ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester Thesis

Autonomous MAV Exploration with Reinforcement Learning

Autumn Term 2020

Supervised by:
Marius Fehr
Alexander Millane

Author:
Hokwang Choi

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Autonomous MAV Exploration with Reinforcement Learning

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Hokwang	Choi

Student supervisor(s)

Marius	Fehr
Alexander	Millane

Supervising lecturer

Roland	Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Zürich, 20.12.2019 — Place and date

J. Hokwang Choi — Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Abstract

The essential part of exploring an unknown area with a robot is the ability to build a map thoroughly and fast. Mostly, current solutions use the global map to plan a path and it incurs a lot of computation time. To alleviate this problem, we use the local map to focus on the local exploration which is much faster and cheaper than the global exploration. As soon as the local exploration is finished, the global planner can be called again to send the robot to a new position to start a local exploration again. In this paper, we suggest a policy network trained with the deep reinforcement learning algorithm to explore a 2D area. Using only the local map around the robot, the network decides the best action based on a trained network. This local exploration method offers an instantaneous decision on path planning and guarantees a consistent speed as the input for the network is a matrix of fixed size. We provide a detailed explanation of the robot's moves and possible exploration scenario in the simulation. The computation time test showed that the action decision can be made with the frequency of 50Hz for a given map of size 128x128.

Symbols

Symbols

s, r, a	state, reward and action
Q	state-action value function
L	loss function
y	target function
A	advantage function
V	state value function
$\mathcal{A}$	action space
γ	discount factor
θ, α, β	parameters of a network
ϵ	exploration fraction
δ	temporal difference error
X	matrix representation of a submap

Indices

j	step number
π	policy

Acronyms and Abbreviations

SLAM	Simultaneous Localization and Mapping
RL	Reinforcement Learning
DQN	Deep Q-Network
CNN	Convolutional Neural Network
USAR	Urban Search and Rescue
RRT	Rapidly-Exploring Random Tree
ESDF	Euclidean Signed Distance Field
TD	Temporal Difference

Chapter 1

Introduction

These days, researchers try to solve the challenges of exploring an unstructured and cluttered area with a robot. To navigate through an unknown area like "Fig. 1.1", the robot should be able to make a map around it and plan a path to discover a new area. This project introduces a new approach in this exploration problem. Incorporating the dense mapping technique and the reinforcement learning algorithm, we train a model which can decide the best action only looking at a local map. This approach will enable the robot to explore the unknown area fast and efficient. Even though we focused on the 2D exploration in a simulated environment, it will be a step to bridge the exploration problem with the reinforcement learning. This project could be conducted thanks to the dense mapping solution Voxblox [1] and the open source reinforcement learning environment by OpenAI Gym [2] and Stable Baselines [3].

Figure 1.1: The quadrotor is deployed to make a map of the tunnel. With a limited battery capacity, it needs to travel the unknown area fast and efficiently while building a map.

In highly cluttered and unstructured environments where human access is limited, the autonomous exploration with a robot is essential. By autonomous exploration, we mean that the robot navigates an unknown area itself and build a map. Since the operation time is always limited by the battery capacity, the autonomous ex-

ploration should be conducted efficiently and fast. In this circumstance, the most fundamental abilities of the robot is mapping and path planning. While SLAM techniques [4] enable the robot to build a map when it is travelling, we can use the map to devise a navigation strategy.
So far, wide range of research have developed methods for the autonomous exploration with a map which is built by the robot. For example, frontier-based exploration has been widely used for USAR search missions [5], [6], [7] and [8]. Frontier-based exploration was first introduced by B. Yamauchi, where the robot travels based on the boundary between the known and the unknown area which is called the frontier [9]. With this approach, a map is built in a global sense and the frontier is selected as the next goal position to go. Also, in the autonomous exploration, RRT method [10] is used to plan a path for exploration. Following the tree of possible trajectories, the robot navigates the area avoiding the obstacles [11], [12]. To improve the RRT method, [13] used the history of exploration to boost the RRT planning performance. However, in both methods, the global map should always be used for path-planning and the amount of computation grows as the discovered map becomes bigger. These methods can be powerful as a global planner but it faces the limitation in the sense of fast local exploration. To solve this issue, [14] suggested a local exploration strategy based on the frontier-based method. Focusing on the local map could speed up the navigation process overcoming the slow computation and making fast moves. Here, the idea is to call the global planner only when the fast local exploration has no more frontiers to travel. See "Fig. 1.2". However, the limitation still exists since the robot blindly follows the local frontiers and the actual navigation includes wasteful moves. For example, the robot in a room would check every corner until there is no more unknown voxels left, while the smart robot would notice the corner without looking into it and make a smooth turn to another corner. To decrease the wasteful moves and improve on smart moves, we introduce a reinforcement learning algorithm to learn the efficient moves directly from the local map which is provided by a dense mapping technique Voxblox [1]. Given an action command, Voxblox provides 3 different information: 1.The local submap of the size 128x128, 2.The number of newly discovered voxels, 3.The collision warning information when the robot tries to go too close to an object.

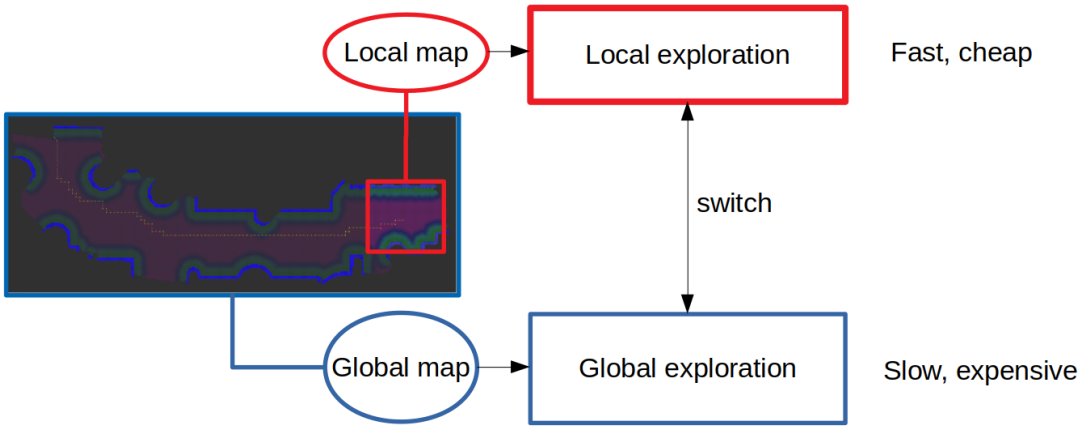

Figure 1.2: Comparison between the global and the local exploration. The global planner is very costly in computation and it incurs the slow movements of the robot.

