A Modular Framework for Socially Compliant Robot Navigation in Complex Indoor Environments

Sara Pohland
Claire Tomlin

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-36
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-36.html

May 4, 2022

A Modular Framework for Socially Compliant Robot Navigation in Complex Indoor Environments

by

Sara Pohland

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire Tomlin, Chair
Professor Sergey Levine

Spring 2022

Abstract

A Modular Framework for Socially Compliant Robot Navigation in Complex Indoor Environments

by

Sara Pohland

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Claire Tomlin, Chair

An important challenge in human-robot interaction is the design of socially compliant robot navigation policies that enable safe navigation around crowds of people. Deep reinforcement learning is a popular and effective method to predict human motion and plan paths that avoid collisions with humans while following social norms. While various deep reinforcement learning methods have proven effective for generating crowd-aware navigation policies, these policies generally assume the robot is operating around humans in a large open environment, which is not reflective of typical indoor spaces. To design a socially compliant robot navigation policy that works effectively in complex indoor spaces with walls and other stationary objects, I combined a deep reinforcement learning policy with a global path planning algorithm and a custom safety controller. Combining all these elements in a modular framework, I enabled a robot to reach its goal from an arbitrary starting position, while limiting close encounters with humans and avoiding collisions with humans and stationary objects. I found that my policy achieves an overall success rate of over 99% when tested in a diverse set of simulation environments comprising different geometries and distributions of humans and stationary obstacles. When compared against a baseline navigation policy that does not utilize learning, I found that my modular approach results in better navigation performance and greater compliance to human social norms. I also implemented my approach on a physical robot to navigate real-world indoor spaces with various humans and stationary objects. These simulation and real-world results demonstrate the value of using learning-based methods as a single component of a larger framework to develop navigation policies that are effective in real-world settings and comfortable for people.

Acknowledgments

I am so grateful for everyone who has supported me through my graduate school experience and who has contributed to the completion of my Master's thesis. I would first like to express my deepest appreciation to my committee. Many thanks to my advisor, Professor Claire Tomlin, who has consistently supported and encouraged me. Thank you so much for all of your valuable advice, unwavering support, insightful feedback, and endless encouragement. Thank you also to my second reader, Professor Sergey Levine. Your course on deep reinforcement learning, decision making, and control was invaluable in developing this thesis, and I appreciate you sharing your extensive knowledge on this research area with me.

I would also like to extend my deepest gratitude to my friend and colleague, Alvin Tan, whose invaluable contributions to this project cannot be overlooked. I very much enjoyed working with you, and I appreciate all of your insightful suggestions and useful contributions. This project certainly would not have been as enjoyable nor successful without you.

I also had the pleasure of working with many collaborators at the Johns Hopkins Applied Physics Laboratory (APL) and the Johns Hopkins University (JHU). Thank you to my former supervisor, Dr. Kapil Katyal, who helped me define my research idea and played a critical role in the early stages of my project. This project would have never begun without your unparalleled support and extensive guidance. Thank you also to my current supervisor, Dr. Jared Markowitz, who has provided me unrelenting support, valuable advice, and insightful suggestions through out the development of this project. I would also like to thank another colleague at APL, Dr. Corban Rivera, who has provided me with so much encouragement and advice, and who has made very useful contributions to this work. I would also like to extend my thanks to Dr. I-Jeng Wang. The suggestions you have given me have been very thought-provoking, and I appreciate all of your constructive criticism and helpful advice. Special thanks to Jared, Corban, and I-Jeng for offering to read my thesis and providing me with valuable feedback. I would also like to thank Professor Chien-Ming Huang at JHU for the useful suggestions and insightful feedback.

Finally, I would like to thank all of my family and friends for their unwavering support and relentless encouragement. The completion of my Master's thesis and all other successes in my graduate program would not have been possible without the constant support I have received from my parents through out graduate school and long before beginning my graduate journey. Thank you so much for all you have done for me. Lastly, I would like to thank my best friend, Adam Lee. While you have not provided technical support, the emotional support you have provided me has been just as valuable as any other's contributions. Thank you so much for your endless patience and relentless support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Socially Assistive Robots

Overview

Socially assistive robots (SARs) are defined as robots that provide some form of assistance to human users through social interaction, rather than physical interaction [16]. These robots have been used to benefit several different populations of users in various different environments. They have been found to be particularly helpful in assisting elderly populations, individuals in convalescent care, individuals with physical impairments, and individuals with social and mental disorders in hospitals, assisted living homes, and schools [16].

