– Senior Thesis Spring 2022 –

Policy Transfer and Limitations of Spatial Action Maps

Chandler Ault
Adviser: Szymon Rusinkiewicz

Abstract

Spatial action maps developed by Wu et al. have shown promising results in applications such as foraging, search and rescue, and exploration [11, 10, 12]. These policies are trained in simulation on simple agents with no onboard sensing. Given the success that these agents demonstrate, the question of whether these policies can be transferred to new agents with more capabilities naturally arises. Here, I implement spatial action maps for foraging in simple environments on a TurtleBot3 Waffle Pi and deploy it in a Gazebo simulation. Results show that the policies only transfer effectively if measures are taken to mimic the movement patterns of the agent the policies were originally trained on. Furthermore, results indicate that path planning from sensor data in adverse environments limits the transferability of the policies.

1. Introduction

1.1. Motivation and Goal

In recent years there has been an influx of interest in the field of robotics due to the limitless applications of such technology. As the prevalence of robots increase, the need to train policies and implement solutions will increase commensurately. The computational load of training these policies combined with the enormous amount of training data associated with it will prove burdensome.

Thus, there is motivation to be able to transfer policies trained on one robot with its hardware capabilities and dynamics to other robots with differing hardware capabilities and dynamics. The ability to do this policy transfer in even a few cases would drastically reduce the computational load of training and would facilitate quicker deployment times for key tasks. However, the criteria under which such transfers are possible and the efficacy of these transfers is unclear. Furthermore, the degree to which performance will drop-off when transferring policies learned in highly idealized simulation conditions to less ideal scenarios is unknown.

Taking a step back, Wu et al. sought to improve performance of goal-oriented navigational tasks by introducing "spatial action maps" which defined the action space in the same domain as the robot's state space. This deviated from typical steering command action representation and facilitated more elaborate, goal-oriented movements. Their original implementation focused on Anki Vector robots tasked with foraging by pushing objects into receptacles. They trained their policy in simulation and tested it in both simulation and reality. Overall, spatial action maps showed improved learning rates of complex behaviors, and it achieved better performance compared to its steering command counterpart.

Although spatial action maps performed well, there are some limitations to the implementation. First, the authors state that their implementation is limited by the assumption of perfect localization and mapping. This is due to training in simulation and utilizing the simulated camera views for Simultaneous Localization and Mapping (SLAM). Second, spatial action maps are limited by what they call "high-level motion primitives." In other words, the implementation abstracts the action to

a destination on a map and allows the underlying software and hardware to decide how to reach the location. This combines the benefit of representing more complex action goals with the drawback of not guaranteeing that all agents will take the same low-level controls to reach them.

Some research indicates that attributes of policies that can be effectively transferred is the abstraction of the action space away from robot-level commands. This is exactly the approach used in spatial action maps which lends credence to the notion that spatial action map policies can be highly transferable between agents. However, the extent to which this is true has not been tested. If spatial action maps are shown to create highly transferable policies, then this method of goal-oriented navigation will be even more attractive for future applications.

With this background of spatial action maps combined with the challenges and benefits of robot-to-robot policy transfer, the question of how well spatial action maps can transfer to new agents naturally arises. Thus, the goal of this project is to explore the extent to which the spatial action map policies can be transferred to new robots and to explore the performance drop off therein. Such an exploration is valuable because it will test some of the limitations discussed in the paper while also contextualizing the project in the realm of robot-to-robot policy transfer.

1.2. Overview of Challenge

The challenge of transferring the spatial action map policy to a new agent is multi-faceted. First, there is the challenge of computing the requisite input channels for spatial action maps with the new agent. Choosing how to create an overhead map with the proper values for each object in the environment is not trivial and requires a blend of laser scans and imagea captured through a front-facing camera. The challenge here is deciding how these sensors will work together to provide the best outcome. The second challenge is configuring the new agent to carry out the actions specified by the spatial action map. Creating movement goals and allowing the underlying path planning capabilities of the agent to determine the best path may not result in ideal performance. Since the policies were trained on one agent, the neural network will learn how to optimize the movement patterns of the original agent. Thus, when carrying out an action, it will likely be

necessary to attempt to mimic the movement patterns of an Anki Vector robot in the new agent.

Furthermore, there is the challenge of evaluation. Running experiments in both simulation and reality are time consuming endeavors though the latter is more time consuming by far. In either case, development and design of the testing environments is critical for evaluation. Additionally, it is necessary to vary different parameters of the implementation to quantify how certain limitations of the spatial action map implementation impact performance. Thus, choosing which parameters to alter and choosing the testing environment carefully will certainly be challenging.

1.3. Summary of Approach

To achieve this goal, I will port the spatial action map policies to a TurtleBot3 Waffle Pi. TurtleBots have a similar overall structure to the Anki Vector robots used in the original implementation which provides a good baseline for comparison. However, TurtleBots have on-board sensing capabilities not seen in the Anki Vector robots used. The additional sensors will allow us to test the policies in imperfect localization and mapping scenarios. Furthermore, testing the policy on TurtleBots will allow us to see how the policy performs on agents with different low-level controls. Thus, transferring the spatial action map policy to TurtleBots will allow us to test key limitations in the original paper without drastically altering the features of the underlying agent.

