Spatial Action Maps Augmented with Visit Frequency Maps for Exploration Tasks

A THESIS
SUBMITTED TO THE FACULTY OF THE
UNIVERSITY OF MINNESOTA
BY

Zixing Wang

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Nikolaos Papanikolopoulos, Adviser

Dec 2021

Acknowledgements

The author would like to thank Doctor Nikolaos Papanikolopoulos for his help. This material is based upon work partially supported by the Minnesota Robotics Institute and NSF through grants #CNS-1439728, #CNS-1531330, #CNS-1544887, and #CNS-1939033. USDA/NIFA has also supported this work through grant 2020-67021-30755.

Abstract

Reinforcement learning has been widely applied in exploration, navigation, manipulation, and other fields. Most of the relevant techniques generate kinematic commands (e.g., move, stop, turn) for agents based on the current state information. However, recent dense action representations based research, such as spatial action maps, pointing way-points to the agent in the same domain as its observation of the state shows great promise in mobile manipulation tasks. Inspired by that, we make the first step towards using a spatial action maps based method to effectively explore novel environmental spaces. To reduce the chance of redundant exploration, the visit frequency map (VFM) and its corresponding reward function are introduced to direct the agent to actively search previously unexplored areas. In the experimental section, our work was compared to the same method without VFM and the method based on traditional steering commands with the same input data in various environments. The results show conclusively that our method is more efficient than other methods.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Exploration serves as one of the fundamental tasks in the field of robotics, and it also widely covers different applications including autonomous vehicles, space exploration robots, etc. Gul et al. (2019). There are many categories of exploration and navigation algorithms which have been proposed to achieve the goal of going to a location. For example, there are deterministic methods including neural networks, fuzzy logic, etc., and non-deterministic method including genetic algorithms, swarm optimization, etc. Gul et al. (2019). In recent years, reinforcement learning based visual navigation methods Gao et al. (2017); Chen et al. (2018); Mnih et al. (2015); Gupta et al. (2017); Pfeiffer et al. (2017); Ross et al. (2013); Silver et al. (2018); Levine et al. (2018) have achieved remarkable success. They generally use a neural network with visual state observation as input to generate commands for agents to achieve different objectives.

One of the common points of the aforementioned methods is that they use steering commands to direct agents to reach their goals. Common steering commands as shown in Figure 3.2 control agents by giving moving direction and distance (e.g., going straight for 0.16 meters). Comparing with all the possible navigation points, the action space of steering commands is relatively limited. If we consider the steering command setting's step size to be fixed, the action space is the number of candidate directions. For example, a 18-direction fixed step size steering commands system's

