state s and its knowledge about past experience, the agent decides which action $a \in A$ to take. After taking this action, the agent reaches a new state s' determined from the state transition distribution $P(s'|s, a)$ and obtains the reward $R(s, a)$. The optimal state-action value function, denoted by Q^*, which calculates the expected returns (sum of discounted rewards) for a given state-action pair (s, a), can be defined by the following equation:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a' \in A} Q^*(s', a'), \quad (12)$$

where $\gamma \in (0, 1)$ is the discount factor. Then, there exists an optimal policy, π^*, which is defined as:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a). \quad (13)$$

Thus, the agent learns how to modify its behavior in the motivation of obtaining maximum cumulative reward over its lifetime.

In general, the RL methods, which use the value function approach, can be divided into two categories: model-free and model-based methods. In contrast to model-free (or *direct*) methods, where the value function is updated by directly using experience (s, a, s', r) obtained from the environment, model-based (or *indirect*) methods update the value function from a model of the environment. In particular, each time when the agent obtains a new experience, it learns a model of the environment by estimating the state transition distribution $P(s'|s, a)$ and the reward function $R(s, a)$ for each state-action pair. Then, the agent performs the exact planning on the learned model to update a policy by using various methods such as policy or value iteration [5]. As a result, the typical structure of the model-based RL methods is sequential as it is depicted in Fig. 3. In addition, the model-based RL methods have a better sample efficiency compared to the model-free methods, since the sample-efficiency of the model-based RL algorithms is only limited by the number of actions the agent performs to learn a decent model of the environment. On the other hand, planning and model learning are computationally expensive, and therefore, most of the model-based RL algorithms are not really feasible for real-time systems. Fortunately, TEXPLORE utilizes a parallel architecture which allows to take actions based on the current policy as quick as required without waiting for the completion of planning and model update [9].

A typical model-based agent consists of three main components, namely *the action selection*, *model learning*, and *planning*. In order to separate the computationally-intensive processes from the action selection part, TEXPLORE places each component on its own parallel thread. Thus, the planning and model learning run in the background; while the action selection part directly interacts with the environment by taking actions as fast as required based on the most recent policy.

As for model learning, TEXPLORE learns a model of the environment by using decision trees, namely C4.5 algorithm implementation [11]. Decision trees learn to predict the relative change in the state (or the transition effects), $s^{rel} = s' - s$, and the reward function based on the state-action pair (s, a). The

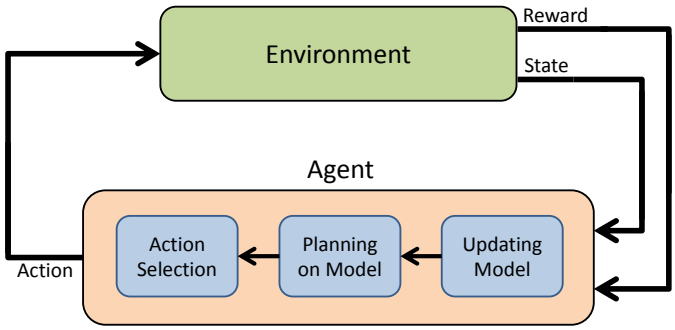


Fig. 3: The typical model-based reinforcement learning scenario.

learning of the transition effects provides a better generalization of the impacts of actions across the states when compared to the learning of the absolute outcomes. However, a single model may incorrectly represent the environment. Therefore, several models of the environment are learned by forming a random forest [12]. The final representation of the environment is the average of the representations of the models derived from decision trees. Because of averaging, the uncertainties in the models are naturally incorporated in the final model. Thus, TEXPLORE incorporates generalization with targeted exploration on states that are both uncertain in the model and promising for the final policy.

In contrast to conventional model-based RL methods, which use value iteration for exact planning, TEXPLORE utilizes Upper Confidence bounds applied to Trees (UCT) algorithm [13] to plan *approximately*. The planning is performed by simulating paths (roll-outs) starting from the current state s to a maximum search depth. Along its way, it updates the value function by using the most recent model of the environment. Since the planning runs in the background without stopping, more accurate policies can be obtained with more roll-outs.

In Fig. 2, the control architecture of the proposed solution is given. In this architecture, the agent is the high level decision making system. As can be seen from Fig. 2, the action selected by the agent is translated to the trajectory commands for UAV to understand. Then, the position control provides the specific low level control commands to UAV.