1.4. Summary of Results

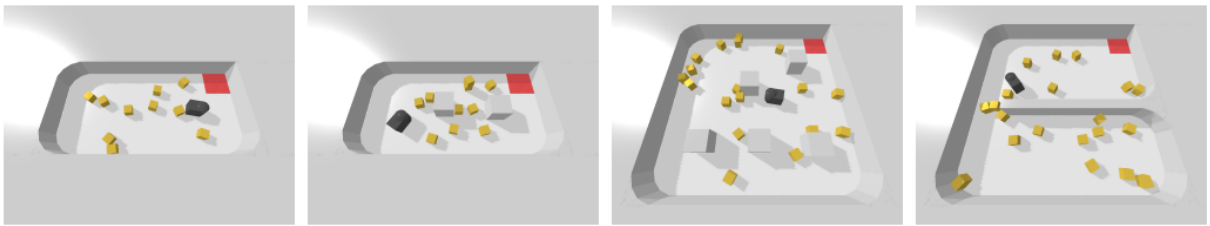
Overall, experimentation showed that spatial action map policies can successfully be transferred between agents. The quality of this transfer is dependent on how carefully the execution of actions is designed to mimic the original agent. In addition to this, it can be seen that imperfect tracking of objects in the setting substantially decreases efficiency but only slightly hinders performance in simple settings. Therefore, it is certainly possible that some limitations of spatial action maps can be mitigated and policies can be efficiently transferred between agents.

2. Problem Background and Related Work

2.1. Spatial Action Maps

The key motivation and fundamental building blocks for this project comes from Spatial Action Maps for Mobile Manipulation by Wu et al. Spatial action maps represent action as a dense map of navigation endpoints [11]. This action representation makes no indication of what trajectories to use to reach these endpoints, and it is up to the agent to decide what linear or non-linear trajectory is best to reach the goal. Allowing the goal to be executed in a single action facilitates goal-driven learning of the deep Q-networks underlying spatial action maps [11].

Additionally, spatial action maps use an input state space of four pixel-aligned overhead maps encoding different values of the environment and an output space that is pixel-aligned with the input space. Since both the input and output states are in the same domain and pixel-aligned, this allows the use of fully convolutional neural networks (FCN) to map output values from the input state. The use of FCN's was motivated in part by the sample efficiency of end-to-end robotic manipulation as well as improved performance in other domains [11].



(a) SmallEmpty (b) SmallColumns (c) LargeColumns (d) LargeDivider

Figure 1: The four environments simulated by Wu et al. for spatial action map training and testing.

For training and testing, Wu et al. utilized four environments: a small and empty environment, a small environment with columns, a large environment with columns, and a large environment with dividers. These environments can be seen in Figure 1. Each of these environments had randomized block locations, and the environments with columns randomized those as well.

In the study, Wu et al. show that spatial action maps not only learn faster than their steering base counterparts, but vastly outperform them as well. The comparison between these two approaches

and the variations of the spatial action map approach can be seen in Figure 2. In all environments, the spatial action map approach was able to recover over 90% of blocks on average which is a very impressive result compared to the baseline.

However, there are a few key limitations of this approach. First, the authors points out that simulated environments are a limitation since they are constrained by simulation accuracy. Additionally, their implementation is limited by the assumptions of perfect localization and mapping. Lastly, there is the inherent limitation of spatial action maps which is the quality and accuracy with which the outputted motion primitives are carried out. These limitations are of particular interest when using spatial action maps on new agents since localization, mapping, and action execution can vary across agents.

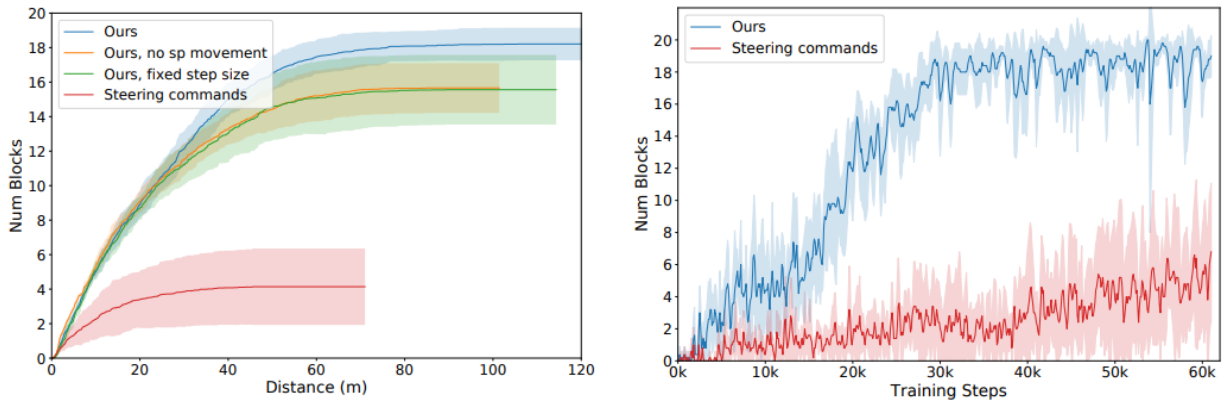


Figure 2: On the left, the number of blocks collected by various implementations in the large divided environment as a function of distance traveled in meters. On the right, the number of blocks as a function of training steps for spatial action map and steering command approaches.

2.2. Application of Spatial Action Maps

2.2.1. Spatial Intention Maps: After researching the use of action maps in single-agent mobile manipulation, Wu et al. sought to apply this technology to generate spatial *intention* maps. Spatial intention maps encode agents intended actions into a 2D overhead map so that multi-agent systems can work together more efficiently to complete tasks [12]. Similar to spatial action maps, spatial intention maps use a series of overhead, robot-oriented, and pixel-aligned local maps to encode both the state of the environment and the state of the agent itself. This implementation, however, utilizes

an additional overhead map called the spatial intention map. This map is created by rendering the trajectories of the most recently broadcasted goals of other agents into another overhead, pixel-aligned map. These inputs maps are then fed into a double deep Q-learning network (DQN) which computes the Q-value of each point in the local map and return the corresponding action map.

