Safe, Autonomous, Robot-Aided Pedestrian Navigation

Eshan Bhargava† and John Bush†

University of Southern California

Abstract. Reinforcement learning (RL) has seen a surge in popularity in the recent years, transforming numerous domains including path planning and navigation in mobile robots. Our project delves into both traditional RL and Deep RL techniques, aiming to develop an autonomous agent capable of assisting a disabled person in safely navigating pedestrian pathways. We approach the problem iteratively, initially applying traditional reinforcement learning (RL) methods in a discrete environment, where obstacles are represented as pedestrians. Gradually, we progress to implementing deep RL in a continuous environment, utilizing Perlin Noise fields[1] to model obstacles. This approach is designed to mimic the unpredictable and chaotic movement found in busy pedestrian pathways. Our method not only bolsters the effectiveness of RL in path planning and navigation but also introduces a policy adept at navigating through unique obstacles.

Keywords: Autonomous Agent · Deep Reinforcement Learning · Perlin Noise.
Author Roles: † - equal contribution authors[1]

1 Introduction

1.1 Motivation

The challenge of navigating through crowded spaces poses a significant hurdle for visually impaired individuals. Traditional guide dogs offer remarkable assistance, but they come with limitations such as training time, cost, and availability. Nevertheless, there are over 20 million Americans living with visual impairments, and one of the main impacts of visual impairment, especially among seniors, is loss of independence due to inability to safely navigate sidewalks and intersections. A robotic guide dog, equipped with advanced navigation capabilities, could offer a complementary or alternative solution, addressing these limitations while providing consistent, reliable guidance. Developments in computer vision, robotics, and machine learning enable robots to intelligently interact with complex environments alongside humans and we seek to harness this technological advancement to prove the viability of robot service companions. Our goal in this project is to create a practical application that would allow a disabled person

[1] https://github.com/john-bush/safety-gymnasium

to navigate a crowded walkway with the assistance of an intelligent robot in a safe fashion. In this report, we work on developing a deep reinforcement learning algorithm capable of avoiding both static and dynamic obstacles. Moving forward, we are committed to advancing our methods with an enhanced focus on aggressively prioritizing safety in robotic pathfinding. This initiative is rooted in our understanding that there is still significant room for improvement in the current state of research in this area. We aim to contribute meaningful solutions to this ongoing challenge.

1.2 Challenges

Creating a realistic simulation for pedestrian navigation presents a significant challenge for our proposed agent, mainly due to the high-dimensional and continuous nature of the observation and action spaces within such an environment. Additionally, the presence of numerous objects, each with its own action space, further complicates the scenario. Therefore, a primary challenge in our experiment was selecting an environment that could accurately replicate these intricate and dynamic conditions.

When beginning our project, we initially focused on leveraging traditional reinforcement learning algorithms. These algorithms are often faster to train, particularly in environments featuring static obstacles, like fire hydrants, mailboxes, and lamp posts due to the lower dimensional state space for a single dynamic agent. However, we recognized the necessity of integrating deep reinforcement learning to enhance our approach to facilitate safe pathfinding through dynamic obstacles fields, akin to a busy intersection. This integration entailed hyperparameter tuning deep learning models to serve as function approximators. Specifically, we considered either a Q-function approximator or a policy function approximator.

Lastly, and potentially the most challenging part of the project was tuning the reward function. Firstly, we would have to iteratively modify the reward function so that the agent best utilizes rewards to navigate to a perceived goal, while avoiding both static and dynamic obstacles. Moreover, this could be a joint process with the hyperparameter tuning of the deep learning algorithm because theoretically certain hyperparameters of a neural network could work for some reward functions but not others.