Robots in Healthcare Centers

SARs are commonly used in hospitals, rehabilitation centers, and other healthcare centers, where they have been found to ease the workload on the healthcare workers, allowing staff members to focus more on critical tasks and caring for patients, thus increasing the overall efficiency of these facilities [11], [30], [19]. Within healthcare facilities, SARs have been used as nursing assitants to fetch and distribute supplies, such as food trays, medicines, and laboratory specimens [11], [30]. Besides reducing the general burden on healthcare workers, SARs are particularly beneficial when dealing with patients with contagious illnesses. Unlike nurses and other staff members, robots are not vulnerable to viruses and other microorganisms. This makes SARs useful for collecting specimens from patients for disease screening, disinfecting hospital environments, delivering supplies to infected patients, and collecting information on the physiological conditions of patients who may be contagious [22].

Robots in Elderly Communities

There is also a particular interest in using SARs to aid elderly populations in nursing homes and assisted living communities. The worldwide population is rapidly aging, and it has been

projected that 21.1% of the population will be above the age of 60 years by 2050, presenting an increasing need for caregivers [30]. Despite this need, it is predicted that there will be a worldwide shortage of more than 100,000 caregivers for the elderly population by 2030 [19]. This presents a strong desire to provide support to caregivers that are increasingly burdened by caring for the ever-growing number of senior patients. SARs are able to assist older adults mentally by providing reminders, emotional support, and motivation, as well as physically by delivering items and assisting with daily tasks [30]. They have also been used as companions to enrich the social lives of elderly populations and address loneliness and isolation, which are very high in elderly populations [19], [11]. SARs are particularly useful in supporting older adults with dementia, Alzheimer's disease, and related cognitive impairments [11], [26], as well as older adults with depression [35], [19]. Overall, SARs can improve the quality of life of elderly populations, reduce depression, increase independent living, enhance well-being, and reduce isolation and loneliness [12], [19]. In addition, studies have found that elderly populations have a positive perception of SARs [12], [17].

Robots in Schools

The final application of SARs that is currently very popular is in education. SARs have been used to diagnose and assist in the education of children with social and mental disabilities [30]. Studies have shown that SARs can therapeutically interact with children who have autism, and this research can be generalized to other cognitive and behavioral disorders [16]. In addition, there is interest in using SARs to serve as tutors and coaches within school settings to support student learning [16]. These robots have also been employed as companions in elementary schools to facilitate interaction among students from different social groups and populations, especially within special education classrooms [16].

1.2 Socially Compliant Robot Navigation

Socially Compliant Behavior

It is desirable for SARs to behave in a way that is socially compliant. In designing socially compliant robot behavior, there are four main goals to consider: comfort, naturalness, effectiveness, and sociability. Comfort refers to the absence of annoyance and stress for humans while interacting with and around robots [28]. Note that comfort is different from safety because humans may not feel comfortable around a robot, even if it is moving in a way that avoids collisions. Rather than simply seeking to avoid collisions, a socially compliant robot should keep a large enough distance between itself and nearby humans to prevent emotional discomfort. Discomfort can also be reduced by appropriately changing the velocity of the robot when it approaches a human and encouraging the robot to approach humans from the front, among other methods. Naturalness refers to the similarity between robots and humans in low-level behavior patterns [28]. A robot that moves naturally seeks to mimic the

navigation patterns of humans. Effectiveness refers to the ability of the robot to complete desired tasks. Finally, sociability refers to the adherence of robot policies to explicit high-level cultural conventions [28]. This aspect of social compliance relates to very particular robot behaviors and is not commonly studied in robot navigation.

Human Trajectory Models

One branch of work in socially compliant robot navigation attempts to explicitly model the trajectories of pedestrians in the environment and plan a path around these trajectories. Within this area, there has been work on capturing relevant aspects of human trajectories to determine the probability distribution that underlies human navigation behavior [29]. Another approach uses a feature-based maximum entropy model to predict joint behavior of heterogeneous groups of agents from onboard data [34]. There has also been work to model humans in terms of mixture distributions that capture both discrete navigation decisions and the natural variance of human trajectories [27]. More recently, there have been efforts to models danger zones for the robot by considering all possible actions that humans can take at a given time [37]. While explicitly modeling pedestrian trajectories works well in settings with very few pedestrians, it does not scale well to environments with a greater number of people. It has become very popular to use reinforcement learning methods because they have been shown to generate socially compliant behavior in larger crowds of pedestrians [7]. There are two main branches of reinforcement learning that have been explored: deep reinforcement learning and interactive reinforcement learning.

