A COMPREHENSIVE IMPROVEMENT OF DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS UAV NAVIGATION USING THE NOVEL REWARD FUNCTION AND ACTOR-CRITIC MODEL ENHANCER METHODS

MANIT CHANSUPARP

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2022

KMITL-2022-SC-D-002-121

Thesis Title	A Comprehensive Improvement of Deep Reinforcement Learning for Autonomous UAV Navigation Using the Novel Reward Function and Actor-Critic Model Enhancer Methods
Student Name	Manit Chansuparp
Student ID	61605012
Degree	Doctor of Philosophy (Computer Science)
Department	Computer science
Year	2022
Thesis Advisor	Asst. Prof. Dr. Kulsawasd Jitkajornwanich

Abstract

The autonomous navigation has gained many attentions in recent years due to many factors such as exponential growth of logistic industry, the need for social distancing in contagious pandemic. Additionally, one thing that also gains attentions parallelly with it is the Unmanned Aerial Vehicle (UAV) or also known as drone. The usual UAV are small vehicles which don't need any pilot and can fly to the destination fast since traffic jam on ground can be avoided for them.

There are many recent works proposing the autonomous UAV navigation method. The most of them chose deep reinforcement learning as the learning model and gain satisfactory results. However, those works are still far from real-world adoption since they only test the methods on static and unrealistic environments and low dimensional action space. Contrarily, in real world, the environment is dynamic and also the UAV can move freely in 6 Degrees Of Freedom (6DOF). Plus, after the test of latest method on our complex environments, we also found that some problems are caused by irrationality of the traditional reward function and the apprehensive behavior of agent (UAV) which is the being that the agent will move back and forth repeatedly when it faces against risky scenes. Hence, the aim of this work is to propose the method which can be conducted on more realistic environment while still retain the high success rate because, in real-world adoption, the collision can get UAV the severe damage. The proposed method consists of parts which are used to solve or alleviate the different problems in the past. The main parts are as followings: First is the point cloud simplification with truncated icosahedron structure which make enormous cloud points handleable even for

microprocessor. Second is the Augmentative backward reward function (ABR^{+}). This function has more rational reward dispensation mechanism to help wipe out the agent's bias against the goal point. Third, LifeGuard, it is an enhancer for the actor-critic model that urges the learning process to recognize the limitation of risk and help reduces the apprehensive behavior. The experimental results shown that the proposed method can solve and alleviate the problems in the past. The success rates were 10% higher than that of the current state-of-the-art method, FORK, in static environment and 2.4% for dynamic.

Keywords : Actor-Critic model, Autonomous navigation, Deep reinforcement learning, Drone, Point cloud simplification, Reward function, UAV

Acknowledgements

I cannot express enough thanks to my advisor, Asst Prof. Dr. Kulsawasd Jitkajornwanich, for his continued support and encouragement. I offer my sincere appreciation for the learning opportunities provided by The King Mongkut's Institute of Technology Ladkrabang. In addition, to the thesis committee, your suggestions are a great motivation for me to make the better work. My completion of this project could not have been accomplished without the support of my family, Mana, Pen, Parida, and Thip. Finally, I would like to praise my teacher who has been consistently guided me the way of research, Dr. Annupan Rodtook. Thanks to your confidence in your students.

Table of contents

	Page
Abstract	i
Acknowledgements	iii
Table of contents	iv
List of tables	vi
List of figures	vii
Chapter 1 Introduction	1
1.1 Research motivation	1
1.2 Objectives of the study	2
1.3 Scope of the study	2
1.4 Benefits	2
Chapter 2 Theory and literature reviews	4
2.1 UAV Kinematics	4
2.2 Reinforcement Learning	7
2.3 Deep Deterministic Policy Gradient (DDPG)	7
2.4 Twin Delayed DDPG	8
2.5 Forward-Looking Actor	10
2.6 Reward function	10
2.7 Autoencoder	11
2.8 Prioritized Experience Replay	12
2.9 Literature reviews	13
Chapter 3 Research methodology	16
3.1 Point cloud simplification with Truncated Icosahedron structure	16
3.2 Sequential transition learning	18
3.3 Unsupervised trajectories classification with all...	18
3.4 Augmentative backward reward function	20
3.5 Actor-Critic-LifeGuard	22
3.6 Model Architecture	23
Chapter 4 Main results and discussion	27
4.1 Experimental setting	27
4.2 Trajectories classification	30
4.3 Autonomous UAV navigation	31
4.3.1 Training in static environment	31

4.3.1 Training in dynamic environment 32
4.4 Complexity 32
4.5 Other navigation task 34
Chapter 5 Conclusions and suggestions 35
5.1 Conclusions 35
5.2 Suggestions 35

References 37
Author biography 42

List of tables

Table		Page
4.1	Parameters and Variables used in this work	28
4.2	Sample complexity (Million steps) of TD3-FORK+ABR$^+$ and TD3-LG+ABR$^+$	33

List of figures

Figure	Page
2.1 UAV Position and Euler angles in a 3D environment	4
2.2 Autoencoder network	11
3.1 LiDAR sensor simulation used in this work, Velodyne VLP-16 model	16
3.2 Left: Truncated icosahedron structure (soccer ball-like geometry), ...	17
3.3 Network architecture of all Positive Siamese Auto Encoder	19
3.4 The process flow of Augmentative Backward Reward function (ABR^{+})	21
3.5 Workflow of the proposed method for autonomous UAV navigation task	24
4.1 Left: static environment (no moving objects except the agent)...	27
4.2 Trajectories data distributions, Left: of one-shot learning...	29
4.3 Success rates of using TD3 with ABR (one-shot learning) and ABR^{+} (PSAE)...	30
4.4 Results of using DDPG, TD3, TD-FORK, and TD-LG with TR and ARB in...	31
4.5 Results of using TD3, TD-FORK, and TD-LG with TR and ARB in dynamic...	32
4.6 Left: BipedalWalkerHardcore task. Right: episodic rewards of using...	34

Chapter 1
Introduction

1.1 Research motivation

Due to gradually increase of ground traffic congestion and contagious pandemic, the Autonomous Unmanned aerial vehicle Navigation (AUN) has been deemed as the new promising way for many industries. Nowadays, the UAV are mainly used only in the tasks that are risky or difficult for human to reach (Jain et al. 2018; Sudhakar et al. 2020). It is so pity that automatically using the UAV on common tasks like an errand for food are still not paid attention as much as it should be. The hindrance may be that the enormous size of sensing data, expensive cost of the UAV and its sensor, and severe damage to the UAV occurred from just moderate collision (MAHMUD 2021).

Despite a number of techniques trying to accomplish the AUN in recent years, the main problems still exist. The success rates of the latest works about AUN, which is the rate that the agent can reach to the goal point, were still moderate even in simplified static environments, low dimensional action space, and unrealistic sensor (Elmokadem and Andrey 2021; Hu et al. 2020). If the environment is complex and dynamic, then the collision occurs frequently in training process and this will cause to the apprehensive behavior of the agent. The agent will be stuck in the loop of moving back and forth in front of a risky scene until running out of limit steps. In addition, the traditional reward function (TR) is composed of three conditions: if arrived, collided and other. The agent gets punished with negative or less reward when it collides and gets some small reward if it's not. The problem is that, when it arrived to the goal point, it usually gets a large reward to highly encourage this decision. This mechanism will work only in the environment that is simple enough to not have many similar patterns of scene and start point is fixed. The agent will have a bias to the scene which is similar to the one at the goal point and it will decide to take a same action whenever it faces that similar one even though it is a bad decision. Plus, the collision rate was still concerned that can't let it happen in real-world adoption. Simultaneous localization and mapping (SLAM) for UAV in precise tasks like agriculture task has been also difficult challenge (Sadeghzadeh et al. 2021). In sum, the AUN's problems in many aspects are just alleviated but not wiped out.

All our experiments were conducted on the realistic simulation due to the fact that the collision rate of even the state-of-the-art method is still concerned and the negligible collision may cause inability to the UAV. If the experiments are conducted with real

devices, the implementation cost will be around 5,100 USD which separated to 1,100 USD for the quadrotor UAV and 4,000 for VLP-16 LiDAR sensor.

1.2 Objectives of the study

1) Construction of realistic 3D simulation for autonomous navigation.

The UAV, sensor, and environment in this work will be simulated in the almost-same detail, size, and weight as reality. The sensor is the same specification as of actual device. Plus, the gravity, aerodynamic, magnetic field, and the physics system are set just as same as of the real world. The difficulty level can be altered by spawning the moving objects into the environment.

2) Point cloud simplification which can reduce enormous size of sensing data into the size which is affordable for even microprocessor.

3) To resolve or alleviate the past problems including irrationality in traditional reward function and apprehensive behavior of the agent.

4) Proposing the new autonomous navigation method which work in both realistic environment and high dimensional action space.

1.3 Scope of the study

This work develops the motion control for AUN based on reinforcement learning. The agent is in form of UAV which can move freely in space and there are external forces including gravitational and aerodynamic forces. In term of environments, realistic simulation is used since the collision in real world can cause inability to the UAV.

1.4 Benefits

1) Realistic and open source frame work for autonomous navigation.

The framework which was proposed in this work can be used in many various kind of navigation tasks even though there are differences in action space or kinematic.

2) Point cloud simplification specifically designed for motion control.

There have been many point cloud simplifications but all of them are designed for generic purposes, like 3D modeling. Point cloud simplification with Truncated Icosahedron structure was designed only for motion control and it made the output of modern LiDAR sensors affordable even with a microprocessor.

3) Novel reward function.

With rational reward dispensation mechanism of the proposed reward function, ABR^{+}, it helped reveal the authentic performance of reinforcement learning algorithms which were degraded by the bias in traditional reward function.

4) Improvement of the Actor-Critic model.

In most of navigation task, high risk means high reward but, on the other hand, a very thin exceeding of risk would turn out to be the punishment instead. This contrary difference can cause catastrophic forgetting to the agent. LifeGuard, the new role in Actor-Critic model, urged the agent to realized the limit of risk which it can take.

Chapter 2
Theory and literature reviews

There are only two physical things in the AUN task that are the agent and environment. The objective is to navigate the UAV to the goal point with no collision. It sounds simple but making it efficient in complex environment need several components. The details are described in this chapter.

2.1 UAV Kinematics

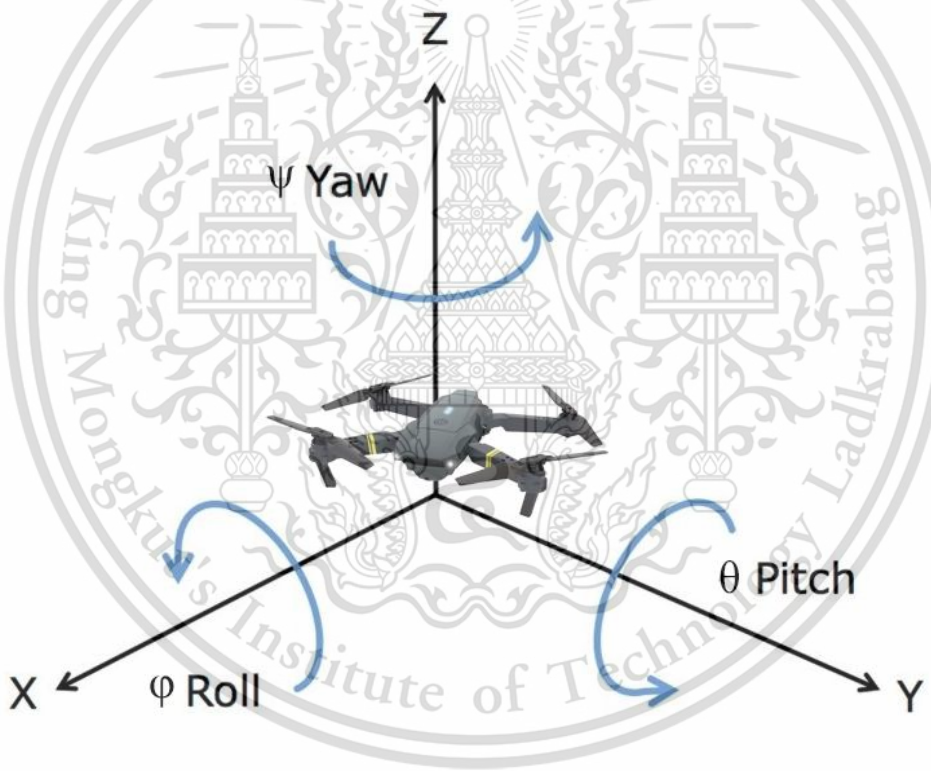


Figure 2.1 UAV Position and Euler angles in a 3D environment.