Overall, spatial intention maps are an interesting application of the ideas of spatial action maps in multi-agent mobile manipulation. Again, some of the same limitations face spatial intention maps as faced spatial action maps. Specifically, the accuracy of mapping and localization. Additionally, when mapping intentions, understanding movement patterns of other agents along with the agent in question are keys for success. Thus, if spatial action maps can effectively complete tasks with a new agent and new movement patterns, it gives hope that policy transfer could also be successful in spatial intention maps as well.

2.2.2. Spatial Action Maps for Exploration: In addition to the application of spatial action maps by Wu et al, others have leveraged spatial action maps for different tasks. Wang et al. applied spatial action maps to effeciently explore novel environments [10]. This approach also utilized four input channels, but it replaced the shortest path to receptacle map with a unique visit frequency map (VFM) as seen in Figure 3. This is done to incentivize agents to explore unfrequented areas on the VFM. Additionally, Wang et al. introduce new environments for training and testing their exploratory implementation of spatial action maps.

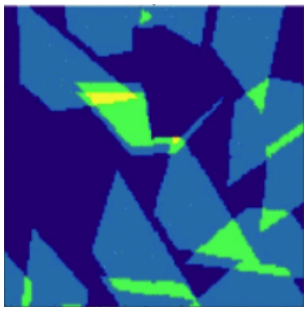


Figure 3: An example of the visit frequency map (VFM) used in Wang et al.

Once again, results show that spatial action maps with VFM's as an input outperformed traditional steering commands [10]. Agents were able to plan paths and execute goals while reducing the

frequency of revisiting areas in the map. The success of this spatial action map implementation opens the door to future implementations with new state spaces acting as inputs. Therefore, spatial action maps demonstrate tremendous potential in foraging, search and rescue, and exploration tasks.

However, the original action map implementation, the spatial intention map implementation, and the VFM implementation all utilize a simulated environment and a simple agent without onboard sensing. Thus, the question of how new agents will work with spatial action maps and whether the policies detailed above will transfer to new agents remains unanswered.

2.3. Policy Transfer

Transferring policies to new robots is an important area of study because of the computational resources that go into training said policies. There have been various instances of research looking into how to transfer policies to new hardware [1, 8, 5, 3, 7]. Some of these papers look at transferring policies to new agents with various degrees of freedom and dynamics [8, 1]. One key observation from one of these works is that their policy transfer approach performs better for tasks whose policies are not heavily dependent on the dynamics of the agent [1]. In other words, policies based on providing actions abstracted from the agent are more easily transferred to other agents. Another attempt at policy transfer led to an algorithm that interpolates between the original robot the policy was trained on and the target agent [8]. This allows the policies to evolve to their final state over the course of the algorithm. However, this approach was specifically targeting robots that varied heavily in their dynamics and controls, so such an algorithm is not as necessary for agents with smaller differences.

One paper attempted to layout the conditions under which policies can be transferred [7]. The findings of their research is that a key component of transferable policies is that there is an accurate mapping of input state and output action between the source and target agent. Provided this holds true, the transfer process is significantly more successful.

Along this line of thinking, some researchers have focused on leveraging modularity to improve transfer accuracy [9, 3, 4]. Research showed that modularizing a system so that neural networks

are learning high-level action policies and low-level controls are either learned or hand-designed separately allowed better transfer to new domains [9]. This can be viewed as decomposing the components of a task into "task-specific" and "robot-specific" modules [4]. Task-specific modules are modules looking at an abstracted version of the task and is independent of the robot's dynamics. Through related work focusing on modularizing policies, we can see that a policy that is abstracted away from the robot specific side of the task provides more opportunity for transferability.

To conclude, spatial action map implementation is rare in the realm of deep reinforcement learning (DRL). Most action spaces in DRL are represented by discrete moving actions, continuous velocity commands, or motor speed commands [13]. With that being said, some methods have taken similar approaches to related tasks, but these fail to modularize the robot-specific and task-specific components of the policies [6, 2]. Thus, it is to be expected that these policies would not be as transfer friendly as spatial action maps.

3. Approach

To test some of the limitations of spatial action maps as well as the efficacy of its policy transfer, I ported the neural network and action dynamics of the original spatial action map implementation to a TurtleBot3 Waffle Pi. The TurtleBot3 has many similar features and dynamics to Anki Vector robots but has the added benefit of onboard sensing. This sensing can be used to perform SLAM, navigate, and perform object detection. This is key because it will allow us to test two of the limitations outlined in the original paper. First, this will provide context on the setting and experimental limitations faced in the Anki Vector setup. Specifically, the quality of motion primitive implementation and the assumptions of perfect localization and mapping. Second, it will allow us to test the inherent limitation of spatial action maps. Namely, that the policy outputs a high-level motion primitive while giving no indication of what the low-level controls should be to arrive at its destination. Thus, we can compare how an agent with different path planning and execution will perform on similar tasks.

Due to physical and temporal constraints of this project, we elected to approach the problem from

simulation alone. Doing so allows quicker test times and more customizable parameters. Although still a simulation, Gazebo provides a more robust simulation environment where more physical parameters can be altered and more accurately represented than in the original paper.

The novel idea of this approach is to isolate the limitations of spatial action maps and quantify their impact on the overall efficacy of the system. In doing so, we can also observe under what conditions the spatial action map policies will transfer to new agents. This is certainly one of the next steps in developing spatial action maps for practical use. Evaluation of key limitations and transferability can inform future research and provide methods of mitigating performance issues.