Deep Reinforcement Learning

As mentioned, one very popular approach to generating socially compliant robot navigation algorithms is to use deep reinforcement learning, through which the robot seeks to learn a policy that maximizes its expected accumulation of a developer-designed reward through trial and error. This approach generally focuses on the comfort aspect of socially compliant robot navigation algorithms by encoding comfort criteria into the reward function. Among the deep reinforcement learning approaches, one of the earlier ones developed a decentralized multi-agent collision avoidance algorithm that offloads interaction pattern predictions to an offline learning procedure [8]. Building on this work, there was later a method proposed that uses long short-term memory (LSTM) to enable the algorithm to use observations of an arbitrary number of other agents [14]. Another algorithm inspired by the original decentralized multi-agent collision avoidance algorithm presented a multi-scenario multi-stage training framework in an effort to reduce the performance gap between decentralized and centralized methods [32]. Another expansion of this work sought to focus on avoiding violations of social norms, rather than specifying precise mechanisms of human navigation [9]. The decentralized learning approach was later expanded to enable the network to identify and pay attention to the humans in the crowd that are most critical to navigation [10]. After this paper, there was a very popular paper that used an attentive pooling mechanism to learn

the collective importance of neighboring humans with respect to their future states [7]. This paper then inspired policies that consider both static and dynamic objects [31], as well as policies that focus on robot navigation behavior that follows group social norms [24].

Interactive Reinforcement Learning

Another very popular approach to generating socially compliant robot navigation algorithms uses interactive reinforcement learning, through which a human delivers explicit or implicit feedback to the agent. This approach generally focuses on the naturalness aspect of socially compliant robot navigation algorithms because the robot seeks to mimic the human demonstrator. Within this area of work, various methods have been explored to learn a good reward function. One approach uses a flexible graph-based representation of socially normative human behavior and extends Bayesian inverse reinforcement learning to use sampled trajectories from this representation [33]. Another approach used to learn a good reward function uses maximum entropy deep inverse reinforcement learning (MEDIRL) [15]. Most recently, there have been efforts to learn a reward function while eliminating the additional sample complexity associated with inverse reinforcement learning [2]. Another area of work in interactive reinforcement learning uses a generative adversarial imitation learning (GAIL) strategy, which improves upon a pre-trained behavior cloning policy to mimic human behavior [40]. There has also been a limited amount of work that combines deep reinforcement learning and interactive reinforcement learning methods to generate robot behavior that is both comfortable and natural [41].

Comparison of Navigation Methods

As mentioned previously, explicitly modeling pedestrian trajectories does not scale well to environments with a large number of people. While both deep reinforcement learning methods and interactive reinforcement learning methods have been effective in generating socially compliant robot navigation policies in environments with an arbitrary number of pedestrians, there are benefits and drawbacks of both approaches. While deep reinforcement learning methods have been effective at generating robot behaviors that are safe and comfortable to the user, it is difficult to design a reward function that can generate a policy that is both comfortable and natural [41]. On the other hand, interactive reinforcement learning can generate robot behavior that is natural, but a lack of negative examples, such as collisions, can lead to a model that only focuses on naturalness without considering comfort or safety [41]. Interactive reinforcement learning also has the drawback that manual feature engineering is necessary to get reasonable performance [41], [1]. Another interesting consideration to note is that policies that aim to mimic human behavior may lead to unfair behavior by replicating, promoting, and amplifying societal issues, such as discrimination and segregation [21]. It is also important to consider whether comfort or naturalness is a better measure of social compliance for robot navigation algorithms. Because robots are not humans, people do not necessarily expect robots to operate exactly like a human would and may be distrustful of

their behavior. For this reason, I would argue that is more important to focus on the comfort aspect of socially compliant robot navigation algorithms. With all of these aspects considered, I believe that deep reinforcement learning is the most promising direction to explore when developing socially compliant robot navigation policies.

1.3 Path Planning Algorithms

Overview

Path planning algorithms are used to find safe and efficient paths that autonomous systems can follow from their initial location to a target destination while avoiding collisions with obstacles [23]. Path planning algorithms can generally be divided into two categories: global planners and local planners [6]. These planners differ based on their assumptions of the environmental information that the robot can acquire during the navigation process. Global planners generally only use a static global map to generate a collision-free path to the goal position that remains fixed throughout the navigation process. Local planners use knowledge of dynamic obstacles to generate shorter paths that may change during the navigation process. I will discuss popular global and local planners, focusing on foundational algorithms. There has been additional research that expands on many of these algorithms to improve their performance or efficiency, but I will avoid discussion of these variations.

Global Planners

Among global planners, some of the first and most widely explored algorithms rely on grid-based search. These algorithms include the Dijkstra algorithm [13], the A* algorithm [20], and the D* algorithm [38], along with their many variations (some of which are discussed in [23]). The Dijkstra algorithm is one of the first proposed algorithms that is still used today, which constructs the tree of minimum total length between nodes in the space and finds the path of minimum length between any two nodes [13]. This algorithm performs blind searches, making it computationally intensive. To address this issue, the A* algorithm incorporates heuristic information from the problem domain into the graph searching algorithm [20]. While this algorithm works well in environments in which the global map is well known, it does not perform well if the map of the environment is not well known before planning. To allow for changes in the global map, the D* algorithm provides a method for efficiently replanning when updates in the map are received [38].