Recently, the reinforcement learning has shown successful results in broad areas of games, control, and robotics. Especially, V. Minh et al.[15] has shown that RGB images can be directly used as inputs to the deep neural network which enables

the agent to understand the semantics of the system to achieve a human level performance in playing games. Since the mapping in SLAM is based on building images, we bridge the concept of playing Atari games with the exploration problem in unknown spaces and train a RL agent with maps provided by the mapping technique Voxblox [1]. In this paper, we focused on the 2D environment to verify the validity of the reinforcement learning algorithm in the exploration problem. The output map from Voxblox is ESDFs of the local map. ESDF represents the geometry of the field where each voxel value means the distance to the nearest surface of the structure. Voxel values are positive when they are outside the object and negative when they are inside the object. Specifically, the known and the unknown areas are distinguished by voxel values. However, to be used as a raw input for our neural network policy, the provided map includes noises and continuous values from the raw data limit the simplicity of the expression of the map. To solve this issue, we preprocess the given image with a CNN auto-encoder and extract the necessary information into binary values. Then, we use these simple state representations as inputs for the RL algorithm. RL methods have also been used to incorporate with the frontier-based method to select the best frontier position [16], [17]. However, without relying on the frontiers, we directly use the map as the input and train the RL agent to make smart action decisions. See "Fig. 1.3".

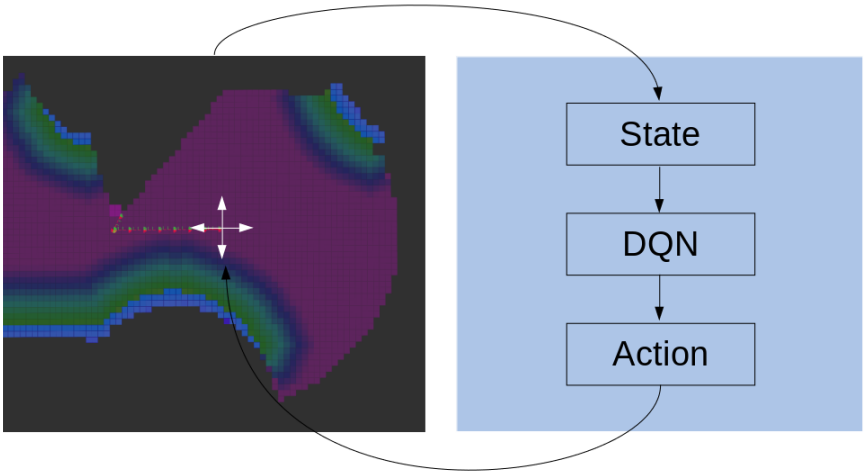

Figure 1.3: With a given local map as a state, the RL agent trained with DQN algorithm decides the best action based on the Q-network. The action space is $\{Up, Down, Right, Left\}$.

In the model-free reinforcement learning problem [18], the agent learns the optimal action in such a way that it maximizes cumulative rewards in one episode. The reward encourages the agent's learning process and it is an essential part of designing a reinforcement learning problem. The reward values have different possibilities from binary to continuous. To frame the exploration problem into a reinforcement learning environment, proper understanding of the goal in exploration is a primary step. For the exploration problem, however, its goal is an abstract concept which is 'building a whole map thoroughly and fast'. This includes detailed movements of avoiding obstacles, passing through a narrow path, continuous travelling to a new place and further more. These implicit goals make sense for a human but it is hard to make a control system which collaborates with all small goals. Even though the exploration is an integrated task, we can extract these essential features from

the map and focus on each goal with a proper set-up of the reinforcement learning problem. First, we should always prefer to go somewhere unknown and it is better when we can discover more in one action. Second, we should not collide with an obstacle but avoid it instead. Third, staying at already discovered area should be discouraged. Based on these three goals, we design a reward system and train an agent.

Chapter 2

Background

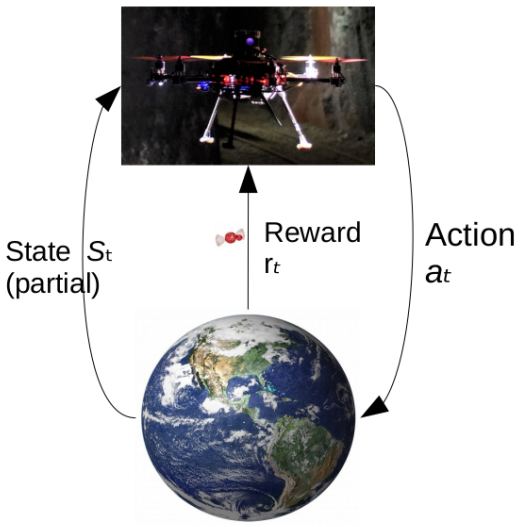

Figure 2.1: Overview of the reinforcement learning system.

In general, reinforcement learning system can be explained by Markov Decision Process (MDP). At a given state S_t, when the agent takes an action a_t, the environment gives back a reward r_t and the next state S_{t+1} in "Fig. 2.1". By definition, the transition of Markov state is only with respect to the current state and the next state. These consecutive processes continue until the episode is finished while updating the policy or the Q-network. Repeating the episodes, the agent is preferably looking for the bigger cumulative rewards and the policy is updated in different ways depending on the type of the algorithms. The presented approach utilizes Deep Q-network (DQN) developed by V. Minh et al. [15]. In this method, CNN structure is used to approximate the state-action value function, $Q(s, a)$. Q-function is equally informative compared to the policy network since we can already select the best action which incurs the highest expected value of the cumulative rewards. Then, based on the ϵ-greedy policy, the state is mapped into an action. This policy selects a random action with probability ϵ. The stochastic policy is appropriate in our situation as it encourages exploration which might send to a new state leading to a bigger reward in an episode. This stochasticity is essential in our case since the robot can easily get stuck anywhere. Also, in a given map, there is not just one way to explore the given space but multiple strategies for explorations are possible.

