Generalizability and Robustness of a Sensor-State Based Reinforcement Learning Strategy for Robot Navigation

Erik Contreras, Russell Hawkins, Danny Yeap
Department of Mechanical and Aerospace Engineering
University of California, Davis
Davis, CA, 95616

Abstract

To design an intelligent system that can maneuver in various warehouses, robots must be equipped with state-of-the-art sensors and algorithms to safety navigate around obstacles and people walking around. In this paper we will consider a robot using radar, lidar, and GPS to detect incoming obstacles and prior knowledge of the dimensions of the warehouse to navigate from a starting position to an end position. This paper will study reinforcement learning specifically sensor state based Q learning and sparse Q learning to evaluate how well each method can navigate a robot safely in different warehouses. Beyond this we will investigate how generalizable our strategy is, that is, how able it is to usefully translate experience from one environment to another, as well as its robustness to sensor failure and the presence of dynamic obstacles.

I. Introduction

Automation and robotics have been making a larger impact in warehouse environments, such as the Kiva Systems in the Amazon warehouses and the Quicktron Robotic systems in the large warehouses found in Huizhou, China. With many of these robots needing to be given pre-programmed tasks and/or designated pathways to efficiently perform their jobs, it would be ideal to use reinforcement learning to reduce the amount of setup time and effort to create an automated warehouse. The ultimate goal would be to create a robot capable of making deliveries out if the box without the help of technical staff or modifications to its environment using reinforcement learning.

We will focus on developing a reinforcement learning approach using Q-learning algorithm for delivery tasks in a warehouse environment. Many studies on Q-learning utilize the absolute location of the robot in warehouse for the states which can suggest overfitting for the particular warehouse it learned from. In other words, when parameters such as the warhouse changes the Q-learning process must be redone on the new warehouse to find a solution. This paper aims to use sensor values for the states to provide a Q-learning and sparse Q-learning method so that it can be used on different variation of warehouses. Normally Q-learning uses a tabular format to store states which is unrealistic to deploy onto a robot that has limited memory and computation power. A sparse representation of Q learning is a method where only distinct states will be stored to save on memory on the robot thus saving on memory and performance of the robot.

To examine these different methods in reinforcement learning, a simulated warehouse environment will be created in MATLAB. This warehouse environment will be modeled as a two-dimensional grid, that represents a top-down view of the warehouse floor with nodes representing delivery locations, and obstacles (both static and dynamic). With this simulation, this paper will show how a sensor based Q-learning will allow a robot to safely navigate around a warehouse from one starting point to the end point.

II. Problem Statement

In this investigation we sought to evaluate the viability of a sensor-based Q-learning strategy for navigating a robot through an obstacle laden environment. We will compare the performance of standard Q-learning to that of the variant Sparse Q-learning. Additionally, we will test the generalizability of our strategy, generalizability defined as the ability of our strategy to benefit from experience in past environments in novel environments, as well as the robustness of our strategy to sensor failure and the presence of dynamic, random obstacles in the environment.

III. Past Work

Reinforcement learning has been used to develop intelligent warehouses and control autonomous robots to help navigate in a dynamic environment. One paper [1] uses multiple robots in a warehouse specifically it uses genetic scheduling to interface between multiple robots and reinforcement learning on single robots. It used Q-Learning for path planning which uses a trial and error method to develop a policy that will allow the robot to effectively navigate the 50x28 grid warehouse. It compared the results found using reinforcement learning to A*, a common path planning algorithm, and found that both algorithms perform evenly. The limitation with A* is that the warehouse environment must be static while reinforcement learning can find a solution in a dynamic environment.

In another research paper [2], the authors used a novel incremental learning scheme for reinforcement learning to adjust the reward function for a dynamic environment. The purpose of the paper was to see if having a reward function that can change will find a solution quicker. The paper used 12 warehouse robots called Kiva which is used in Amazon warehouses with shelves as obstacles. The paths the Kiva robots followed were pre-planned and must pick up packages from one location to another. The dynamic environment portion is the environment drift where in reality the position of the shelves and pick stations may change over time. After running the experiment it found that incremental learning can significantly increase the time for the reinforcement learning algorithm to find a solution.

Reinforcement learning can be applied quite generally to navigation for any sort of robot in an unknown, dynamic environment. In [3], the authors consider a search and rescue mission for a UAV. The goal is for the UAV to take the shortest path between its starting point and a stationary target (the missing person), navigating around obstacles that are in its way. The environment is modelled as a discrete grid, with each point in the grid representing free space, part of an obstacle, or the desired target. The UAV is taken to possess a radar, allowing it to determine its distance from the target, and laser range sensors, allowing it to determine its distance from obstacles. Importantly, it does not have access to a complete map of the obstacles in its environment, but must learn how to navigate the obstacles over many trials. The reward function is designed to give a large punishment for hitting an obstacle, a large reward for finding the target, and a mild punishment for all other states.

