Learning to Save Lives: Using Reinforcement Learning with Environment Features for Efficient Robot Search and Rescue

Rowan Border

University of Oxford
rborder@robots.ox.ac.uk

Abstract

This project proposes a reinforcement learning approach to robot search and rescue (SAR). The approach uses Q-Learning with a neural network representation to learn collision avoidance, exploration and victim discovery behaviours. Visual features in the environment are used to learn correlations between detected features and the likelihood of victim discovery. The system was built using the ROS framework and designed for the TurtleBot mobile platform. The performance of the approach was evaluated in a simulation environment and compared to the performance of a random walk.

I. Introduction

I. Search and Rescue

The success of many search and rescue (SAR) operations could be greatly improved with the effective use of autonomous robotic agents. Robotic agents have many advantages over their human counterparts. They can explore areas deemed unsafe for humans, analyse and communicate vast amounts of sensory input and perform feats of strength and precision impossible for a human. However, robots have historically only been able to perform reliably in highly controlled, static environments. In contrast, SAR scenarios are usually in dynamic environments where communication is limited and information is sparse.

A number of approaches have been proposed to overcome the difficulties of SAR operations with autonomous robotic agents. Many of these involve the use of multiple agents cooperating to explore an environment, either by providing reliable communication via a relay network [6] or by sharing information and distributing tasks to jointly explore the environment [12] [13] [2]. Some approaches use a role-based structure to allocate tasks either on a state-wise basis (e.g. amount of resources available, proximity) [5] or explicitly by using heterogeneous agents (e.g. a UAV coordinating multiple UGVs) [8] [3] [9]. Other approaches use reinforcement learning methods, allowing the agents to learn how best to explore the dynamic environment.

The most important evaluation criteria for SAR performance is the time taken to explore the environment and identify potential victims. An ideal robot SAR system would be able to search an environment in the shortest time possible and accurately identify potential victims. The effectiveness of a robot SAR system is dependant on the exploration strategy used. The most common strategy used is frontier exploration [15]. This strategy discretizes the boundary between explored and unexplored regions into a set of 'frontiers'. A frontier is chosen to be explored and the set of frontiers is updated accordingly, until the frontier set is empty. This is an efficient and exhaustive exploration strategy that can be scaled to multiple agents [1].

However, environment exploration is not the only goal in a SAR scenario and exhaustive exploration may not be necessary. For example in a SAR scenario with a known number of victims a targeted exploration strategy that learns where victims are most likely to be found would reduce the search time required. A frontier exploration approach would not be ideal in this case as it would not be able to apply a targeted strategy without a learn-

ing system. This project proposes a reinforcement learning approach to robot SAR using Q-Learning. The learning system is designed to be used by an agent to jointly learn collision avoidance, exploration and victim discovery behaviours in order to develop the targeted exploration strategy described.

II. Reinforcement Learning

Reinforcement learning (RL) provides a framework for learning the state-action mappings in an environment which produce the highest cumulative reward. It has proven useful for developing computer game AIs, swarm intelligence and robotic control. A particularly successful application was the use reinforcement learning by DeepMind to teach an AI to play Atari games [11]. This success is relevant to the application of RL proposed in this project as the DeepMind AI had to learn to explore a game world and collect rewards for interacting with in-game targets, similar to the exploration and victim discovery behaviours proposed. In the field of robotic control, RL approaches to path planning and exploration have been proposed but the problem of jointly learning exploration and target discovery has not been investigated.

The RL approach used in this project was Q-Learning. Q-Learning is a RL algorithm which iteratively updates the values associated with state-action pairs (called q-values) in order to learn an optimal action selection policy. It has been used in approaches where an agent learns to navigate a collision free path to a goal in an environment with static obstacles [10]. The state-action mapping for Q-Learning is usually discrete and uses a matrix representation. However, it is possible to use a continuous state-action mapping with Q-Learning by implementing a neural network representation [14]. The specifics of Q-Learning and its two representations is discussed later. Both representations were evaluated for this project and the neural network representation was used for the final implementation as it allowed for the use of continuous sensory inputs.