4. Implementation

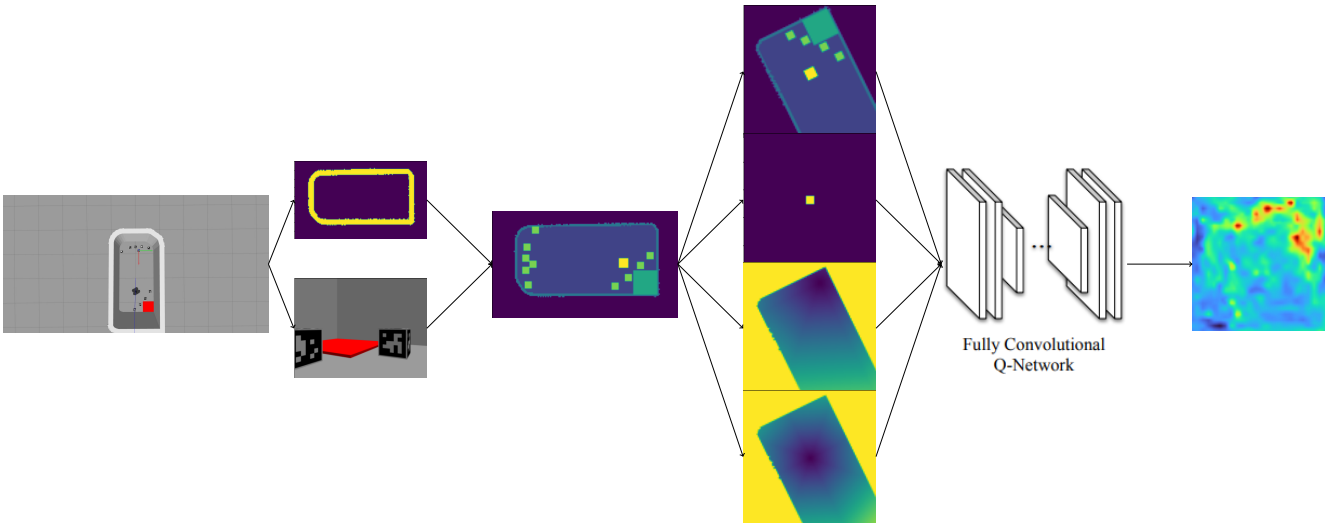


Figure 4: The state to output flow of the implementation. From left to right, the environment, the sensor observations, the reconstructed global map, the pixel-aligned local maps, the Q-network, and the pixel-aligned output.

4.1. System Overview

Overall, the system was constructed in the same manner as the original spatial action map paper. The primary differences relate to how the input channels are created and output actions are executed in TurtleBots and Anki Vector robots. The entire process can be seen in Figure 4.

For spatial action maps, there are four input channels for the neural network. First, there is an

overhead map encoding objects in the environment with specific values. To create this overhead map while also navigating it, I utilized the built-in ROS package GMapping for SLAM which provides a grid map indicating the occupancy status of cells. However, this method only constructs an overhead map containing the obstacles in the environment. To place the robot onto the overhead map, I sampled the values of the odometer and drew the robot into the corresponding location in the map. To place the blocks into the map, I used two methods separately. The first method assumed the blocks were perfectly tracked and localized and placed them in the exact location. The second method used the RGB camera to detect Aruco markers on the blocks and updated the location of the detected tags. Finally, the exact location of the receptacle is assumed to be known, so its exact position was calculated and placed into the overhead map. Spatial action maps need a local 96x96 pixel robot-oriented map, so one can crop a section out of the global map with the robot's location at the center and rotated it according to the odometer's readings. After doing this, one has the first input channel for spatial action maps.

In brief, the second input channel is a simple binary mask of the robot's location. To get this, I initialized a completely black 96x96 pixel image and altered the center pixels corresponding to the robot to their necessary values.

Finally, the third and fourth channels are very similar in their design as they 96x96 pixel-aligned robot-oriented distance maps to the agent and receptacle respectively. To calculate these, we used the occupancy grid maps detailed above and the respective locations for the goals to calculate the

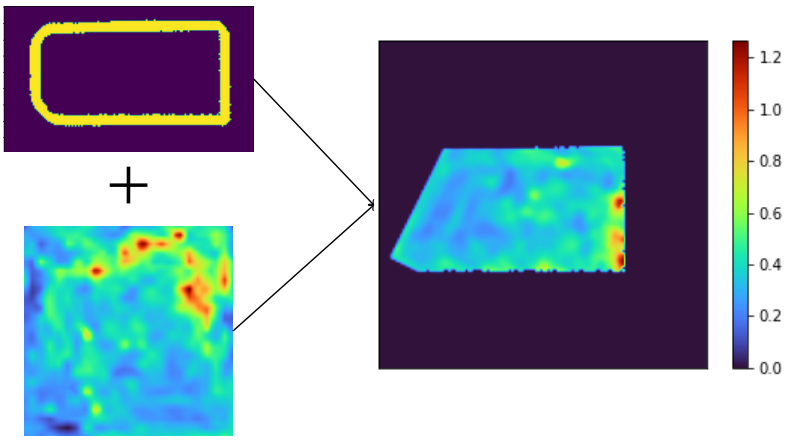
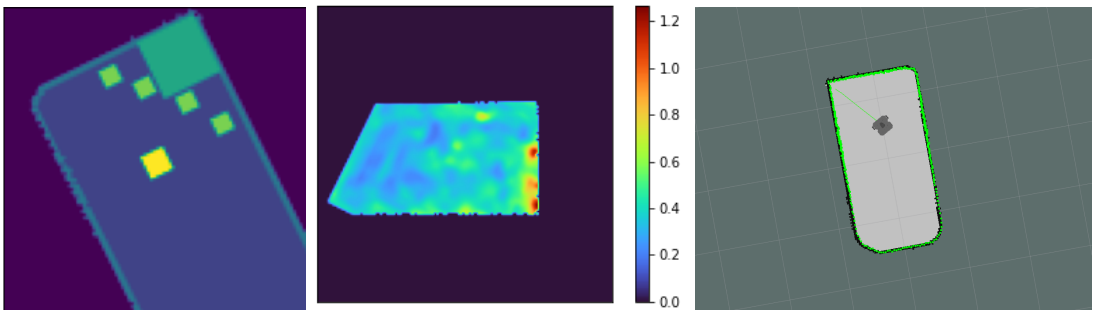


Figure 5: The occupancy map is combined with the spatial action map to create the global action map

shortest path using the Shortest Path Faster Algorithm (SPFA). This is the same implementation as found in the original spatial action map code.