The study compared the performance, in terms of number of learning episodes required to find the optimal path, of two learning procedures. The first was standard Q-learning, as described above, with the construction of a complete, exhaustive table of Q values for each state/action pair. The second method was a refinement of Q-learning that uses a technique called Fixed Sparse Representation approximation to map the Q table to a parameter vector θ. The mapping is defined as follows:

$$Q_k(s_k, a_k) = \sum_{l=1}^{n} \phi_l(s_k, l_k)\, \theta_l .$$

This is an approximation that significantly reduces the number of parameters necessary. The authors found a substantial improvement on the amount of time required for training using this method. The learning scheme implemented in this paper was used as the starting point of our study.

IV. Simulation

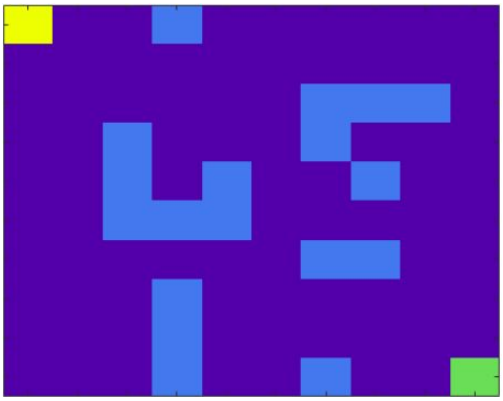


Figure 1: The simulated environment. The yellow square represents the robot, the green square the goal position, and the light blue squares are obstacles. This is the exact obstacle distribution used for most of the analysis.

To ensure that our navigation problem was tractable, we made our robotic simulation as simple as possible without being trivial. Our environment consists of a discrete grid, with each space being either an obstacle, a free space, or the desired goal. The simulation treats the edges of the environment as obstacles, ensuring the robot stays within bounds. The environment is represented in Figure 1 above.

Generally the obstacles in the environment are static, however in a few of our simulations we introduced a dynamic obstacle. This dynamic obstacle occupies one space, and at each time step it would move one space in a random direction. This was intended to model a human ambling about in the robot's environment.

The robot itself occupies one space and can at each time step can move one space up, down, left, or right. Diagonal movements are not allowed. The robot is equipped with two types of sensors. One sensor, which can be thought of as a radar ranging system, indicates the robot's distance from the goal. Importantly, this sensor does not give a sense of direction, only a distance. The other type of sensor is an obstacle detector. The robot has 4 of these, one for each direction. Each one indicates the distance to the nearest obstacle in that direction, just as a LIDAR or ultrasonic rangefinder might.

V. Algorithm

We implemented a reinforcement learning technique called Q-learning for our simulation. In Q-learning, the robot is assumed to exist in a finite number of states *S,* and is assumed to be capable of taking a finite number of discrete actions *A*. We construct a *Q-table*, which stores a number for each state/action pair that reflects the "goodness" of that action. By choosing the action with the largest Q-value in a given state, the robot will navigate in a way that maximizes reward, if the Q-table has been learned properly [11]. In our simulation, the Q-table was learned via the Bellman update equation,

$$Q_{k+1}(s_k, a_k) = (1 - \alpha)\, Q_k + \alpha\,(r_{k+1} + \gamma \cdot max\, Q_k(s_k, a'))$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are the learning rate and discount factor respectively, and r_{k+1} is the reward associated with taking the action a_k in state s_k. Details of learning and discount factors will be given in a later section.

Reward is assigned as follows. If the robot collides with an obstacle or the environment boundary, it receives a reward of -10 points, if it reaches the goal, it receives 100 points, and otherwise, if it simply moves to a free space, it receives -1 points. Also important to note is that if the robot chooses an action that results in collision with an obstacle, that action is not taken and the robot remains at its current location for the next time step.

Actions are selected by an $\boldsymbol{\epsilon}$-greedy algorithm. This means that rather than automatically selecting the action with the highest Q-value, instead at each step there is a small probability $\boldsymbol{\epsilon}$ that a random action will be taken, regardless of its value. Having this encourages the algorithm to "explore", giving it a chance to escape local maxima of reward. It also serves to prevent the algorithm from getting stuck in loops.

Perhaps the most central component of our model is the definition of the states of the robot. We defined the state of our robot as being equivalent to the state of its sensors. This requires that the state of the sensors be discrete. For the distance sensor, we discretized the distance to be one of 5 values. For proximity sensors, they could each take a value of 0,

1, or 2, corresponding to an obstacle being immediately adjacent to the robot, 1 space away, or 2 or more spaces away respectively. With our distance sensor taking 5 possible values, and each of our 4 proximity sensors taking 3 possible values, our robot has a total of $5 \cdot 3^4 = 405$ possible states.

In addition to this Q-learning scheme, we also implemented and tested a variation know as Sparse Q-learning. Sparse Q-learning is a methodology that is similar to regular Q-learning except that the Q table is represented in a different method. In regular Q-learning, a tabular method is used to express Q value of the states and actions of robot. At the first iteration all the Q values for the state and actions are all zero. However, there is a major disadvantage to this because there is a possibility that all the states will not be used once the solution converges. This leads to excess zeroes being stored and depending on the number of states the robot's hardware may not be able to support the size of the Q table. One method to solve this issue is to only store values that are non-zero which is what a sparse Q-learning representation does in order to save on memory. This makes it more realistic to deploy onto a robot.