III. Target Discovery

It has recently become possible to develop object recognition algorithms with unprecedented accuracies - in many cases approaching human level performance. This has largely been thanks to work done in the field of deep learning, which use neural networks with many interconnected layers that can be trained to recognise objects based on visual descriptors. Object recognition in cluttered and dynamic environments (such as SAR scenarios) was previously not reliable enough to be used in SAR systems. However, the ability to identify features in an environment can provide useful information for developing a targeted exploration strategy. Feature recognition in a SAR scenario could be used to classify regions based on topology (e.g. to determine accessibility for different agent types), risk (e.g. analysing structural integrity) or to determine the likelihood of victim discovery given the presence of certain features. For example by correlating visual features with victim discovery it would be possible for an agent to learn that the presence of blood is indicative of a victim, even if the victim themselves cannot be seen. Approaches to robot SAR have been presented which use visual features to target exploration [13] or classify regions based on their accessibility [7]. In [13] visual features are used to generate a prior probability distribution over a search grid. In [7] visual features are used to classify piles of rubble based on their drivability. This project proposes to use feature identification together with RL to learn a search strategy which targets regions with the highest likelihood of victim discovery.

II. Methodology

I. Q-Learning

Q-Learning is a reinforcement learning algorithm that can be used to learn an optimal policy for the state-action behaviours of an agent in a given environment. The state representation is a set of input variables describing the current state of the agent and environment. The action representation is a set of possible actions available to the agent. The joint set between states and actions are the state-action

pairs, each of which has an associated q-value. Each q-value represents the value of choosing an action given the state. At each time-step an action is selected given the current state and the state representation is updated according. The most common method of action selection is soft-max selection using a Boltzmann distribution (1).

$$e^{Q(s,a)/\tau} / \sum_{a} e^{Q(s,a)/\tau} \tag{1}$$

Soft-max selection is a probabilistic method which normalises the q-values of the action set (given the current state) to generate a probability distribution over possible actions. The Boltzmann equation is then applied to transform this distribution based on the temperate parameter τ. As $\tau \to 1$ the distribution becomes more uniform and the action space is explored more. As $\tau \to 0$ the distribution remains unchanged and the action with the highest q-value is selected. During the training phase the value of τ is usually set high to explore the action space and then decremented to zero by the end of the training phase.

$$\begin{aligned} Q(s_t, a_t) = (1 - \alpha_t) Q(s_t, a_t) \\ + \alpha_t \Big[R(s_t, a_t, s_{t+1}) \\ + \gamma \max_a Q(s_{t+1}, a) \Big] \end{aligned} \tag{2}$$

When an action is selected given the current state a reward is generated based on the value of performing the selected action. This reward value is used to update the q-value of the state-action pair given the learning rate α. Q-Learning can also update the q-value using the expected reward of future states. Actions for these future states are selected based on the maximum q-values. The value of the expected reward is scaled by the γ parameter and added to the immediate reward value (2).

II. Neural Network

The most common implementation of Q-Learning uses a state-action matrix (the set of state-action pairs) of q-values and a state-action matrix of reward values. As each state is represented by a set of variables describing the state, these variables must be discrete and finite in order for the matrices to be computationally feasible. This limits the expressiveness of the state-space representation and therefore the complexity of the learned policy. It is possible to implement Q-Learning with a continuous state-space representation by replacing the state-action matrix with a feed-forward neural network (Fig. 1).

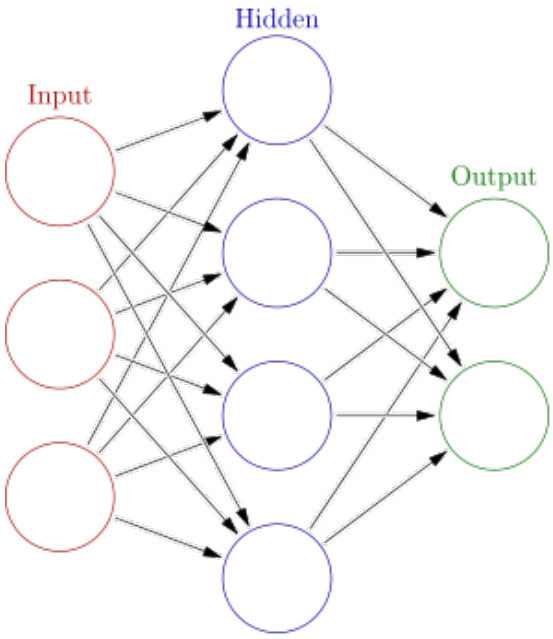


Figure 1: *Feedforward Neural Network*

The input neurons are the state variables which can have continuous values. The output neurons are the action variables which output q-values for each action given the input state values. The hidden layer contains neurons which propagate values from the input to output layers based on their activation state. The number of neurons in the hidden layer is typically the mean of the number of input and output neurons. The feed-forward network is a fully connected network and therefore every input neuron is connected to every hidden layer neuron and every hidden layer neuron is in turn connected to every output neuron. Each pathway is assigned a weight which scales values propagated along the pathway. Each hidden neuron has a sigmoid activation function whose output value is determined by the sum of the weighted inputs to the neuron.