1.3 Related Work

Localization and Mapping Localization of a robot relative to nearby obstacles and identification of those obstacles is a key challenge in robotic path planning. In Kayukawa, et al. [2], they used LiDAR simultaneous localization and mapping (SLAM) to build a local 2D map of the environment combined with an RGB depth camera module to identify and place key points of interest within that local 2D map. In our work, we have adopted a similar sensor stack, using simulated LiDAR to sense nearby obstacles. Other implementations, such

as in [3] and [4] use embedded RFID tags throughout the environment to perform localization on a known environment space with a precomputed map. These methods work within a structured framework, where the environment is known in advance, and the focus is on localization within that environment. Furthermore, path planning within such structured environments with known static RFID tag locations presents a simpler, lower dimension state space representation with a single dynamic ego agent. In our implementation, we consider an unknown environment with many dynamic obstacles (pedestrians), that the ego must identify and avoid collision with.

Algorithmic Path Planning Methods Algorithmic Path Planners, built upon graph search algorithms such as A* and Dijkstra are widespread in their use for assistive navigation, as used in [5]. A* has widespread use due to its efficiency and optimality in shortest path finding, and has many applications throughout the field of robotics. However, when applied to human-assistive robotics, A* search has some limitations due to its computational cost that scales exponentially with state and action space complexity. Furthermore, paths generated by discrete planners will have optimality that may be suitable for a robot to follow, but awkward and potentially dangerous for the human being led [6]. A* generated paths are considered Turn-By-Turn, since they are generated through a superimposed grid representation of the state space, utilizing a simplified, discrete action space. This results in jerky, sharp turns in the generated paths. Additionally, these paths becomes more expensive to compute when considering a changing state space with other dynamic agents, resulting in continual replanning. In our implementation, we leverage a continuous action space and an online planning structure to generate paths that are smoother and easier to follow. These smoother paths lend themselves to better safety and understandability for a real world application.

Inherent Noise-Aware Insect Swarm Simulation Since integrating pedestrian agents in our simulation space is key to our safe planning model, we looked into different existing methods of crowd and pedestrian simulation. Crowd dynamics are known to apply to large numbers of agents who occupy the same space and whose movements are predominantly defined by local interactions [7], which led us to look at swarm simulations, which have similar characteristics to human crowds. The research presented in [8] offers a pertinent example of perlin noise to model insect swarms, which offers many parallels to crowd dynamics. In their work, perlin noise is utilized to simulate the continuous, smooth random movements of an insect swarm. The perlin generator produces a random, but continuous noise field that mimics the local interactions that influence crowd dynamics. While the scale of noise particles employed in their model significantly exceeds ours, the underlying principle remains applicable to our use case. Our aim is to model the unpredictable and chaotic movement patterns of pedestrian swarms, akin to those commonly observed in bustling, urban environments, like New York City.

1.4 High-Level Research Questions

In this project we attempt to answer the following questions:

1. What constitutes an accurate environment that encapsulates the multi-faceted interactions and inherent complexities of pedestrian walkways?
2. In what manner can this environment be effectively modeled to reflect its true essence?
3. Which Reinforcement Learning algorithms demonstrate the potential for proficient navigation in our proposed environmental?
4. How simple or complex does the reward function have to be to accurately allow the system to perceive the dynamics of these kinds of environments and aptly navigate them?

2 Problem Definition

As previously mentioned, our proposed project involves the development of an autonomous agent designed to safely navigate through crowded and chaotic pedestrian pathways, similar to those found in large, urban environments. While this technology has potential for a broad range of applications, our primary focus is on its utility as a guidance system for disabled individuals, enabling them to traverse these complex environments with ease.