For the reinforcement learning problem, the system is defined by observation space, reward and action space. These three components are the most important designs to define a RL system. DQN algorithm assumes its input as images and CNN structure interprets the situation based on the image change. Also, it preprocesses the consecutive 4 images into an internal state representation. This encourages the network to get the meaning of motion and it can learn a policy from the consequences of movements. In our exploration environment, we use local ESDF map from Voxblox and convert it into two channel images. Both channels are represented with 128x128 size matrix with two values. The first channel gives the information of the frontiers and the second one gives the wall information. Here, we should notice that this state representation is very implicit information. Given a state, by taking a certain action, we cannot expect a specific transition to another state since we do not know the map. When the agent updates the policy based on the transition experiences, there are several sub-goals. In [19], the concept of the intrinsic motivaiton is well explained in the problem of exploration. Firstly, the robot has to catch the meaning of every action. It means that the robot has to learn the consequences of the action by trying. Secondly, the robot has to learn to avoid the walls and the objects to navigate the space. Lastly, the robot should always try to go forward to the unknown area. To achieve these sub-goals, a sophisticated rewarding system is necessary. Our reward system is based on the number of newly discovered voxels which quantify the size of the area the robot has discovered in a given step. To prevent a collision with a wall and to discourage the robot staying at the same position, negative rewards are given in two cases. For action space, we restrict it in four discrete actions(up, down, right and left) with a magnitude of 0.5m. This restrictive action space is inevitable since the state representation is implicit and we have to rely on the DQN internal state representation for the agent to capture the meaning of each motion. Details are explained in section 3.
The goal of the RL agent is to maximize the cumulative rewards at time t

$$R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau \tag{2.1}$$

toward the end of the episode at time T. Discount factor γ for the future reward is introduced for the simplicity of mathematical expression and to stress the importance of immediate rewards. It also avoids the infinite reward in a cyclic system. However, without the knowledge of consequence of an action at a certain state, we cannot get the cumulative future reward directly. According to [15], Q-network is approximated with CNN structure which maps a state and an action into an expected reward sum. For each action step, one move with a given reward is saved in a buffer and these experiences are randomly selected to update the Q-function based on a natural gradient descent method. Here, the loss function is defined by the parameter θ as a network representation,

$$L(\theta) = \left(y_j - Q(s_j, a_j; \theta)\right)^2, \text{where} \tag{2.2}$$

$$y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_a Q(s_{j+1}, a; \theta) & \text{otherwise} \end{cases} \tag{2.3}$$

The network parameter θ is changed whenever the gradient descent is performed for Q-network update. For a gradient descent update, it randomly selects experience transitions (s_j, a_j, r_j, s_{j+1}) from the experience buffer with a minibatch size. Here, s_j means the internal state representation which is preprocessed with 4 consecutive observation inputs. This internal state representation is a crucial part for the agent

to capture the meaning of a motion. In this way, Q-function is approximated and the ϵ-greedy policy selects the best action,

$$a = max_a Q(s, a; \theta) \tag{2.4}$$

with probability $1 - \epsilon$.
On top of this DQN algorithm [15], we utilize four extensions of this algorithm. First, Prioritized Experience Replay [20] is used. Instead of randomly choosing an experience from the buffer, the experience is prioritized based on a TD error,

$$\delta = y_j - Q\left(s_j, a_j; \theta\right) \tag{2.5}$$

High TD error means that this experience gives more information for learning. Also, prioritizing these experiences can speed up the learning process since the agent frequently learns more important moves.
Secondly, we use Double-Q learning [21]. In Q-learning process, each gradient descent updates the network parameter θ. However, changing θ in every update can result in overestimation of Q-function [21]. To prevent this issue, double Q-learning introduces two Q-functions. By holding the target function with the parameter θ_{target}, it performs gradient descent with the online network parameter θ_{online}. The target function and the online Q-network are as follows:

$$y_j = r_j + \gamma max_a Q(s_{j+1}, a; \theta_{target}) \tag{2.6}$$

$$Q(s_j, a_j; \theta_{online}) \tag{2.7}$$

After a given number of updates, θ_{target} is synchronized with θ_{online}. We synchronize the parameters every 500 updates.
Thirdly, Dueling Network [22] introduces a new network structure. The advantage function means the benefit you get by taking a certain action with a given policy π and state-value function $V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s, a)\right]$, which is expressed as:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{2.8}$$

Here, we use the fact that the expected value of the advantage function with the optimal policy is zero.

$$\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s, a)\right] = 0 \tag{2.9}$$

Using two more network parameters α and β, Q-function is approximated with a sum of the state-value and the advantage function.

$$\begin{gathered} Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \\ \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A\left(s, a'; \theta, \alpha\right)\right) \end{gathered} \tag{2.10}$$

Lastly, Noisy Networks for Exploration [23] is used to encourage further exploration. This method perturbs the network weights with a noise and the consequence is that it encourages the exploration to reach the global optimum.

Chapter 3

Methods

3.1 Environments

As our reinforcement learning environment, we used OpenAI Gym [2]. It supports various algorithms and the customized environment can be shared, so it is beneficial for the reproducibility of the reinforcement learning problem. For our dense mapping technique, we used Voxblox [1] which is operated in Robot Operating System(ROS) [24]. Two environments communicate when the robot takes a step. Overview of these environments is shown in "Fig. 3.1". Using ROS topic broadcasting system, the Gym environment gives an action command and Voxblox returns necessary information to feedback new observation and reward. This information is composed of local maps, number of newly discovered voxels and collision information. By 'collision', it is a detection system when a robot tries to go too close to an object. It is only a warning system which will be later used to set up the reward system.