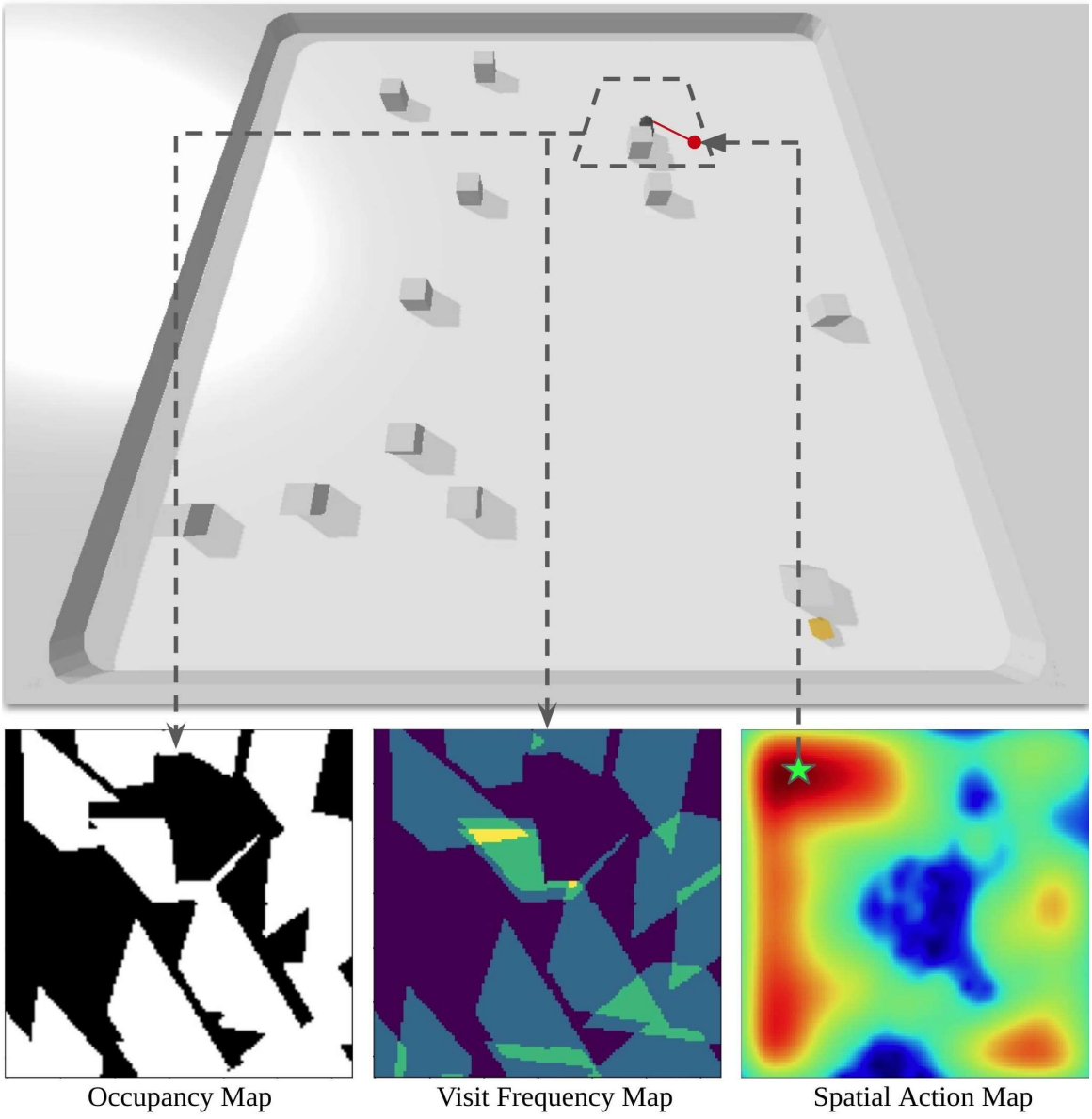

Figure 1.1: Occupancy map on the left encodes the explored area (white), unexplored area (black), obstacles (black), and walls (black). The VFM in the middle encodes the number of times each pixel is visited by the agent. Each color represents a specific frequency number. We capture the egocentric local maps including VFMs and other maps to a double DQN to generate commands on a spatial action map (right), where each pixel holds the Q-value of the corresponding command. The agent moves to the position with the highest Q-value. In this image, the pixel marked by a green star, whose corresponding position in the real environment is marked by a red dot, is the destination of the agent. (Note that we reduce the scanning frequency for better visualization and showing the shape sensing range. The Figure 3.1 shows more details about the approach.)

action space is 18 in each movement. Except for limited action space, it is hard for a single steering command to generate a complex path, such as a zigzag path to avoid obstacles, because the fixed dimension of the output includes limited information.

Wu et al. Wu et al. (2020) proposed a new action command representation for mobile manipulation: spatial action maps (Figure 1.1) to solve the aforementioned limitation of steering commands. An agent associated with it was compared with the traditional method of mobile manipulation: agents need to push as many cubes as possible to the receptacle area in the testing environments where we have obstacle cubes and dividers. Their results show that the spatial action commands have much better performance than the steering commands.

The spatial action maps are developed so that action commands can be represented as pixels on a map, whose action space is much larger than the low dimensional steering commands. Since the command only serves as a navigation endpoint, the agent can arrive to the corresponding position through linear or non-linear trajectories generated by the shortest path planner in Wu et al. (2020). It is worth noting that the spatial action maps lie in the same domain as state observations, and each pixel in the map encodes the corresponding Q-value if the agent chooses it as the end navigation point, so it is easier for the neural network to learn and predict the Q-value of each point based on the features of the state representation.