VI. Analysis

Parameter Selection

Three parameters were tested for their optimal performance: 1) the learning rate (**α**), and 2) the discount factor (**γ**) from the Bellman equation, as well as 3) the greedy factor (**ϵ**) from the **ϵ**-greedy algorithm. The goal was to find the best parameter values that would allow the robot to quickly and consistently find the optimal pathway on the warehouse floor.

Optimal performance was measured on: 1) the lowest amount of 'spikes' in step count after settling, 2) the lowest value of steps performed by the robot, and 3) lowest amount of variability in the paths taken. The test trials were done using a static testing environment. 2000 trails were taken for each test. Each variable was tested separately.

The learning rate (**α**) was the first parameter tested. This parameter sets the extent newly acquired information overrides old information. This parameter has a value range of 0 to 1, where 0 has the robot learning nothing (exclusively exploiting prior knowledge), and 1 has the robot considering only the most recent information (ignoring prior knowledge to explore possibilities).This behavior was reflected in our tests shown in Figure 2a. From our tests, we found the highest variability occurred when **α**=0.0, the highest spikes in step count when **α**=0.25. In the end we found that optimal performance was achieved when **α**=0.1.

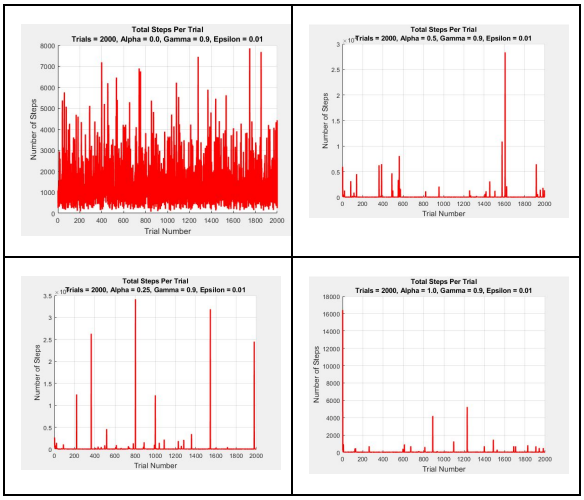


Figure 2a: Test results for **α**=0.00 (top left), **α**=0.25 (bottom left), **α**=0.5 (top right), **α**=1.00 (bottom right).

The second parameter tested was the discount factor (**γ**). This parameter sets the importance of future rewards versus immediate rewards. This parameter has a value range of 0 to 1, where 0 has the robot considering only current/nearby rewards, and 1 has the robot considering only future/distance rewards. "If the discount factor meets or exceeds 1, the action values may diverge. For $\gamma = 1$, without a terminal state, or if the agent never reaches one, all environment histories become infinitely long, and utilities with additive, undiscounted rewards generally become infinite" [12]. This behavior was reflected in our tests shown Figure 2b. From our tests, we found the highest variability occurred when **γ**=0.75, and there was a consistent amount of step spikes however the amplitude of the step count decreased as **γ** increased towards 1. In the end we found that optimal performance was achieved when **γ** = 0.9.

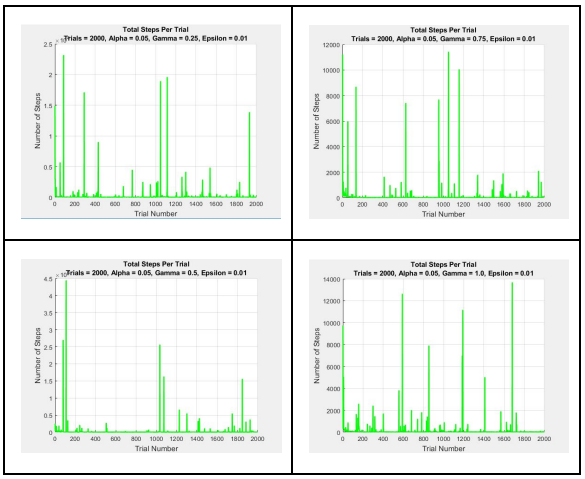
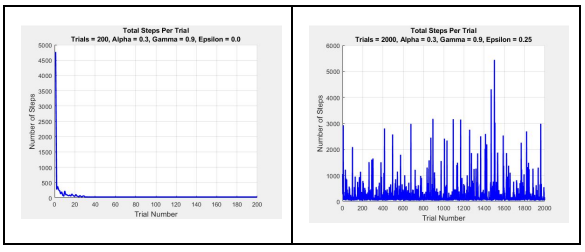


Figure 2b: Test results for γ=0.25 (top left), γ=0.50 (bottom left), γ=0.75 (top right), γ=1.00 (bottom right).

The third parameter tested was the greedy factor (ϵ). As stated in the previous section, the greedy factor sets how often the robot will takes a pathway with the highest "payout" versus a random pathway [10]. This parameter has a value range of 0 to 1, where 0 has the the robot will take pathways with the highest payout, and 1 has the robot will only take random pathways. This behavior was reflected in our tests shown in Figure 2c. It was found that optimal performance was achieved when ϵ=0.01. While the best results occured when ϵ=0.0, it posed as a potential risk for the robot to get stuck if a new obstacle was placed on the map; by setting ϵ greater than 0 the robot had a potential method of getting out of unplanned situations.