While grid-based search algorithms are effective and widely used in relatively simple two-dimensional environments, they are generally too computationally expensive to use in large, complex environments. This limitation led to research on random sampling algorithms. Among random sampling algorithms, the most popular is Rapidly-exploring Random Trees (RRTs), which is a randomized data structure that is iteratively expanded by applying control inputs that drive the system towards randomly selected points [39]. Because RRTs

are efficient and effective in dynamic and high-dimensional environments, there has been an extensive amount of research on methods to expand and improve on this approach.

Another class of algorithms that were designed to be used in high-dimensional environments are roadmap planners. Among these are the Probabilistic Roadmap (PRM) planner [25] and the Voronoi Roadmap planner [5]. The PRM planner constructs a probabilistic roadmap of feasible paths between collision-free configurations, then searches this map for a path joining the initial and goal positions of the robot [25]. The Voronoi Roadmap planner performs a similar procedure, utilizing a Voronoi diagram [5].

Local Planners

Compared to global planners, there has been less research into local planners. The three main local planners that are most foundational are the Velocity Obstacle (VO) planner [18], the Reciprocal Velocity Obstacle (RVO) planner [3], and the Optimal Reciprocal Collision Avoidance (ORCA) planner [4]. The VO planner generates maneuvers to avoid static and moving obstacles by selecting robot velocities outside of the set that would result in a collision with an obstacle that moves at a fixed velocity for some time [18]. While this method considers passively moving obstacles, the RVO planner takes into account the reactive behavior of other agents by assuming that the other agents follow similar collision-avoidance procedures [3]. The ORCA planner is an expansion of this work, which allows the robot to avoid collisions with moving obstacles while obeying acceleration constraints [4].

1.4 Project Goals

In order to employ SARs in real-world environments, they need to be equipped with socially compliant navigation policies that perform effectively in complex indoor spaces with a variable number of people and stationary objects. While there has been a significant amount of research conducted on socially compliant robot navigation algorithms, most of these algorithms have been trained and tested in large open environments without stationary obstacles, greatly limiting their application to real-world environments. On the other hand, path planning algorithms can be employed in more complex environments if a map of the environment is known, but they do not consider the need for socially compliant behavior. My goal is to combine the benefits of these approaches and enable a robot to navigate in complex indoor spaces while following social norms and avoiding collisions with humans and stationary objects. With this goal in mind, I design a modular framework that uses a socially-aware deep reinforcement learning policy as a single component.

Chapter 2

Deep Reinforcement Learning

2.1 Overview of Reinforcement Learning

Reinforcement learning (RL) can be defined as a mathematical formalism for learning-based decision making. While controllers and policies governing a system are often designed by hand using a model of the system, RL is an approach for decision making and control used to develop a controller or policy from experience. Suppose there is an agent in some environment with an assigned task. Rather than explicitly telling the agent how to complete its task, the agent learns how to complete the task through trial and error by interacting with its environment. The agent is able to receive an observation reflecting the state of the environment, take an action, and receive the resulting reward and a new observation from the environment. Based on the rewards it receives, the agent learns which actions are best to take for a given observation. Figure 2.1 below depicts this learning process.

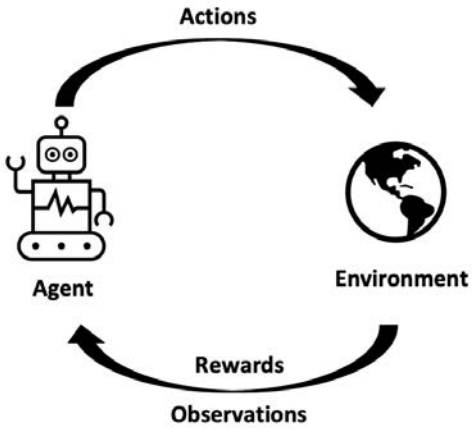

Figure 2.1: Overview of the reinforcement learning process.

Neural networks (NNs) are a very popular mathematical structure that can effectively approximate nonlinear relationships between data and labels. The model of this relationship can then be used to label previously unseen data. A neural network model is learned through optimization of a cost function, tuned via hyperparameters. The model architecture specifies the flow of information between the network layers, which defines the composition of functions that the network performs from input to output. The model aims to minimize a cost function, which depends on the true labels and predicted values, and the optimization algorithm is used to minimize the cost function. The neural network also depends on a set of hyperparamters, including the learning rate and batch size, among others.