2.1 Experiment Setup and Formalism

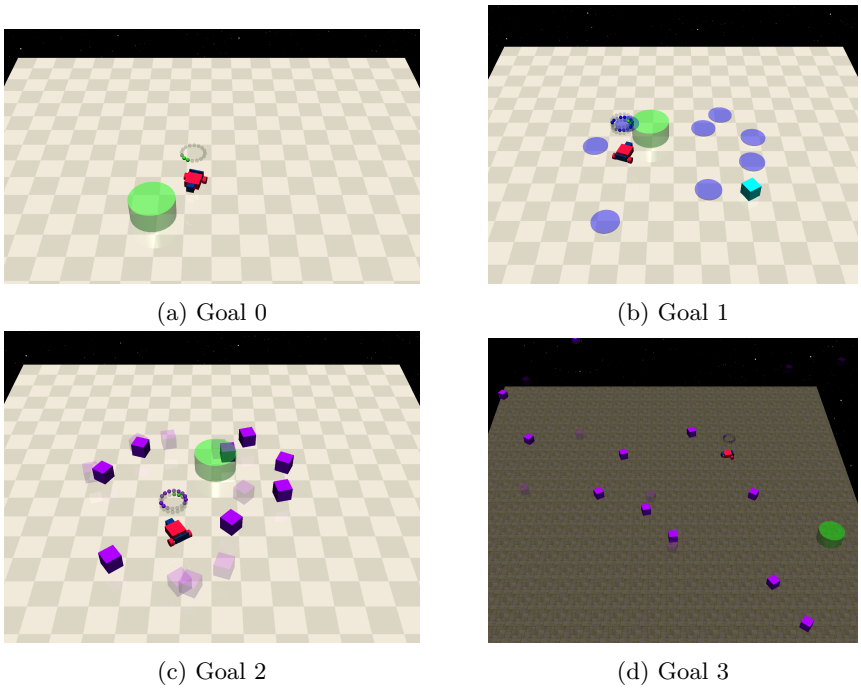
In this experiment, we employ two distinct simulation environments: discrete[2] and continuous[3]. In the continuous environment, the agent's action space is continuous, whereas in the discrete environment, it is discrete. Initially, we focused on developing and utilizing the discrete environment, which served as a foundational step towards the continuous environment. This approach was strategically chosen because reinforcement learning algorithms operate more efficiently in a discrete setting. This expedited learning in the discrete environment provided valuable insights and groundwork, facilitating a smoother transition to the more complex continuous environment.

Additionally, we have broken the project up into 4 stages, acting as checkpoints for our progress:

1. Goal_0: The agent must navigate to a goal position. (discrete and continuous)
2. Goal_1: The agent must navigate through static obstacles to a goal position. (discrete and continuous)
3. Goal_2: The agent must navigate through dynamic obstacles, that move circularly, to a goal position. (continuous)
4. Goal_3: The agent must navigate through dynamic obstacles, modeled as a perlin flow field, to a goal position. (continuous)

[2] This environment was created in Gymnasium[9].
[3] This environment was created in Safety Gymnasium[10].

We use model-free reinforcement learning strategies in order to solve each project stage. In the next sections, we will go over how each simulation environment is setup.



(a) Goal 0 (b) Goal 1

(c) Goal 2 (d) Goal 3

Fig. 1: Safety Gymanisum environments built for each of the proposed goal tasks.

Continuous Environment In this environment, our agent is modeled as a wheeled robot that moves in two dimensions, has two independently driven parallel wheels, and one free-rolling rear wheel. Both steering and forward/backward movement require coordination of the two drives. Table 1 describes the action space. Table 2 is the agent observation space. We chose to use a wheeled robot rather than a quadrupedal robot to avoid the additional computational cost of complex internal joint coordination during training, allowing us to focus our efforts on path planning.

For each goal task, the environment is a continuous, 3-dimensional grid. To maintain feasibility in solving the task, we have strategically constrained the explorable space of this environment, which is theoretically unbounded. This involves closely positioning the spawning points of the goal, obstacles, and the agent, ensuring that the task remains manageable despite the vast potential space. It is also important to note that the agent receives information about

Table 1: 2-dimensional, continuous action space of the proposed agent.

Action	Control Min	Control Max	Joint/Site
Force(N) applied on the left wheel	0	1	hinge
Force(N) applied on the right wheel	0	1	hinge

Table 2: Continuous observation space of the proposed agent. Note the Quaternions of the rear wheel are turned into 3x3 rotation matrices.

Size	**Observation**	**Control Min/Max**	**Joint/Site**
9	Quaternions of the rear wheel	-inf/inf	ball
3	Angular velocity(rad/s) of the rear wheel	-inf/inf	ball
3	Accelerometer (m/s^2)	-inf/inf	site
3	Velocimeter (m/s)	-inf/inf	site
3	Gyroscope (rad/s)	-inf/inf	site
3	Magnetometer (Wb)	-inf/inf	site

its immediate environment through simulated LiDAR sensors to find the goal(s) and avoid the obstacles. These LiDAR sensors have limited observable range and field of view, which closely mimics real world sensor capabilities deployed on robots. The specific environment observation space is defined in Table 3 for each of our respective goals.