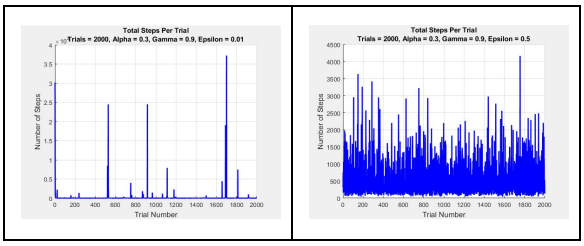


Figure 2c: Test results for ϵ=0.0 (top left), ϵ=0.0 (bottom left), ϵ=0.25 (top right), ϵ=0.50 (bottom right). Note that only 200 trials were measured for ϵ=0.0 given how quickly the pathway settled.

Standard Q-learning and Sparse Q-learning Performance

Using the optimal values selected from the previous section, ϵ=0.01, γ = 0.9 and α=0.1, we ran standard Q-learning which uses a fixed size for the Q table. All the Q values in the table are initialized to 0 at the start of the algorithm and as it iterates through the Q values are updated based on the previous and current states. The graph (Figure 3a) shows an exponential decay which is expected because we want to see a decrease in the number of steps, y axis, and as the number of trials increases. Each datapoint in the number of steps is the average over 50 runs because Q-learning can "lucky" and find a solution early. Averaging it across 50 runs will give a better idea on the performance of Q-learning instead of only observing one run. Number of steps indicates how many actions it took for the robot to reach the final destination of the warehouse. The number of steps appears to converge at around the 175th trials signifying Q-learning having successfully learned the warehouse layout and the best solution from start to end.

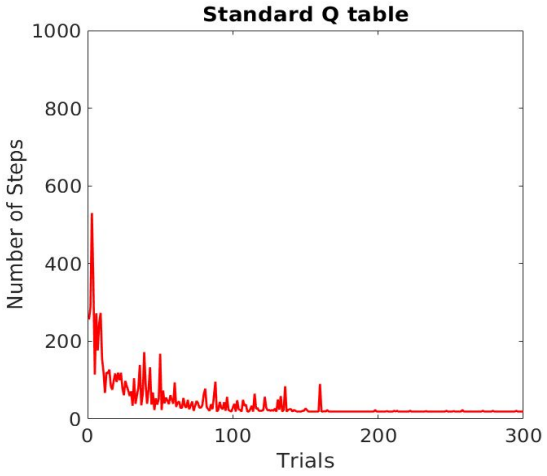


Figure 3a: Plot of standard Q-learning executed on static environment where the number of iterations (y axis) represents an average over 50 runs

Many times the size of the Q table does at the end of learning process has many states that have a Q value of 0 for each action. This is because not all states must be explored for reinforcement learning to understand the maze environment. In order to be more realistic, we used a spare Q-learning method in which only states with non-zero Q values are stored on the robot. The performance of our results are shown below.

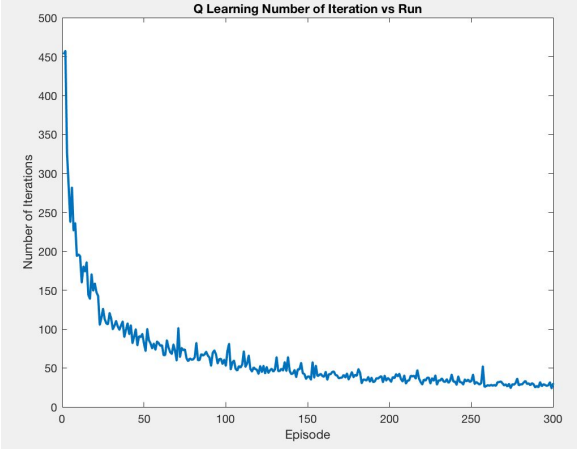


Figure 3b: Plot of Sparse Q-learning executed on static environment where the number of iterations (y axis) represents an average over 50 runs.

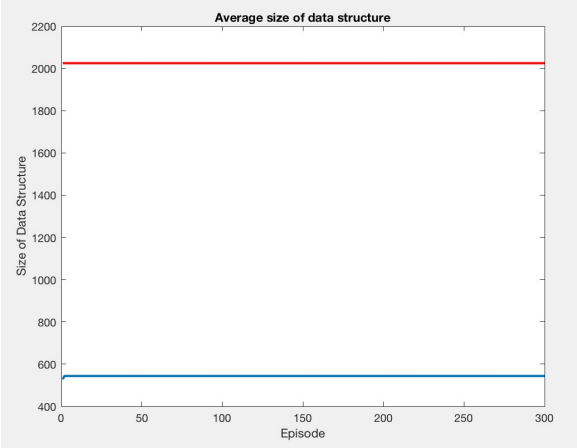


Figure 3c: Size of Q-table where red line shows the regular Q-learning table and blue line shows sparse Q-learning table

The sparse Q-learning methodology is repeated 50 times and the number of iterations for each episode is averaged (Figure 3b). The graph (Figure 3b) shows an exponential decay where the number of iterations start to convergence at around the 175th run. Observing the number of iterations converges after the 125th run, there are still small peaks where the number of iterations increases to 100. This is not an issue because overall the number of iterations is steady at about 25 iterations. This is expected because initially all the states have a Q values are zero and the robot has no knowledge of the maze. As the Q values get updated by the bellman equation the robot starts understanding which path to take to reach the end. There are some cases where the robot gets stuck and goes back and forth between two states. This was seen in earlier stages of implementing the Q-learning algorithm where the robot never stops stepping forward and stepping backward which is why having epsilon, 0.01, be non-zero help "kick it out" of an infinite loop of indecisiveness. This is also why we decided to execute 50 repeated runs of the algorithm to see how well Q-learning runs overall not just on one particular run. Observing standard Q-learning (Figure 3a) and sparse Q-learning (Figure 3b), it can be seen that both can find a solution and the algorithms are comparable, algorithm wise.