In this work, we employ modified spatial action maps for the exploration task to evaluate their performance. Exploring a new environment is an important task for robots. Taking a foraging robot as an example, before it finds target objects and transports them to the designated position, it will keep exploring the previously unseen area and build maps for later use Winfield (2009). This kind of task requires that agents explore the environment as efficiently as possible, which means agents should explore as much new area as possible and reduce the frequency of revisiting already explored areas. We mimic this setting in simulated environments as the

Figure 1.1 shows.

There are three main contributions of this work. First, we introduce the spatial action maps to the exploration tasks. Second, VFMs (Figure 1.1) are added to the state representation to urge agents to explore new areas. Finally, a sigmoid penalty function is designed for VFMs.

Chapter 2

Related Works

Steering commands based learning methods are currently one of the main stream choices to train an agent. AI2-THOR Kolve et al. (2017), MINOS Savva et al. (2017), House3DWu et al. (2018), Gibson Virtual Environment Xia et al. (2018), and Matterport-3D Anderson et al. (2018) are popular simulated indoor environments to train vision based reinforcement learning navigation methods. Lowe et al. Lowe et al. (2017) proposed a method to learn policies for multi-agent coordination. Chen et al. Chen et al. (2019) study a method that enables agents to navigate 3D environments without the rewards structure associated with tasks. Gupta et al. Gupta et al. (2017) proposed cognitive mapping and planning mechanisms that convert a first-person view to a top-down view map and plan a path towards the destination on that map. Chen et al. Chen et al. (2018) present an approach to enable wheel-legged robots to navigate in complex and dynamic environments by mapping the observations of the environment height to the action space. Gao et al. Gao et al. (2017) introduce a two-level approach including intention-net map commands to the image and a path planner to generate trajectories in 2D domain. Bellemare et al. Bellemare et al. (2016) propose to encourage agents to explore "novel" states. Chen et al. Chen et al. (2020) use an end-to-end method to enable agents to navigate in dynamic environments where obstacles are not stationary. These and other related work apply

different architectures to the exploration tasks and allow different types of robots to navigate the diverse environments. However, the control loop and types of action commands are similar: the agent acquires observations of the most recent state from on-board sensors, then feeds the observation to neural networks, then execute the generated commands, which are mostly low-dimensional, and repeats the loop by start observing new states.

Zeng et al. Zeng et al. (2018) use dense action representations in pick-and-place tasks as their system infers probability maps (top-down view) so that each pixel represents the affordances for different primitives associated with grasping. Song et al. Song et al. (2020) use a similar approach. Their grasping model predicts the future states for all possible actions to maximize the cumulative reward based on an action-view simulation. Moreover, Wu et al. Wu et al. (2020) introduce the spatial action maps as a new action representation and space that enable mobile robots to learn to navigate and manipulate by selecting navigational point on a map of local state observation. Xia et al. Xia et al. (2020) proposed ReLMoGen, whose deep Q-Learning variant is similar to spatial action maps. Their work enables robotic arms to navigate and manipulate.

Chapter 3

Methods

Our work is implemented in PyBullet Coumans and Bai (2017), a physics simulation tool for various applications including robotics, reinforcement learning, etc. The training and testing environment is a 3×3 meters square playground enclosed by walls with a random number (1 to 20) of randomly positioned 0.1×0.1 meters unmovable obstacles (cubes) and a 0.044×0.044 meters target cube. A running time image of it is shown in Figure 1.1. The agent is equipped with a simulated depth camera with a 0.25 meter sensing range and a degree of 90 field of view facing in the moving direction. There is no prior information available to the agent, so it needs to incrementally build a map over time based on point clouds generated by the depth sensor.

3.1 Problem Formulation

As most reinforcement learning based methods, our method can be modeled as a sequential decision process. At the beginning of a task, the agent's position and pose are randomly initialized inside this environment. The agent explores the environment until the target object is found (any part of the target cube is within the sensing range of the depth camera). At each time t, given a representation of the current state s_t, a policy π is employed to provide an action command a_t for the agent to execute as

follows:

$$a_t = \pi(s_t).$$

Once a_t is finished at timestamp $t+1$, the agent receives a reward r_t and observes the updated state to get the new representation s_{t+1}. The agent is expected to maximize the γ-discounted cumulative future reward (Q-value) until the task is finished, $\gamma \in [0, 1]$:

$$Q(s_t, a_t) = \sum_{i=1}^{\infty} \gamma^{(i-t)} r_i.$$

Using a Q-learning Mnih et al. (2015) algorithm, the optimal Q-function can be approximated in an iterative way by the Bellman equation:

$$Q(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}).$$

In order to get a higher Q-value in s_{t+1}, we trained a policy π to choose the a_{t+1} that maximizes the Q-function:

$$a_{t+1} = \arg\max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}),$$

where θ is the weight of the neural network (Q-network). Similar to Wu et al. (2020), the agent is trained by a double DQNVan Hasselt et al. (2015) to minimize the loss at each iteration i:

$$\mathcal{L} = |r_t + \gamma Q_{\theta_i'}(s_{t+1}, \arg\max_{a_{t+1}} Q_{\theta_i}(s_{t+1}, a_{t+1})) - Q_{\theta_i}(s_t, a_t)|,$$

where θ' represents the weights of the target-network.

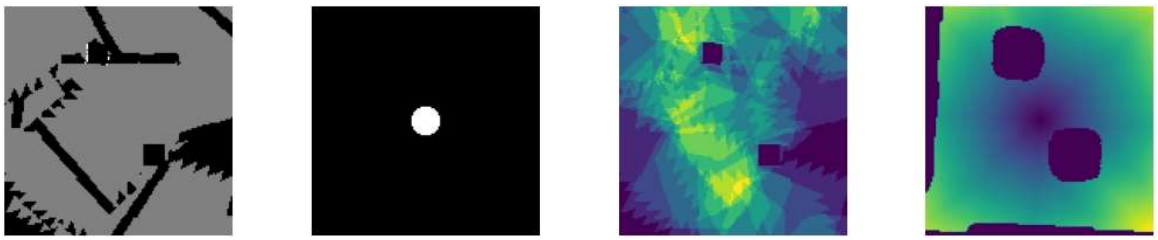

Figure 3.1: Each channel of state representation. All of these channels are defined in the same domain. From left to right: overhead map, robot position map, VFM, and shortest path map.

3.2 State and Action Representation

State Representation. The current state of the environment and the agent are represented by egocentric local maps based on the new observations and the already built map of the agent. The state representation consists of 4 top-down view channels as Figure 3.1 shows in the same domain encoding different pieces of information: overhead map, robot position map, VFM, and the shortest path on the agent map. All of these maps are cropped and rotated so that the agent's facing direction is always vertical up. The overhead maps represent the pixel-wise occupancy and segmentation, which allow the agent to avoid collisions and recognize the target object. The robot position map is a simple binary image representing the robot's location in the general local maps. The shortest path on the agent map holds the shortest path distance from the agent to each pixel in the local map of the current state. The VFM encodes the number of times each pixel is scanned by the depth sensor, which directs the agent to explore more unseen areas. All of these channels except the robot position map are empty at the beginning of a task and are updated over time as the depth sensor captures new environmental structure data, which is a typical online-SLAM behavior.

This setting is similar to the work of Wu et al. Wu et al. (2020). Their state representation is the same as our method but they use the shortest path to recep-

tacle map instead of the VFM to execute a mobile manipulation task. Except the tasking difference, these two state representations reflect different situational settings. As stated in the paper Wu et al. (2020), their settings illustrate the situation that the agent has access to the state observation by the sensor, the coordinates from an outside positioning system, egocentric local mapping, and the task-related goal coordinates. For comparison, our settings reflect the situation that the agent has access to the state observation by the sensor, the SLAM-based visit frequency record, and the local mapping, which simulate the situation that an agent is sent to a novel environment without the support of a global map and a positioning service.

Action Representation. We use an identical dense action representation as the spatial action maps Wu et al. (2020) defined in the same domain as the state representation. Each pixel belonging to the floor (an agent is unable to reach to points above obstacles or walls) represents an action at a location (the location can be viewed as a potential way-point in the current movement of the agent). After the process of state representation by a neural network, each pixel holds a Q-value, and the agent goes to the point where the pixel has the highest value as the Figure 3.2 shows.

When executing spatial action commands, there are many options, such as a straight line and a shortest path moving primitives, adaptive and fixed step size, etc. In the original paper, the effects of these options were comprehensively evaluated. So we follow the evaluation results and employ the best combination of the available options: our agent will use the shortest path moving primitive without the limitation of the step size to execute the spatial action commands.