UAV is an aircraft which has no any pilot in it. This work chose one of the most popular kinds of UAV, the quadrotor, since its advantage over other kinds that it can maneuver ideally like capable of taking-off and landing vertically and also has powerful lifting power. As the name suggests, this kind of UAV has four rotors at the border of a square frame spinning independently to get a move over the six degrees of freedom (6DoF). Hence, to control its

motion requires six parameters including three linear velocities (l_x, l_y, l_z) and three angular velocities (a_x, a_y, a_z). The rotation along these axes were called roll, pitch, yaw angles ($^{\varphi,\theta,\psi}$), respectively. In blunt environments, the translation $(\dot{x}, \dot{y}, \dot{z})$ and rotation $(\dot{\varphi}, \dot{\theta}, \dot{\psi})$ of UAV on three dimensional cartesian coordinate space can be done by this:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} l_x \\ l_y \\ l_z \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \cos\theta \cos\psi \\ \sin\theta \cos\psi \\ \sin\psi \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ a_x \\ a_y \\ a_z \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad (2.1)$$

But in a realistic environment like used in this work, the moments and forces given by rotors are dampened during the flight by external forces including aerodynamic and gravity forces. So, to find the changes of UAV's motion can be done as follows.

$$C = \begin{bmatrix} F_{mx} \\ F_{my} \\ F_{mz} \\ M_{mx} \\ M_{my} \\ M_{mz} \end{bmatrix} - (E_g + E_a) \qquad (2.2)$$

$$F_{mx} = m(\dot{l}_x + a_y l_z - a_z l_y) - S_x(a_y^2 + a_z^2) - S_y(\dot{a}_z - a_x a_y) + S_z(\dot{a}_y + a_x a_z)$$

$$F_{my} = m(l_y + a_z l_x - a_x l_z) + S_x(\dot{a}_z + a_x a_y) - S_y(a_x^2 + a_z^2) - S_z(\dot{a}_x - a_y a_z)$$

$$F_{mz} = m(\dot{l}_z + a_x l_y - a_y l_x) - S_x(\dot{a}_y + a_x a_z) + S_y(\dot{a}_x + a_y a_z) - S_z(a_y^2 + a_x^2) \qquad (2.3)$$

$$M_{mx} = I_x \dot{a}_x - (I_y - I_z) a_y a_z - I_{xy}(\dot{a}_y - a_x a_z) - I_{xz}(\dot{a}_z + a_x a_y) - I_{yz}(a_y^2 - a_z^2) + S_y(\dot{l}_z + a_x l_y - a_y l_x) + S_z(a_x l_z - a_z l_x - \dot{l}_y)$$

$$M_{my} = I_y \dot{a}_y - (I_z - I_x) a_z a_x - I_{xy}(\dot{a}_x + a_y a_z) - I_{yz}(\dot{a}_z - a_x a_y) - I_{xz}(a_z^2 - a_x^2) - S_x(\dot{l}_z + a_x l_y - a_y l_x) + S_z(\dot{l}_x - a_z l_y + a_y l_z)$$

$$M_{mz} = I_z \dot{a}_z - (I_x - I_y) a_x a_y - I_{yz}(\dot{a}_y + a_x a_z) - I_{xz}(\dot{a}_x - a_z a_y) - I_{xy}(a_x^2 - a_y^2) + S_x(\dot{l}_y - a_x l_z + a_z l_x) - S_y(\dot{l}_x - a_z l_y + a_y l_z)$$

C denotes vector of translational and rotational changes in UAV's motion regarding to gravity force E_g and aerodynamics force E_a impacting against UAV's flight.

where F = force, M = moment, m = the mass of UAV, I_x, I_y, I_z = moments of inertia related to UAV axes, I_{xy}, I_{yz}, I_{zx} = moments of UAV deviation, S_x, S_y, S_z = static moments related to UAV axes.

$$E_g = mg \begin{bmatrix} -sin\theta \\ cos\theta\ sin\varphi \\ cos\theta\ cos\varphi \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.4}$$

g = gravitational acceleration ($9.81\ m/s^2$) (Since gravity performs through the center of gravity point of the UAV, no moments are created.)

$$E_a = \begin{bmatrix} F_a \\ M_a \end{bmatrix} \quad \alpha = \arctan\frac{l_z}{l_x} \quad \beta = \arcsin\frac{l_y}{l_{y0}}$$

$$TM = \begin{bmatrix} cos\alpha\ cos\beta & cos\alpha\ sin\beta & -sin\alpha \\ -sin\beta & cos\beta & 0 \\ -sin\alpha\ cos\beta & sin\alpha\ sin\beta & cos\alpha \end{bmatrix} \quad A = \frac{1}{2}\rho S_f l_{y0}^2$$

$$F_a = ATM \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} + \begin{bmatrix} X_y a_y \\ Y_x a_x + Y_z a_z \\ Z_y a_y \end{bmatrix} \tag{2.5}$$

$$M_a = A \begin{bmatrix} 0 & -z_a & y_a \\ z_a & 0 & -x_a \\ -y_a & x_a & 0 \end{bmatrix} TM \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} + ATM \begin{bmatrix} C_l \\ C_m \\ C_n \end{bmatrix} + \begin{bmatrix} L_x a_x + L_z a_z \\ M_y a_y \\ N_x a_x + N_z a_z \end{bmatrix}$$

where α, β = UAV approach and slide angles respectively, $\rho(Alt)$ = air density at altitude Alt, S_f = the surface of UAV, x_a, y_a, z_a = distances between the aerodynamic center and UAV's center of gravity, C_x, C_y, C_z = coefficients of components of aerodynamic forces : resistance, side and carrier force, C_l, C_m, C_n = coefficients of reclination, tilt and slumping aerodynamic moments, $X_y, Y_x, Y_z, Z_y, L_x, L_z, M_y, N_x, N_z$ = derivatives of components of the aerodynamic forces and moments respected to components of linear and angular velocities.

In this work, the UAV is allowed to move in four dimensions (l_x, l_y, l_z, a_z). The reason behind this is that almost all applications have required the UAV to always stay toward the goal

since many devices equipped on the UAV such as cameras and sensors have limited field of view (FOV).

2.2 Reinforcement Learning

Reinforcement Learning (RL) (Sutton and Barto 1999) is the process that the agent learns to interact with the environment to maximize the cumulative reward and what determines the action that the agent should take is called the policy (μ_θ). The interaction transition between the agent and environment is often in form of Markov decision process (Bellman 1957) represented as tuple (S, A, P, R, γ). S denotes a set of possible states in environment. A denotes a set of possible actions. P denotes a probability of executing action a at state s to reach next state $\acute{s}$ (where $a \in A$ and $s, \acute{s} \in S$). R denotes a reward gained after the change of state from $s \rightarrow \acute{s}$. $\gamma \in [0,1]$ a discount factor determining the agent's preference to the reward achieved in the past, present, and future. $\gamma = 0$ means that the agent will be myopia and only learns on the actions which provide an intermediate reward. At each time step t, the agent receives an observation o_t, which may be a part or whole of state s_t, and then takes an action a_t, which is generated by the policy, to environment. After that, the agent receives reward r_t and its state is replaced with a next state s_{t+1}.

In term of Deep Reinforcement Learning, it is RL that having experience replay buffer which is used to break the strong temporal correlations caused by sequentially generated states in RL. The experience replay leverages the memory to store past transitions. For each iteration, the fixed size array of transition is randomly selected as an input for the update of network parameters. As a result, DRL can surpass human in certain tasks, Atari (Mnih et al. 2013) for example.

2.3 Deep Deterministic Policy Gradient (DDPG)

DDPG, one of the most popular methods to deal with continuous domain. Lillicrap et al. (2015) broke the limitation in discrete action space of DQN (Deep Q Network) with DPG and a state-of-the-art actor-critic model. The policy in DQN can be formulated as follows:

$$\mu(s) = argmax_a \, Q(s, a) \qquad (2.6)$$

In DQN, the optimal action is received from taking argument max over the Q-values of all possible actions. DDPG bypasses this by letting the actor directly determines the action. The

actor-critic framework consists of two eponymous networks including actor-net and critic net and each network also has doppelganger network of itself called eval-net and target-net. The purpose of having these is to be time-delayed network of itself to alleviate instability issue of Q-learning in Deep Neural Network (DNN). The actor, which is used as a policy $\mu(s|\theta^{\mu})$, selects an action expecting to give a high reward for a given state s and the critic tell the actor how good the selected action is (value function $Q(s,a|\theta^{Q})$. To update the parameters, actor adopts DPG algorithm proved by Silver et al. (2014):

$$\nabla_{\theta^{\mu}} J(\theta^{\mu}) = \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s_i, a|\theta^{Q})|_{a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu}) \qquad (2.7)$$

where θ^{μ} , $\theta^{\mu'}$, θ^{Q} , $\theta^{Q'}$ denote the parameters of the eval-net, target-net in actor and critic network respectively. N denotes batch size.

As can be seen in formula 2.7, it is basically the sum of Q-value hence it needs to maximize this result. In term of the critic, the parameters θ^{Q} are updated by minimizing this loss function which is just a simple TD-error. It presumably represents how surprising the transition is to the agent:

$$L_c(\theta^{Q}) = \frac{1}{N}\sum_{i=1}^{N}((y_i - Q(s_i, a_i|\theta^{Q}))^2 \quad y_i = r(s_i, a_i)\gamma \acute{Q}(s_{i+1}, \acute{\mu}(s_{i+1}|\theta^{\acute{\mu}})|\theta^{\acute{Q}}) \quad (2.8)$$

$\acute{\mu}, \acute{Q}$ denote the parameters of actor and critic in target-nets.

The target-nets were used as a time-delayed copy of eval-net so everything in both networks is exactly the same unless the update process which occurs intervalley (soft update):

$$\theta' = p_{tau}\theta + (1 - p_{tau})\theta' \qquad p_{tau} < 1 \qquad (2.9)$$

This update mechanism makes the learning process gradually change and make it steadier. DDPG also exploits the experience replay buffer to store transitions (s_t, a_t, r_t, s_{t+1}) gained by the interactions between agent and environment. The batch, array of random transitions, is sampled from the buffer to be input of the network update process.

2.4 Twin Delayed DDPG

One of the DDPG's problems is that it tends to overestimate the Q-values and lead to policy's collapse since it directly associates with Q-function. Fujimoto, Hoof, and Meger (2018)

introduced the new method called Twin Delayed DDPG (TD3) to address this problem. Their concept is simple just increasing and doubling certain components of the DDPG. The differences from DDPG consist of three things as follows:

The first is target policy smoothing regularization. If the error approximator (Q-function) gives an incorrect narrow peak for some actions then the policy will be variant quickly. TD3 alleviates this by using the regularization in which the action used in target Q-function is added with clipped noise to all dimensions and then clipped to the valid action range. This process helps smooth the Q-function over the changes in action and, as a result, it will be harder for the policy to exploit the errors. The clipped action can be formulated as follows:

$$a_i = clip\left(\mu'\left(s_{i+1}|\theta^{\mu}\right) + clip(\epsilon, -c, c), a_{low}, a_{high}\right) \quad \epsilon \sim N\,(0, \sigma) \qquad (2.10)$$

Where σ denotes policy noise, c denotes noise bound.

The second is clipped double-Q learning. As the name suggests "Twin", TD3 has two Q-functions and the learning process is almost as same as of the DDPG but there are also some different details. TD3 exploits two Q-functions by modifying the initial TD-error calculation in formula (2.8) to this:

$$y_i = r(s_i, a_i) + \gamma(1 - d_i) \min_{j=1,2} Q'_j\left(s_{i+1}, a'_i \mid \theta^{Q'_j}\right) \qquad (2.11)$$

where d denotes episode status, 1 = done, 0 = not done.