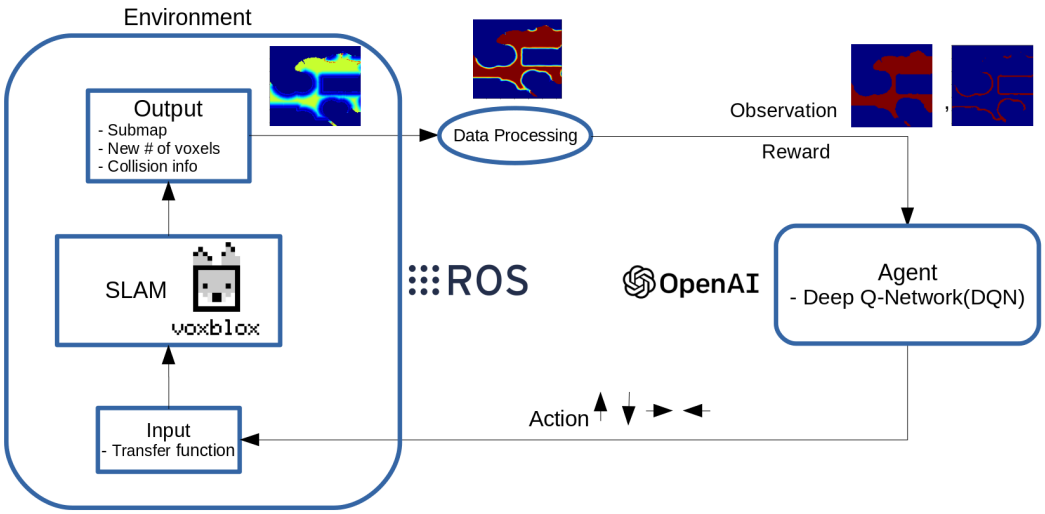

Figure 3.1: Overview of environments

3.2 Data preprocessing

A local map given by Voxblox package is ESDFs around the robot. To reduce the noise and make reliable binary data, we use a CNN auto-encoder and a color mapping scheme. Our auto-encoder is mainly used to smooth the map and keep the information in the map. Before we give inputs into auto-encoder, the raw map is first masked to differentiate the known and the unknown area. Then, we have two channels as our inputs. For the auto-encoder output, we only get one channel which is similar to the original map. After training with 5000 sample maps, a reliable auto-encoder is obtained. In "Fig. 3.2", we can see the map is smoothed and de-noised. After that, this data is further processed to generate an observation for our reinforcement learning agent. After normalizing the map values into $[0, 1]$, color mapping scheme is performed to extract near wall information,

$$X_{new} = 255 \cdot min(max(4(X_{old} - 0.25), 0), 1) \tag{3.1}$$

Here, all the values are calculated in element-wise and mapped to integer. The effect of it is shown on top of 'Data Processing' in "Fig. 3.1". Then, we extract only the wall information with a threshold value. This threshold is obtained with a heuristic way considering the thickness of the wall in the image and the amount of noise. Since we still have the mask map, we concatenate these two maps and use them as observation for RL agent. The final observation for the agent is shown as 'Observation' in "Fig. 3.1".

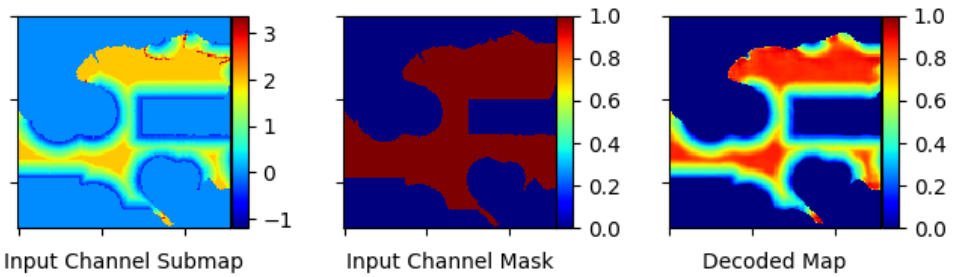

Figure 3.2: Auto-encoder processing

3.3 Action space

In 2D exploration problem, the robot can move on a plane. However, loading a local map with high frequency is computationally costly and it will slow down the exploring system . For our agent, we use discrete action space of size four. This space is defined as $\{Up, Down, Right, Left\}$ with the magnitude of 0.5m in a global frame. The magnitude of action depends on the size of voxels which we set as 0.2m. Appropriate action magnitude is imperative for the agent to understand the semantics of movements. If the action magnitude is too large, it will lose the movements since the observation spaces will not have any translational relationship. The small discrete action space is chosen to reduce the number of possible trajectories and it is also beneficial to find more efficient trajectories. Alternatively, we experimented with action space of size eight including the diagonal moves. However, it did not

learn the efficient moves and it made zagged moves which should be avoided in a real robot. Also, continuous action space is considered with Soft Actor-Critic method [25] - [26] and Policy-gradient method [27], but the agent could not make any progress in learning. Presumably, the observation space we have as the state of the agent is too implicit to learn the meaning of motions with continuous action values. Also, the best action with a given observation cannot be converged since the global map is unknown and the robot does not know what it can expect from taking different magnitude of actions.

3.4 Reward system

Compared to most Atari 2600 games, our agent's goal is not just surviving and collecting points. Rather, our agent should prefer discovering the area as much as possible and finish the episode when it is stuck at the dead end. Then, we can call the global planner to send the robot to a next position to start again. Additionally, in our case, discovering a bigger area should be rewarded more than discovering a small area. Since we get the number of newly discovered voxels from Voxblox, this becomes our criterion to give a reward to the agent. Considering the depth sensor range and angle, we scale the reward in a magnitude of 100. When a collision happens, even if the robot discovered new voxels, we give a negative reward of -20 because it is not desired to go forward to an object. Also, when the robot does not discover any new voxels, it is penalized with negative rewards growing proportionally. This reward system is designed to encourage the agent to always prefer a new exploration rather than easily getting stuck with a local minimum.