In addition to these 4 channels, an occupancy map, which is an overhead map without segmentation, is generated by the system. The shortest path from agent map and the path to execute the shortest path moving primitive are computed based on this map. Any area that is unexplored in this map will be treated as free space.

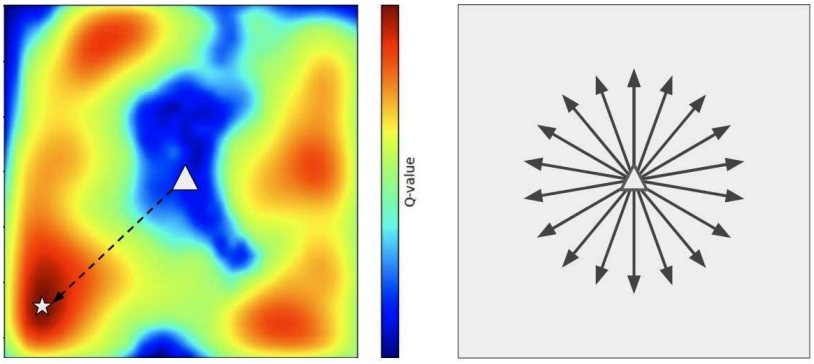

Figure 3.2: The left graph shows spatial action maps. The star indicates the position with the highest Q-value. The action space of it is all the free area, which covers all the possible movements in the local map. The right graph shows the 18-direction steering commands, whose size of the action space is only 18.

3.3 Rewards and Penalties

Reward Function. The reward system encourages the exploration of new areas and penalizes revisits. When a previously unseen area A is found, the agent will be rewarded by the pixel size of it:

$$r = \text{size}(A).$$

For instance, a 4×4 square newly found area will reward the agent with a value of 16. It worth noting that finding the target cube is not rewarded since there is no target related information available to the agent. The agent's policy only affects the relative efficiency of finding the target cube.

Penalty Function. The most intuitive penalty function design is calculating the pixel size or scaled pixel size of revisited areas:

$$p = \frac{\sum_i^{\text{size}(A_e)} f_i}{\text{size}(G)},$$

where $\text{size}(A_e)$ represents the pixel size of a previously explored area revisited by

the agent, f_i is the number of times a pixel has been visited inside A_e, and size(G) is the pixel size of the whole experimental environment, which is a scale factor.

However, this penalty function has two critical disadvantages. First, there is no upper limit on the penalty value, but newly explored areas can only reward the agent once. This would cause penalty explosion: an unreasonably high penalty value. Second, since the penalty value for non-movement and collision is limited, the agent would stuck around walls or obstacles if visiting a nearby area would receive a higher penalty value than the normal one. So we propose a sigmoid penalty function:

$$p = \sum_{i}^{\text{size}(A_e)} \frac{2}{1 + e^{-f_i}}.$$

This function has two main advantages. First, the max number penalty that a pixel can penalize an agent is 2. This can avoid the aforementioned two shortcomings. Second, from our observations we noticed that some areas (e.g., a channel connecting multiple areas) would be unavoidably visited more than once by the agent. Our design would tolerate and penalize the first several times of visiting these pixels. We also compared these two penalty functions in Section Chapter 4. In addition, collisions and non-movement also negatively affect the whole process, so the agent will be penalized by 15000, which is the double of maximum possible sigmoid penalty of a revisit in a single action under the setting of our experimental environment, if such behaviors occur.

3.4 Neural Network

We leverage the same network architecture and strategy as the original spatial action maps method Wu et al. (2020). The ResNet-18 He et al. (2016) without batch normalization layers serves as the backbone network of the fully convolutional neural network, which takes the aforementioned 4-channel state representation, and then

generates a command on the spatial action maps. In order to obtain outputs whose size are the same as the inputs, there are 3 convolutional layers, which use 1×1 kernels, at the end of the backbone network replacing the AvgPool and the fully connected layers. In addition, there are 2 bilinear upsampling layers (scale factor: 2) inserted in the 3 convolutional layers. Since the area of our environment is larger than the original setting, the size, 128×128 of our local map (agent's observation of the state) is larger than 96 in order to ensure there is an adequate receptive field for the fully connected network.

