Navigation for the Visually Impaired of Virginia

Ingy ElSayed-Aly
University of Virginia
ie3ne@virginia.edu

Jyoti Kumari
University of Virginia
jk6ca@virginia.edu

Nabeel Nasir
University of Virginia
nn5rh@virginia.edu

***Abstract*—Navigation using geolocation technology has become an essential function of our personal smartphones. It comes in handy when walking, cycling, driving or using public transport in areas we are unfamiliar with. However, because of GPS limitations it falls short when it comes to indoor navigation. In our project, we focus on indoor navigation specifically for visually impaired individuals who face this challenge every day even when moving within their homes. The idea is to use the camera of a handheld phone for vision, and a reinforcement learning based algorithm for navigation. After the modeling of an indoor environment and training, the resident will be able to navigate from any point in the indoor environment to another, avoiding walls and obstacles. The purpose of this document is to report our findings and approach to solving the problem. This project uses a simulation of the desired indoor environment for training. In this document, we propose several experiments using Q-Learning and our results.**

I. Introduction

Reinforcement learning is an active area of research especially for games and robotics [1]. Reinforcement Learning (RL) refers to a kind of Machine Learning method in which the agent receives a delayed reward in the next time step to evaluate its previous action. Google DeepMind became very famous for its success at beating top Go players as well as Atari games with a higher performance than humans [2] . In the context of robotics, reinforcement learning has been used to attempt to teach different types of behaviors one of them being parking. Kolter et al. describe how they were able to create a state of the art model for "extreme" parking (high speed slide into a parking spot) using reinforcement learning [3].

The aim of this project is to help navigate a person through an environment based on visual input. This would be done though an app on a phone that has a camera. The camera will use our algorithm to identify and avoid obstacles while navigating the impaired person to the goal. This would be geared towards areas where Google maps does not have defined roads/clear paths such as inside a house with multiple rooms. For this project, we have used reinforcement learning within a graphical simulated environment. Within the environment, we will have a first person actor (agent) navigating in a world with obstacles to try to reach a target location. The actor will eventually learn not to bump into obstacles and attempt to get to the target location within the best time possible. In the case of reinforcement algorithm, we will be rewarding steps in the right direction and penalizing the actor for bumping into obstacles.

II. Reinforcement Learning

A Reinforcement Learning setup is composed of two components, an agent and an environment. Figure 1 demonstrates a typical reinforcement learning setup, where the environment refers to the object that the agent is acting on (e.g. the game itself in the Atari game), while the agent represents the RL algorithm. In our case, the environment would be the world in the simulation, and the agent would represent the first-person actor in the world. In reinforcement learning, a dataset is not required for training the actor in the simulated world. The learning is instead based on rewards or penalties the agent receives from the environment. There are several reinforcement learning algorithms including Q-Learning, State-Action-Reward-State-Action (SARSA), Deep Q Network (DQN), Deep Deterministic Policy Gradient (DDPG) etc. We choose to use Q-Learning which is one of the most common algorithms used for maze-like problems.

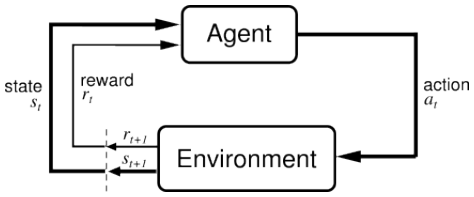

Fig. 1. Reinforcement Learning

A. Q-Learning

Q-Learning is an off-policy (off-policy learns the Value(V) based on the action A at time t obtained from another policy), the core of this model algorithm is the Bellman equation which is used to update the value of the previous state based on the reward received and the action chosen. The Bellman equation is as shown below in Figure 2:

$$Q^{\pi}(s_t, a_t) = \underline{E}[\underline{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...}|\underline{s_t, a_t}]$$

Fig. 2. Bellman Equation

The algorithm views each cell in the game as a state and there are a fixed set of actions that can be selected by the agent in each state. The algorithm uses a table called Q-table to keep track of the maximum expected future reward for each action at each state. For instance, if there is a game in which

25 grid game as shown in Figure 3, in which the agent can pick four actions - left, right, up, and down, the Q-table will have 25 x 4 = 100 cells. The columns in the Q-Table will be the four actions (left, right, up, down). The rows will be the states. The value of each cell will be the maximum expected future reward for that given state and action.