3.5 Episode scenario

How to finish an episode is a crucial component in our exploration system. Since there is no point in getting stuck and obtaining no information, we finish the episode when it does not discover any voxels for thirty consecutive steps. Also, if the agent is consecutively running into a collision status more than five steps, we finish the episode. The numbers are chosen to give the agent a chance to learn how to get out of the situation which might lead to a bigger reward. Since we have to compare the reward of each episode, we also limit the maximum steps as 500 in one episode. However, for validation purpose after training, we can change the parameters to define an episode. For example, in a real situation, we might change the maximum collision number as one.

3.6 Hyper-parameters

Parameters for our DQN algorithm are described in "Table. 3.1". As extensions of DQN algorithm, we use 1.Prioritized Experience Replay [20], 2.Double Q-learning [21], 3.Dueling Network [22], 4.Noisy Networks for Exploration [23].

3.7 Evaluation criterion

For our exploration problem, evaluation of the performance is hard to be defined. To solve the generalized exploration problem, we randomly generate objects in a map and place the robot in a random position. Then, we evaluate the cumulative reward for an episode. Howevery, this is very noisy data since it highly depends on the structure of the map and the position where the agent is spawned. Thus, an

Table 3.1: Parameters

Variable	**value**
Discount factor γ	0.99
Learning rate	0.0005
Buffer size	1 million
Exploration fraction ϵ	0.2
Training frequency	10
Batch size	32
Target network update frequency	500

absolute evaluation is limited with the cumulative reward and the human evaluation is also necessary. To evaluate the agent, we first check the smoothed cumulative reward for every fixed map. When the reward grows over a same map, it means that the agent is learning meaningful moves in the map. To check the agent's quality, we run the simulation to see if the agent makes sensible moves in every situation.

Chapter 4

Experiments

4.1 Training

Once the reinforcement learning model is trained, we only use the trained policy network to navigate in a place. Therefore, robustness is crucial in training process. To generalize our solution for exploration in any place, we change the map every 10000 steps and spawn a robot in a random place every episode. The original 3D map is generated with cylinders, spheres and cubes. Then we fix the level of height and it becomes our 2D map. The training progress is checked with rewards, loss and TD error. Here, the important measure is rather the tendency than the magnitude of rewards. Training progress can be viewed at `https://youtu.be/ZiqzRN5-lp8`.

4.2 Random map generation effect

To see the effect of different maps, we compared the rewards, loss and TD error over 100000 steps. From "Fig. 4.1", we can see the magnitude of rewards per episode can vary depending on the map structure. However, in all cases, loss function decreases over time steps and we can verify the learning process is conducted correctly. TD error in "Fig. 4.1" shows that the effect of random spawning in a same map at each episode. If it started at the same place in every episode, TD error would decrease because it would explore and find a good way to travel. Since our aim is focused on general exploration, we start at a different position at each episode and the effect of random start shows that TD error can increase because the robot has more freedom to make a different trajectory which cannot be predicted.

4.3 Methods comparison

Additionally, we trained agents using different deep reinforcement learning algorithms. DQN is compared with Asynchronous Actor Critic(A2C) [25], Proximal Policy Optimization(PPO2) [27] and Actor Critic with Experience Replay(ACER) [26] methods. Over 50000 action steps, reward is compared in "Fig. 4.2". The map is changed every 10000 steps and the qualitative evaluation is performed after the training. A qualitative evaluation is performed by looking at the agent's moves to check the quality of acions. Even though the reward can be gained with the one-directional movement, we could verify if the agent is actually learning with a qualitative evaluation. As we could expect from "Fig. 4.2", only DQN algorithm succeeded to learn. Possible explanation for this result can be from key aspects of DQN algorithm. First, DQN algorithm is possible to catch the meaning of motion