Both Q-functions in target-net receive a single state-action pair and only the smaller Q-value is used. The loss is calculated by mean square Bellman error of the outputs of these Q-functions Q_1, Q_2.

$$\nabla_{\theta_j^Q}\left(\theta_j^Q\right) = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\theta_j^Q}\left(y_i - Q_j\left(s_i, a_i \mid \theta^{Q_j}\right)\right)^2 \qquad (2.12)$$

This smaller target Q-value and regressing with it will make sure that although the Q-value may not be accurate but it will be not that much over as before.

The third is slower target and policy updates. The target-net has been known that it could be used to reduce the error on multiple updates. The policy updates on high-error estimated value (Q-value) cause divergence then the update rate of policy network (actor) should be significantly lower than that of the value network (critic) to reduce the error before conveying it to the policy. Their results showed a better performance from enlarging the gap of update

rate between the policy and target networks. This accords to the two-time scale algorithm (Konda and Tsitsiklis 2003), often encourages to the convergence in linear setting.

These three changes help to alleviated the Q-value overestimation of DDPG and then greatly improves in both the learning speed and performance.

2.5 Forward-Looking Actor

In general actor-critic model, the policy is amended with the value the critic provides. The general loss function of this amendment is as followed:

$$L_o(\theta) = -Q(s_t, \mu(s_t|\theta)) \qquad (2.13)$$

Honghao Wei and Lei Ying (2020) proposed the Forward-Looking Actor (FORK), the policy amendment algorithm having foresight. It should be better if the action selection regards not only a reward pairing with the state but also the future benefit. To do this, the extra two neural networks called system-net and reward-net are added for learning the system model and the reward function respectively. The system-net learns to forecast a next state s_{t+1} which will happen after a given state-action pair ($F(s_t, a_t|\theta^F)$) but this network is not quite deterministic or stochastic policy optimization which mainly based on the model. In term of the reward-net, it will forecast a reward which receive after state-action pair ($R(s_t, a_t|\theta^R)$) was executed. Their loss function of the actor network could be formulated as follows:

$$L_{aF}(\theta^\mu) = -Q(s_t, \mu(s_t|\theta^\mu)|\theta^Q) - R(s_t, \mu(s_t|\theta^\mu)|\theta^R) - \gamma R(\tilde{s}_{t+1}, \mu(\tilde{s}_{t+1}|\theta^\mu)|\theta^R) - \gamma^2 Q(\tilde{s}_{t+2}, \mu(\tilde{s}_{t+2}|\theta^\mu)|\theta^Q) \qquad (2.14)$$

Where $\tilde{s}_{t+1}, \tilde{s}_{t+2}$ denote future states of s_t forecasted from system-net.

It is just forecasting of the next state and reward for a given state-action pair and let the forecasts participate in the policy amendment. Therefore, any actor-critic methods added FORK are still model-free. Their experiments on the open AI gym demonstrated that applying FORK upon the state-of-the-art model-free RL algorithms can improves the performance in term of cumulative reward. Plus, their work still pointed out that FORK outperforms other model-based RL algorithms.

2.6 Reward function

Reward function, as mentioned before, is the function which greatly drives the learning process of the RL. Its output indicates how good the action is. It is usually composed of three conditions as follows:

$$R(s,a) = \begin{cases} r_g & \text{if arrived the goal point} \\ r_c & \text{if collided} \\ \left(D_{prev} - D_{curr}\right) & \text{other} \end{cases} \quad (2.15)$$

r_g, r_c = large and low constants. D_{prev}, D_{curr} = previous and current distances between the agent and goal point, it may be Euclidean distance.

This function will urge the agent to reach the goal point with some large reward and dissuade the collision with less or negative reward.

2.7 Autoencoder

The autoencoder is a feature extraction method based on neural-network which not need any supervised knowledge. The network is composed of three parts: The first is an encoder compressing input data into a small representative while still containing its identity. The second is a bottleneck (or latent space), the most compressed representative of input data is stored here and is thus the gist of the network. The third is a decoder decompressing a compressed representative from the previous part to reconstructs it back to the almost original input data. The network is expected to give an output which is the same or similar to input to make sure that no gist is lost in compression process. The architecture of the whole network is shown in Figure 2.2.

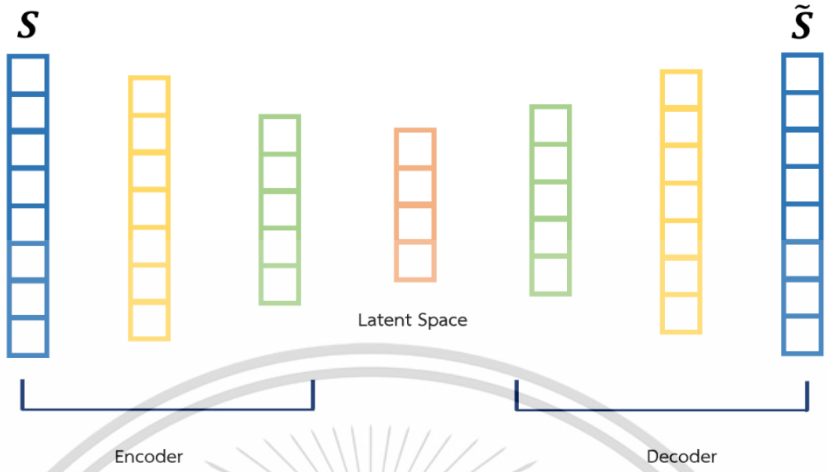


Figure 2.2 Autoencoder network.

The autoencoder have been developed into variants such as followings: Denoising autoencoder (Vincent et al. 2008), they infuse noise over the input to make the network more robust against discrepancy. Sparse autoencoder (Ng 2011), the method that regularize autoencoder by sparsity constraint, and so forth. But the interesting one is a combination between autoencoder and Siamese network model with triplet loss (Schneider et al. 2019) exploiting the good representative in latent space and distance-based classification ability. The triplet loss function helps the features or encoded data in same class get closer than from the other classes. This loss function can be formulated as follows:

$$L_T = \max\left(d(a,p) - d(a,n) + m, 0\right) \tag{2.16}$$

Where d denotes a distance matrix function. a, p denote an anchor sample and sample in the same class (positive sample). n denotes the different class (negative sample). m denotes the margin.

2.8 Prioritized Experience Replay

Prioritized Experience Replay (PER), Schaul et al. (2015), this method is based on the hypothesis that the agent can learns from certain transitions more than others. So, this is opposite to traditional DQN in which use randomly sampling and that means all transitions in the buffer are equal. PER uses the absolute TD-error value as the indicator of how surprise the transition is. In addition, many RL algorithms have already computed the TD-error for

network parameters update purpose and appending this value to each transition is a little storage burden. Even though the TD-error could be a representative of the amount which the agent can learn from the transition but selecting transitions greedily by its TD-error is still not deemed as a good choice because it can cause the less diversity in the batch of transitions. Thus, PER prioritizes transitions in buffer with absolute TD-error and also use this value as probability of being selected. After the update, those sampled transitions get updated their priority according to the loss which is the by-product of network parameters update. Despite this stochastic sampling, there is still no guarantee that batch's diversity is sufficient since some transitions with large priority may be ceaselessly replayed. Hence, two hyper-parameters α and β were brought to adjust the influence of priority on sampling process. These two parameters were annealed during the training process to make the sampling more uniform. The probability of being selected for transition i can be calculated as follows:

$$P(i) = \frac{p_i^{\alpha}}{\sum_{j=1}^{N} p_j^{\alpha}} \qquad (2.17)$$

where P = probability, p = priority (absolute TD-error)

However, PER tends to make the network frequently receiving transitions with large TD-error. As a result, the change of the parameter in the network is large and then the network will be prone to swing. The importance-sampling weights (Mahmood, Van Hasselt, and Sutton 2014) has taken the role of reducing the change of gradient magnitude. The importance-sampling weights formula is shown below.

$$w_i = \frac{1}{N^{\beta} P(i)^{\beta} max_j w_j} \qquad (2.18)$$

2.9 Literature reviews

Youn et al. (2020) proposed a UAV navigation technique which compounds the advantages of variant Rapidly Random Trees (RRT) algorithms. With the help of error state Kalman filter integration, their technique reduces redundancy on random node generating of the conventional path planning by new directional node generation. This new node

generation has the comparison of neighboring node costs (RRT-Smart) and dynamic directional path length (RRT*-GD) to better prevent unnecessary nodes.

Elmokadem and Andrey (2021) proposed the AUN method with also RRT algorithm called RRT-connect. In this method, the trajectory path of the UAV is generated by the convergence of the two trees toward each other from the start and goal points, differ from the conventional ones which path generated from only one root. They handled the collision avoidance with reactive control law which is the law designed controlling the UAV to move around the obstacle until it met the safe condition. Then, the UAV comes back along to the original planned path. Though, this method no needs any prior information of the environment but it needs some expertise knowledges in structuring process of the obstacle avoidance procedure which is hardly comprehensive for realistic environment. In addition, these two methods (Youn et al. 2020; Elmokadem and Andrey 2021) are able to afford only discrete spaces so the movement is rigid and not full potential.

Tong GUO et al. (2020) mentioned that although there are many deep reinforcement learning methods receiving quite good result but none of them can converges to satisfied point in high dimensional state-action spaces tasks like AUN. Hence, they tried bringing the divide-and-conquer strategy to the AUN. They divide the AUN into three tasks including collision avoidance, goal approach, and decision. These tasks were represented as three recurrent neural networks: avoid-net, acquire-net, and packed-net. Avoid and acquire networks have responsibility to offer packed-net a next action. After that, packed-net decides whether which one is a better action for current state, avoiding the obstacle or approaching the goal. The packed-net will get punished in form of a negative reward if it makes a wrong decision. Their experimental results shown that this model improves the collision rate, success rate, and velocity when compared with variant deep Q networks. Nonetheless, their method is still insufficient for the real-world adoption but supports more realistic conditions and action than path planning methods.

ZhiBin et al. (2020), In spite of many empirical proofs of the state-of-the-art method, TD3, which are almost maximum scores in many tasks (Fujimoto, van, and David 2018; Haarnoja et al. 2018), but there still have been tasks where TD3 can't achieve or be time consuming (Wei and Lei 2020). Hence, they addressed the slow convergence rate in TD3 by applying some prior knowledges which are in form of an alternative action. This action is produced from error and three parameters which are tuned to suit the environment.

This action and action coming via the actor are estimated for their Q values and then decide which one is more suitable. This alternative action helps boost the training speed and enhances the stability for TD3.

Apostolopoulos, Marcos, and Eirini (2019) introduced the framework which is the cooperation between Fully Autonomous Aerial Systems (FAAS) and Mobile Edge Computing (MEC) to navigate UAV for crowd surveillance purpose. They mainly addressed the data offloading problem. The devices need to satisfy their individual Quality of Service (QOS) while also regarding the energy consumption and they overcame these problems by using an on-policy RL algorithm named state-action-reward-state-action (SARSA) (Rummery and M. Niranjan 1994) and Satisfaction Equilibrium concept (SE).

Hu et al. (2020) proposed a supervised learning model which integrates human pilot experiences to DRL. This method is called Multi Experience Pools-DDPG (MEP-DDPG). It divides the buffer into X separate pools. $X-1$ is the number of human pilots giving the experiences and the last pool is for autonomous exploration. The batch is generated from randomly sampling transitions from all pools. They use one hyper-parameter η to anneal the proportion between the human pilot and exploration transitions. In the early stage of training, η is set to 1 and will be gradually decreased at each step to 0 and, simultaneously, the learning method also gradually turned up to initial DDPG. Around the second half of training, it will be fully DDPG. These storing and sampling mechanism ensures that the agent learns from good transitions and the batch is sufficiently diverse until the agent becomes fledged. In addition, they also proposed Model Predictive Control-Simulated Annealing (MPC-SA) for full filling the pilot experiences due to the fact that training process requires numerous pilot's transitions and it will be a lot of manpower to gather. With MPC-SA, human pilots have to only give some guidance, the rest will be generated automatically. Their experimental results showed that MEP-DDPG can deal with AUN in a high complexity environment better than DDPG, the success rates are around 60% and 37% respectively. But for a low complexity environment, DDPG is slightly better than MEP-DDPG instead.