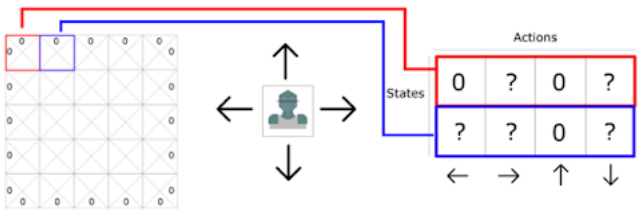

Fig. 3. Possible actions in a given state

At each step, the agent either does one of exploration (pick a random action) or exploitation (pick the action with the best maximum expected future reward) and then updates the Q-Table. As the agent does more and more exploration, it learns much better about the best way to approach the environment. At each step we encourage the agent to choose to exploit the Q-table more by introducing a exploration rate hyperparameter that is decreased at each step. The goal is that at the end of training the agent should be use mostly exploitation and not random actions.

III. Choice of Environment

We researched and experimented on several of the available environments and then chose DeepMind Lab. DeepMind Lab is a 3D learning environment based on id Software's Quake III Arena using the open source ioquake3 platform. It provides a set of challenging customizable environments for training learning agents. In this section, we present the various environments that we had tested out and the rationale behind our selection. We experimented with the following environments: OpenAI Gym [4] with Gazebo, Microsofts Project Malmo, Unreal Engine with OpenAI, and DeepMind Lab.

We discuss a few key things that we look for in an environment. First, we looked at the support for reinforcement learning algorithms. Second, we look at the ease of use of the environment. This includes the ease of setting up the environment and what middleware needs to be in place for things to work. Gazebo is a robot simulation environment designed to be used with Robot Operating System (ROS). Gazebo provides a way to use 3D models. However, we found that using this system was complex and also agents in the Gazebo worlds can only be controlled via ROS. This means all our code needs to be tweaked to use ROS, which is also fairly complex.

Project Malmo is a platform for AI experimentation on top of the Minecraft game. The object placement in the simulation world is straight forward using an XML style descriptor. However, the major drawback is that the platform does not have a good community support. Moreover, the reinforcement learning support for the platform is also limited. This has led to the Multi Agent Reinforcement Learning using Malmo (Marlo) platform which is also in its nascent stages.

Unreal engine is a free and very comprehensive game development engine which can be used to create realistic simulated environment and natural feeling agent movement. Unfortunately, Unreal Engine does not officially support OpenAI. Therefore, we have tried using community supported plugins to Unreal Engine, with very little success. Moreover, the world development requires some modeling knowledge in Unreal Engine.

Deepmind lab provides a customizable environment in which we can generate worlds using Text Levels, which are simple, human-readable text files to specify walls, spawn points and other game mechanics [5]. Although, the environment might look a little game-like, we believe that the physics of the engine works reasonably well to be utilized for our problem.

IV. Method

A. Map generation

Deepmind lab provides a couple of modules written in Lua for random maze generation with the guarantee that there is a path and that the player does not spawn too close to the goal. One of the challenges was to learn enough Lua and go through the Lua code (the documentation was not sufficient to understand the mechanism clearly) to be able to create a map with the characteristics we wanted. Finally, we customized our map to be a 7 by 7 grid map, with one goal and 7 apples (intermediary goals of sorts). In order to make the movement more realistic, we decided to make the resolution of the cell 100x100 possible coordinates (10,000 possible positions for our character within one cell).

B. Rewards and Penalties

A sample environment generated by Deepmind Lab for our map is shown in Figure 4. The objective is for the agent to reach the goal using minimum number of steps (i.e the shortest path). We use game objects to make the agent learn about navigating in the environment. The goal object has a reward of 10, each apple has a reward of 1, and the agent is penalized by -1 for each step that it takes. We penalize the agent for every step to incentivize it to find a shorter way to the goal and not waste time in a corner. The apples are placed strategically close to the goal to lure the agent to the goal.