Table 3: Continuous observation space of the environments for each of our goals. Each LiDAR sensor can sense up to a max distance of 3; they return a vector of dimension size(seen in the table) to the agent.

Environments	**Means of Observation**	**Size**	**Observation Low/High**
Goal_0	Goal LiDAR	16	-inf/inf
Goal_1	Goal LiDAR, Hazards LiDAR, Vases LiDAR	48	-inf/inf
Goal_2	Goal LiDAR, Hazards LiDAR	32	-inf/inf
Goal_3	Goal LiDAR, Hazards LiDAR	32	-inf/inf

Our reward scheme has two modes. At each time step, the agent receives a positive reward when it moves closer to the Goal, and conversely, a negative reward is applied if it moves farther away. We refer to this as the "reward_distance" mode, described by the below equation:

$$r_t = (D_{last} - D_{now}) * \beta \tag{1}$$

The next mode is called the "reward_goal" mode, where the agent gets a positive reward of 1 if the goal is reached. The metric we use to model success for each episode is average reward.

Lastly, in order to accurately replicate a pedestrian walkway environment, it's essential to include various types of obstacles for our agent to navigate around.

This includes static obstacles, representing immovable items like fire hydrants, dynamic obstacles to simulate moving pedestrians, and pushable objects, akin to movable items such as cardboard boxes. This variety ensures a realistic and comprehensive simulation for our proposed agent that mimics a real world intersection crossing.

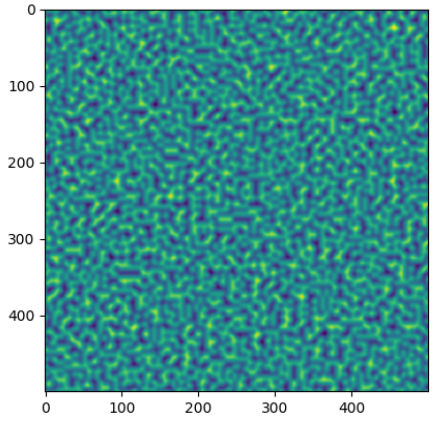


Fig. 2: Randomly Generated Continuous Perlin Noise Field

The most sophisticated and realistic obstacle we designed is a pedestrian traffic generator that mimics an all-way pedestrian intersection. We generated N dynamic obstacles, referred to as *Gremlins*, each with a psuedo-randomly assigned characteristics defining their movements within the state space. To simulate the local interactions between dynamic obstacles, we implemented a Perlin noise field superimposed onto the state space that injects a disturbance velocity v_{noise} to each obstacle agent in the simulation, as shown in Figure 2. This approach was employed to emulate the dynamic and ever-changing patterns of pedestrian movement.

In order to learn safe behaviors, our environments assess penalties for the ego agent upon collision with various obstacles in the environment. A practical example of incurring such a penalty would be if the robot guide collided with a lamp post or other pedestrian while guiding its user to the destination. Through these penalties, we train the agent to avoid collision with obstacles that it observes in its nearby vicinity. Table 4 details the costs associated with each obstacle in this environment.

For our Gremlin obstacles, we created a specific motion generator to mimic the dynamics of a realistic crowded pedestrian intersection. To define the motion of these pedestrian fields, we formalize the initialization of Gremlin_i as having a phase offset ϕ_i, a speed modifier $\gamma_i \in [0.75, 1.5]$, moving direction $\delta_i \in \{-1, 1\}$, moving radius r_i and an oscillation center coordinate defined as $C_i = (a, b) \mid a, b \in \{-dim, +dim\}$.

Table 4: Costs associated with each obstacle. Note, the vase is a pushable obstacle that no cost, as the agent simply needs to push it out of the way, and continue.