The output of the neural network is a pixel-aligned action map where the value at each pixel represents the score of expected value if the robot went to the corresponding location on the map. Once this action map is calculate, it must be rotated from the agent's orientation back to the global map orientation. Once this is done, an occupancy mask is applied to eliminate values known to be unreachable through the occupancy grid map (see Figure 5). Finally, the pixel with the greatest value is selected and converted to coordinates in the global map frame.

To navigate to the specified location, I utilized the built-in *move_base* node which provides a ROS interface for interacting with the navigation stack of the robot. A goal pose containing the x-y coordinates in the overhead map as well as a direction is passed to the *move_base* node. Because the input to the *move_base* node requires orientation to be specified, I was forced to choose an orientation. Rather than choosing an arbitrary orientation for the pose goal, I elected to make the orientation follow the line passing through the agent's starting and goal locations. I did this in an attempt to eliminate rotational uncertainty after the TurtleBot reaches its goal position. Theoretically, the agent should be following the shortest path between two points which is a line in an empty environment. Thus, this implementation is justified for some environments. However, for other environments, methods such as reducing the rotational tolerance for the global and local planners could also be justified.



(a) Local state (b) Spatial action map (c) Path planned in green

Figure 6: The path planned from the given state space and spatial action map.

The *global_planner* package was used for global path planning in this implementation. This ROS package allows for a variety of customizations in how the global path is calculated. By default, this package utilizes Dijkstra's algorithm although the A* algorithm can be used if stipulated. See Figure 6 for an example of the path created for a given environment state. For the global path planner, all default values and arguments were used for consistency. Additionally, this implementation utilizes the *dwa_local_planner* package in ROS for the local planner of the TurtleBot. This package implements the Dynamic Window Approach (DWA) for local robot navigation. DWA samples trajectories and selects the best one given an objective cost function. This cost function takes the form:

$$\begin{aligned} Cost = {} & path\_distance\_bias * (\text{distance}(\text{trajectory endpoint} \rightarrow \text{path})) + \\ & goal\_distance\_bias * (\text{distance}(\text{trajectory endpoint} \rightarrow \text{local goal})) + \\ & occdist\_scale * (\text{trajectory's obstacle cost}) \end{aligned} \quad (1)$$

All default values for *path_distance_bias*, *goal_distance_bias*, and *occdist_scale* were used. The *max_scaling_factor* parameter, the factor by which to scale the robot's footprint, was reduced to 0 to allow the local planner to plan trajectories nearer to walls without predicting erroneous collisions.

To further test how navigational differences between robots could affect performance, a feature was added to turn the agent in the direction of the desired goal prior to moving. This was done after observing movement patterns of the Anki Vector bots upon which the original neural network was trained. The Anki Vector robots appear to travel linearly. In contrast, the path planning component of TurtleBots often move in looping trajectories if not directly facing the goal. Consequently, this movement pattern often forfeits the intended benefit of the action. Thus, rotating towards the goal will allow us to compare how customizing movement patterns to mimic those of the original agent will improve performance.

4.2. Implementation Issues

During the course of developing this implementation, a variety of issues arose. These issues arose from differences in agents, in mapping, and in action execution. The primary issues have been laid out here.

4.2.1. Distance Map Scale: One issue that arose in the construction of the distance maps lied in scale. Due to the differences in resolution as well as environment size, the distance values calculated through the SPFA implementation were far larger than in the original implementation. This was discussed in the paper as they tuned a scaling hyperparameter for each environment which gave the best performance. One possible solution for our implementation would be to tune a hyperparameter ourselves, but this seemed not only tedious but prone to error. Thus, our implementation had a constant scaling factor corresponding to the product of the resolution of our map, the scaling factor of the TurtleBot to Anki Vector bot, and their original scaling factor. This took the form of:

$$resolution * sizescale * shortest\_path\_map\_scale = .025 * \frac{1}{3} * .25 \approx .002 \quad (2)$$

There were undoubtedly other ways to tune this parameter, but this method gave values in the same order of magnitude as the original implementation and provided qualitatively good results.

4.2.2. Path Planning and Obstacle Inflation: Another issue that arose in the implementation of spatial action maps was in the obstacle inflation of local path planning. One of the key behaviors that emerged in the training process for the spatial action map neural networks was that the agent would maneuver blocks to the walls of the environment and proceed to sweep many blocks along the wall and into the receptacle at once. The problem with this is that the path planners used to navigate the agent around the map are extremely averse to getting near obstacles such as walls. This is clearly untenable for the policies attempting to be implemented. Not only did the agent fail to

carry out the policies well due to obstacle inflation, but it would also get stuck near the walls and corners because it could not find a viable path away from the wall without colliding with it.

There are a few ways to address this issue. First, one could reduce the cost factor associated with approaching obstacles. Although this helps somewhat, this still does not allow the robot to approach as close to the walls as necessary for spatial action maps to perform well since there is still an inflation around the obstacles. Additionally, the agent can still get stuck in certain configurations due to the path planning not allowing any collisions. Second, one could reduce the inflation factor of obstacles in the map. This allows the robot to get closer to the walls and behave how it is intended to, but ultimately, this does not prevent the robot from getting frozen near walls due to collision avoidance. Third, one could reduce the agent's footprint size passed to the local planner. This allows the agent to get out of tight situations near obstacles and in corners, but it comes at the cost of a significant amount of scraping against obstacles and possibly even more violent collisions.