C. Environment States and Actions

At first, our Q-table comprised of all the cells in the grid (as rows) and 3 actions (as columns). Our three actions were : rotate left 20 degrees, rotate right 20 degrees, and move forward 20 world units (in the 700x700 map). However, our agent would stay stuck in the corners for a very long time and we further discovered that telling it to rotate 20 was not equivalent to 20 degrees. We therefore adjusted the rotation and increased the rotation angle to 35 degrees to make sure it has a better chance to get out of a corner. What was happening

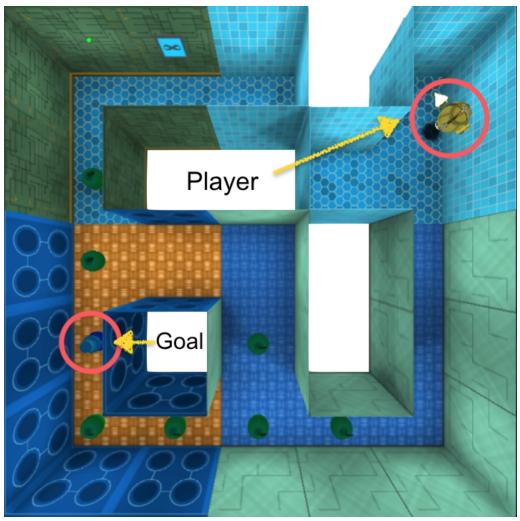

Fig. 4. Project game environment in Deepmind Lab

is that in exploration mode (random actions) it would choose -20 degrees, then forward, then rotate left 20 degrees and forward, etc. The probability of it rotating in the same direction twice or thrice and then moving forward to get out of a corner was very low. Moreover, the state simplification from 700x700 to 7x7 and not taking into account the current rotation made it very hard for our agent to learn anything. Being in cell 1,1 and moving forward and the resulting state and the same action would be the same. Also, moving forward when facing a wall or a path were represented equivalently. Therefore, we decided to include rotation in the states. The states of the agent are now based on the agents position and current orientation. The 2D positional coordinate of the agent between (0,0) and (700, 700) is scaled to a value between (0,0) to (7,7) implying 49 positional states. The agents orientation can be between any angle between 0°to 360°. This is simplified into quadrants of 45°accounting to 8 states. Therefore, a total of 49 x 8 = 392 states are possible.

D. Q-Learning

The Q-Table has a dimension of 392 x 3 (392 states and 3 actions) as discussed above. The agent is trained for 200 episodes. An episode lasts until the agent reaches the goal or it runs out of time after 800 actions. In order to control the ratio of exploration to exploitation, we use a parameter called decay rate to 0.05 which computes the rate of decay of the exploration rate ϵ. The decay rate represents how much the exploration is decreased for the next step.

V. Experiments

Because the goal of this project was to provide navigation indoors (mostly for their homes), it is assumed that there is an exploration/training phase in the indoor environment before it can be used reliably. Exploration of the environment is encouraged in the beginning of the training by setting the exploration rate to 1. As we first started to train the agent the model it bumped into the walls more and was less directed towards the goal. This was initial stage and as it explores more and more, the Q-table gets updated at every action. After several episodes, it has recorded a sequence of actions that gives it better rewards, less penalty and is shortest to the goal. In Figure 5, we record the number of steps the agent has taken to reach the goal (or run out of time if it is 800).