Another important analysis the amount of memory that is saved when using a sparse representation of the Q table. The size of Q-learning table is represented as the red line and the size of the sparse Q-learning is represented as the blue line (Figure 3c). Initially, it was expected that the blue line would start exponentially saturate towards the red line. This would show that Q-Learning is slowly exploring different states to find the best solution for the warehouse. However, it can be seen that all possible states are explored at the 5th iteration and saturates out at about 550. This means that after the 5th iteration, Q-learning is adjusting the Q-values to find the best solution to the maze. It also means that for this specific maze, the max amount of sensor states is 550. When regular Q-learning was run (red line), the data structure size is about 2025. If sparse Q-learning is implemented, there is approximately 73% memory saved on the robot.

Generalizability

To test how generalizable our sensor-state based Q-learning strategy is, we decided to test the performance of a robot navigating using a Q-table trained in our default environment in a different environment, one with a different

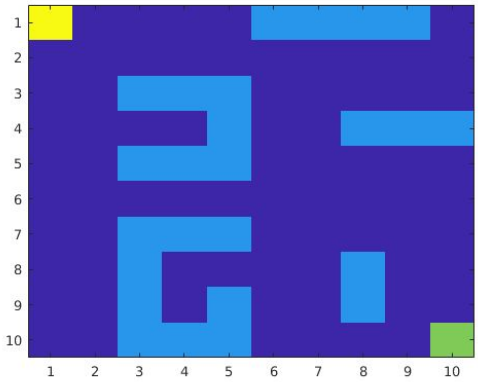


Figure 4: Alternative environment used to test generalizability

distribution of obstacles. The alternative environment is shown above in Figure 4.

At first, we tested the robot in the new environment with no additional learning, to see how an unmodified Q-table would perform in a novel environment. Unfortunately we found that the robot was generally unable to navigate to the goal. It seemed that the robot would tend to get stuck in infinite loops, taking actions back and forth between two states.

With it clear that the Q-table learned in one environment cannot on its own perform well in another, we decided to look for an advantage in learning. Perhaps with the experience of another environment, the robot in a new environment will be able to learn more rapidly? Certain things, such as not crashing into obstacles that you are immediately adjacent to, should be equally true in all environments, and hopefully this experience will translate. Below Figure 5 gives a comparison of performance.

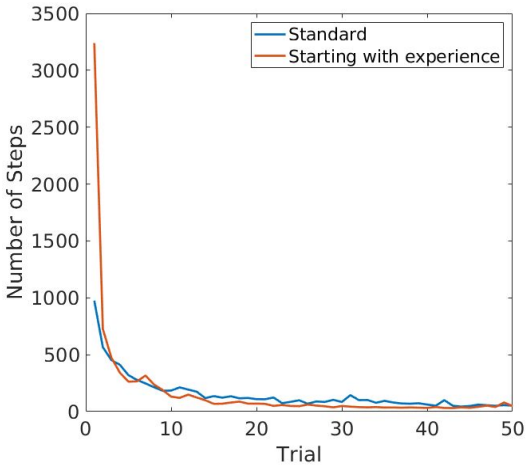


Figure 5: Number of steps required to reach goal vs. Trial number, averaged over 50 learning iterations. Blue line indicates starting from blank Q-table, red line starting with experience in default environment.

It is clear that for early trials, the number of steps required to reach the goal is much larger for the robot with experience in the default environment than the standard, which is starting from a blank, that is, all zeroes, Q-table. After the initial few trials, their performance is quite similar by this metric. It appears that prior experience in a different environment only degrades performance.

After seeing this result, we thought we would look at a different performance metric, the total accumulated reward throughout each trial. Perhaps it was the case that experience was not enhancing the rate at which the robot was reaching the goal, but was lowering the rate at which the robot crashed into obstacles, which would be evident in the total accumulated reward. Below Figure 6 show a comparison of total accumulated reward for the robot starting with a blank Q-table and one starting with experience in a different environment.