Chapter 3
Research methodology

3.1 Point cloud simplification with Truncated Icosahedron structure

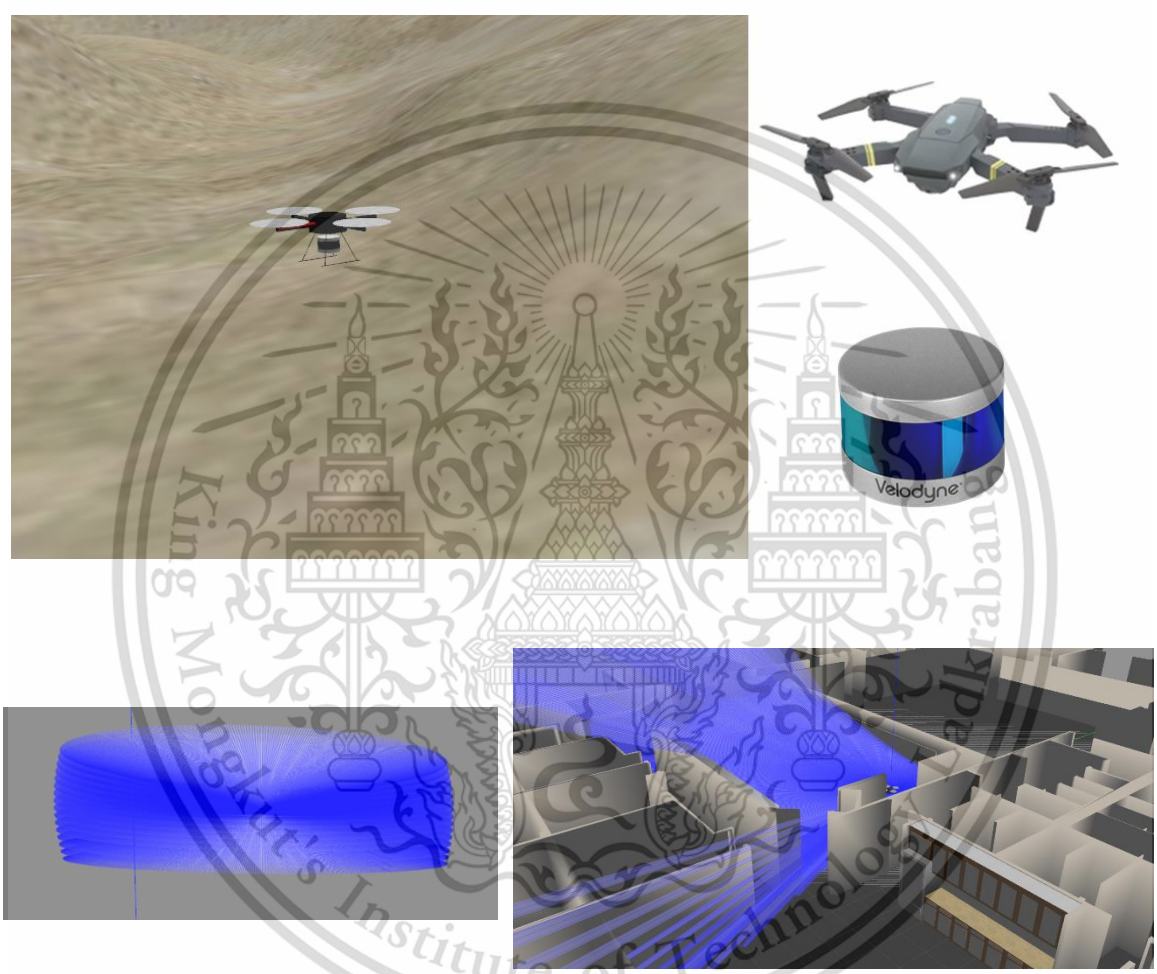


Figure 3.1 LiDAR sensor simulation used in this work, Velodyne VLP-16 model.

The soccer ball-like geometry shown in Figure 3.2 is called Truncated icosahedron. It is one of the Archimedean solids consisting of twelve pentagons and twenty hexagons. This ball has ever been formed physically for many UAV's purposes such as preventive cage (MAHMUD 2021) and moving sport balls (Nitta et al. 2015). For this work, it is used in novel way which the ball is visualized for both simplifying point cloud data and avoiding collisions simultaneously. The point cloud data (PCD) from simulated Velodyne VLP-16 LiDAR sensor is

the only perception the UAV has in this work. The sensor can provides a 100-metre sensor range, 360° 30° horizontal and vertical FOVs, and ~300,000 cloud points per second. With this huge of PCD, directly feeding it to the neural networks is unaffordable. If takes fixed-interval sampling to PCD until it becomes a moderate number, certain necessary data may be lost.

Figure 3.2 Left: Truncated icosahedron structure (soccer ball-like geometry), Middle: UAV in environment, Right: A UAV's perception of surrounding.

That is to say, the ball is used to extract only necessary data for AUN from the PCD. The UAV will be covered with the impalpable ball having 32 faces. The radius of the ball p_{radius} equals to a safe distance between the UAV and obstacles. Hence, the observation received from the environment at each time step will be simplified to a vector $(f_1, f_2, ..., f_{32})$, each f_i denotes a distance between the nearest cloud point, which trespasses into the ball at face i. f_i is 0 when there are no obstacles at the face. To find f, first, cloud points inside the ball are segmented with an implicit equation of the plane. This process is also known as an inclusion test.

$$F_i(p) = a_i p_x + b_i p_y + c_i p_z + d \qquad -d = n \times P \qquad (3.1)$$

The face of the ball is an oriented plane with a normal vector oriented outside $n_i(a_i, b_i, c_i)$ and p is the coordinate of the cloud point. P denotes a known point on the plane. The point where $F_i(p) < 0$ for at least one is outside the ball and can be ignored. After cutting out all outside points, it still needs to locate which face the point belongs to. This process is done by calculating a Euclidean distance of cloud point to the center point of each face c_i as follows:

$$f_{i=1,32} = \min_{j \in J} \sqrt{(p_{jx} - c_{ix})^2 + (p_{jy} - c_{iy})^2 + (p_{jz} - c_{iz})^2} \qquad (3.2)$$

where j denotes the number of cloud points inside the ball.

The shortest distance of point p with will be representative of the face f.

3.2 Sequential transition learning

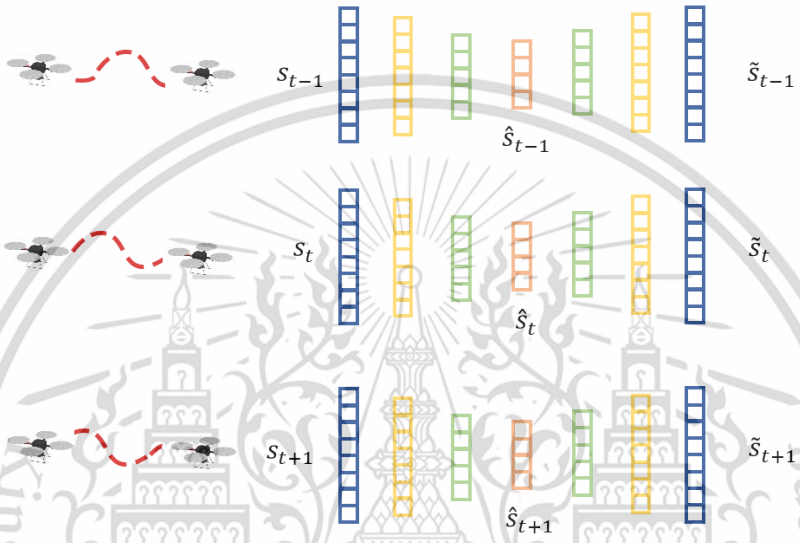
The single state s_i is only able to tell what the surrounding is at time i but no directional insight. It is difficult to extract useful information from the one frame transition. For example, if we have a photo of two humans playing tennis and be asked *"what direction the ball will move to, up or down?"*. Hence, there were thoughts that the agent could learn better on sequential transition (Kapturowski et al. 2018; Z. Lei et al. 2020) and their experimental results showed that it improves the performance in various RL tasks. Normally, s refers to one observation received from the environment but, for sequential transition, it is a sequence of observations. In this work, At each time, the agent takes an action to the environment and receives a new observation o which is combined vector of simplified PCD $(f_1,\ldots, f_{32})$, altitude of UAV z_t, and two angles including the head's UAV (camera direction for headless UAV) with a goal position h_t and a UAV position with a goal position g_t. Thus, the observation looks like this $(f_1,\ldots, f_{32}, z, h, g)$ and will be stacked up inside the fixed-size state in FIFO fashion. The state is sequence of o $\left(o_1,\ldots,o_{p_{frames}}\right)$, p_{frames} is the limit number of observations in state.

3.3 Unsupervised trajectories classification with all positive Siamese autoencoder

In unsupervised classification, one-shot learning is one of the traditional methods which the model has very few trainings before it has to be used. It often comes with convolutional neural network (CNN) since the convolutional feature is robust against wide range of domains. After the CNN extracts the features, many similarity indicators can be used to the features and find that which cluster the feature belong to. But in fact, this traditional method can only deal with modest number of clusters.

As mentioned in chapter 2.7, Siamese model with triplet loss for autoencoder can extract the features of data in same class to be close and be far for different class. Apparently, this is an ideal for the classifier but thing is that this model requires positive and negative samples and, in unsupervised learning, there are barely label on samples. In term of denoising autoencoder, the input has to be distorted but, unlike many kinds of data, a little shift on UAV's trajectory can totally change the meaning. All these problems have made the

unsupervised trajectories classification hard and also harder to concretely estimate the accuracy. Therefore, a pilot study on unsupervised trajectories classification is proposed in this work. This method adapts the ideas of Siamese autoencoder with triplet loss. With the absence of positive samples for UAV's trajectory, we set the hypothesis that the past and future states (s_{t-1}, s_{t+1}) are approximately similar to the state s_t. In sum, the adjacent states



of a given state are used as surrogates of positive samples. Siamese-triplet model is adopted as an autoencoder to make the encoding process resilient to both aspects, past and future, and this proposed method is called all Positive Siamese Auto Encoder (PSAE). The architecture of our PSAE is shown in Figure 3.3.

Figure 3.3 Network architecture of all Positive Siamese Auto Encoder.

Our Siamese-triplet model is composed of three identical networks which centrally share their parameter. Two positive states (s_{t-1}, s_{t+1}) and an anchor state (s_t) will be the input of its own network respectively. The output of network is the reconstruction of input $\tilde{s}$ which is expected to be almost the same to input according to the autoencoder theory. The features $\hat{s}$ in latent space for all networks are expected to be close to the anchor's feature in term of distance according to the concept of triplet loss. Hence, the aim of the loss function is to minimize the Mean Squared Error (MSE) between the input and output and to maximize the cosine similarities of the features pairs between the anchor with positives. The real product of this model stays in the latent space of the anchor network.

$$L_{PSAE} = p_{wae} \sum_{i=-1}^{1} || s_i - \tilde{s}_i ||_2^2 - (1 - p_{wae}) \sum_{j=-1,1} coss(\hat{s}_0, \hat{s}_j) \quad (3.3)$$

where p_{wae} denotes the tradeoff level between two terms, $coss$ denotes cosine similarity function.

After the feature extraction process, all the features $\hat{s}$ of each transition in buffer are clustered by the K-mean algorithm. The number of clusters k is determined by the elbow method, the classical method for unsupervised clustering (Liu and Yong 2021; Shi et al. 2021). The elbow method determines the optimal k number by iteratively calculating Sum Square Error (SSE), the sum of average Euclidean distances between each data points and its centroid, and then pick the point where SSE is starting to standing. This point looks like an elbow in the graph.

$$SSE = \sum_{k=1}^{K} \sum_{x \in C_k} || x - C_k ||_2^2 \quad (3.4)$$

where K denotes the maximum cluster number, C denotes the centroid of cluster.

For high dimensional state-action spaces, trajectory patterns can be enormous so there should be many steps of k values to try finding. After optimal k is found, all states in buffer can be labeled its cluster number. These pairs of state's feature and label are used as training set for the classifier to label the states in motion control learning process.