The neural network learns an optimal action selection policy by using back-propagation

to update the weights on the connection pathways. The optimal q-value output for a given state is estimated using the q-value update equation (2). This is used as a target output state for the neural network. The mean-squared error between the current output and the target output is calculated and minimized iteratively using gradient descent.

III. Feature Learning

Given that it is now possible to train object detection algorithms to reliably recognize features even in cluttered environments, this capability can be used to developed targeted exploration strategies for robot SAR systems. This project proposes using visual features and target detection as part of a RL exploration strategy to learn correlations between environment features and the likelihood of victim discovery. The detection states of visual features and targets are input into the neural network and an output state indicates whether a victim is present. The current polar coordinates of the robot are also input states to the neural network, making it possible to learn localised correlations for victim discovery.

III. Implementation

I. Development Platform

The development platform used for this project was a TurtleBot. TurtleBot is a mobile platform with a Kinect RGB-D camera and bump sensors on the sides of the mobile base. The platform runs the Robot Operating System (ROS). ROS is an open source framework for developing robotic control systems. The framework facilitates communicate between different modules using a centralised message passing interface. Multiple modules can be used as components of a larger system.

The RL system presented in this project was developed using the ROS framework. The RL algorithm was written as a ROS node which receives state information from simulated TurtleBot sensors (e.g. the Kinect RGB-D camera and bump sensors), trains the neural network using the state input and sends velocity commands to a TurtleBot in the simulation environment based on the action selected from the output state. The simulation environment Gazebo was used to evaluate the behaviour of the RL algorithm in a physical simulation. The `cost-map_2d` package was used to maintain a cost-map, representing the unexplored and explored regions of the simulation environment, which was used to generate state variables for the neural network. The `RViz` package was used to visualise the current environment state during simulation, including the detection of feature markers.

II. Simulation Environment

The performance of the RL algorithm was tested in the Gazebo simulation environment. Using Gazebo it was possible to build a 3D world for a simulated TurtleBot to explore. Simulated sensory information for the Kinect RGB-D camera and bump sensors was generated based on the world state and communicated via ROS to the RL node. The visual Kinect data was used for detecting feature markers placed in the 3D world. The disparity map data was transformed into a laser scan representation using the `depthimage_to_laserscan` package. The laser scan and bump sensor data was used to generated state inputs for the neural network. The simulated TurtleBot was controlled by publishing velocity commands to a controller which updated the model state in Gazebo (and RViz).

III. Learning Framework

The RL algorithm used for this project was an implementation of Q-Learning. Both state-action matrix and neural network implementations were considered.

The first algorithm implemented used the state-action matrix representation for learning collision avoidance. The state-space for the algorithm was the state of the bump sensors on the left, front and right of robot and three state variables generated from the laser scan data. The laser scan data was divided into three sectors (left, front and right) and the average scan distance for each sector was calculated. As the state-action matrix representation requires discrete state variables, the average distance

values were transformed into a binary representation based on a threshold parameter. If the distance value was less than the threshold (2.5m) the proximity state variable was set to true. The action-space of algorithm was turn left $\pi/2$, move forward 0.5m and turn right $\pi/2$. The reward structure was chosen to negatively reward action choices which resulted in a collision state (activating a bump sensor) or a state approaching collision (a move forward action in a state where the proximity variable for one of the sectors was true). The robot was otherwise positively reward for moving forward in order to prevent the Q-Learning from converging to a policy of continuous rotation. The state-matrix representation was sufficiently expressive for the agent to learn collision avoidance, however in order to extend the algorithm to include exploration and victim discovery goals it was necessary to transition to the neural network representation, allowing for the use of continuous state variables.