Ultimately, the solution that I settled on was to not touch the cost factor associated with approaching obstacles, but to significantly decrease the inflation factor of obstacles and the footprint size of the agent. I did this because I still wanted the agent to avoid obstacles if there was a better path, but I also wanted true behavior of spatial action maps to emerge without the agent getting frozen along a wall. Furthermore, I observed that it was not unusual for Anki Vector bot to make light contact with the walls of its environment. In non-simulated implementations, a different, more nuanced approach would certainly be more reasonable. However, given the low risk setting, some light collisions with the wall was deemed acceptable.

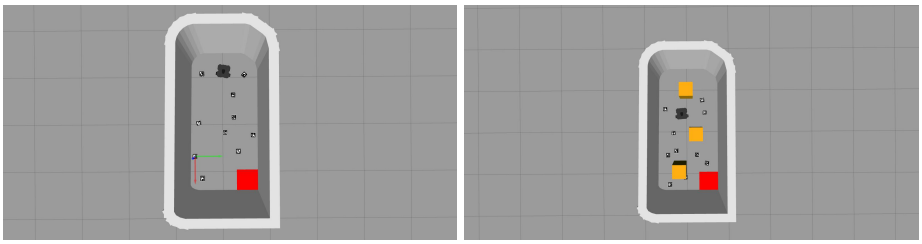
5. Evaluation

5.1. Experiment Design

5.1.1. Overview: The experiment consists of running the spatial action map policy on a TurtleBot3 WafflePi in a Gazebo simulation and collecting the number of blocks pushed into a receptacle and the distance traveled to do so.

5.1.2. Task: In each iteration of the experiment, the agent's goal is to push as many blocks as possible into a receptacle located in the corner of its environment. The agent does this by navigating to the coordinates of the best position outputted by the spatial action map.

Each variation of the experiment will iterate 10-15 times. At the beginning of each iteration, the location of the agent and blocks is randomized. The iteration concludes when either all blocks have been pushed into receptacles or 100 actions have been executed with no additional blocks entering the receptacle.



(a) Small, Empty (b) Small, Columns

Figure 7: Two environments used for testing spatial action maps on a TurtleBot.

5.1.3. Environment: The experiment consists of two environments. These two environments can be seen in Figure 7. The first environment is a replicated version of the small, empty environment from the original paper. This environment along with the blocks and receptacles contained therein are scaled to match the ratio of the original environment. This means that all objects are scaled to roughly three times their original size. Thus, the walls enclose an approximately 3m x 1.5m area, there are 10 blocks each measuring 10 cm, and the receptacle is a 40cm square placed in one of the corners of the environment.

The second environment has the same enclosure, block size, and receptacle size and location as the first, but it differs in that it contains 1-3 additional 30cm x 30cm stationary columns. This second setup will test the path planning capabilities of the new agent in carrying out its goals in complex environments. Both environments will have the starting location of the agent,blocks, and

columns randomized.

5.1.4. Variations: There are four different experimental variations for the small empty environment. The four variations for the small empty environment will aim to test how different limitations impact performance. First, the baseline action map will be implemented with no control stipulations and no assumption of perfect block tracking. Second, the action map policy with perfect block tracking will be implemented. Third, the action map policy without perfect tracking, but with a pre-action rotation towards the goal will be implemented. Finally, the action map policy with both perfect tracking and pre-action rotation will be implemented.

For the small environment with columns, only one variation of the experiment will be used. This variation will be the best performing variation on the small, empty environment. The results will be used to judge the efficacy of the policy transfer of spatial action maps in adverse conditions.

5.1.5. Metrics: At each execution of an action in the experiment, the number of cubes in the receptacle as well as the distance traveled by the agent up to that point in time will be collected. From this data, the average number of blocks collected for each variation can be calculated. Additionally, we can see how each variation collects blocks as a function of the distance traveled by the agent.

5.2. Results

Test	Average Blocks	Distance per Block (m)
Turning and Tracking	$\mathbf{9.38 \pm 0.47}$	$\mathbf{11.7 \pm 2.42}$
Turning Only	8.00 ± 0.96	19.9 ± 3.25
Tracking Only	6.35 ± 1.00	24.4 ± 3.54
Baseline	3.6 ± 1.42	48.0 ± 11.6

Table 1: Number of objects pushed into receptacle per iteration and average distance traveled per block in the small, empty environment.

The results showed that the spatial action map policies could be efficiently transferred for the small, empty environment. As seen in Table 1, the optimized implementation for the TurtleBot swept

an average of 9.38 blocks into the receptacle in the empty environment which is approximately 0.5 blocks less than the Anki Vector implementation. These results are promising and indicate that in certain settings policy transfer is possible.

Additionally, the optimizations of the implementation had a noticeable impact. The baseline with no assumption of perfect block tracking and no pre-movement rotation was nearly inept. It pushed in less than half as many blocks as the optimized implementation and traveled over three times the distance to collect each block. Qualitatively, the baseline model would follow looping paths often circumventing the blocks that its optimized counterparts would have moved. Thus, the baseline model had inefficient movements that not only failed to collect blocks but also caused it to travel farther in each step.

The interpolated variations between the baseline and optimized model shed light on some of the limitations facing spatial action maps. The variation that assumed perfect block tracking severely under-performed its optimized counterpart by over 3 blocks per iteration. However, there is also a greater confidence interval in this variation suggesting that it could handle some randomized configurations but struggled with others. Overall, this variation was a significant improvement over the baseline variation. Observational analysis showed that the scenarios it struggles with are when blocks get collected near corners. In these scenarios, it has difficulty reaching the exact point indicated by the spatial action map, and it often pushes blocks further away.