3.4 Augmentative backward reward function

There are many examples for that augmenting certain processes significantly boosted the convergence to optimal policy (Hu et al. 2020; Zhang et al. 2020). Hence, we develop a novel reward function called Augmentative Backward Reward function (ABR$^+$) in this work. The motivation of this development is the irrationality of traditional reward functions used in many recent AUN methods (Hu et al. 2020; Zhang et al. 2020). As can be seen in chapter 2.6, when the agent arrives to goal point, it will receive a large reward. This means that only the last transition of the success episode has a large reward and the rest in the episode are negligible compared to the last one. Although this part will help increase the success rate, it only works in the setting that is almost constant and very simple due to the fact that it produces a bias on the observation which is similar to the one at the goal. Our hypothesis is that it is true giving more reward for the success episode can urge the good behaviors but it should not be only the last transition to earn that much credit. Every transition in the success episode should be taken into account. The thing is how to assess the contribution of each transition. Hence, ABR$^+$ is designed to dispense the goal reward more appropriately. Our new

reward dispensation mechanism will allow all transitions in the success episode receive an additional reward ‘proportional’ to its proficiency. ABR+ is composed of the following processes:

First, when the agent arrives to the goal, all transitions in the episode are fed as the input into the backward reward dispensation process. Transitions at index i ,where $i\%p_{frames} = 0$ are selected to extract $p_{features}$ features $(e_1, \dots, e_{p_{features}})$ with PSAE to be a representative of trajectory τ of p_{frames} steps motion.

Second, classify τ to get the class l it belongs to with the classifier trained in offline. After that, insert floored sum reward $\left\lfloor \sum_{k=i}^{i-p_{frames}} r_k \right\rfloor$ of p_{frames} previous transitions into the unique stack of reward of trajectory’s history H at index l. The trajectory’s history is just the two-dimensional array which first dimension refers to the cluster number of trajectory (label) and the second refers to the rank of floored sum reward. Additional reward $r_{additional}$ will be added to rewards r of transitions since $i - p_{frames}$ to i in the buffer. The additional reward can be calculated as follows.

$$r_{additional} = m \times p_{addReward} \quad (3.5)$$

where $p_{addReward}$ denotes some small value, m = index of floored sum reward in the unique stack.

Apparently, $r_{additional}$ will be low or high depending on the rank of the sum reward in the unique stack. This mechanism helps drive the agent to move in a trajectory which gives a larger sum reward than before.

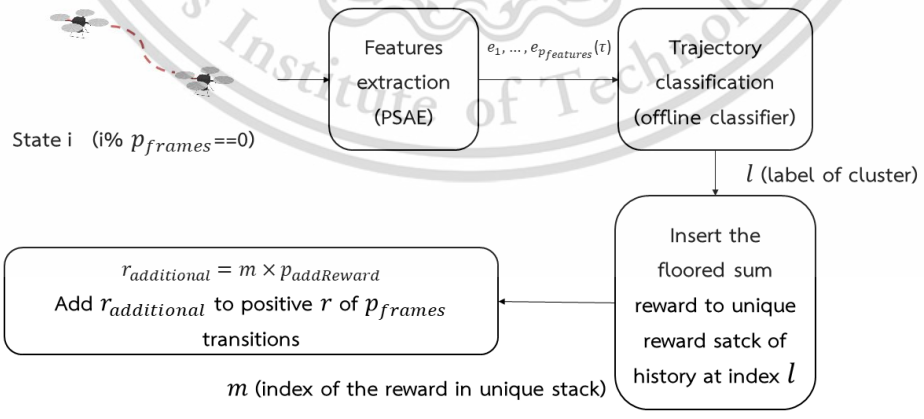


Figure 3.4 The process flow of Augmentative Backward Reward function (ABR+)

The reward function used in this work only has two conditions and can be formulated as below:

$$R(s,a) = \begin{cases} r_{collided} & \text{if collided} \\ w_1(D_{pre} - D_{cur}) + w_2\left(1 - \left(\frac{(z - p_{approAlt})}{p_{approAlt}}\right)\right) + & \\ w_3\left(1 - \left(\frac{v_h}{180}\right)\right) + w_4\left(1 - \left(\frac{v_g}{180}\right)\right) & \text{if not collided} \end{cases} \tag{3.6}$$

$$v_h = \begin{cases} 360 - h & \text{if h} > 180 \\ h \end{cases} \qquad v_g = \begin{cases} 360 - g & \text{if } g > 180 \\ g \end{cases}$$

Where W_{1-4} denote weight-parameters, D_{pre}, D_{cur} denote the previous and current Euclidean distances between the agent and the goal point, z denotes an altitude of UAV, h,g denote angles of the UAV's head and the position to goal point respectively, $p_{approAlt}$ denotes an appropriate altitude or flight level. Every step the agent takes an action, it receives the reward from the above function but if in case it arrives the goal point, the additional reward from ABR$^+$ will be added to all transitions in the episode proportionally.

3.5 Actor-Critic-LifeGuard

We propose adding the new third entity to Actor-Critic model, the LifeGuard (LG). It was inspired by the FORK method which, as mentioned in chapter 2.5, it uses supplementary forecast to improve the policy. After the trial of FORK on AUN, we found that the system-net's loss values during the training wag significantly. It can imply that forecasting the future state and reward is unaffordable for the complex environments having high dimensional state- action spaces. Hence, we aimed to develop policy supplement having lower dependency. The LG is simply a binary classifier which forecasts whether the agent will collide or not if it takes an action a_k at state s_k. The Stochastic Gradient Descent algorithm (SGD) was adopted for learning the classifier because its scaling capability and linear complexity that make the computation still fast even with enormous and high dimensional data (Mittal, Dev, and Sanjiban 2015). Let (x_n, y_n) is a set of training samples, $x \in R^m$ is features (for this work, x is the last observation in state) and $y \in \{1,0\}$ is label (collided and not). One thing which should be aware is that the SGD is sensitive to feature scaling so all

attributes in features must be scaled to (0,1). The output of SGD classification can be gained by this:

$$c(x) = w^T x + j \qquad (3.7)$$

where $w \in R^m$ denotes model parameters and $j \in R$ denotes intercept.

w is found by minimizing the regularized training error with this cost function:

$$E(w, j) = \frac{1}{n}\sum_{i=1}^{n} L\big(y_i, c(x_i)\big) + \alpha R(w) \qquad (3.8)$$

where R denotes regularization term, the L2 norm. α denotes a regularization hyper-parameter.

L denotes the loss function and the logistic regression is used for that.

$$L\big(y_i, c(x_i)\big) = \log\left(1 + \exp\left(-y_i c(x_i)\right)\right) \qquad (3.9)$$

Every update time step, a training sample is taken and the model parameters are updated according to this rule:

$$w \leftarrow w - lr\left[\alpha\frac{dR(w)}{dw} + \frac{dL(w^T x_i + j, y_i)}{dw}\right] \qquad lr^{(t)} = \frac{1}{\alpha(t_0 + t)} \qquad (3.10)$$

where lr, t_0 denote learning rate and initial time.

After the LG is trained to some extent then the collision forecast can be used to improve the policy (Actor) like this:

$$L_{aL}(\theta^{\mu}) = -(Q(s_t, \mu(s_t|\theta^{\mu})|\theta^{Q}) - c\big(o_t + \mu(s_t|\theta^{\mu})\big)p_{cp}) \qquad (3.11)$$

where o denotes the last observation in state s, p_{cp} denotes the hyperparameter controlling the diminishing of Q value.

3.6 Model Architecture

The main entities in this work consist of: the agent (UAV), goal point, motion controller, and environments. Every time step t, the agent perceives the observation of surrounding o_t which is abundant points in space, around few hundred thousand cloud points. Hence, it will be simplified with point cloud simplification with truncated icosahedron structure to screen

only the cloud points the agent should focus on. The PCD is turned into 35 attributes $o_t = (f_1, \dots, f_{32}, z, h, g)$ including the 32 representatives of each ball face, altitude of agent, two angles of the agent's head and position to goal point respectively. The o_t will constitute a state $s_t = (o_{t-p_{frames}}, \dots, o_t)$. The motion controller receives s_t as input and then output the action a_t. Behind this process is the policy update which there are three networks, Actor-Critic-LifeGuard, contribute each other to gain optimal policy. N transitions in the replay buffer B are sampled randomly to form a batch and sent to the actor, critic, and lifeguard networks as an input. The model parameters update is almost the same as of the TD3 but the eval-net actor update is replaced with Formula 3.11. After executing a_t, the agent will move to new step and receive o_{t+1}, reward r_t, episode ending status d_t from the environment. All these attributes constitute the transition $(s_t, s_{t+1}, a_t, r_t, d_t)$, $s_{t+1} = (o_{t-p_{frames}+1}, \dots, o_t, o_{t+1})$ and then be stored in B with FIFO fashion. If d_t is 1 and the agent arrives to the goal point then all transitions of the episode in B are brought to calculate additional rewards. Rewards r in the transitions will be increased with $r_{additional}$ from the ABR$^+$ function. The work flow of the proposed method can be illustrated in Figure 3.5.

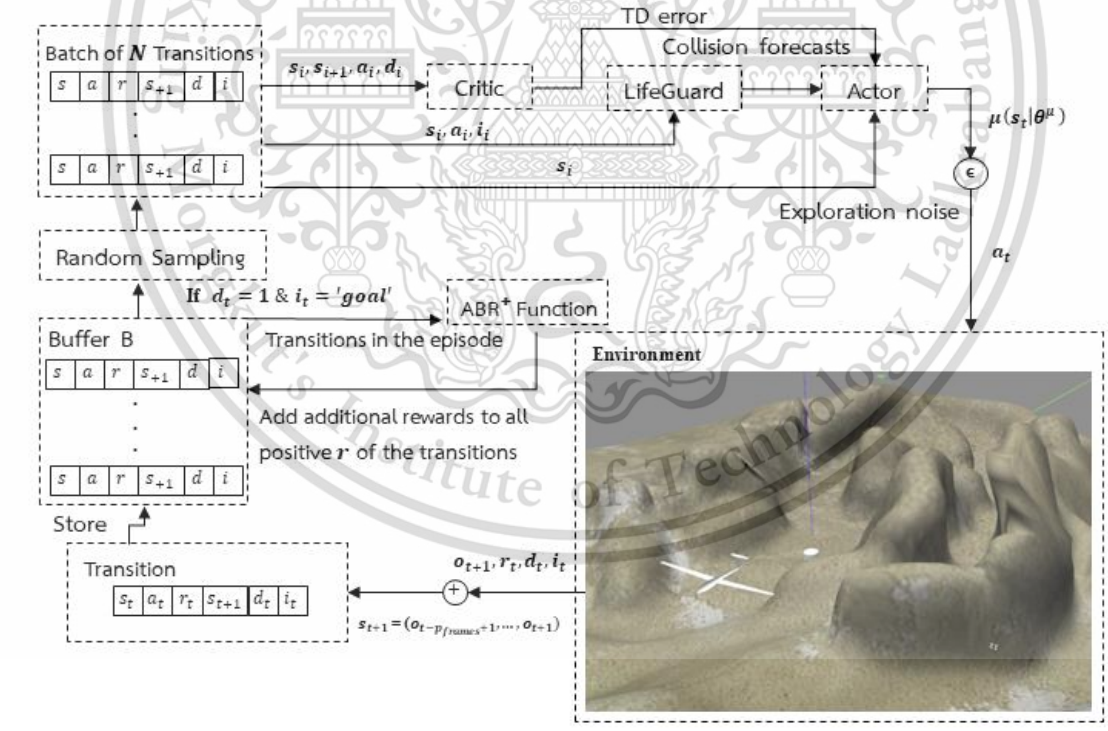


Figure 3.5 Workflow of the proposed method for autonomous UAV navigation task.

Algorithm 3.1: AUN with TD3-LG and ABR+

Initialize eval-networks of critic Q_1, Q_2 and actor μ with random parameters $\theta^{Q_1}, \theta^{Q_2}$ and θ^{μ}.

Initialize target-networks with same parameters as in eval-networks $\theta' = \theta$.

Initialize buffer B with size p_{buffer} and trajectories history H.