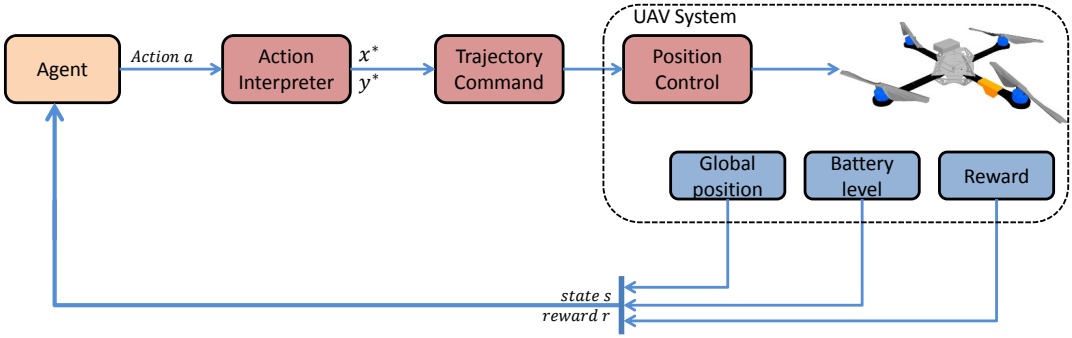


Fig. 2: The control architecture of the UAV.

IV. Experiments

To achieve a safe autonomous flight, the quadcopter should always monitor its battery level, motor conditions, sensor readings, and if necessary, it should make intelligent decisions, for example changing the route for the intermediate recharging of batteries. In this section, the experiment is arranged to illustrate a quadcopter with a mission of finding an efficient route to the goal location when it is limited in battery life. During its flight, the quadcopter constantly monitors its position and battery level and decides when it should change its route for the intermediate recharging. In addition, usually when the quadcopter reaches the goal, it does not have enough battery to make further surveillance in the target region. To handle such cases, our quadcopter agent should also learn the best recharging location in order to have the highest possible battery level when it reaches the goal.

In order to represent the aforementioned scenario, the quadcopter agent is simulated within a grid map as shown in Fig. 4. The quadcopter agent starts randomly in the left part (within the red box) of the map and tries to reach the goal location (the blue cell) in the right part of the map. The bottom (BRR) and top (TRR) rows represent the possible recharging locations for the quadcopter agent. It should be noted that at each episode the quadcopter agent starts with the maximum battery level, but it cannot reach the goal location without recharging at least once.

The state of the agent is described by 3 parameters, namely (x, y) location on the grid ($s_x \in [0, 40], s_y \in [0, 20]$) and the battery level ($s_{battery} \in [0, 25]$). In each state, the quadcopter agent has 8 possible actions, i.e. it can move horizontally, vertically, and diagonally to any adjacent square. The actions, which result in horizontal or vertical movements, have a reward of -1; while the actions with diagonal movements have a reward of -1.4. With each performed action the battery level decreases by one and when it drops down to 0 the agent terminates its episode with reward -500 which can be considered as the crash of UAV. In addition, when the quadcopter hits the boundaries with an action, it will remain in the same location and receive reward -10. When the quadcopter agent reaches the goal location, the episode will finish with a reward

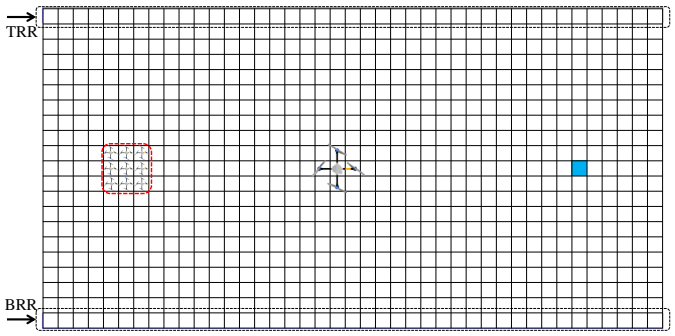


Fig. 4: The grid map. Starting states are within the red box, the goal is in blue. TRR and BRR stand for the top and bottom recharging rows, respectively.

of 0. Finally, the reward function for recharging batteries in coordinate s_x in the top or bottom row is defined as:

$$R(s_x) = b + s_x \bmod 41, \tag{14}$$

where $b = -50$ for the top row and $b = -100$ for the bottom row; while s_x dictates how the rewards change across columns. The difference in the recharging costs between the top and bottom row is introduced in order to address the cases when UAV is flying on top of a river and it has difficulties to recharge on one of the banks of the river due to rocks or dense forests. Thus, the grid world has 21×41 cells, where at each cell there exist 8 possible actions and 26 possible battery levels.