On the other hand, the variation that had no assumption of perfect block tracking but did incorporate a pre-movement rotation in the direction of the goal performed well. Despite not having the most up to date overhead map, it managed to sweep in just over 8 blocks per iteration. Though this is still significantly worse than its optimized counterpart, it is significantly better than the perfect tracking variation. Observing the behavior of this variation showed that it struggled when many blocks were clumped together. In these scenarios, the Aruco tags became obfuscated from the camera, and the map would not update their positions as the blocks shifted to new locations. Because of this, the agent would believe that blocks were still in their original location and would often return to collect them despite their absence. The effects of this can be seen in Figure 8. Both

the optimized version and the turning only version converge to similar values as a function of distance, but the optimized version collects blocks with far less superfluous movement.

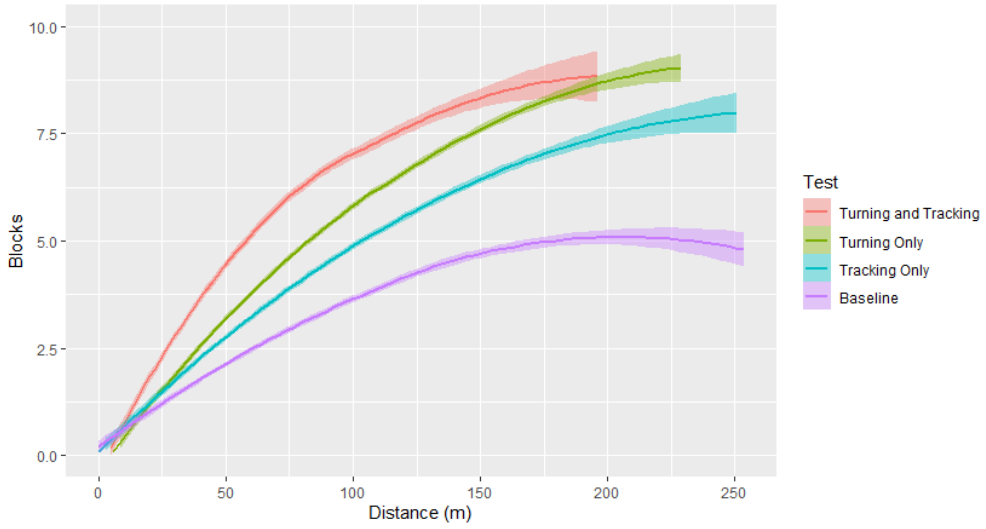


Figure 8: The number of blocks collected as a function of distance traveled by the agent in meters.

When the new idealized variation was tested on the small environment with columns, the true shortcomings of this implementation could be seen. The performance difference between the empty environment and the columned environment was a drop of nearly two blocks and an increase of about 5 meters per block. Part of this can be contributed to the pre-movement rotation optimization created for the empty environment. Rotating towards the goal in an empty environment sets it on a straight path, but in the columned environment, there could be an obstacle along that path.

When compared to the Anki Vector implementation in the original paper, we can see in Table 2 that the TurtleBot implementation can hold its own in an empty environment. However, as more complexity is added to the environment, this TurtleBot implementation struggles to match the Anki Vector implementation. Observing the TurtleBot's behavior in the small, columned environment, one can see this discrepancy once again comes down to path planning. In many iterations of the experiment, the TurtleBot will perform as good as in the empty environment. However, if the randomly placed columns are placed near each other or near walls, the TurtleBot will struggle to plan a path through the obstruction. Sometimes, this leaves blocks between spaces that are

theoretically navigable by the TurtleBot but practically hard to reach. For this reason, we see a higher confidence interval since one configuration could lead to perfect block recovery while another leads to only one or two blocks to be recovered. Similarly, this explains the larger distance per block confidence interval.

Environment	Anki Vector - Cubes	TurtleBot - Cubes	TurtleBot - Distance per Block (m)
Small, Empty	9.91 ± 0.11	9.38 ± 0.47	11.7 ± 2.42
Small, Columns	9.18 ± 0.14	7.20 ± 1.70	16.5 ± 5.28

Table 2: Comparison of the average blocks collected by the new TurtleBot implementation vs the original Anki Vector implementation for the small, empty environment and the small environment with columns.

6. Summary

6.1. Conclusion

In conclusion, the experiments conducted in this paper show a promising future for spatial action maps. In simple settings, the TurtleBot3 can efficiently collect blocks into a receptacle following the same policies trained on Anki Vector robots. This suggests that spatial action maps are suitable for robot-to-robot policy transfer. Of course, there are mitigating factors. As can be seen in the variations tested on the small, empty environment, mimicking the movement patterns of the original agent that the spatial action map neural network was trained on is essential. In the experiments here, performance almost exactly doubled with the addition of movement mimicking. This performance increase from baseline to movement mimicking is true for both the average number of blocks collected per iteration (3.6 to 7.2) as well as in the distance traveled per block (48.0m to 23.9m).

Moreover, the experiments conducted here help to alleviate certain limitation concerns outlined in the original spatial action map paper. Firstly, as mentioned above, the fact that spatial action maps only make use of high-level motion primitives is devastating for performance as seen in the baseline. However, these can be mitigated by small measures to ensure consistency across the lower-level controls. Secondly, the limitation of perfect localization and mapping was shown to be neither negligible nor particularly devastating for the small, empty environment. Imperfect mapping

showed a drop off in performance of roughly 1.3 blocks per iteration and about 8 meters per block recovered. However, this implementation generally trended towards the same result as the optimized implementation though at a slower rate. Additionally, this limitation proved less consequential than discrepancies in how agents executed actions.