For episode e = 1 to N do

Receive initial observation o_1 from env then form state $s_t(0, \ldots, o_1)$.

For t = 1 to T do

Select an action with exploration noise

$a_t = clip(\mu(s_t|\theta^{\mu}) + \epsilon, a_{low}, a_{high})$.

Execute a_t and receive reward r_t, new observation o_{t+1},

and episode ending status d_t.

Generate s_{t+1} by insert o_{t+1} to s_t in FIFO fashion.

Store transition $(s_t, a_t, r_t, s_{t+1}, d_t, i_t)$ in B.

If d_t == 1 (done) & i_t == 'goal' then

Select All transitions $(s_i, a_i, r_i, s_{i+1}, d_i)$ in e from B.

For k = 1 to t / p_{frames} do

$Z = k + p_{frames}$.

$\left(e_1, .., e_{p_{features}}\right) = \tau_k$ = trajectory extracted from s_z with PSAE.

Classify τ with classifier trained offline to get its class l.

Calculate floored sum reward $R = \left\lfloor \sum_{j=l}^{l-p_{frames}} r_j \right\rfloor$.

Insert R to unique reward stack of H_l and sort stack

then return m, the index of R in stack.

$r_{(z-p_{frames})-z} \mathrel{+}= m \times p_{addReward}$

end for

end if

Sample mini-batch of N transitions $(s_i, a_i, r_i, s_{i+1}, d_i)$ from B randomly.

$a'_i = clip\left(\mu'\left(s_{i+1} \mid \theta^{\mu'}\right) + \varepsilon, a_{low}, a_{high}\right)$, $y_i = r_i + \gamma(1 - d_i) \min_{j=1,2} Q'_j\left(s_{i+1}, a'_i \mid \theta^{Q'_j}\right)$.

Update critic $\nabla_{\theta_j^Q}\left(\theta_j^Q\right)=\frac{1}{N}\sum_{i=1}^{N}\nabla_{\theta_j^Q}\left(y_i-Q_j\left(s_i,a_i\mid\theta^{Q_j}\right)\right)^2$.

if t mod $p_{updateDelay}$ == 0 then

Update actor θ^{μ} using deterministic policy gradient and collision forecast

$\nabla_{\theta^\mu}J(\theta^\mu)=\frac{1}{N}\sum_{i=1}^{N}\nabla_{\theta^\mu}Q_1(s_i,\mu(s_i|\theta^\mu)|\theta^{Q_1})-c\big(o_i+\mu(s_i|\theta^\mu)\big)p_{cp}.$

Update target networks

$\theta^{Q'}=p_{tau}\theta^{Q}+(1-p_{tau})\theta^{Q'}$, $\theta^{\mu'}=p_{tau}\theta^{\mu}+(1-p_{tau})\theta^{\mu'}$.

end if

if $t > p_{limitStep}$ or d_t == 1 (done) then

break loop.

end if

end for

end for

Chapter 4

Main results and discussion

4.1 Experimental setting

To make this experiment realistic, the proposed method is proceeded under the conditions which are as realistic as possible. The agent and environments are simulated by the well-known robot testing simulator, GAZEBO (Koenig and A. Howard 2004). The size, appearance, and weight of objects are almost the same as of the real world. The environments regard two external forces, gravitational and aerodynamic forces. The Robot Operating System (ROS) (Stanford Artificial Intelligence Laboratory et al. 2018) was in charge of UAV's perception and control. All the UAV experiments in this work were operated under the point cloud simplification with truncated icosahedron structure and sequential transition. At each episode, the start point of the agent and goal point (the white cylindrical in Figure 4.1) were randomly determined. In term of environment, besides complex geography, what it should be worried about in real world is moving objects that may obstruct the UAV's flight like birds and planes so there are two types of environment in this work, static and dynamic. The static environment has no any moving object except the agent on the other hand the dynamic environment has three planes flying at the same altitude of the UAV ($\boldsymbol{p}_{approAlt}$) with constant speed which is greater than a_{high}. The episode will be end if one of three conditions occurs: First, the agent arrives the goal, Second, the agent collides, Third, the agent runs out of step limit. The average success rate, which is the probability of agent arriving to goal point for last 500 episodes, was used to measure the performance of each method. And in this work, the recent and state-of-the-art methods for AUN were compared with the proposed method. In this experiment of AUN, 5,000 training episodes will take around 36 hours. The UAV's motion control was implemented with Python3 and all processes ran on a laptop computer with Core ™ i7 CPU @2.20 GHz, GeForce GTX 1050 Mobile Graphic chip, 32 GB memory, and Ubuntu 20.04 operating system. All Attributes and hyper-parameters for this work were shown in Table 4.1.

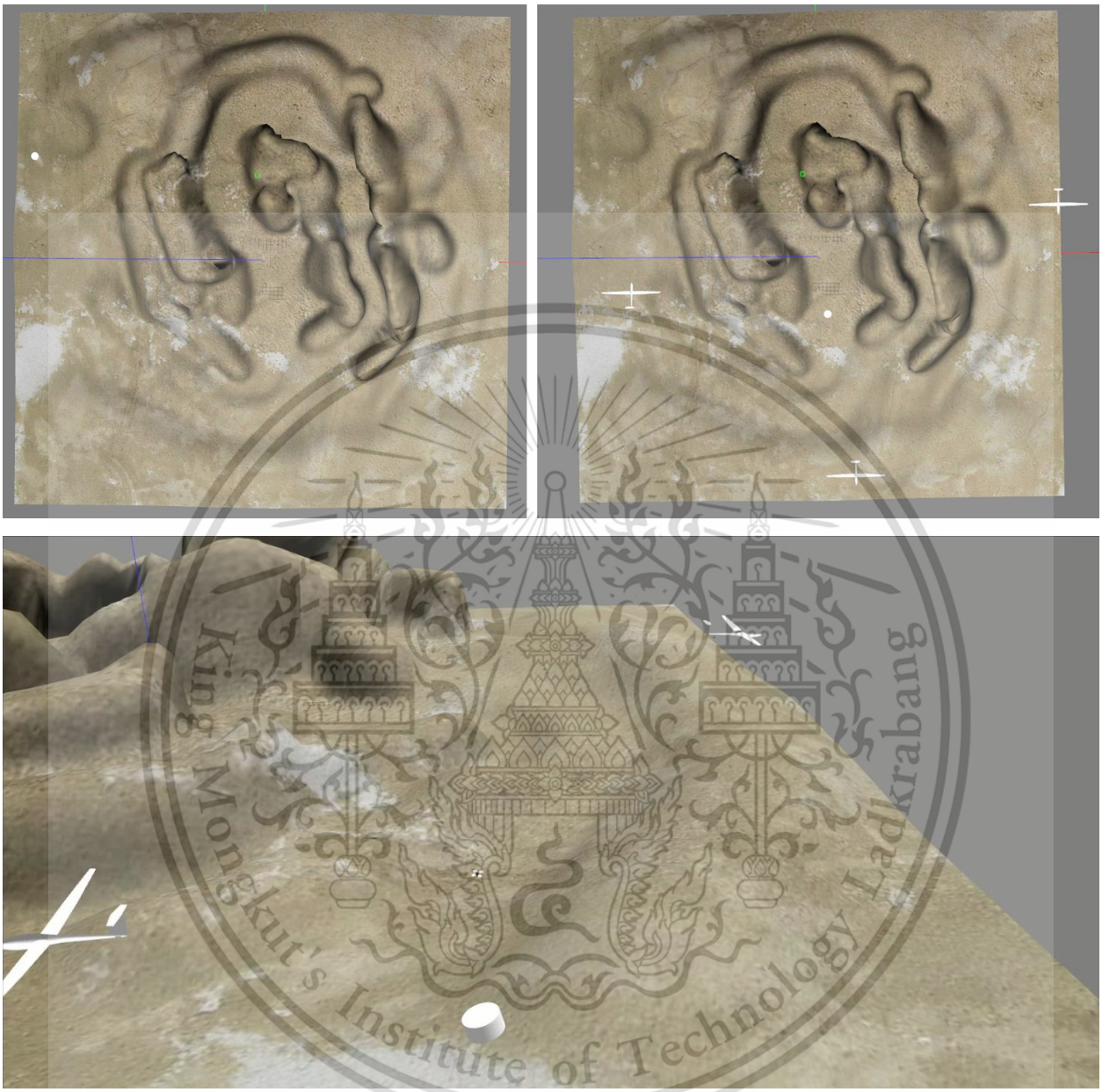


Figure 4.1 Left: static environment (no moving objects except the agent), Right: dynamic environment (three moving planes), Below: side view of the environment.

Table 4.1 Parameters and Variables used in this work**.**

No.	Name	Variable	Value
1	UAV	-	Quadrotor
2	LiDAR sensor	-	Velodyne vlp-16
3	Environment-valley	env	1200 m^2
4	Action space	-	4
5	Observation space	-	32
6	Action limit	a_{low}, a_{high}	-2, 2
7	Discount factor	γ	0.99
8	Soft update rate	p_{tau}	0.005
9	Noise bound	c	0.5
10	Observation sequence length	p_{frames}	3
11	Feature size (Latent space size)	$p_{freatures}$	20
12	Appropriate altitude or Flight level	$p_{approAlt}$	3.5 m
13	Radius of Truncated icosahedron structure, Safe flying distance	-	4 m
14	Learning rate	lr	0.001, 0.0001 for critic and actor
15	Level of tradeoff in PSAE	p_{wae}	0.7
16	Q-value diminishing value in LG	p_{cp}	10
17	Traditional reward function: goal reward, collision punishment	-	50, -50
18	Weight-parameters in reward function	w_{1-4}	8,1,0.5,0.5
19	Extra reward multiplier in ABR and ABR$^+$	p_{exR}	0.2

20	Collision punishment	$r_{collided}$	-10 (when using LG), -50
21	Delay to slow the update of actor and target networks	$p_{updateDelay}$	2
22	Size of buffer	p_{buffer}	10^6
23	Maximum number of steps per episode	$p_{limitStep}$	300
24	Batch size	N	100
25	Similarity threshold used in one-shot learning method.	-	98%

4.2 Trajectories classification

The 200,000 states randomly sampled from 5,000 training episodes were used to train the PSAE. After that, as mentioned in chapter 3.3, finding probable optimal number of clusters (k) with elbow method need iterative calculation of SSE so all the features of those states extracted by PSAE were clustered with k-mean method for k since 100-6,000. Finally, 4,000 was the number of clusters where SSE is starting to standing. To measure the performance, trajectories classification with PSAE was compared with the convolution based one-shot learning method which the algorithm accords to chapter 3.3 and uses cosine as similarity indicator. The Standard Deviation (SD) was used to indicate the spread of the data distributions.

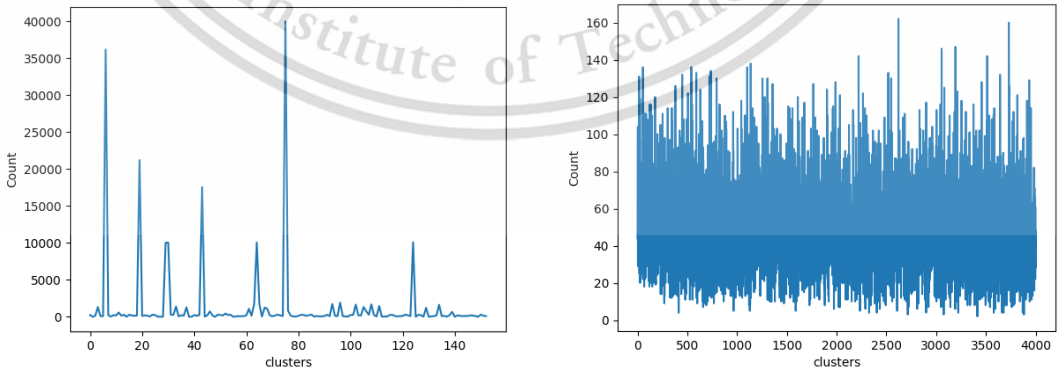


Figure 4.2 Trajectories data distributions, Left: of one-shot learning, Right: PSAE.