The neural network RL algorithm implementation was the primary focus of this project. Using this implementation the agent was able to learn collision avoidance, exploration and victim discovery behaviours using a common state representation of the environment. The state-space consisted of bump sensor states, laser scan distances, an exploration grid surrounding the robot, the polar coordinates of the robot in the world, exploration state variables and feature marker states. In order to reduce the computational complexity of the network, the laser scan array was divided into 11 sectors and the average distances for the sectors were used as the state variables. The action-space was move forward 0.5m, the set of turn angles $\{-\pi/3, -\pi/6, -\pi/12, \pi/12, \pi/6, \pi/3\}$, and an output state for victim discovery. Multiple turn angles were used to allow learning of different turning behaviours (e.g. a large turn to explore an area vs. a smaller turn to avoid a wall). To ensure uniform exploration of the action space between moving forward and the two sets of turn directions, the moving forward action was given a weighting of three.

The structure of the neural network was generated using an open source C++ library [4]. The network used was fully connected with a single hidden layer and had 44 input states, 8 output states and 26 hidden nodes (the mean of the number of input and output states) (Fig. 2). At each time-step the input state would be updated by retrieving the latest state information from the simulation environment. The feed-forward network would then calculate the resulting output state q-values and an action would be selected using the Boltzmann distribution. The velocity command associated with the selected action would be sent to the simulation and the state of the world was updated. At the next time-step, given the updated state of the world, the optimal q-value output of the previous state was calculated using the q-value update function (2) and used to train the network weights with back-propagation. The reward value used in the q-value update function was the sum of the reward values for the collision avoidance, exploration and victim discovery behaviours. The reward structure for the collision avoidance behaviour is the same as described for the state-matrix representation, except using continuous values. The reward structures for the exploration and victim discovery behaviours are detailed in later sections.

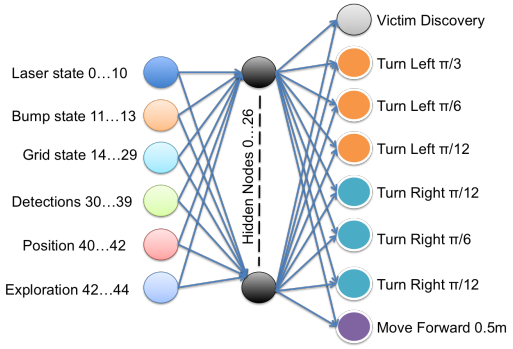


Figure 2: *RL Neural Network*

IV. Exploration Cost-map

The purpose of the exploration cost-map is to store the current exploration state of the world (i.e. which cells have been explored). In ROS this functionality is provided by the `costmap_2d` package. The costmap is initialised to an empty map of the world with all cells marked as unexplored. As the robot moves around the simulation world, the sensor data generated (e.g from the laser scan and bump sensors) is used to update the state of

the costmap and mark cells in the map as free space or obstacles. The costmap is used by the RL algorithm to generated an exploration grid surrounding the robot and the cell values are input states for the neural network. The exploration grid used is 5x5 cells and the size of the grid is 10x10m. The grid size is based on the range of the Kinect laser which is 5m. The robot is positioned in the center of the middle cell. The inner square of 8 cells surrounding the robot is the exploring grid and the outer square of 16 cells is the sensing grid (Fig. 3). The number of unexplored points in each cell is calculated and these values are input states to the neural network. Two other input states calculated from the exploration grid are the decrease in unexplored cells in the explored grid and the increase in unexplored cells in the sensing grid. When training the neural network, rewards for exploration are assigned proportional to the decrease in unexplored cells in the exploring grid and the increase in unexplored cells in the sensing grid. The principal behind this reward structure is that cells in the exploring grid are within range of the Kinect laser and can be explored. For the sensing grid, an increase in the number of unexplored cells indicates the robot is moving into a new area of the map.