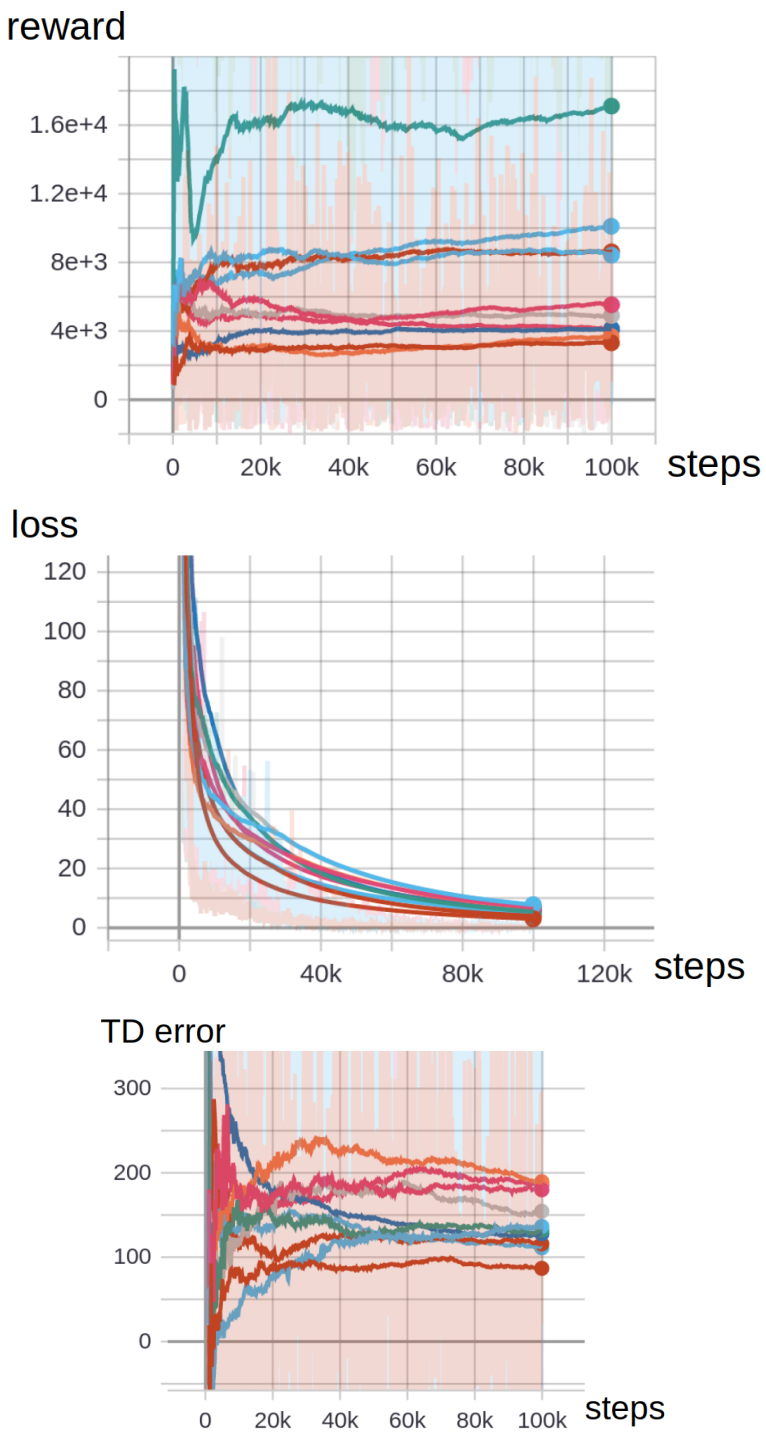

Figure 4.1: Rewards per episode (top), Loss function (middle) and TD error (bottom) for randomly generated maps

with the preprocessed internal state representation. This is crucial in our case since the robot makes actual movements and it results in different states which are local maps. Second, the training process of DQN algorithm is based on the experience replay. The previous transition experiences are kept in a buffer and Q-function is updated based on the selected experiences from the buffer. Third, DQN algorithm is an off-policy method. In Q-learning process, the target function is considered with the entire action space rather than the action taken based on the policy. This will respect the possible consequences of taking action at a certain state, and the exploration can improve over episodes.

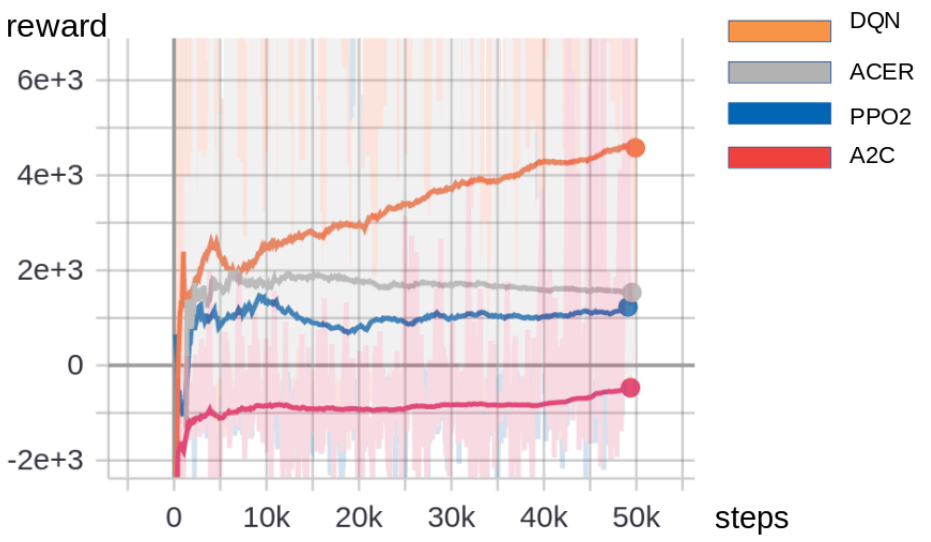

Figure 4.2: Methods comparison

Chapter 5

Result and Discussion

5.1 Integrated simple global planner

The model is trained for 48 hours with 2.6GHz CPU for Voxblox simulation and GeForce GTX 1650 (GPU) to update neural networks. The robot learned with 2.15 million action steps. To see the intended behavior of the whole 2D exploration, we introduce a simple global navigation planner which sends a robot back to a position where it discovered the largest number of voxels. The global planner is called when a robot is stuck at a dead end or when a collision warning is on. From the point where the robot is sent, local exploration is conducted again to continue discovering the new area. The video is available at `https://youtu.be/gdb0svZJyTc`. "Fig. 5.1" shows the whole exploration calling the global planner twice.

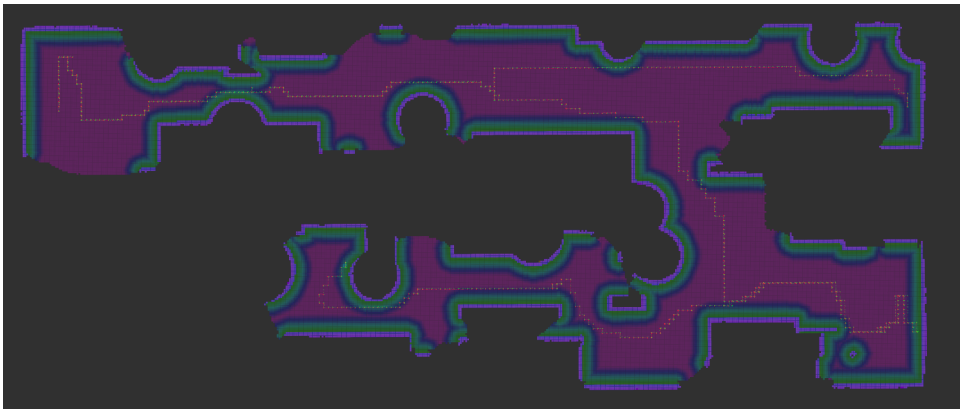

Figure 5.1: Full exploration with a global planner. The robot initially started at the top left position of the map and it called the global planner at each dead end at the top right and the bottom right. Map size is 100m x 40m.