Apparently, the data distribution of the PSAE method was much more diffuse than of the one-shot learning and had SD at 24.89. In term of the one-shot learning method, the data largely clustered in merely four clusters and had SD at 5,017.52. But only data distribution may not be enough to tell which one is better for AUN. Hence, the PSAE and one-shot learning were used in ABR to again confirm its performance. The ABR with PSAE trajectories classification was named ABR+.

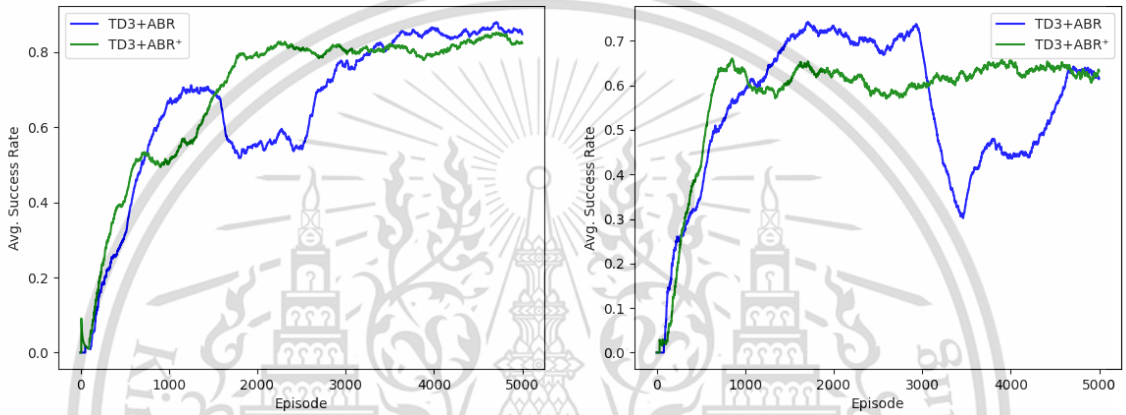


Figure 4.3 Success rates of using TD3 with ABR (one-shot learning) and ABR+ (PSAE) in AUN task, Left: in static environment, Right: in dynamic environment. (The figure is designed for coloured version)

The success rates of these two methods, PSAE (ABR+) and one-shot learning (ABR), were very close. For static environment, the success rates were 82.4%, 84.8% respectively and 61.6%, 63.2% for dynamic environment. But one obvious thing is that the PSAE helps the learning process significantly steadier.

4.3 Autonomous UAV navigation

4.3.1 Training in static environment

The gist this work wants to know consist of two things. The first is that TD3 is still better than DDPG in the combination with ABR+ to both methods. Despite the evidence (Fujimoto, van, and David 2018) that the TD3 outperforms the DDPG, it still needs to test the ABR+ on DDPG since there are some cases (Hu et al. 2020) which the DDPG is slightly better than the state-of-the-art methods. The second is to know whether LG is better than

FORK or not. With the fact that FORK relies on learning the reward function to admen the policy and ABR$^+$ gives reward varying with the rank of trajectory, this may suffer the forecasting process of FORK. As a result, we have to test FORK for AUN with both reward functions, traditional reward (TR) and ABR$^+$. Hence, the seven AUN experiments were conducted and the results can be seen in Figure 4.4.

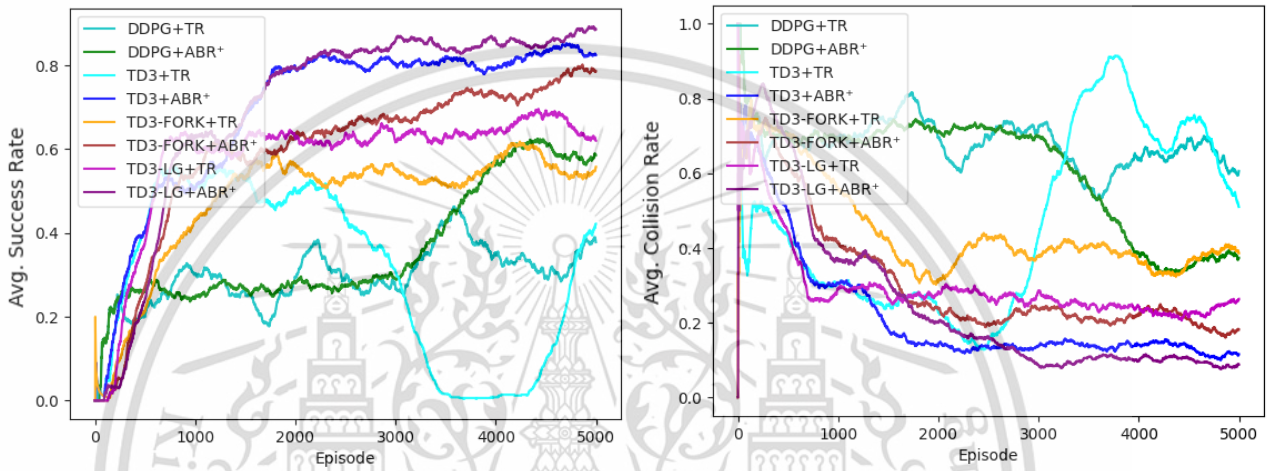


Figure 4.4 Results of using DDPG, TD3, TD-FORK, and TD-LG with TR and ABR$^+$ in static environment, Left: success rate, Right: collision rate (the out of step limit rate is a residual from success and collision rates). (The figure is designed for coloured version)

It is obvious that TD3 outperforms DDPG in all cases, TR and ABR$^+$. The success rates were close for DDPG and TD3 with TR which were 37.8% and 42.2% but, when the reward function was replaced with ABR$^+$, the difference was clearly stretched. The success rates raised up to 58.6% and 82.4% for DDPG+ABR$^+$ and TD3+ABR$^+$. For FORK and LG, even though our first assumption was that ABR$^+$ tends to harm the FORK and may lead to the worse result but the results showed that ABR$^+$ is still far better than TR. The success rates of FORK were 55.8% and 78.6% for TR and ABR$^+$. For the proposed method, TD3-LG gained the highest success rate which is 62% and 88.6% for TR and ABR$^+$. In term of the collision rates were 60.6%, 37.4%, 51%, 11.6%, 38.6%, 18.2%, 26.4%, and 8.8% for DDPG, TD3, TD3-FORK, TD-LG with TR and ABR$^+$ respectively.

4.3.2 Training in dynamic environment

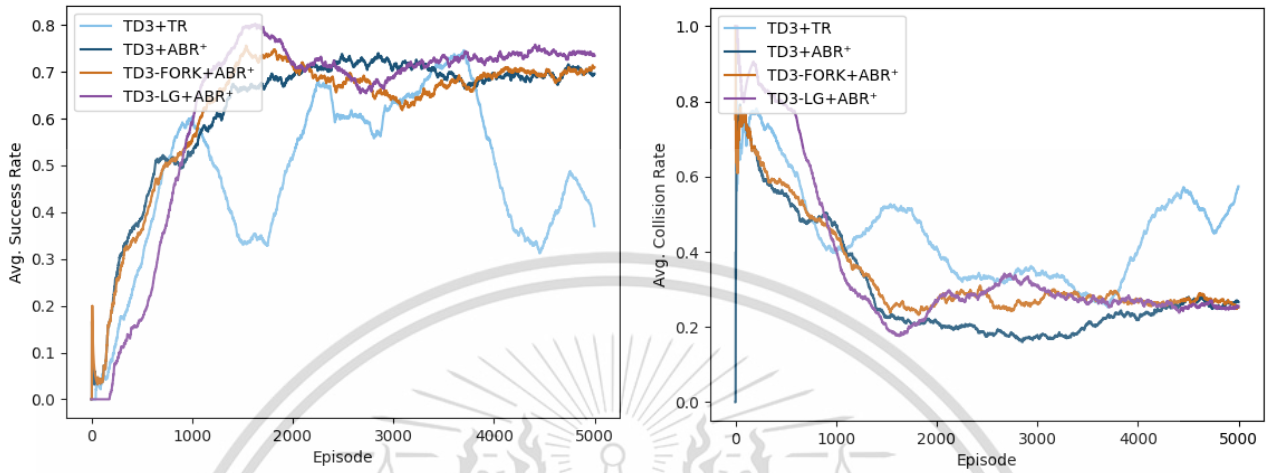


Figure 4.5 Results of using TD3, TD-FORK, and TD-LG with TR and ARB in dynamic environment, Left: success rate, Right: collision rate. (The figure is designed for coloured version)

The success rates of all methods dropped down in dynamic environment. TD3 with TR and ABR$^+$ gained 37% and 69.6% success rates. This again confirmed how superior ABR$^+$ is compared to TR. The success rates of TD3-FORK and TD3-LG with ABR$^+$ were 71% and 73.4% respectively. Although the proposed method still achieved the highest result but one thing should take into account is the stability of FORK method. The success rate of FORK had the lowest drop compared to all methods which is 9 % when the others are around 15-17%. In term of the collision rates were 57.4%, 26.6%, 25.2%, and 25.6% for TD3+TR and ABR$^+$, and TD3-FORK, TD-LG with ABR$^+$ respectively.

4.4 Complexity

The improvement in term of complexity was shown in this section. Our two methods (LG, ABR$^+$) mainly utilized the DNN so their computational complexity can be formulated as follows :

$$c_{LG} = O(N) \qquad (4.1)$$

Where c_{LG} denotes the computational complexity of LG.

As can be seen in Algorithm 3.1, the computational complexity of ABR$^+$ depends on the number of transitions in an episode. Hence, it can be done by this :

$$c_{\mathrm{ABR}^+} = O\left(T\left(\left(N_{PSAE}{M_{PSAE}}^2\right) + \left(N_C{M_C}^2\right)\right)\right) \quad (4.2)$$

Where c_{ABR^+} denotes the computational complexity of ABR$^+$ and N, M denote the number of hidden layers and maximum number of neurons for DNNs of features extraction (PSAE) and trajectories classification. T denotes number of transitions in an episode.

But, in RL, only computational complexity is insufficient to tell how fast the method is. So, we decided to use the straighter way, the sample complexity. It is the number of training steps (million) required to reach the same point. In this work, the best success rates of TD3-FORK+ABR$^+$ were used as the point which are 78% and 74% for static and dynamic environments respectively.

Table 4.2 Sample complexity (Million steps) of TD3-FORK+ABR$^+$ and TD3-LG+ABR$^+$.

Environment	TD3-FORK+ABR$^+$	TD3-LG+ABR$^+$
Static environment	0.95	0.41
Dynamic environment	0.39	0.32

In the static environment, TD3-FORK+ABR$^+$ required 0.95 million training steps to achieve its best success rate (78%) and TD3-LG+ABR$^+$ only need 0.41 million steps to reach the same success rate, reducing the training steps by more than 50%. But, in the dynamic environment, the numbers were not much different, the required training steps for TD3-LG+ABR$^+$ was 18% lower than of TD3-FORK+ABR$^+$.

4.5 Other navigation task

To confirm performance and robustness of the proposed method against the navigation tasks, BipedalWalkerHardcore was used to prove that. It is the well-known testbed for reinforcement learning which provided by the OpenAI gym (Brockman et al. 2016). This task consists of the two legs robot with sensor at the head to perceive a scene at its front and the objective which is to not fall before arriving the goal point. From (Wei and Lei 2020), they suggested the three heuristic changes to overcome this difficult task. Two of three in the changes are about original reward modifications therefore, to be compliance, the ABR^{+} wasn't applied in this task. The result of TD3-FORK was received by running their source code, which have been available on websites, on our computer. For TD3-LG, we used the same code as for AUN but with the three heuristic changes. The results of these two methods are shown in Figure 4.6 and are very close but the TD3-LG shows the sign of significantly faster convergence.