Environments	**Task**	**Cost**
Hazard	Goal_0	1
Vase	Goal_1	0
Circular Gremlin	Goal_2	1
Perlin Noise Field Gremlin	Goal_3	1

The angular position of each Gremlin is defined through Eq. 2 in terms of a global simulation time step, $t_j \mid j \in [0, n]$, and $i \in [1, N]$ identifying each dynamic obstacle agent. Agents have randomly assigned directions to simulate bidirectional traffic in each segment of an intersection, and randomly assigned speeds within a constrained range to emulate different walking speeds.

$$\theta_i(t_j) = 2\pi(1 + \delta_i) - (\gamma_i \delta_i(t_{sim} + \phi_i)) \tag{2}$$

Given a Perlin noise matrix generated at the beginning of the simulation episode with a dimension of 10 times the continuous state space, we calculate a noise angle at each time step for each Gremlin based on its previous position, described in Eq. 3, which provides the basis for our random, continuous disturbance for the Gremlin agents.

$$\rho_i(t_j) = \text{Perlin}(position_i(t_{j-1}) \cdot 10) \tag{3}$$

The actual perturbation for each agent is then calculated as the Cartesian unit vector associated with that angle ρ_i in Eq. 4.

$$v_{noise}(t_j) = \begin{bmatrix} cos(\rho_i) \\ sin(\rho_i) \end{bmatrix} \tag{4}$$

Finally, we determine the position for each Gremlin as the sum of its sinusoidal, time-dependent position, the local perturbation, and its rotational axis, as shown in Eq. 5.

$$position_i(t) = r_i \cdot \begin{bmatrix} cos(\theta_i) \\ sin(\theta_i) \end{bmatrix} + v_{noise}(t_j) + C_i \tag{5}$$

Equations 2-5 define the dynamics of our moving obstacles within the state space. It is worth noting that the dynamic obstacles have roughly circular trajectories, with only a portion of the trajectory fitting inside of the simulation's state space. This is due to momentum constraints of the physics engine which prevent position resetting during simulation episodes. To address this limitation, we decided to give Gremlins cyclic, partially random trajectories that are psuedo-linear within the arc length contained in the state space.

By injecting a disturbance generated through a Perlin noise matrix onto each agent, we can mimic the formalisms of crowd dynamics described in [8,

7]. Simulated pedestrians in localized regions are jointly affected by the continuous noise matrix, which produces realistic results of propagating disturbance through nearby agents. Another benefit of this method is the cheap computational overhead per time step per Gremlin added to the simulation, which helps to reduce the training time per episode.

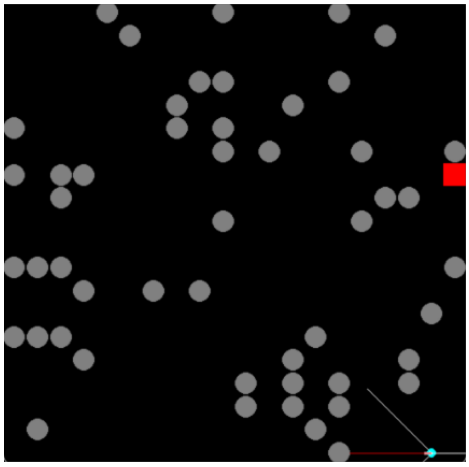


Fig. 3: Discrete environment built with Gymansium.

Discrete Environment This environment is notably simpler compared to the continuous one. Here, the agent is modeled as a point with four possible actions: turn left, turn right, turn backwards, and move forward, with the direction dependent on its current orientation. The entire observation space is defined as a 20 by 20 grid. The agent is equipped with a lidar sensor for detecting obstacles and the goal. This sensor is depicted as three lines emanating from the front of the robot, each separated by a 45-degree angle, and an additional line extending from behind the robot.

Furthermore, the reward and cost functions in this environment are distinct from those in the continuous environment. This variation is attributed to the different algorithms we experimented within this simpler setting. In this environment, the reward system for Goal 0 is straightforward: the agent receives a reward of 0 when it has not reached the goal, and a reward of 1 upon successfully reaching the goal.