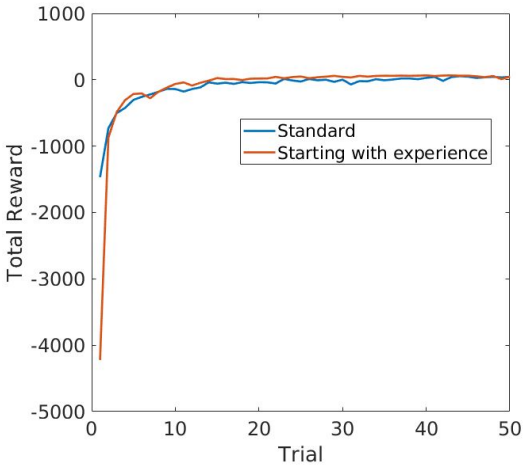


Figure 6: Total Reward vs Trial number, averaged over 50 iterations

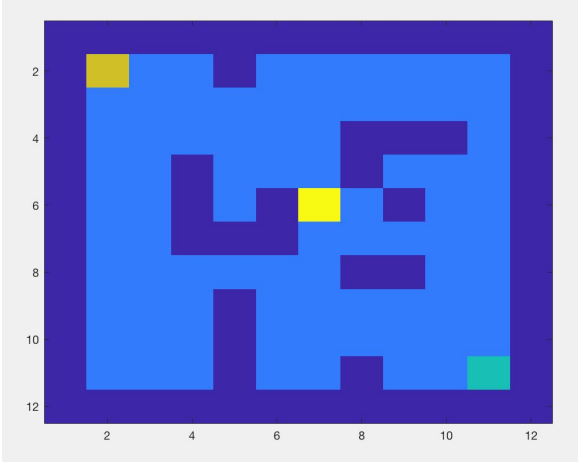


Figure 7: The simulated environment for the dynamic environment. The yellow square represents the robot, the green square the goal position, and the light blue squares are obstacles.

Here it seems the conclusion is the same. In early trials, the experienced robot accumulated much more negative reward, and after that initial period, performed comparably. From this and the previous analysis we conclude that, at least as we have tested it, experience in one environment does not enhance performance in another environment. The robot navigates better if it forgets everything it learned from its previous experience.

Dynamic Environment

To test how well sparse Q-learning performs when there is a dynamic obstacle wandering around the warehouse, a human (bright yellow) is placed in the center of the maze (Figure 7). At every iteration, the human will randomly move one direction up, down, left or right. If the human will run into an obstacle, wall or robot, it will stay in the same spot. The robot (brown) moves in a similar fashion but with Q-learning to help it reach the final destination (teal).

The results of the dynamic environment shows an exponential decay Figure (8a) and converges at around the 250th run. This is to be expected because adding a human in the warehouse requires more iteration for Q-learning to understand how the random human walking around affects warehouse dynamics. This is also why the amount of iterations in the dynamic environment (Figure 8a) is higher when compared to sparse static environment (Figure 3a). This can be attributed to the human randomly walking around and the possibility that the human can block the robot from reaching the final destination in a timely fashion. Another observation is that the there is more fluctuations and more peaks. The static environment only had a few peaks and disappears once the solution converges. The dynamic environment continues to have some occasional peaks and this is due to the possibility that the robot may prevent the robot from reaching the final destination because it is blocking a hallway and the robot must turn around and take the long way around. Interestingly, the size of the Q-table also shows an exponential growth where the first iteration starts at size 400. Previously, it was seen that if the size of the data structure, Q-table, reaches 550 then all the states of the static environment has been explored (Figure 3b). Similarly, the size of the Q-table for the dynamics environment reaches 550 at the 5th run as well. Since there is a dynamic obstacle the robot will sense and see more states compared to the static environment. It continues to increase until it reaches

the 200th run and levels out to 1850. This only says about 10% of the memory which does not save as much as the 73% for the static environment. Future work can implement a system where all states and actions are mapped to distinct Q values. Also at the 100th run, the data structure size has mostly saturated meaning that between the 100th run and the 250th run, where the solution converges, Q-learning has seen all possible states and the Q-values are being changed to find the best path to the final destination.

There is some randomness involved with Q-learning and every time you run the simulation again the results may be different. It is possible to get lucky and Q-learning may find the optimal solution quickly for that particular run. As a result, it is important to run Q-learning multiple times and average the number of iterations for each run to find out on average how long it take to complete the maze. Q-learning was run 50 times and the average number of iteration across all 50 runs show a smoother exponential decay plot (Figure 9a). The curve also converges but instead of converging at 20 iterations it converges at around 65 iterations. As a result, it more accurate to state that on average with a dynamic obstacle it takes around 65 steps to complete the maze once the solution converges. The average size of the Q-table (Figure 9b) shows the same exponential growth as only one run (Figure 8b). The memory saving is around 10% as well.

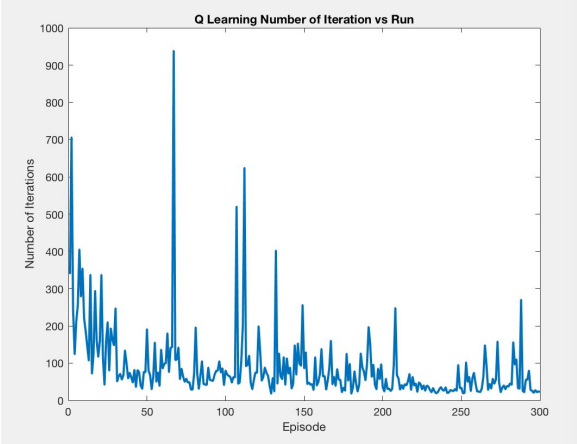


Figure 8a: Plot of Sparse Q-learning executed on dynamic environment.