Chapter 4

Experiments

In this section, we evaluate and report the performance of the spatial action maps with the VFM and the sigmoid penalty function (SAM-VFM-SIG) on an exploration task, the same method with the incremental penalty function (SAM-VFM), the spatial action maps (SAM) without VFM, the traditional steering commands based method (ST-COM) taking the same 4-channel state representation as the SAM-VFM and SAM-VFM-SIG in Section 4.3, and the random method (RAND) that randomly picks a destination within the local state space map. Note that we provide the RAND results only for reference since the input of it is totally different from the other methods. The evaluation metrics and model training details are in Section 4.2 and Section 4.1.

Regarding ST-COM, we modified the neural network so that it does not generate spatial action maps but traditional steering commands. The action space of ST-COM is 18, which means there are 18 possible directions that the agent can move along, and the step length (25cm) of each action is fixed.

4.1 Training

In our experiments, we run a 1000 transitions of the random policy to fill the replay buffer (6000) with initial experience. Then we train our neural network for 35000 iterations with a stochastic gradient descent (SGD) optimizer whose learning rate, momentum, and decay are 0.01, 0.9 and 0.0001. The training environment will be reset when an episode is finished or reached the maximum steps allowed for an episode, which means the target is found by the agent. The training is implemented on a RTX TITAN X 11GB GPU.

4.2 Metrics

GRER: Global Repetitive Exploration Rate (lower is better). The GRER evaluates the efficiency of the exploration: the lower the GRER is, the less a unit size area has been visited. This metric is calculated by the formula:

$$GRER = \frac{\sum_1^{\text{size}(V)} f_i - \text{J}}{\text{size}(V)},$$

where V is the global VFM, f is the value of each pixel on V, and J is a matrix of ones whose shape is the same as V. Each f_i is reduced by 1 since we only pay attention to the numbers of the revisit frequency.

CE: Command Efficiency (Higher is better). The CE evaluates the average new area explored by each command. This metric is calculated as:

$$CE = \frac{\text{size}(E)}{\text{num}(C)},$$

where E represents the explored area and num(C) represents the number of commands generated before the target cube is found.

PE: Path Efficiency (Higher is better). The PE evaluates the quality of a generated path in a global view by calculating the average new area explored per unit length of path. This metric is calculated as:

$$PE = \frac{\text{size}(E)}{\text{length}(D)},$$

where E represents the cumulative explored area and length(D) represents the cumulative distance the agent travelled before the target cube is found. Note that we only calculate the cases that length(D) > 1 in order to reduce the offset.

FR Failure Rate (Lower is better). In some environmental settings some methods are unable or take too long time to eventually finish the task. It is worth noting that if we add those data points to the final results, the average value will be greatly affected. So we mark a task lasting longer than 10 minutes as failed and add the data points at 10 minutes to the final average results calculation.

We evaluated all candidate methods for 200 episodes. For each episode they are run in a randomly generated environment, whose basic setting is the same as the training environment except that the number and positions of the obstacle cubes are different. We ensure the environment settings are the same for each of these methods by applying the same random seed. Agents in all experiments are with an ϵ-greedy policy whose $\epsilon = 0.01$.

4.3 Performance and Ablation Analysis

The detailed experimental results are reported in Tables Table 4.1 to Table 4.4, and the result trends of methods are illustrated in Figure 4.1. In summary, SAM-VFM-SIG shows better performance on all metrics and the performance is quite stable in all testing episodes. In addition, SAM-VFM has more stable performance for some metrics.

	$\mu(GRER)$	$\sigma(GRER)$
SAM-VFM-SIG	**0.502**	0.452
SAM-VFM	0.839	0.470
SAM	1.659	1.738
ST-COM	1.151	1.421
RAND	3.420	2.603

Table 4.1: Average and Standard Deviation of GRERs

	$\mu(CE)$	$\sigma(CE)$
SAM-VFM-SIG	**2906.379**	890.254
SAM-VFM	1847.799	377.058
SAM	1882.847	995.570
ST-COM	1437.152	406.720
RAND	834.312	576.124

Table 4.2: Average and Standard Deviation of CEs

	$\mu(PE)$	$\sigma(PE)$
SAM-VFM-SIG	**6479.335**	2380.952
SAM-VFM	5402.662	2076.554
SAM	4326.754	2914.199
ST-COM	5480.642	2281.578
RAND	3083.404	1763.122