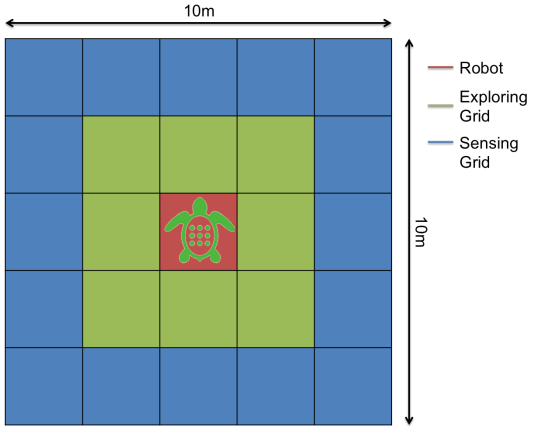


Figure 3: *Exploration Grid*

V. Feature Markers

Given that the focus of this project was reinforcement learning and not object recognition, the vision problem was simplified by using AR tags instead of complex visual features. The AR tags could be placed in the simulation world and reliably tracked using the `ar_track_alvar` ROS package. This package can provide reliable pose estimates for multiple detected tags and their associated tag IDs. The detection states for the tags were used as input states to the neural network. In order to simulate the problem of detecting a target given visual features in the environment, the AR tags were used in $(feature, target)$ pairs (Fig. 4). Certain tag pairs were assigned to be valid for victim discovery. If a valid set of detection states was input to the neural network, the value of the victim discovery output state should be greater than 0.5. Therefore when training the network if a valid set of states existed then a reward would be assigned to update the victim discovery q-value, otherwise no reward would be assigned.

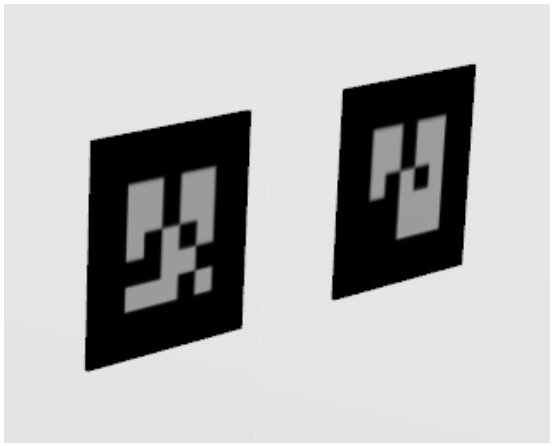


Figure 4: *AR Tags*

IV. Evaluation

I. Environment

The simulation world created in Gazebo to evaluate the performance of the RL algorithm was a 10x10m room with a 2x2m obstacle at the center (Fig. 5). There were 10 AR tag pairs placed on the walls of the room and the central obstacle, 5 of which were valid for victim discovery. The AR tag models used were 0.5x0.5m.

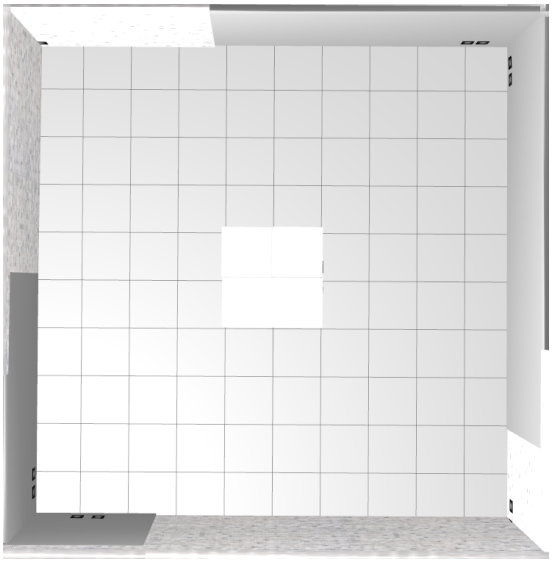


Figure 5: *Simulation Environment*

II. Parameter Selection

The following parameters were used for training the neural network and the q-value update function. The neural network training parameters (NN) were the recommended values used in [4]. The q-value update function parameters (RL) were the values used in [14].