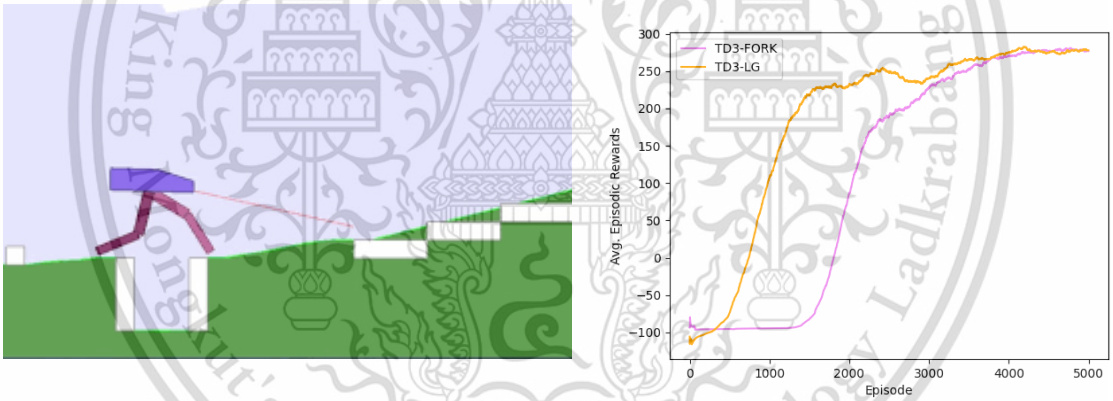


Figure 4.6 Left: BipedalWalkerHardcore task. Right: episodic rewards of using TD-FORK and TD-LG in BipedalWalkerHardcore task. (The figure is designed for coloured version)

Chapter 5
Conclusions and Suggestions

5.1 Conclusions

To get closer to AUN for real-world adoption, we had experiments on more realistic environments than other works and there were four novel methods proposed in this work. The first is the Point cloud simplification with truncated icosahedron structure which exploits the soccer ball shape to extract only necessary data from the massive point cloud. It turns three hundred thousand points into few points which is affordable even with a microprocessor. Second, a pilot study on unsupervised trajectories classification, our All Positive Siamese Auto Encoder (PSAE) had the show that its extracted features make a more proper data distribution in classification process and lead to the steady learning graph. Third, the main problems for AUN, the irrational reward dispensation of traditional reward function was addressed with our Augmentative Backward Reward function (ABR^{+}). It gives rewards proportional to the contribution of each transition by using the reward rank of transition itself in the history. The experimental results for AUN were clear that replacing TR with ABR^{+} tremendously boost the success rate for all learning methods and in both static and dynamic environments. The fourth is LifeGuard (LG), the new role in Actor-Critic model. Due to the fact that, for many task, high risk means high reward, it will take long time or never occur for the learning model to realize the maximum risk limit. Therefore the LG used collision forecast to stimulate the model to realize this much faster and the results also accord to it. The LG gained the lowest out of step limit rates in all experiment, 2.6% and 1% for static and dynamic environments respectively. This indicates that LG can alleviate apprehensive behavior of the agent and, as a result, also lead to higher success rates. In addition, LG also worked great in other navigation task and this was the evidence that it is robust. In sum, the proposed method achieved higher performance in all aspects, (success, collision, and out of step limit rates, convergence rate), than the state-of-the-art method, FORK.

5.2 Suggestions

There are still two drawbacks in the proposed method. First, PSAE requires pre-training with comprehensive patterns dataset. If not that case, the convergence rate will be slower

for an online parameters update. Second, like FORK, the hyper-parameter controlling the level of policy adjustment rather affect the performance.

The videos of experiments are available here: https: / / youtu. be/ OkNSfLbknIE, https://youtu.be/OkNSfLbknIE. The source code of the proposed method was uploaded to https://github.com/Manit-Chan/TD3-LG.

References

Apostolopoulos, Pavlos Athanasios, Marcos Torres, and Eirini Eleni Tsiropoulou. 2019. "Satisfaction-Aware Data Offloading in Surveillance Systems." Proceedings of the 14th Workshop on Challenged Networks - CHANTS'19. https://doi.org/10.1145/3349625.3355437.

Bellman, Richard. 1957. "A Markovian Decision Process." *Indiana University Mathematics Journal* 6 (4): 679–84. https://doi.org/10.1512/iumj.1957.6.56038.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. "OpenAI Gym." arXiv, https://doi.org/10.48550/ARXIV.1606.01540.

Elmokadem, Taha, and Andrey V. Savkin. 2021. "A Hybrid Approach for Autonomous Collision-Free UAV Navigation in 3D Partially Unknown Dynamic Environments." *Drones* 5 (3): 57. https://doi.org/10.3390/drones5030057.

Fujimoto, Scott, van Hoof, and David Meger. 2018. "Addressing Function Approximation Error in Actor-Critic Methods." ArXiv.org. 2018. https://arxiv.org/abs/1802.09477.

GUO, Tong, Nan JIANG, Biyue LI, Xi ZHU, Ya WANG, and Wenbo DU. 2020. "UAV Navigation in High Dynamic Environments: A Deep Reinforcement Learning Approach." *Chinese Journal of Aeronautics*, June. https://doi.org/10.1016/j.cja.2020.05.011.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." arXiv, https://doi.org/10.48550/ARXIV.1801.01290.

Hu, Zijian, Kaifang Wan, Xiaoguang Gao, Yiwei Zhai, and Qianglong Wang. 2020. "Deep Reinforcement Learning Approach with Multiple Experience Pools for UAV's Autonomous Motion Planning in Complex Unknown Environments." *Sensors* 20 (7): 1890. https://doi.org/10.3390/s20071890.

Jain, Trevor, Aaron Sibley, Henrik Stryhn, and Ives Hubloue. 2018. "Comparison of Unmanned Aerial Vehicle Technology-Assisted Triage versus Standard Practice in Triaging Casualties by Paramedic Students in a Mass-Casualty Incident Scenario." *Prehospital and Disaster Medicine* 33 (4): 375–80. https://doi.org/10.1017/s1049023x18000559.

Kapturowski, Steven, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. 2018. "Recurrent Experience Replay in Distributed Reinforcement Learning." Openreview.net. September 27, 2018. https://openreview.net/forum?id=r1lyTjAqYX.

Koenig, N., and A. Howard. n.d. "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator." *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Accessed October 20, 2019. https://doi.org/10.1109/iros.2004.1389727.

Konda, Vijay R., and John N. Tsitsiklis. 2003. "OnActor-Critic Algorithms." *SIAM Journal on Control and Optimization* 42 (4): 1143–66. https://doi.org/10.1137/s0363012901385691.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. "Continuous Control with Deep Reinforcement Learning." ArXiv.org. 2015. https://arxiv.org/abs/1509.02971.

Liu, Fan, and Yong Deng. 2021. "Determine the Number of Unknown Targets in Open World Based on Elbow Method." *IEEE Transactions on Fuzzy Systems* 29 (5): 986–95. https://doi.org/10.1109/tfuzz.2020.2966182.

Mahmood, A. Rupam, Hado P van Hasselt, and Richard S Sutton. 2014. "Weighted Importance Sampling for Off-Policy Learning with Linear Function Approximation." Neural Information Processing Systems. Curran Associates, Inc. 2014. https://proceedings.neurips.cc/paper/2014/hash/be53ee61104935234b174e62a07e53cf-Abstract.html.

MAHMUD, AL. 2021. "Development of Collision Resilient Drone for Flying in Cluttered Environment." *Graduate Theses, Dissertations, and Problem Reports*, January. https://doi.org/10.33915/etd.8022.

Mittal, Dishant, Dev Gaurav, and Sanjiban Sekhar Roy. 2015. "An Effective Hybridized Classifier for Breast Cancer Diagnosis." *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, July. https://doi.org/10.1109/aim.2015.7222674.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning." ArXiv.org. 2013. https://arxiv.org/abs/1312.5602.

Ng, Andrew. 2011. "Sparse autoencoder." CS294A Lecture notes.

Nitta, Kei, Keita Higuchi, Yuichi Tadokoro, and Jun Rekimoto. 2015. "Shepherd Pass." *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, November. https://doi.org/10.1145/2832932.2832950.

Rummery, Gavin Adrian, and M. Niranjan. 1994. "On-Line Q-Learning Using Connectionist Systems." Undefined. https://www.semanticscholar.org/paper/On-line-Q-learning-using-connectionist-systems-Rummery-Niranjan/7a09464f26e18a25a948baaa736270bfb84b5e12.

Sadeghzadeh-Nokhodberiz, Nargess, Aydin Can, Rustam Stolkin, and Allahyar Montazeri. 2021. "Dynamics-Based Modified Fast Simultaneous Localization and Mapping for Unmanned Aerial Vehicles with Joint Inertial Sensor Bias and Drift Estimation." *IEEE Access* 9: 120247–60. https://doi.org/10.1109/access.2021.3106864.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. 2015. "Prioritized Experience Replay." ArXiv.org. 2015. https://arxiv.org/abs/1511.05952.

Schneider, Stefan, Graham W. Taylor, Stefan Linquist, and Stefan C. Kremer. 2019. "Similarity Learning Networks for Animal Individual Re-Identification -- Beyond the Capabilities of a Human Observer." arXiv, https://doi.org/10.48550/ARXIV.1902.09324.

Shi, Congming, Bingtao Wei, Shoulin Wei, Wen Wang, Hai Liu, and Jialei Liu. 2021. "A Quantitative Discriminant Method of Elbow Point for the Optimal Number of Clusters in Clustering Algorithm." *EURASIP Journal on Wireless Communications and Networking* 2021 (1). https://doi.org/10.1186/s13638-021-01910-w.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. "Deterministic Policy Gradient Algorithms." Proceedings.mlr.press. PMLR. January 27, 2014. https://proceedings.mlr.press/v32/silver14.html.

Stanford Artificial Intelligence Laboratory et al. Robotic Operating System (version ROS Melodic Morenia), 2018. https://www.ros.org.

Sudhakar, S., V. Vijayakumar, C. Sathiya Kumar, V. Priya, Logesh Ravi, and V. Subramaniyaswamy. 2020. "Unmanned Aerial Vehicle (UAV) based Forest Fire Detection and monitoring for reducing false alarms in forest-fires." Computer Communications 149: 1–16. https://doi.org/10.1016/j.comcom.2019.10.007.

Sutton, Richard, en A. G. Barto. "Reinforcement learning". Journal of Cognitive Neuroscience 11 (01 1999): 126–34.

Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. "Extracting and Composing Robust Features with Denoising Autoencoders." *Proceedings of the 25th International Conference on Machine Learning - ICML '08*. https://doi.org/10.1145/1390156.1390294.

Wei, Honghao, and Lei Ying. 2020. "FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning." arXiv, https://doi.org/10.48550/ARXIV.2010.01652.

Youn, Wonkeun, Hayoon Ko, Hyungsik Choi, Inho Choi, Joong-Hwan Baek, and Hyun Myung. 2020. "Collision-Free Autonomous Navigation of a Small UAV Using Low-Cost Sensors in GPS-Denied Environments." *International Journal of Control, Automation and Systems* 19 (2): 953–68. https://doi.org/10.1007/s12555-019-0797-7.

Zhang, Qichen, Meiqiang Zhu, Liang Zou, Ming Li, and Yong Zhang. 2020. "Learning Reward Function with Matching Network for Mapless Navigation." *Sensors* 20 (13): 3664. https://doi.org/10.3390/s20133664.

Author biography

Name	Mr. Manit Chansuparp
Date of Birth	11 October 1992
Address	166/1 Soi Khu Bon 27 yaek 11, Tha Raeng, Bang Khen, Bangkok, 10220, Thailand
Education	2013 Bachelor of Science in Ramkhamhaeng University / computer science 2016 Master of Science in Burapha University / informatics

Academic Publication(s)

1. “Emotion recognition of affective speech based on hybrid classifiers”, fund of the National Research Council of Thailand (NRCT) 2014

2. “The Automated Skull Stripping of Brain Magnetic Resonance Images using the Integrated Method”, BMEiCON 2015

3. “Novel features for Classification of Hydrocephalus and Cerebral Atrophy”, JCSSE 2016

4. “Segmentation and Analysis of the Ventricle Deformation of Brain MR image”, fund of the National Research Council of Thailand (NRCT) 2017

5. “Moving object detection using Integrated Spatial and Motion-based method”, JCSSE 2019

6. “A Novel Augmentative Backward Reward Function with Deep Reinforcement Learning for Autonomous UAV Navigation”, Applied Artificial Intelligence 2022

7. “LifeGuard: An Improvement of Actor-Critic Model with Collision Predictor in Autonomous UAV Navigation”, Applied Artificial Intelligence 2022