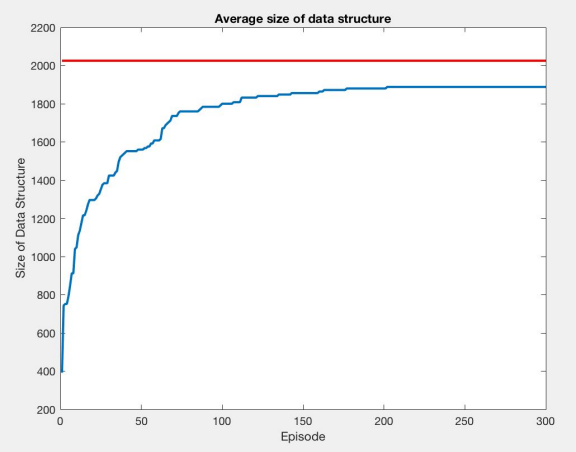


Figure 8b: Size of Q-table. Red line represents size of regular Q table. Blue line represents size of sparse Q table.

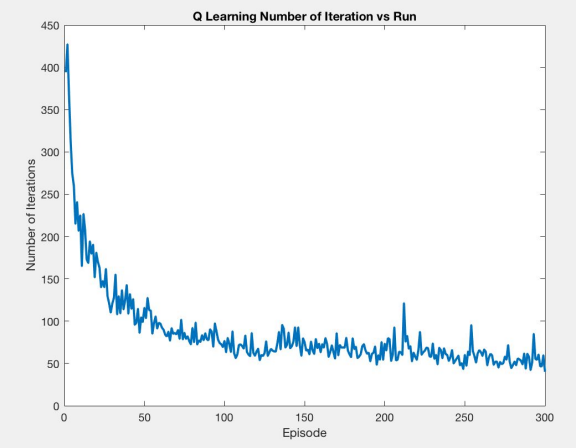


Figure 9a: Plot of Sparse Q-learning executed on dynamic environment where the number of iterations (y axis) represents an average over 50 runs.

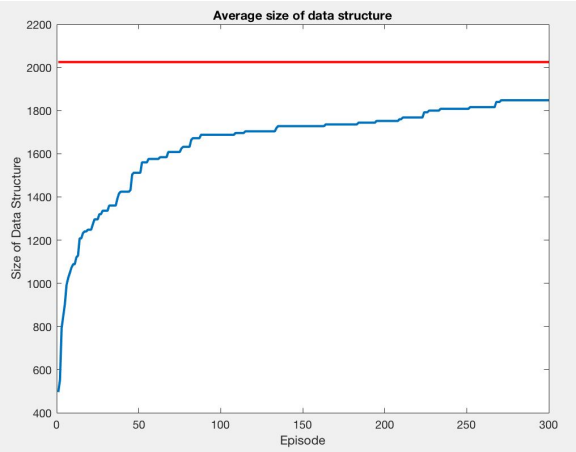


Figure 9b: Size of Q-table. Red line represents size of regular Q table. Blue line represents size of sparse Q table.

Robustness to Sensor Failure

In any real world application, internal system failure is a possibility. An ideal system would be robust to some of the sorts failures that are possible. With this in mind, we tested how robust our system is to a failure of one of its sensors.

To test this, we selected one of the proximity sensors to be faulty. We introduced a fault by having the chosen sensor return a completely random value X percent of the time, and we tested performance for X = 50%, 75%, and 95%. The results are shown in Figure 10 below.

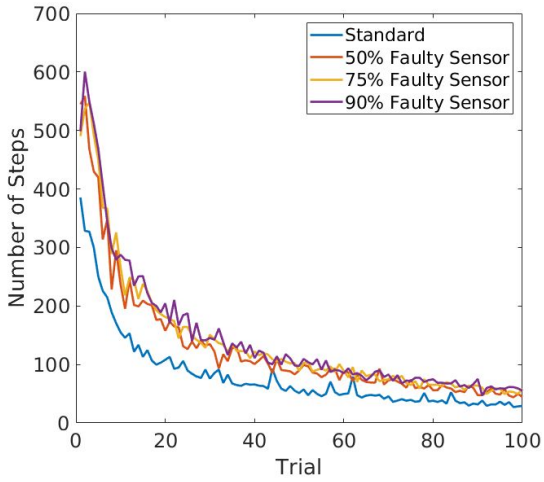


Figure 10: Number of steps to reach goal vs. Trial number for a robot with perfect sensors and 3 different degrees of faulty sensor

Here we can see that the faulty sensor negatively impacts the performance of the robot, increasing the amount of time required to learn how to navigate to the goal. However, in all the faulty cases, including the case where one of the sensors emits a random value 95% of the time, rendering it virtually useless, the robot still managed to make it to the goal eventually. After 100 trials, the performance is within 20% or so. In Figure 12 below, a similar comparison is done but for total reward.