$$\alpha_{NN} = 0.3$$

$$\beta_{NN} = 0.1$$

$$\alpha_{RL} = 0.1$$

$$\gamma_{RL} = 0.15$$

$$\tau_{max} = 0.8$$

$$\tau_{min} = 0.01$$

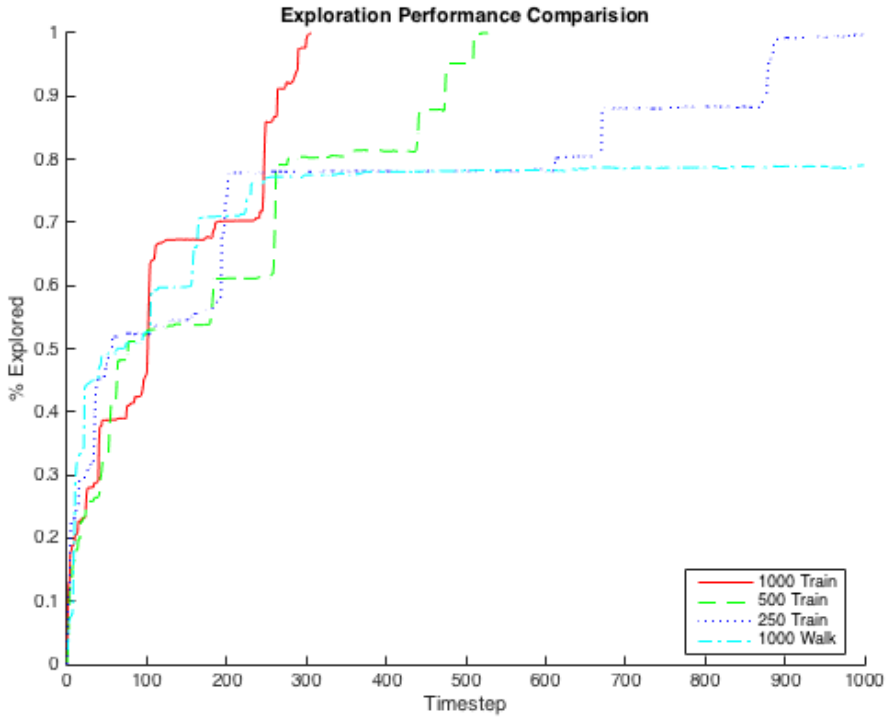
III. Results

III.1 Exploration Performance

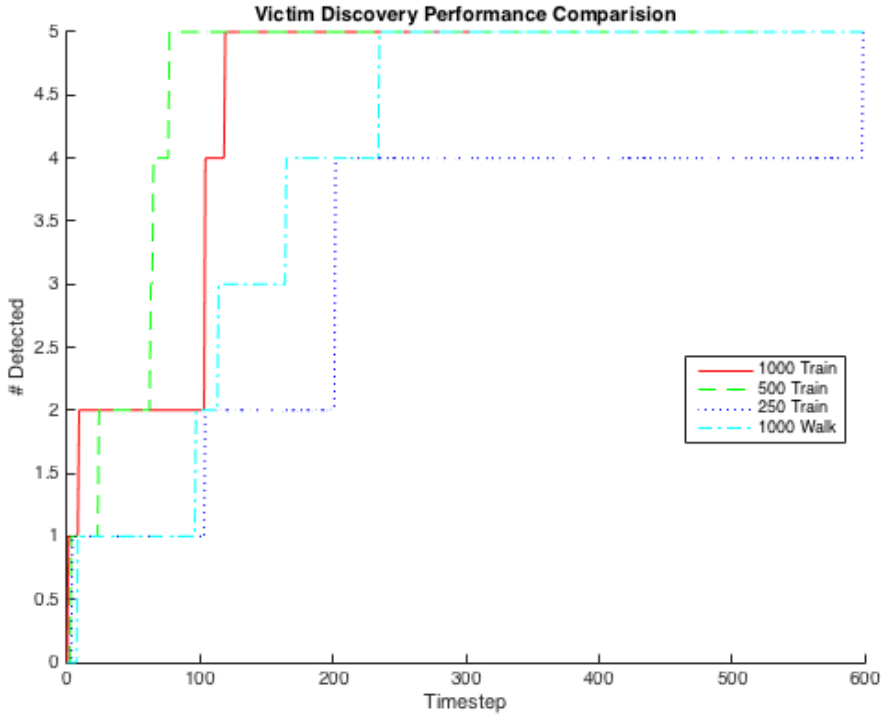
Figure (a) shows a comparison between the exploration performance of the RL algorithm and a random walk in the environment. The exploration performance was evaluated by training the neural network for 250, 500 and 1000 steps and then measuring the number of time-steps taken for complete exploration of the environment. The random walk in the environment was a run of the RL algorithm for 1000 time-steps with $\tau = 1$ to enforce exploration of the action-space. The RL algorithm consistently outperformed the random walk in terms of exploration performance and the random walk did not achieved complete exploration. The exploration rate of the RL algorithm was proportional to the number of training steps. After training for 250 steps the robot explored the environment in 1000 steps, after 500 training steps it took 528 steps and after 1000 training steps it took 306. No significant performance improvement was observed for using more than 1000 training steps. The periods in the graph where the exploration rate plateaus occur when the robot has explored its surrounding environment in both the exploring and sensing grids and therefore has to search for an unexplored region. The exploration cost-maps for the RL algorithm training runs and the random walk are included in the Appendix.