Table 4.3: Average and Standard Deviation of PEs

	Failure Cases
SAM-VFM-SIG	**0 / 200**
SAM-VFM	**0 / 200**
SAM	2 / 200
ST-COM	4 / 200
RAND	7 / 200

Table 4.4: Number of Failures in Exploration

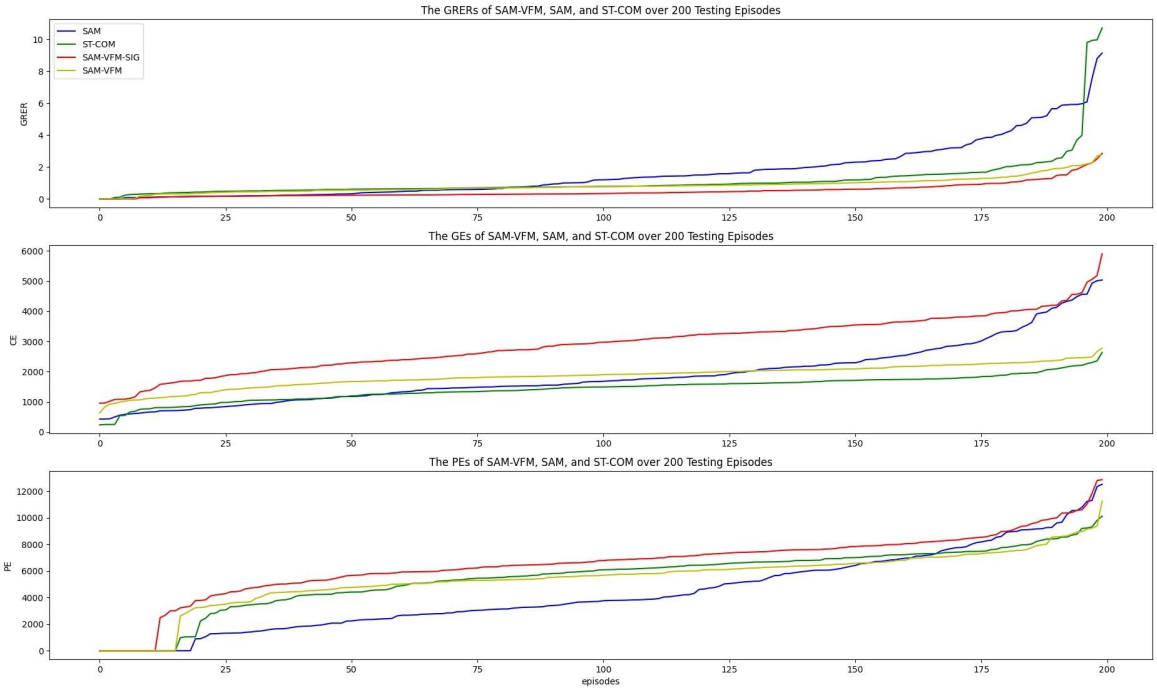

Figure 4.1: The GRER, CE, and PE scores of SAM-VFM-SIG, SAM-VFM, SAM and ST-COM over 200 testing episodes. SAM-VFM-SIG outperforms SAM-VFM, SAM, and ST-COM among all metrics. The spatial action maps based methods show great advantage over the steering commands based method. Note that we sorted each method's scores from low to high, so scores of different methods at the same x-axis position cannot be directly compared with (These plots share the same legend).

Effect of the VFM. By comparing the performance of SAM-VFM-SIG and SAM-VFM with SAM, we are able to recognize the effect of the VFM. Based on the results, the VFMs are able to help spatial action maps based methods improve the exploration efficiency by reducing the frequency of redundant revisits and generate commands that explore many more previously unseen areas. It is worth noting that the performance of SAM-VFM-SIG and SAM-VFM is also more stable, which shows the adaptability brought by the VFM to the different environmental settings. We conjecture that there are two reasons for the better performance. The first is that the VFM can penalize agents heavier for undesirable behavior. The second is that even though all the pixels on a local state map are explored, their corresponding visit frequency values are different. So the agent can still be directed to a place where the penalty is relatively low. However, this cannot be achieved in SAM. So it is hard for the agent to make reasonable decisions and highly possible that it just randomly chooses a position without concrete reasoning. It is worth noting that ST-COM equipped with VFM even outperformed SAM with respect to the GRER and PE, which conclusively demonstrated the fundamental impact on reducing the revisiting instances by VFMs.