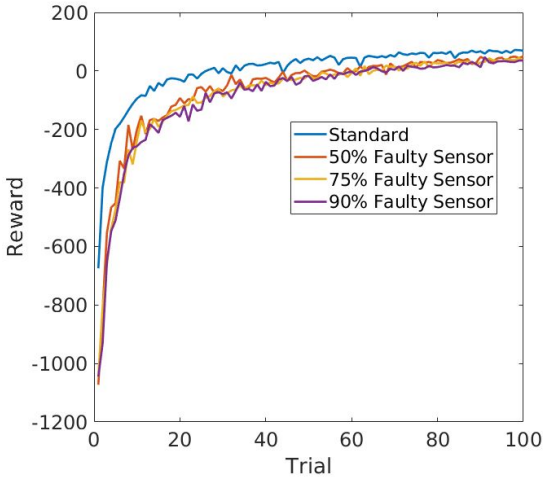


Figure 11: Total reward vs Trial number for a robot with perfect sensors and 3 different degrees of faulty sensor

The conclusion here is the same: even with one faulty sensor, the robot still manages to accumulate reward comparable to that of the undamaged robot. From this analysis we can conclude that our robot navigation strategy is robust to at least single sensor failure.

VII. Conclusion

Overall we can conclude that the strategies that we've explored, Q-learning and Sparse Q-learning, with robotic states defined in terms of a limited set of sensors, are suitable for the task of navigating a robot through an obstacle filled environment. After varying the learning parameters to optimize performance, we compared the Q-learning and Sparse Q-learning methods, and found their learning performances comparable, with Sparse Q-learning having a substantial memory advantage. We then tested the generalizability of our model, which was the main motivating factor for its sensor-state based design. Unfortunately, it seems this strategy is unable to leverage previous experience in novel environments. Finally, we explored the robustness of our model, by testing its performance in an environment containing a random, dynamic obstacle, and testing its performance in a situation where one of its sensors was failing. In both cases, the performance was negatively affected, but still managed to function. This indicates that our strategy is fairly robust. This paper shows that reinforcement learning can be applied the robotics

field in artificial intelligence to find appropriate solutions for robotics path planning.

VIII. Future Directions

This work leaves us with many opportunities for future exploration. Each of the tests we did would benefit from further exploration. Particularly, the question of generalizability could use more investigation. Testing more environments, and perhaps using more refined statistics, might be able to tease out some positive effect of experience when learning a new environment. We feel that given the construction of this strategy, there must be some benefit to experience, but that it is too subtle effect for our current analysis to detect. Detecting this would not substantially change our conclusion, as a positive benefit that subtle is not terribly useful.

Generally, exploring the effect of the details of the geometry of environments would be interesting. There is a huge space of possible environments, and it would be interesting to characterize and classify some different categories of environment, and see how the algorithm performance is affected by these.

There are also plenty of possible extensions, things to do that are entirely beyond the scope of what we have done. One potential direction for would the implementation of additional robots working together to complete tasks on the warehouse floor and how to automate the path-planning process, perhaps using an auction algorithm to distribute tasks between the group of robots. Another extension could be to add more sensor states to reduce the learning time of Q-learning because there are some states where Q-learning take a long time to learn and this can be improved by adding a sensor to get out of these situations more efficiently instead of relying on epsilon or randomness.

References

[1] J. Dou, C. Chen, and P. Yang, "Genetic Scheduling and Reinforcement Learning in Multirobot Systems for Intelligent Warehouses," *Mathematical Problems in Engineering*, vol. 2015, pp. 1–10, 2015.

[2] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, "A novel incremental learning scheme for reinforcement learning in dynamic environments," in 2016 12th World Congress on Intelligent Control and Automation (WCICA), 2016.

[3] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen, "Reinforcement Learning for Autonomous UAV Navigation Using Function Approximation", in *Safety, Security, and Rescue Robotics (SSRR), 2018 IEEE International Symposium on.* IEEE, 2018.

[4] N. Imanberdiyev, C. Fu, E. Kayacan, and I. Chen, "Autonomous Navigation of UAV by Using Real-Time Model-Based Reinforcement Learning", in *Control, Automation, Robotics, and Vision (ICARCV), 2016 14th International Conference on.* IEEE, 2016, pg. 1-6.

[5] N. Pinkam*, F. Bonnet, and N.Y. Chong, "Robot Collaboration in Warehouse" in 2016 *16th International Conference on Control, Automation and Systems.* IEEE, 2016, pg. 269-272

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

[7] L. Mohan and J. Ignatious, "Navigation of Mobile Robot in a Warehouse Environment". *2018 International Conference on Emerging Trends and Innovations in Engineering and Technological Research (ICETIER),* pg. 1-5

[8] H. Ma, Y. Gong, J. Song, X. Wu, Y. Cui, "A Task-Grouped Approach for the Multi-Robot Task Allocation of Warehouse System". *2015 International Conference on Computer Science and Mechanical Automation.* Pg. 277-280

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

[10] D. McCaffrey, "The Epsilon-Greedy Algorithm," November 30, 2017 [accessed March, 1

2019].
https://jamesmccaffrey.wordpress.com/2017/11/30/the-epsilon-greedy-algorithm/ .
[11] Sutton, R., & Barto, A. (). Reinforcement learning: An introduction. Cambridge, MA: MIT Press.

[12] 2. Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (Third ed.). Prentice Hall. p. 649. ISBN 978-0136042594