In the conducted experiments, TEXPLORE utilizes models with 15 decision trees. The discount factor, γ, is set to 0.9; while UCT algorithm has $\lambda = 0.1$ for planning. We also choose to act greedily relative to the learned model of the environment. As for Q-Learning method, it is used with $\epsilon = 0.1$ (for ϵ-greedy exploration), the learning rate $\alpha = 0.3$ and the action values are set to 0 at the initial stage.

Similar to [7], we initialize the model learning by giving sample seeding experiences from the environment, namely one experience from each recharging row, a single experience of the goal state and a single experience of the terminal state when the battery level is zero ($s_{battery} = 0$). In addition, when the agent selects a particular action there exist 0.2 probability that it will move to another adjacent square instead of going as intended. This was done to increase stochasticity in the experiment. All initial reward values and constants (γ, λ, α and ϵ) were chosen from an experienced guess, and then they were tuned by trial and error for better performance.

V. Results

In this section, we describe the results obtained from the conducted experiments. The experiments are implemented on the simulated quadcopter using a combination of ROS and Gazebo. To simulate the grid world, the dimension of a cell is set to $0.2\mathrm{m}$ in Gazebo world; while the height of the quadcopter flight is fixed to $2\mathrm{m}$. The (x, y) location of UAV is obtained by taking a global position of UAV from Gazebo simulator. This position is then discretized in order to work in the grid world. The quadcopter starts around the origin and moves towards the goal located at $x = 6\mathrm{m}$ and $y = 0\mathrm{m}$ in Gazebo world. Since the model learning and planning run in the background, we can choose any required action selection rates. For our simulation, the action rate is set to $10\mathrm{Hz}$.

Firstly, TEXPLORE based approach is compared with the baseline model-free method known as Q-Learning. Figure 5 illustrates the average rewards per episode over 10 trials of these methods. Since Q-Learning implements an exhaustive exploration to achieve a near-optimal behavior, it fails to reach the goal location over the first 150 episodes. As for TEXPLORE, it performs targeted exploration on states that are uncertain in the model and promising for the final policy. Therefore, it quickly accrues more rewards than Q-Learning. In the beginning, the agent explores the environment and receives the similar rewards as Q-Learning. However, due to

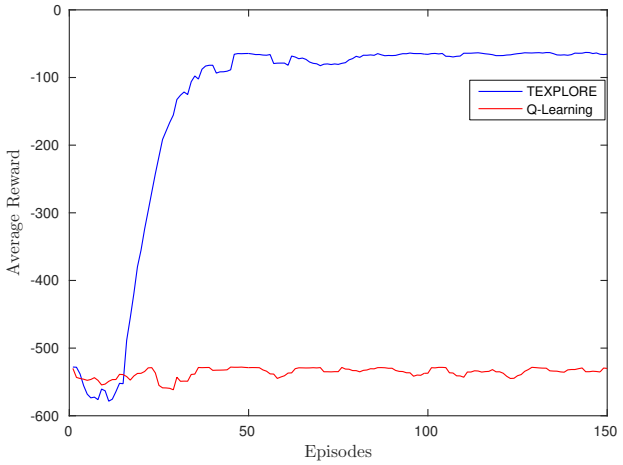


Fig. 5: Average reward per episode over the first 150 episodes. Results are averaged over 10 trials.

the targeted exploration, the agent gradually improves its performance and starting from episode 25, it obtains significantly more rewards per episode than Q-Learning based method.

To see the learning process in detail, we also present heat maps (averaged over 10 trials) which show the states visited by the agent. The brightness of the colors shows the number of times the quadcopter visited that specific cell. More visits are indicated by brighter colors. The heat map over the first 150 episodes is shown in Fig. 6a. Before the learning starts, the color of the heat map is fully monotonous as shown in Fig. 6b. In the first 20 episodes, the quadcopter agent explores the map and tries to find states that are promising for the final policy (Fig. 6c). After finding promising states, the agent continues the exploration on states close to the recharging cells and the path to the goal as depicted in Fig. 6d. In Fig. 6d, we can also observe that cells in the top part are brighter than cells in the bottom part. This can be easily explained by recalling that recharging at cells in the bottom row results in more negative rewards compare to the top row. Because of more negative rewards, the agent concludes that it is not worthwhile to recharge at the bottom row, and after episode 50, it only recharges from the top row and acts greedily relative to its learned model as illustrated in Fig. 6e.