Deep learning is a branch of machine learning that uses neural networks with multiple layers. Deep reinforcement learning is the combination of deep learning and reinforcement learning, which allows agents to make decisions from unstructured input data without manually engineering the feature space. The benefit of deep reinforcement learning algorithms is that they are able to receive very large and complex sensory inputs and decide what actions to perform to optimize an objective. These algorithms are able to make decisions based on data, without the need for well-defined rules about what actions to take in different scenarios.

2.2 Reinforcement Learning Problem

The RL problem can be formulated as an optimization problem, using the notation below:

$\mathcal{S}$ – state space describing the set of all possible states of the system

$\mathcal{O}$ – observation space describing the set of all possible observations of the system

$\mathcal{A}$ – action space describing the set of all possible actions performed by the agent

$s_t \in \mathcal{S}$ – true state of the system at time t

$o_t \in \mathcal{O}$ – observed state of the system at time t

$a_t \in \mathcal{A}$ – action take by the agent at time t

$p(s_{t+1}|s_t, a_t)$ – probability of the next state given the current state and action

$p(o_t|s_t)$ – probability of an observation given the current state

$\pi_\theta(a_t|o_t)$ – probability of taking an action given the current observation

θ – set of parameters defining the policy governing the agent

$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – reward function

$r(s_t, a_t)$ – reward earned for taking some action from the current state

Figure 2.2 depicts the relationship between the components of the RL problem.

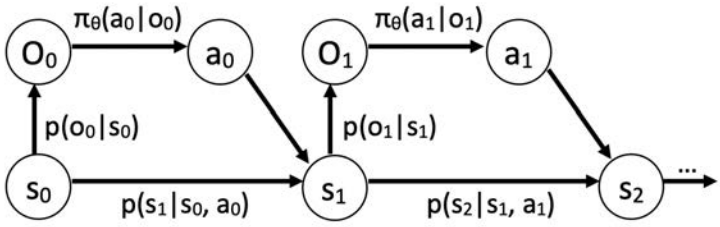

Figure 2.2: State transitions in the reinforcement learning decision process.

The goal of reinforcement learning is to find the best parameters θ^*, which define the optimal policy $\pi_{\theta^*}(a_t|o_t)$, such that the optimal policy maximizes the expected sum of rewards. Note that, for some parameters θ, the policy $\pi_\theta(a_t|o_t)$ governing the action of the agent for a given observation is a probability distribution over the action space, conditioned on the observation. When the agent is running its policy, it receives an observation o_t and samples an action a_t from the distribution $\pi_\theta(a_t|o_t)$. The environment then responds with a reward $r(s_t, a_t)$. Using the probability distributions defined, the state transition probability is

$$p_\theta(s_{t+1}, a_t, o_t, s_t) = p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|o_t)p(o_t|s_t)p(s_t)$$

Often, we are interested in the sequence of state transitions over T discrete time steps. I will use τ to denote this trajectory of T state transitions, which can be defined as

$$\tau = \big\{(s_1, a_0, o_0, s_0), \ldots, (s_T, a_{T-1}, o_{T-1}, s_{T-1})\big\}$$

The probability distribution of the trajectory induced by the policy is then

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|o_t)p(o_t|s_t)$$

With this notation, the reinforcement learning problem can formally be expressed as

$$\max_\theta \ \mathbb{E}_{\tau\sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} r(s_t, a_t)\right]$$

2.3 Value Network Algorithms

Value Functions

To solve the RL problem, it is useful to define the Q function as the total expected reward earned over time for taking action a_t from state s_t, which is expressed as

$$Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{(s_{t'}, a_{t'}) \sim p_\theta(s_{t'}, a_{t'} | s_t, a_t)} \big[r(s_{t'}, a_{t'}) \big]$$

It is also useful to define the value function as the expected value of the Q function over all possible actions sampled from the policy $\pi_\theta(a_t|o_t)$, which is the total expected reward from the given state s_t. The value function can be expressed more formally as

$$V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \big[Q^{\pi}(s_t, a_t) \big]$$