Effect of the Sigmoid Penalty. Regarding the performance of GRER, CE, and PE, the sigmoid penalty function brought considerable advantages to SAM-VFM-SIG over SAM-VFM. The main reason behind this, based on our observations, is that the agent with the sigmoid penalty acts more closely to what we desire. At the early stages of a task, the agent dares to explore low-visit-frequency areas. Though these visits would bring some penalty, which is minor, they supply the agent with the ability to pass through less explored areas to reach unvisited places. On the contrary, SAM-VFM with the incremental penalty function has no such flexibility.

Effect of the Spatial Action Map. There is a great performance advantage of SAM based methods over the ST-COM based method when they have the same

inputs. We believe that it is the difference between the spatial action map commands and the steering commands that lead to these results. We assume that the input and output defined in the same domain helps the neural network better understand the spatial relation between state observation and action space. In addition, we noticed that in many situations agents with the ST-COM move following a circular trajectory and never get out of it until the system execution is suspended. Regarding this we conjecture the reason is that the action space of the ST-COM is only 18 with fixed step size, so it keeps trying to reach a position that is always showing in the local observation with higher rewards or lower penalty, but due to the limited action options, the agent can never reach that point.

Chapter 5

Conclusions

We make the first step towards using spatial action maps to perform exploration tasks and introducing the VFM and the corresponding rewards structure to help the agent effectively explore diverse environments. The results show that our methods are more efficient than traditional steering commands based methods, and the VFM is able to help the agent reduce the frequency of revisiting places. Additionally, the use of VFM provides a source of meaningful rewards for training the agent without requiring additional information from the simulation environment. This is appealing when considering deployment of reinforcement learning systems to real environments.

References

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and van den Hengel, A. (2018). Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, 29:1471–1479.

Chen, G., Pan, L., Chen, Y., Xu, P., Wang, Z., Wu, P., Ji, J., and Chen, X. (2020). Robot navigation with map-based deep reinforcement learning. *arXiv preprint arXiv:2002.04349*.

Chen, T., Gupta, S., and Gupta, A. (2019). Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*.

Chen, X., Ghadirzadeh, A., Folkesson, J., Björkman, M., and Jensfelt, P. (2018). Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3110–3116. IEEE.

Coumans, E. and Bai, Y. (2017). Pybullet, a python module for physics simulation in robotics, games and machine learning.

Gao, W., Hsu, D., Lee, W. S., Shen, S., and Subramanian, K. (2017). Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In Levine, S., Vanhoucke, V., and Goldberg, K., editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 185–194. PMLR.

Gul, F., Rahiman, W., and Nazli Alhady, S. S. (2019). A comprehensive study for robot navigation techniques. *Cogent Engineering*, 6(1):1632046.

Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2017). Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A., and Farhadi, A. (2017). Ai2-thor: An interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G.,

Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Pfeiffer, M., Schaeuble, M., Nieto, J., Siegwart, R., and Cadena, C. (2017). From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1527–1533. IEEE.

Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive UAV control in cluttered natural environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 1765–1772. IEEE.

Savva, M., Chang, A. X., Dosovitskiy, A., Funkhouser, T., and Koltun, V. (2017). Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Song, S., Zeng, A., Lee, J., and Funkhouser, T. (2020). Grasping in the wild: Learning 6DOF closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters*, 5(3):4978–4985.

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*.

Winfield, A. (2009). *Foraging Robots*, pages 3682–3700. Springer New York.

Wu, J., Sun, X., Zeng, A., Song, S., Lee, J., Rusinkiewicz, S., and Funkhouser, T. (2020). Spatial action maps for mobile manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*.

Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. (2018). Building generalizable agents with a realistic and rich 3D environment. *arXiv preprint arXiv:1801.02209*.

Xia, F., Li, C., Martín-Martín, R., Litany, O., Toshev, A., and Savarese, S. (2020). Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. *arXiv preprint arXiv:2008.07792*.

Xia, F., Zamir, A. R., He, Z., Sax, A., Malik, J., and Savarese, S. (2018). Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079.

Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., et al. (2018). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3750–3757. IEEE.