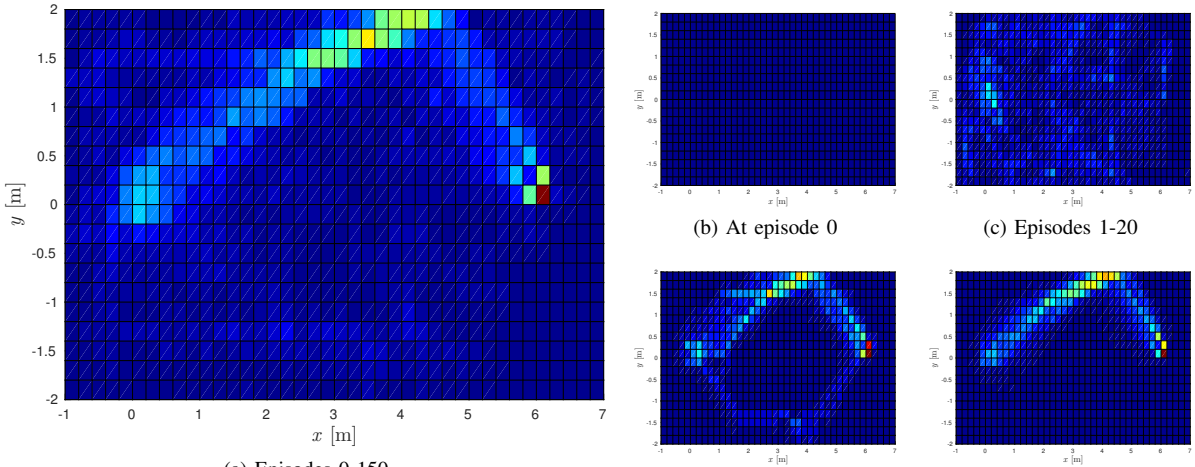
Figure 7 shows the policy map at the maximum battery level ($s_{battery} = 25$) after 150 episodes. We used the policy map with the arrows to illustrate in what direction the agent will move from any specific cell at the maximum battery level. Initially, the quadcopter agent does not know how to reach the goal location and where to recharge the batteries. However, after 150 episodes, we can clearly see that the agent learns a proper route and knows in which direction to move in order to reach the goal. It should be noted that the policy map is obtained for the battery level 25, for another battery level, we will obtain a different policy map that will show the agent's movements for that particular case.

Figure 8 shows a sequence of snapshots of the flight. The sequence consists of four snapshots representing the quadcopter's flight stages. The trajectory of the quadcopter's flight is depicted in Fig. 9.

VI. Conclusions and Future Works

This paper presents an implementation of model-based RL method as the high level control of UAV that has to find an efficient route when it is constrained in battery life. The proposed approach has been tested with a simulated quadcopter UAV in ROS and Gazebo environment. The experimental results have shown that our method is able to learn a good behavior in a few iterations and performs actions in real-time. Moreover, we have shown that our approach significantly outperforms Q-learning based method. The obtained results also demonstrate that RL algorithms, especially TEXPLORE, is a promising choice for improving the autonomous behavior of UAVs.

This work is just one step towards the fully autonomous navigation of UAVs. Future studies will explore the challenges



(a) Episodes 0-150

(b) At episode 0

(c) Episodes 1-20

(d) Episodes 21-50

(e) Episodes 51-150

Fig. 6: The heat map of the number of visits averaged over 10 trials. More visits are indicated by brighter colors.

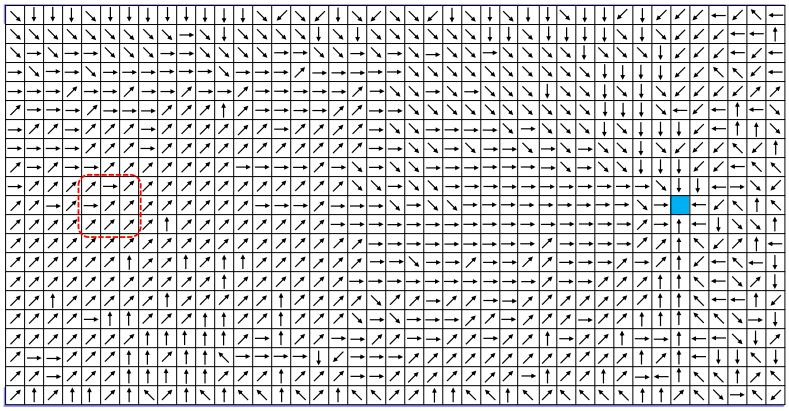


Fig. 7: The policy map at battery level 25 after 150 episodes. Starting states are within the red box, the goal is in the blue cell.