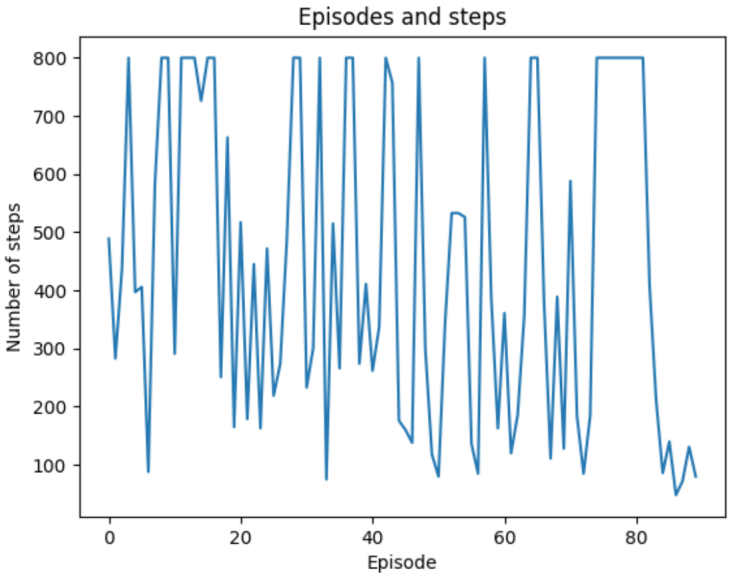

Fig. 5. Plot showing number of steps the agent takes to reach the goal

Over time, the agent is able to reduce the amount of steps to get to the goal on successful runs but it still seems to get stuck. We think this is because of our penalty and reward ratio. Our penalty for each step is large (-5) whereas our rewards are low (10 for the goal and 1 for the apples). Thus, the agent still has a large penalty for successful runs. Therefore, entries that do not lead to anything over time seem to have a better reward than the correct path. The solution to this would be to increase very much the reward (implemented in the code as the reward_factor, but we could not test it because of time reasons). In the next section we suggest several ideas to make the agent more performant.

VI. Limitations and Future Work

We believe that the most challenging aspect of reinforcement learning is to find the best value function to incentivize the agent without giving away too much information. The second most challenging aspect was to find the best set of actions and states. The third most challenging aspect is that the training for the agent takes a very large amount of time. Running one episode takes about 2 minutes and if we are running it for 200 episodes for it to start using the Q-table (when the exploration rate has decayed enough), that makes a total of 6 hours and 40 minutes. For next steps, the state space can be increased to subdivide each grid cell into smaller cells to differentiate the parts of the cell close to the wall versus the subcells close to the path. This increase with the rotation information should give the agent a more detailed map and

more decisions. Moreover, subdividing cells could give the opportunity of penalizing with respect to the distance between the agent and the wall. Furthermore, the initial goal of the project was to use the camera input as input to the algorithm. However, with Q-learning that is not the case. A next step is to adapt our work with Q-Learning into a model using another reinforcement learning method that would take an image as input.

VII. Conclusion

We believe our project provides a first step to an indoor navigation model solution for the visually impaired in Virginia. For instance, in real life, an elderly person with faded vision wants to go to the kitchen for taking their meal. This reward/penalty reinforcement learning model will ensure that they move in the direction of kitchen since that is the highest reward within the shortest possible path as each step is a penalty and without bumping into walls (as those incur a penalty). Therefore, we believe that our project proves that there is an unexploited potential in reinforcement learning algorithms that can satisfy the requirements of our problem.

Reinforcement learning is a hot topic among researchers. The idea of self-learning makes it a go-to technology for autonomous systems. Our project here just demonstrated this fact. However, reinforcement learning we believe is still in its early stages. Further development in this field will likely yield better strategies and modeling opportunities for more complex problems and systems.

VIII. Contributions

Ingy-ElSayed-Aly: Researched Unity and Unreal Engine, Lua API and map generation, Q-Learning testing and tuning. Nabeel Nasir: Researched Gazebo and Malmo, Q-Learning implementation, Q-Learning testing and tuning. Jyoti Kumari: Researched Deepmind lab, Q-Learning testing and tuning, Report Writing.

References

[1] J. Kober, J. Andrew (Drew) Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, July 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] David T. Jackson Andrew Y. Ng Sebastian Thrun J. Zico Kolter, Christian Plagemann. A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. http://ai.stanford.edu/%7Eang/papers/icra10-ExtremeAutonomousDriving.pdf . Accessed: 2018-10-09.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.