Target Values

In value network algorithms, a neural network is used to obtain a model $\hat{V}^{\pi}_{\phi}$ of the value function V^{π}. If the length T of each episode is large, the value function can get infinitely large. To deal with this issue, it is beneficial to use a discount factor $\gamma \in [0, 1]$ with the idea that the policy should favor more immediate rewards. The Q function is then defined as

$$\begin{aligned}
Q^{\pi}(s_t, a_t) &= \sum_{t'=t}^{T} \gamma^{t'-t} \mathbb{E}_{(s_{t'}, a_{t'}) \sim p_\theta(s_{t'}, a_{t'} | s_t, a_t)} \big[r(s_{t'}, a_{t'}) \big] \\
&= r(s_t, a_t) + \sum_{t'=t+1}^{T} \gamma^{t'-t} \mathbb{E}_{(s_{t'}, a_{t'}) \sim p_\theta(s_{t'}, a_{t'} | s_t, a_t)} \big[r(s_{t'}, a_{t'}) \big] \\
&= r(s_t, a_t) + \gamma \sum_{t'=t+1}^{T} \gamma^{t'-(t+1)} \mathbb{E}_{(s_{t'}, a_{t'}) \sim p_\theta(s_{t'}, a_{t'} | s_t, a_t)} \big[r(s_{t'}, a_{t'}) \big] \\
&= r(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}, a_{t+1}) \sim p_\theta(s_{t+1}, a_{t+1} | s_t, a_t)} \big[Q^{\pi}(s_{t+1}, a_{t+1}) \big] \\
&= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p_\theta(s_{t+1} | s_t, a_t)} \big[V^{\pi}(s_{t+1}) \big]
\end{aligned}$$

If the next state s_{t+1} is known, then the Q function can be expressed as

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) \approx r(s_t, a_t) + \gamma \hat{V}^{\pi}_{\phi}(s_{t+1})$$

As mentioned, the goal of value function learning is to find a model $\hat{V}^{\pi}_{\phi}$ that best approximates the true value function V^{π}. Therefore, the target value y_t for a state s_t is

$$y_t = V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \big[Q^{\pi}(s_t, a_t) \big] \approx \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \big[r(s_t, a_t) + \gamma \hat{V}^{\pi}_{\phi}(s_{t+1}) \big]$$

Optimal Policy

The Q function and value function can be used to measure the advantage of taking action a_t from state s_t. The advantage function is defined as the Q function minus the value function and can be viewed as a measure of how the action a_t compares to the average over all of the actions obtained from the policy $\pi_\theta(a_t|o_t)$. The optimal action for a given state s_t is then

$$a_t^* = \arg\max_{a_t \in \mathcal{A}} \Big(Q^\pi(s_t, a_t) - V^\pi(s_t) \Big) = \arg\max_{a_t \in \mathcal{A}} \; Q^\pi(s_t, a_t) \approx \arg\max_{a_t \in \mathcal{A}} \Big(r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1}) \Big)$$

For this choice of action, the conditional probability $\pi_\theta(a_t|s_t)$ is implicitly defined as

$$\pi_\theta(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \arg\max_{a_t \in \mathcal{A}} \Big(r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1}) \Big) \\ 0 & \text{otherwise} \end{cases}$$

If it assumed that the policy π_θ always chooses the optimal action, the target is simply

$$y_t \approx \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \big[r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1}) \big] = r(s_t, a_t^*) + \gamma \hat{V}_\phi^\pi(s_{t+1})$$

The loss function used to update the value network parameters will be denoted ℓ, thus the value network seeks to minimize the objective $\ell\big(\hat{V}_\phi(s_t), y_t\big)$. Typically, mean squared error (MSE) is used as the loss function, but other functions may be used as well. Given a positive learning rate α, the parameters of the value network are updated using gradient descent:

$$\phi \leftarrow \phi - \alpha \frac{d}{d\phi} \ell\Big(\hat{V}_\phi(s_t), y_t \Big)$$

Epsilon-Greedy Policy

While the optimal action is found by computing the argmax of the reward plus the discounted value function over the action space, this may not be the best choice of action when modeling the value function using a neural network. If the model of the value function does not provide a good reflection of the true value function, the "optimal" policy may learn bad actions that never lead to high rewards. A better policy to use during training is the epsilon-greedy policy, which takes the action believed to be optimal with probability $1 - \epsilon$ and some other action in the action space with probability ϵ. For this policy, $\pi_\theta(a_t|s_t)$ is given by

$$\pi_\theta(a_t|s_t) = \begin{cases} 1 - \epsilon & \text{if } a_t = \arg\max_{a_t \in \mathcal{A}} \Big(r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1}) \Big) \\ \frac{\epsilon}{|\mathcal{A}| - 1} & \text{otherwise} \end{cases}$$

By using this policy during training, the agent can explore more states, improving its model of the value function. Often, ϵ is chosen to decay over time as the model of the value function improves. This is done to balance the exploration-exploitation trade-off. The robot is able to explore more states early in training, then exploit its model towards the end of training. Once the value function model has been trained and can provide a descent estimate of the true value function, the agent can be deployed using the optimal policy.