III.2 Victim Discovery Performance

Figure (b) shows a comparison between the victim discovery performance of the RL algorithm and a random walk in the environment. The victim discovery performance was evaluated by training the neural network for 250, 500 and 1000 steps and then measuring the number of time-steps taken to discover all valid AR tags pairs in the environment. The RL algorithm outperformed the random walk in victim discovery using 500 and 1000 training steps but when using 250 training steps the algorithm was outperformed by the random walk. There was no increase in the rate of victim discovery after training for more than 500 steps and in fact the 500 step training marginally outperformed the 1000 step training. Successful detection of all valid AR tag pairs for the 500 and 1000 step training runs took less than half the time taken to completely explore the environment. This demonstrates the ability of the RL algorithm to target exploration to regions where a high likelihood of victim discovery has been learnt.



(a) Exploration Performance Comparison



(b) Victim Discovery Performance Comparison

V. Discussion

I. Limitations

While the results presented demonstrate that the RL algorithm outperforms a random walk at exploration and victim discovery, the utility of this comparison is limited. A better comparison candidate would have been a frontier exploration implementation, however this was not possible due to difficulties encountered using the `frontier_exploration` ROS package. A limitation of the RL approach itself is that convergence to an optimal action selection policy is not guaranteed in non-deterministic state-spaces such as SAR scenarios. By simplifying the visual feature detection problem using AR tags it was not possible to evaluated the robustness of the victim discovery learning to uncertainties in feature detection or to feature misclassification. As the Gazebo simulation environment is a physical simulation and runs in real-world time, it was not feasible to evaluate the performance of the algorithm in larger and more complex environments. A more comprehensive evaluation of the RL algorithm performance would have been possible by deploying the system on a physical TurtleBot, but unfortunately this ended up being beyond the time constraints of the project.

II. Open Questions

- How does the exploration and victim discovery performance compare to state-of-the-art implementations (e.g. frontier exploration)
- How well does the algorithm perform in a real-world SAR scenario?
- Could the algorithm learn to adapt to different types of SAR scenarios (e.g. a scenario with a known number of victims vs. an unknown number)?
- Is it possible for the algorithm to work with multiple agents? How would it scale?
- Can the algorithm be used by a team of heterogeneous agents (e.g. a UAV coordinating a team of UGVs)?

VI. Conclusion

This project presents an reinforcement learning approach to robot SAR using Q-Learning with a neural network implementation. The algorithm is capable of training the neural network to learn collision avoidance, exploration and victim discovery behaviours. After training the robot is capable of avoiding obstacles in an environment and can explore an environment completely with successful victim discovery. The exploration performance of the algorithm outperforms a random walk of the environment and the rate of victim discovery is higher when 500 or more training steps are used. There is also evidence that the algorithm successfully learns to target exploration to regions where there is a high likelihood of victim discovery.

VII. Acknowledgements

I would like to thank my project supervisor Prof. Niki Trigoni for her willingness to support this project and the guidance she has given me throughout it.

Thanks to Sen Wang for his ideas and help troubleshooting problems with ROS.

Thanks to Alison Lowndes and NVIDIA for providing the original project proposal.

References

[1] T. Andre, D. Neuhold, and C. Bettstetter. Coordinated multi-robot exploration: Out of the box packages for ros. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 1457–1462, Dec 2014.

[2] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider. Coordinated multi-robot exploration. *Trans. Rob.*, 21(3):376–386, June 2005.

[3] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, June 2005.

[4] Tejpal Singh Chhabra. Back-propagation neural net, 2006.

[5] J. d. Hoog, S. Cameron, and A. Visser. Role-based autonomous multi-robot exploration. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World:*, pages 482–487, Nov 2009.

[6] Julian de Hoog, Stephen Cameron, and Arnoud Visser. Selection of rendezvous points for multi-robot exploration in dynamic environments. In *Workshop on Agents in Realtime and Dynamic Environments, International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, May 2010.

[7] B. Doroodgar and G. Nejat. A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments. In *2010 IEEE International Conference on Automation Science and Engineering*, pages 948–953, Aug 2010.

[8] Cai Luo, A. P. Espinosa, A. De Gloria, and R. Sgherri. Air-ground multi-agent robot team coordination. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 6588–6591, May 2011.