5.2 Analysis of detailed moves

Local exploration is always preferred since it can speed up the exploration process. Our model could achieve impressive moves which overcomes the problem of following the frontiers blindly. First, it managed to travel along a narrow path which connects to another space. "Fig. 5.2" shows the robot passing through a narrow path. Second, the agent learns how to ignore a trivially unknown area. For human,

small unknown regions can be deduced by previous experiences when they have certain patterns. For our agent, this region means a small amount of rewards if it travels and discovers. Surprisingly, the agent learns how to think in long-term for continuing an efficient travel. “Fig. 5.3” demonstrates the ability to travel efficiently ignoring small immediate rewards. It is indeed beneficial when we want to have a big picture of the map rather than small details. Those small regions can still be discovered later when the global planner is called. The benefit of this behaviour is mostly about the shortening of travel time which can save the battery of the robot in a real situation. Additionally, the agent also learned how to see the reward far in the future. When the local map does not show any unknown area nearby, we can conclude that there is no information on the map. However, the agent could expect a future reward in a specific direction and it travels back to where it came from and continue travelling. This example is described in “Fig. 5.4”. Successful learning of these detailed moves can be tremendously beneficial in a global sense of exploration. The video can be viewed at `https://youtu.be/V291v-nXfCU`.

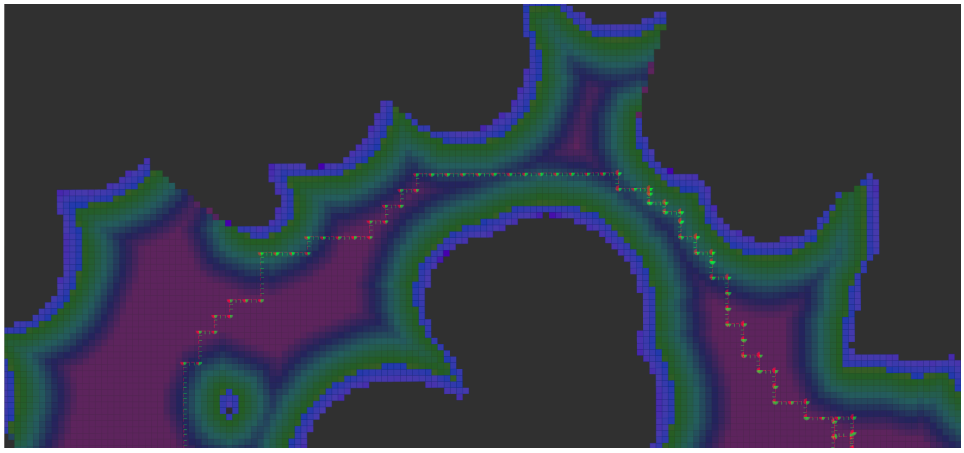

Figure 5.2: The robot passes through a narrow path.

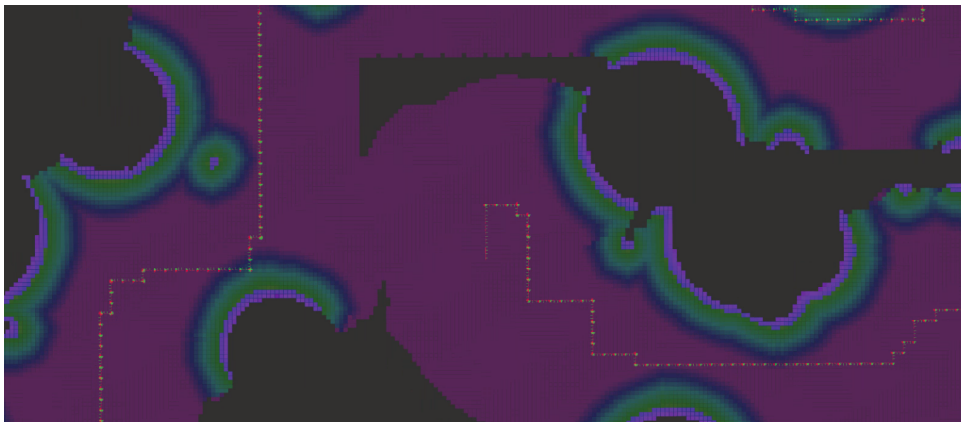

Figure 5.3: The robot ignores trivial regions and goes down looking for a bigger reward.

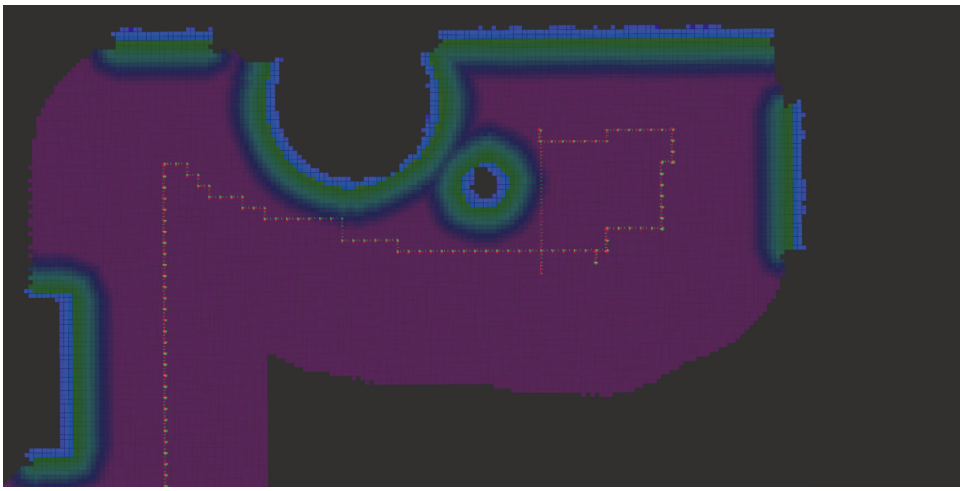

Figure 5.4: The robot sees the future reward out of local map and traverses already discovered region.