Replay Buffer

Because sequential states are strongly correlated, iterative algorithms that rely on state transitions from entire trajectories can lead to overfitting. One way to address this issue is to use a replay buffer to store state transitions, consisting of the initial state s, the action a, and the next state s' after taking this action. Using this solution, the algorithm will obtain a batch of M state transitions sampled randomly from the replay buffer at each iteration and will use this batch to update the model of the value function. Now that the samples are chosen randomly, rather than sequentially, the samples are no longer highly correlated. I will use s_i to denote the initial state, a_i to denote the action, and s'_i to denote the the next state in the ith sample. The target value y_i corresponding to this sample is now

$$y_i = r(s_i, a_i) + \gamma \hat{V}^{\pi}_{\phi}(s'_i)$$

The parameters of the value network will now be updated using stochastic gradient descent:

$$\phi \leftarrow \phi - \alpha \frac{1}{M} \sum_{i=1}^{M} \frac{d}{d\phi} \ell\Big(\hat{V}_{\phi}(s_i), y_i\Big)$$

Target Network

Notice that because the target value given previously depends on the model of the value function, it changes with each gradient descent step. Because the target value is constantly changing, it is difficult for the algorithm to converge to the true value function. One way to address this issue is to use a target network to obtain an additional model of the value function to be used when computing the target value. The target network is designed to track the value network, but this network updates more slowly. This ensures that the target values do not change in the inner loop of the algorithm. Now the target network will estimate the value function using the model $\hat{V}^{\pi}_{\phi'}$ and the target values will be defined by

$$y_i = r(s_i, a_i) + \gamma \hat{V}^{\pi}_{\phi'}(s'_i)$$

Algorithm Parameters

In the value network algorithm, there are several parameters that can be chosen for the specific RL problem: the discount factor γ, learning rate α, batch size M, number of training episodes N, target update interval τ_{TU}, and epsilon ϵ. As mentioned previously, the discount factor γ is used to capture the idea that the agent should favor more immediate rewards. If the discount factor is one, the agent gives equal weight to the reward at each time step of the trajectory, and as the discount factor is decreased, the agent becomes more short-sighted. As shown previously, the learning rate α is used in gradient descent and determines the rate at which the value network weights are updated. The batch size M is the number of state

transitions sampled from the replay buffer to update the value network parameters at each iteration. The number of training episodes N is the number of times the full set of steps in the value network algorithm is run. The target update interval τ_{TU} specifies the frequency with which the the target network is updated using the value function network parameters. Finally, the parameter ϵ used in the epsilon-greedy policy determines how often the agent is expected to take an exploratory action and how often it will take the action believed to be optimal. If epsilon is zero, the agent will always take the optimal action, and if it is one, it will always take a random action. As mentioned previously, it is generally desirable to choose epsilon to decay to a value closer to zero as training progresses.

Algorithm Pseudocode

Algorithm 1 provides the general structure of the value network algorithm.

Algorithm 1 Value Network Algorithm

1: Initialize the parameters ϕ and ϕ' of the value network and target network.
2: Initialize the replay buffer using state transitions from some policy.
3: **for** episode $= 1 \rightarrow N$ **do**
4: Set $t \leftarrow 0$
5: **repeat**
6: Determine the optimal action for the current state using the epsilon greedy policy:

$$a_t^* \sim \pi_\theta(a_t|s_t)$$

7: Take the optimal action, observe the state transition, and add it to the replay buffer.
8: Randomly sample a batch of M state transitions from the replay buffer.
9: Compute the target value for each of the M samples in the batch:

$$y_i = r(s_i, a_i) + \gamma \hat{V}_{\phi'}^{\pi}(s_i'),\ i = 1, \ldots, M$$

10: Update the value function network parameters using stochastic gradient descent:

$$\phi \leftarrow \phi - \alpha \frac{1}{M} \sum_{i=1}^{M} \frac{d}{d\phi} \ell\Big(\hat{V}_\phi(s_i), y_i\Big)$$

11: Increment the current time step: $t \leftarrow t + 1$.
12: **until** s_t reaches a terminal state or $t \geq t_{max}$
13: **if** episode mod $\tau_{TU} = 0$ **then**
14: Update the target network parameters: $\phi' \leftarrow \phi$.
15: **end if**
16: **end for**
17: **return** $\hat{V}_\phi$

Chapter 3

Proposed Robot Navigation Policy

3.1 Problem Formulation

Goals of the Policy

The goal of my socially compliant robot navigation policy is to enable a robot to efficiently navigate in complex indoor spaces while following social norms and maintaining safety. More specifically, from any arbitrary starting position, the robot should be able to navigate to any predefined goal position, while limiting uncomfortable encounters with humans and avoiding collisions with humans, small stationary objects, and walls. Because SARs generally operate in familiar, known environments, I assume the robot will have access to a map of the walls in its environment during navigation. Most real-world robots are also equipped with cameras or motion sensors used for odometry measurements, so I assume the robot will also have some knowledge of its own state. In addition, it can be assumed that the robot has cameras or lidar sensors that allow it to detect and track obstacles in its field of view, so the robot will also have some knowledge of the state of nearby humans and objects.