[9] Cai Luo, A. P. Espinosa, D. Pranantha, and A. De Gloria. Multi-robot search and rescue team. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 296–301, Nov 2011.

[10] K. Macek, I. Petrovic, and N. Peric. A reinforcement learning approach to obstacle avoidance of mobile robots. In *Advanced Motion Control, 2002. 7th International Workshop on*, pages 462–466, 2002.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. Letter.

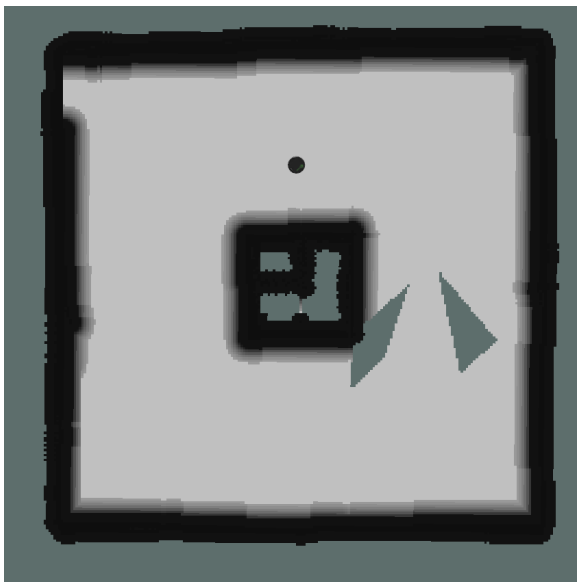
[12] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and Bernhard Rinner. An autonomous multi-uav system for search and rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, DroNet '15, pages 33–38, New York, NY, USA, 2015. ACM.

[13] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *International Symposiom on Robots and Security*, 2010.

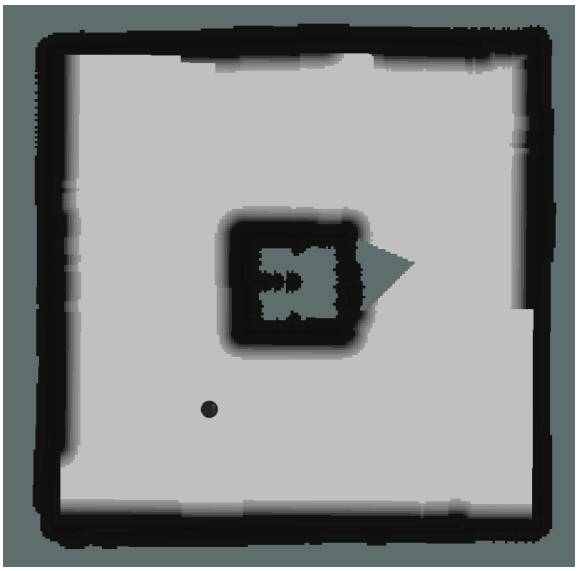
[14] Chen XIA and A. EL KAMEL. *A Reinforcement Learning Method of Obstacle Avoidance for Industrial Mobile Vehicles in Unknown Environments Using Neural Network*, pages 671–675. Atlantis Press, Paris, 2015.

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

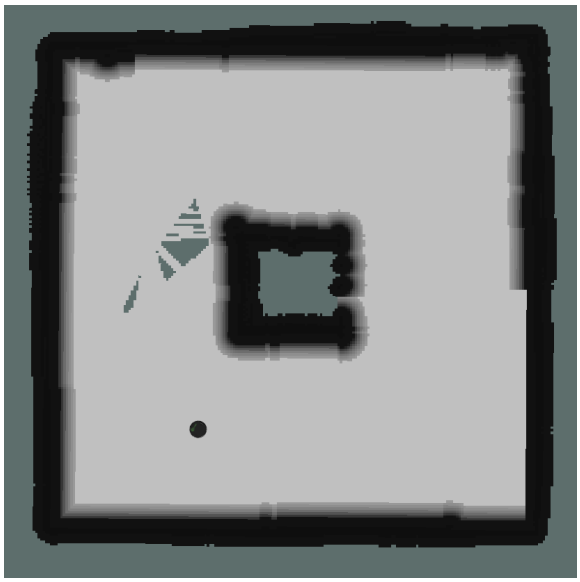
VIII. Appendix



(a) 250 Step RL Training



(b) 500 Step RL Training



(c) 1000 Step RL Training

(d) 1000 Step Random Walk