For Goal 1, the reward structure is slightly more intricate. The agent earns incremental rewards based on the reduction in distance to the goal compared to its previous state. Achieving the goal garners a substantial reward of 50. Additionally, a minor penalty is incurred if the lidar sensor detects proximity to an obstacle, encouraging spatial awareness. In cases of collision with an obsta-

cle, the agent is penalized with a cost of -10, underscoring the importance of navigation without contact.

3 Solution Strategy

Our solution strategy primarily employs model-free reinforcement learning algorithms. Within this framework, we delve into both gradient-free and gradient-based methods. Specifically, we utilize Q-learning, a prominent example of gradient-free techniques, alongside policy gradient optimization, which represents the gradient-based approach.

3.1 Q-table

In the discrete environment, we implemented Q-learning using a Q-table[11], and to accommodate the varying nature of tasks within this environment, we experimented with two distinct types of Q-tables[4]. The first Q-table was the standard form, consisting of 400 rows and 4 columns. These dimensions represent the possible states the agent could encounter and the number of actions it could perform, respectively. This simple Q-table proved effective for Goal 0 but fell short in the context of Goal 1. Recognizing this limitation, we then tested a modified Q-table variant designed to primarily consider local obstacles. Unfortunately, this adapted approach also failed to yield success in addressing Goal 1, highlighting the challenges posed by the task's complexity and the limitations of standard Q-learning techniques in such scenarios.

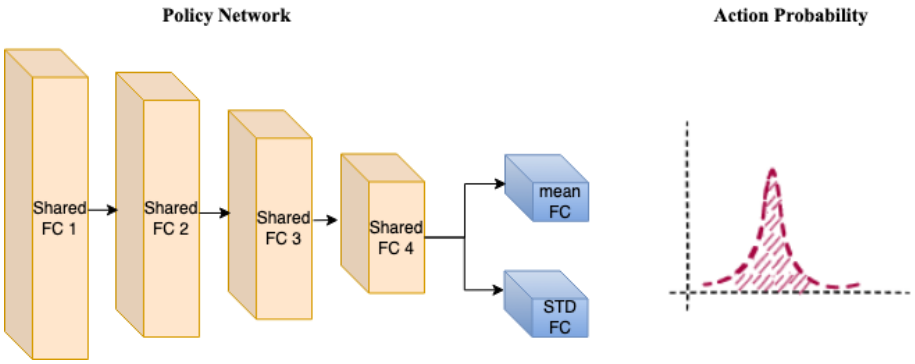


Fig. 4: Policy network for our proposed agent.

[4] The hyperparameters of this experiment are as follows: learning rate = 0.9, discount factor = 0.9, epsilon = 1, epsilon decay rate = 0.0001.

3.2 Policy Network

Our other approach was to build a policy that the agent will learn using REINFORCE[12][5]. The policy, parameterized by a neural network, maps the current environment observation to a probability distribution of the actions to be taken. It consists of 4 hidden linear layers[6] that are shared between both the predicted mean and standard deviation. The hyperbolic tangent non-linearity is applied between the hidden layers. Furthermore, the output representation of the last hidden layer is fed into single individual linear layers, which are used to estimate the mean and the standard deviation. It is expected for the policy to learn appropriate weights to output the mean and standard deviation based on the current observation. This approach is used in the developed continuous environment.

4 Results

4.1 Discrete Environment

The Q-table approach, applied in the discrete environment, effectively solves Goal 0 due to the environment's simplicity. However, this standard Q-table method struggles with Goal 1, where static obstacles are introduced. We believe this is caused by of the 'curse of dimensionality' coupled with a weak reward structure.

It's worth noting that the reward structure effective for Goal 0 was not able to assist the agent in achieving Goal 1, as the agent consistently ended up with a reward of 0 per episode. However, problems that were modeled similarly, such as the Frozen Lake Environment[9], have been successfully solved using this reward structure. Furthermore, when we iteratively modified the reward structure, as detailed in our experiment setup and formalism, the rewards across episodes eventually converged to 0.

In this scenario, the agent prioritizes avoiding obstacles over progressing towards the goal. This behavior results from the reward structure: when the agent nears an obstacle, it receives a negative reward. Consequently, the agent values the neutral outcome of a zero reward more than the risk of a negative reward, leading to limited exploration confined to areas without obstacles.