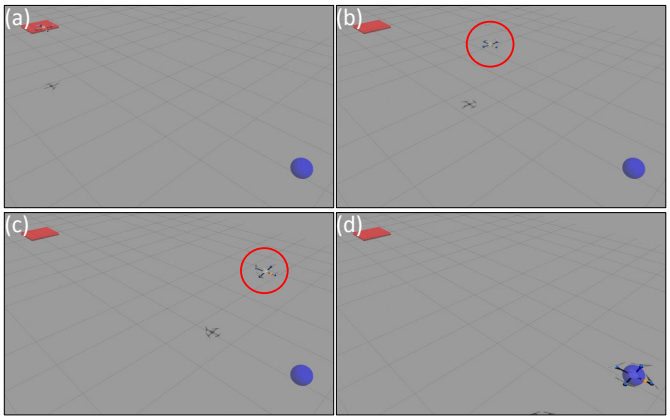


Fig. 8: A sequence of snapshots of learned trajectory. The location of UAV: (a) at initial position; (b) at turning stage towards recharging (c) at recharging location; (d) at the goal. The video is available at https://goo.gl/eLD8fJ.

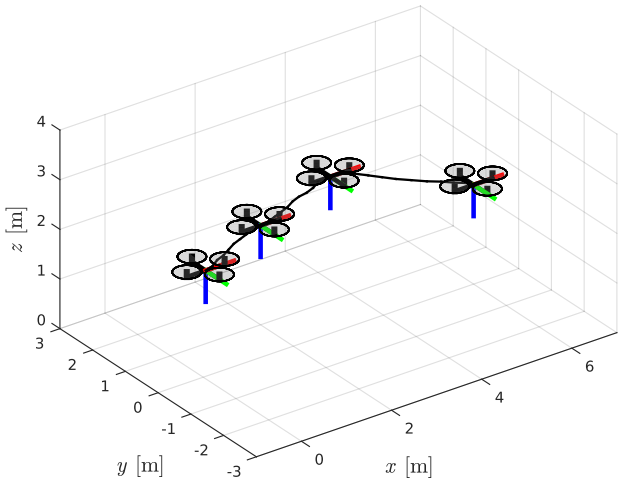


Fig. 9: The learned trajectory at episode 150. UAV starts around the origin and moves towards the goal.

of operating in the continuous domain and focus on conducting experiments on real-world quadcopter UAVs. In addition, we will explore other model learning and planning methods for TEXPLORE in order to further improve the performance of the proposed approach.

Acknowledgment

The research was partially supported by the ST Engineering - NTU Corporate Lab through the NRF corporate lab@university scheme.

References

[1] C. Fu, A. Carrio, M. A. Olivares-Mendez, R. Suarez-Fernandez, and P. Campoy, "Robust real-time vision-based aircraft tracking from Unmanned Aerial Vehicles," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5441–5446.

[2] M. A. Olivares-Mendez, C. Fu, P. Ludivig, T. F. Bissyandé, S. Kannan, M. Zurad, H. Voos, and P. Campoy, "Towards an autonomous vision-based unmanned aerial system against wildlife poachers," *Sensors*, vol. 15, no. 12, pp. 31 362–31 391, 2015.

[3] A. Birk, B. Wiggerich, H. Bülow, M. Pfingsthorn, and S. Schwertfeger, "Safety, Security, and Rescue missions with an Unmanned Aerial Vehicle (UAV)," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 1, pp. 57–76, 2011.

[4] C. Fu, A. Carrio, and P. Campoy, "Efficient visual odometry and mapping for Unmanned Aerial Vehicle using ARM-based stereo vision pre-processing system," in *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, 2015, pp. 957–962.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An Introduction*. Cambridge, MA: MIT press, 1998.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A Survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[7] T. Hester and P. Stone, "TEXPLORE: real-time sample-efficient reinforcement learning for robots," *Machine Learning*, vol. 90, no. 3, pp. 385–429, 2013.

[8] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge England, 1989.

[9] T. Hester, M. Quinlan, and P. Stone, "RTMBA: A real-time model-based reinforcement learning architecture for robot control," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 85–90.

[10] D. Zhou and M. Schwager, "Vector field following for quadrotors using differential flatness," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6567–6572.

[11] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.