5.3 Robustness

Since we use a trained agent in real explorations, robustness is important to explore a complete new world where the agent has no experiece with the area. To guarantee the robustness, it is important that the agent actually learns the semantic meaning of the given map such as walls, objects and possible paths. If the agent is trained only based on a map shape, it would not be able to travel in a complete new map. Through our validation process by exploring randomly generated maps, we could verify that the agent actually learns the information of the map and not the map itself. Several episodes for validation are available at `https://youtu.be/XP9fBk-kJzU`. We can see that the agent is reacting to the wall and objects trying to discover new areas.

5.4 Supplementary study

Especially, fast local exploration is very important since it can save the battery of the robot and extend the range of the exploration. With our reinforcement learning agent, the average action decision time is calculated as 0.02 seconds which is 50Hz in frequency. This is the time taken from a given local map to make an action command. This fast process can enable the robot to make consistent moves without lags. Since the action command can be updated with 50Hz, we can lower the magnitude of one action move and the robot will be able to make more sophisticated moves in a crawlspace if needed.

Chapter 6

Conclusion

This paper introduces a new local exploration strategy based only on a local map around a robot. Using the dense mapping technique and the reinforcement learning algorithm, we trained a model which can instantaneously make an action among four directions on 2D environment. Surprisingly, the agent could also learn ingenious moves as shown in 5.2 and explore the area autonomously. Especially, this method is beneficial since the computation speed is constant as the robot explores more space. It is based on the fact that we only input a local map to the agent and CNN policy is already trained to decide the best action. Combined with a proper global planner, the robot will be able to explore an unstructured and unknown area fast and thoroughly. Since the robot is trained on randomly generated maps, the agent has learned the general exploration strategy and we can expect it to travel in any area.

Currently, this strategy is limited on 2D exploration. In the future work, we plan to extend the space to 3D world and build an agent which can be deployed on aerial vehicles. There are two possible approaches. First, using 2D exploration with a fixed altitude, we can still call a 3D global navigator to change the level when the robot is stuck on the 2D map. Secondly, we can also try to do autonomous 3D local exploration based on the trained agent. In this case, the local map would have one more dimension and we have to add at least two more actions to move in z-axis. From a short experiment, the simulation in 3D space takes much more time than 2D exploration. Therefore, speeding up the simulation will be another challenge to train an agent.

Bibliography

[1] M. F. J. N. R. S. Helen Oleynikova, Zachary Taylor, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning," *arXiv:1611.03631*, 2017.

[2] L. P. J. S. J. S. J. T. W. Z. G. Brockman, V. Cheung, "OpenAI Gym," *arXiv:1606.01540*, 2016.

[3] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[4] J. L. S. Thrun, *Simultaneous Localization and Mapping*. Springer, Berlin, Heidelberg, 2008.

[5] B. S. F. Niroui and G. Nejat, "Robot exploration in unknown cluttered environments when dealing with uncertainty," *IEEE*, pp. 224–229, 2017.

[6] W. B. S. Oßwald, M. Bennewitz and C. Stachniss, "Speeding-Up Robot Exploration by Exploiting Background Information," *IEEE*, vol. 1, no. 02, pp. 716–723, 2016.

[7] Y. L. B. Doroodgar and G. Nejat, "A Learning-Based Semi-Autonomous Controller for Robotic Exploration of Unknown Disaster Scenes While Searching for Victims," *IEEE*, vol. 44, no. 12, pp. 2719–2732, 2014.

[8] C. S. G. L. Y. Mei, Y. Lu and Y. C. Hu, "Energy-efficient mobile robot exploration," in *Proceedings 2006 IEEE International Conference on Robotics and Automation*, 2006, pp. 505–511.

[9] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97*, 1997, pp. 146–151.

[10] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Iowa State University*, 1998.

[11] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," *IEEE*, pp. 1396–1402, 2017.

[12] U. S. S. O. M. B. A. Bircher, K. Alexis and R. Siegwart, "An incremental sampling-based approach to inspection planning: the rapidly exploring random tree of trees," *Robotica*, vol. 35, p. 1327, 2017.

[13] R. B. H. O. C. Witting, M. Fehr and R. Siegwart, "History-Aware Autonomous Exploration in Confined Environments Using MAVs," *IEEE*, pp. 1–9, 2018.

[14] E. K. T. Cieslewski and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," *IEEE*, pp. 2135–2142, 2017.

[15] D. S. A. G. I. A. D. W. M. R. V. Mnih, K. Kavukcuoglu, "Playing Atari with Deep Reinforcement Learning," *NIPS Deep Learning Workshop 2013*, 2013.

[16] Z. K. F. Niroui, K. Zhang and G. Nejat, "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments," *IEEE*, vol. 4, no. 02, pp. 610–617, 2019.

[17] L. Tai and M. Liu, "A robot exploration strategy based on Q-learning network," *IEEE*, pp. 57–62, 2016.

[18] A. Barto and R. S. Sutton, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1992.

[19] S. H. A. Aubret, L. Matignon, "A survey on intrinsic motivation in reinforcement learning," *arXiv:1908.06976*, 2019.

[20] I. A. D. S. T. Schaul, J. Quan, "Prioritized Experience Replay," *arXiv:1511.05952*, 2016.

[21] D. S. H. van Hasselt, A. Guez, "Deep Reinforcement Learning with Double Q-learning," *arXiv:1509.06461*, 2016.

[22] M. H. H. v. H. M. L. N. d. F. Z. Wang, T. Schaul, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv:1511.06581*, 2016.

[23] B. P. J. M. I. O. A. G. V. M. R. M. D. H. O. P. C. B. S. L. M. Fortunato, M. G. Azar, "Noisy Networks for Exploration," *arXiv:1706.10295*, 2017.

[24] B. G. J. F. T. F. J. L. R. W. A. N. M. Quigley, K. Conley, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[25] M. M. A. G. T. P. L. T. H. D. S. K. K. V. Mnih, A. P. Badia, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783*, 2016.

[26] N. H. V. M. R. M. K. K. N. d. F. Z. Wang, V. Bapst, "Sample Efficient Actor-Critic with Experience Replay," *arXiv:1611.01224*, 2017.

[27] P. D. A. R. O. K. J. Schulman, F. Wolski, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347*, 2017.