This issue often stems from an imbalance in the reward structure, where the agent perceives avoiding negative rewards (associated with proximity to obstacles) as more crucial for its cumulative reward than pursuing the positive reward of reaching the goal. To address this, we experimented with various adjustments, such as modifying the magnitudes of both positive and negative rewards, penalizing inactivity, adjusting the decay rate in our epsilon-greedy strategy for action selection, and fine-tuning the discount factor and learning rate.

[5] Experimental parameters are learning rate = 1e-4, discount factor = 0.99, epsilon = 1e-6

[6] The hidden layer dimensions are as follows: 132, 100, 64, 32

Given these challenges, we believe that even in a seemingly simple environment, more advanced techniques, like deep reinforcement learning, are required to achieve a more balanced and effective learning process for the agent.

Table 5: Results of policy network with REINFORCE applied to the continuous space. Note Average Reward is calculated per episode. Currently, goals 2 and 3 are still being trained.

Environments	**Avg. Reward**	**Train Time**	**# of episodes**	**Status**
Goal_0	17	20 hours	30,000	Complete
Goal_1	15	40 hours	60,000	Complete
Goal_2	-2	4 days	92,000	In Progress
Goal_3	-9	6 days	27,000	In Progress

4.2 Continuous Environment

Table 5 presents the outcomes of applying the policy network in the continuous environment for each designated goal task. The results indicate success in the first two goals, as evidenced by positive average rewards per episode. Each episode consists of 1,000 steps, regardless of whether the agent achieves the goal or not.

Specifically, with Goal 0, the average reward converges at 17 after approximately 30,000 episodes, indicating that the agent has effectively solved this problem. Similarly, Goal 1 is successfully tackled using the same network, though it requires a longer training period of about 60,000 episodes. The average reward for Goal 1 is slightly lower, at 15, reflecting the increased complexity or different challenges presented in this goal compared to Goal 0. These results underscore the efficacy of our network in adapting and learning to achieve the set goals in the continuous environment to a certain extent.

However, both Goal 2 and Goal 3 are undergoing training but remain unsolved in this environment. Currently, the average reward per episode is decreasing over time, suggesting that the learning process is ongoing, yet convergence is still not achieved. The training duration for these tasks is considerably longer compared to the first two, as shown in Table 5. This extended training time is attributed to the larger observation space in these environments, which results from an increased number of obstacles. Specifically, for Goal 3, where obstacles are represented as pedestrians, we initially set the pedestrian count at 80. This count was intended to realistically simulate the crowded conditions of a sidewalk segment. But under these conditions, it took 20 hours to complete 1000 episodes. However, when we reduced the number of obstacles to 20, a less complex representation of our scenario, the training duration decreased significantly. Thus,

the training time emerged as a critical challenge in this project, significantly influencing our approach[7].

5 Discussion

Although we achieved our first two goal tasks, one of the key oversights in our project was underestimating the training duration required for reinforcement learning models. The unexpectedly lengthy training times for even basic algorithm efficacy tests limited our ability to explore more advanced methods. To enhance the effectiveness of our current pipeline, we propose several adjustments:

1. **Implementing Model Checkpoints**: Utilize model checkpoints from each goal as starting parameters for subsequent levels. This approach was initially unfeasible due to varying observation space sizes across tasks, affecting network input dimensions. To address this, we suggest standardizing the input size for all goal tasks. For instance, if the maximum observation space is 48 for the most complex task, then all tasks would have a network input of 48 neurons. For simpler tasks, the input representation, say 24, would be paddded with 24 zeros to maintain consistent dimensionality.
2. **Periodic Rewards Strategy**: Introducing intermediate rewards could enrich the agent's learning experience. Placing rewards at strategic locations on the map as the agent progresses towards the goal can serve as checkpoints. These checkpoints would not only reward progress but also facilitate the agent's exploration and navigation towards the final objective.
3. **Obstacle Navigation Rewards**: Offering rewards for successfully navigating between obstacles, with additional incentives for moving away from obstacles in the direction of the goal, could be beneficial. This strategy aims to prioritize the importance of making progress and moving past obstacles, especially in scenarios where the agent is confined between them.