6.2. Limitations

Although the results were promising, the implementation is still severely hampered by its limitations. Without further testing, no definitive conclusions can be drawn about the efficacy of robot-to-robot transfer of spatial action maps nor about the efficacy of spatial action maps given more severe sensor uncertainty. Here I have laid out a few of the key limitations that need to be addressed.

6.2.1. Limitation of Setting: As in the original paper, a key limitation is still that experiments were conducted in simulation. Despite qualities such as friction, lighting, and inertia being customizable in Gazebo, this still does not match the robustness of a real setup. Beyond physical parameters, mapping and navigation are also more uncertain in physical setups. Although uncertainty can be approximated, Gazebo provides too clean of an environment. For instance, the odometer values published in a Gazebo simulation are usually accurate to within centimeters. This is not the case in a physical setup. Thus, the simulation environment is a key limitation in being able to draw conclusions about the true efficacy of spatial action maps.

6.2.2. Limitation of Navigation: Although my implementation introduced a simple pre-movement rotation towards the goal which improved performance, the overall performance of the system is still limited by movement patterns and navigation. This limitation will only be compounded in more complex environments. This is because the TurtleBot's planned path will deviate more and more from the path planned by the Anki Vector robot as more complex trajectories are required. Due to the environments used in these experiments, this limitation was mitigated. However, navigation will become very important when conducting additional tests on more complex environments.

Related to this is the limitation that came from reducing the inflation rate of the TurtleBot's cost maps as well as the size of its footprint. Doing so allowed the agent to navigate more freely

and not get stuck, but it came at the price of permitting some collisions with walls. In a physical implementation of spatial action maps on TurtleBots, this could be a serious limitation since scraping against walls could result in damage to the robot.

6.3. Future Work

6.3.1. New Environments: A simple area of future work would be developing more environments for testing the spatial action map policies. As it stands, this project only replicated two of the four environments from the original paper. Both the large environment with columns and with a divider would provide a unique set of conditions for testing without much additional work. Additionally, one could vary conditions of the environment to test how low-light, different frictions, and different obstacles could impact performance.

6.3.2. Path Planning and Execution: Another key area for future work would be customizing the path planning capabilities of the TurtleBot. As seen in these experiments, even small changes to how the agent moves can have a huge impact on performance. Developing a spatial action map specific path planner that chooses trajectories similar to the Anki Vector robots in the original implementation could have a significant impact on performance especially in more complex environments.

6.3.3. Physical Implementation: Without a doubt, the next key step for future work would be to bring spatial action maps to TurtleBots in a physical setting. Doing so would answer many questions surrounding sensor uncertainty and mapping. Furthermore, it would test the applicability of spatial action maps in real scenarios and settings and provide feedback on what needs improved before deployment.

6.4. Reflections

Although I hope to contribute something to the research of spatial action maps, I cannot help but feel that this project is an endeavor that taught me. From taking my first robotics class in the fall to learning how to use ROS on my own, I feel like this thesis has been an uphill battle from the start. Certain decision made here were certainly not optimal, and I have no doubt that the future applications of spatial action maps for TurtleBots will outperform my implementation. Those more

knowledgeable and more skilled than me in ROS and robotics generally should certainly take the torch and eclipse the results I laid out here.

However, I do not believe that my results are without merit. My knowledge base and skill level should put the results into perspective. Across the environments I tested, the TurtleBot with untuned policies and limited adjustments still greatly outperformed the steering command baseline from the original paper. Thus, the future is bright for the application of spatial action maps in new agents.

Overall, I am pleased with the findings in this thesis and proud of the contributions I have made. Moreover, I am proud of how much I have learned from the process.

7. Acknowledgements

First, I want to thank Dr. Szymon Rusinkiewicz for advising me as well as his help in developing my future. Second, I would like to thank Dr. Anirudha Majumdar for being willing to be my second reader and teaching me my first robotics course. Finally, I would like to thank those who helped me along the way especially Jimmy Wu for answering my questions.

References

[1] T. Chen, A. Murali, and A. Gupta, "Hardware conditioned policies for multi-robot transfer learning," in *Advances in Neural Information Processing Systems*, S. Bengio *et al.*, Eds., vol. 31. Curran Associates, Inc., 2018. Available: https://proceedings.neurips.cc/paper/2018/file/b8cfbf77a3d250a4523ba67a65a7d031-Paper.pdf

[2] X. Chen *et al.*, "Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments," 2018.

[3] I. Clavera, D. Held, and P. Abbeel, "Policy transfer via modularity and reward guiding," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1537–1544.

[4] C. Devin *et al.*, "Learning modular neural network policies for multi-task and multi-robot transfer," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169–2176, 2017.

[5] S. Gao and N. Bezzo, "A conformal mapping-based framework for robot-to-robot and sim-to-real transfer learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1289–1295.

[6] W. Gao *et al.*, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," 2017.

[7] M. K. Helwa and A. P. Schoellig, "Multi-robot transfer learning: A dynamical system perspective," 2017.

[8] X. Liu, D. Pathak, and K. M. Kitani, "Revolver: Continuous evolutionary models for robot-to-robot policy transfer," 2022.

[9] M. Müller *et al.*, "Driving policy transfer via modularity and abstraction," 2018.

[10] Z. Wang and N. Papanikolopoulos, "Spatial action maps augmented with visit frequency maps for exploration tasks," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3175–3181.

[11] J. Wu *et al.*, "Spatial action maps for mobile manipulation," in *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

[12] J. Wu *et al.*, "Spatial intention maps for multi-agent mobile manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[13] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.