By integrating these improvements, we anticipate a reduction in training times and an enhancement in the average reward per episode, thereby optimizing the overall performance of our RL models.

6 Conclusion

In our project, we set out to create an autonomous agent capable of safely navigating crowded pedestrian areas, designed to lead a disabled person. Our progress to date represents a significant stride towards achieving this goal. We have developed an autonomous agent, powered by deep reinforcement learning, that adeptly maneuvers through zones with hazardous areas and actively clears obstacles from its path to reach its destination. Additionally, we are nearing the completion of an enhancement that enables the agent to adeptly navigate around

[7] Videos of simulations of the four goal tasks can be seen in the codebase.

objects moving in circular patterns within confined spaces. Furthermore, we have made promising advancements in enabling the agent to efficiently circumnavigate particles moving with Perlin noise perturbation.

We are confident that the algorithm we have developed is proficient in navigating through continuous environments, and we anticipate that achieving complete success is primarily a function of extending the training period. If our current approach does not yield the desired results, we plan to explore the modifications outlined in the discussion section. Moreover, we'd like to consider other traditional deep reinforcement learning techniques like Deep Q-Networks (DQNs), as well as experimenting with state-of-the-art RL methodologies that are at the forefront of current research.

In order to deploy our algorithm in a real world application, we would need to both validate the performance and safety of the navigation algorithm in real world as well as implement an interface for the robot to communicate with the human. We could draw upon other work done in the field to integrate haptic audio, or physical feedback based on the robot's planned path to allow the user to intuitively follow the robot. The combination of these interfacing technologies with our algorithm would allow us to test and improve the safety and efficacy of the algorithm through real world trials. In order to safely navigate these real world intersections, we could investigate adding vehicular obstacles, crosswalk boundary identification, and traffic and crosswalk signal interpretation to our training environment. These additions would allow us to identify when the robot is allowed to start leading the use and enter certain spatial zones, depending on real pedestrian traffic laws.

References

1. Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002.
2. Seita Kayukawa, Tatsuya Ishihara, Hironobu Takagi, Shigeo Morishima, and Chieko Asakawa. Blindpilot: A robotic local navigation system that leads blind people to a landmark object. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–9, New York, NY, USA, 2020. Association for Computing Machinery.
3. AM Kassim, T Yasuno, H Suzuki, HI Jaafar, and MSM Aras. Indoor navigation system based on passive rfid transponder with digital compass for visually impaired people. *International Journal of Advanced Computer Science and Applications*, 7(2), 2016.
4. Vladimir Kulyukin, Chaitanya Gharpure, John Nicholson, and Grayson Osborne. Robot-assisted wayfinding for the visually impaired in structured indoor environments. *Autonomous Robots*, 21:29–41, 01 2006.
5. Jizhong Xiao, Samleo L. Joseph, Xiaochen Zhang, Bing Li, Xiaohai Li, and Jianwei Zhang. An assistive navigation framework for the visually impaired. *IEEE Transactions on Human-Machine Systems*, 45(5):635–640, 2015.
6. Xiaochen Zhang, Xiaoyu Yao, Yi Zhu, and Fei Hu. An arcore based user centric assistive navigation system for visually impaired people. *Applied Sciences*, 9(5):989, 2019.

7. Dorine C. Duives, Winnie Daamen, and Serge P. Hoogendoorn. State-of-the-art crowd motion simulation models. *Transportation Research Part C: Emerging Technologies*, 37:193–209, 2013.
8. Xinjie Wang, Xiaogang Jin, Zhigang Deng, and Linling Zhou. Inherent noise-aware insect swarm simulation. In *Computer Graphics Forum*, volume 33, pages 51–62. Wiley Online Library, 2014.
9. Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.
10. Jiaming Ji, Borong Zhang, Xuehai Pan, Jiayi Zhou, Juntao Dai, and Yaodong Yang. Safety-gymnasium. *GitHub repository*, 2023.
11. Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Kluwer Academic Publishers, Boston.*, pages 279–292, 1992.
12. Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992.