State Space

The full state of the robot, the full state of the ith human, and the full state of the jth stationary object are represented by $s^{(r)}$, $s^{(h_i)}$, and $s^{(o_j)}$ respectively and are given below:

$$s^{(r)} = \begin{bmatrix} p_x^{(r)} & p_y^{(r)} & v_x^{(r)} & v_y^{(r)} & r^{(r)} & g_x^{(r)} & g_y^{(r)} & v_{pref}^{(r)} & \theta^{(r)} \end{bmatrix}$$

$$s^{(h_i)} = \begin{bmatrix} p_x^{(h_i)} & p_y^{(h_i)} & v_x^{(h_i)} & v_y^{(h_i)} & r^{(h_i)} & g_x^{(h_i)} & g_y^{(h_i)} & v_{pref}^{(h_i)} & \theta^{(h_i)} & 1 \end{bmatrix}$$

$$s^{(o_j)} = \begin{bmatrix} p_x^{(o_j)} & p_y^{(o_j)} & v_x^{(o_j)} & v_y^{(o_j)} & r^{(o_j)} & g_x^{(o_j)} & g_y^{(o_j)} & v_{pref}^{(o_j)} & \theta^{(o_j)} & 0 \end{bmatrix}$$

where (p_x, p_y) is the current position, (v_x, v_y) is the velocity, r is the radius, (g_x, g_y) is the goal position, v_{pref} is the preferred velocity, and θ is the turning angle corresponding to the

robot, human, or stationary object. The current position, velocity, goal position, and turning angle are all measured with respect to a fixed world frame. The radius is measured assuming that each robot, human, and object can be represented using a circle with a fixed radius. The preferred velocity is the maximum possible speed of each agent, which is the speed it would travel at if nothing is obstructing its path. The last term of the human state $s^{(h_i)}$ and object state $s^{(o_j)}$ is a flag used to distinguish between humans and stationary objects. If the environment contains N humans and M objects, the joint state is

$$s = \begin{bmatrix} s^{(r)} & s^{(h_1)} & \dots & s^{(h_N)} & s^{(o_1)} & \dots & s^{(o_M)} \end{bmatrix}$$

Observation Space

It is assumed that the robot has access to its full state but only has knowledge of a portion of the state of each human and object. The observable state of the robot, ith human, and jth object are represented by $o^{(r)}$, $o^{(h_i)}$, and $o^{(o_j)}$ respectively, as given below:

$$o^{(r)} = \begin{bmatrix} p_x^{(r)} & p_y^{(r)} & v_x^{(r)} & v_y^{(r)} & r^{(r)} & g_x^{(r)} & g_y^{(r)} & v_{pref}^{(r)} & \theta^{(r)} \end{bmatrix}$$

$$o^{(h_i)} = \begin{bmatrix} p_x^{(h_i)} & p_y^{(h_i)} & v_x^{(h_i)} & v_y^{(h_i)} & r^{(h_i)} & 1 \end{bmatrix}$$

$$o^{(o_j)} = \begin{bmatrix} p_x^{(o_j)} & p_y^{(o_j)} & v_x^{(o_j)} & v_y^{(o_j)} & r^{(o_j)} & 0 \end{bmatrix}$$

If the environment contains N humans and M objects, the complete observable state is

$$o = \begin{bmatrix} o^{(r)} & o^{(h_1)} & \dots & o^{(h_N)} & o^{(o_1)} & \dots & o^{(o_M)} \end{bmatrix}$$

In existing socially-aware RL policies used for robot navigation, it is generally assumed that all of the humans and stationary objects are visible to the robot at all times. However, in real-world environments, the robot is generally not able to view all of the obstacles it will encounter at all times during navigation. To make my RL policy viable for real-world environments, I created an option to restrict the observations of the robot to only those that could be realistically obtained. Assuming a 360 degree field of view with a four meter detection range, I ensure that only humans and stationary objects that are within the robot's detection range are visible to the robot. I also prevent the robot from viewing humans and objects that are obstructed by another human, object, or wall.

Action Space

The robot receives x and y velocity commands as actions under the assumption that the velocity of the robot can be achieved instantly after the action command is received. The

action space is discretized into 5 speeds exponentially spaced in the range $(0, v_{pref}^{(r)}]$ and 16 rotations evenly spaced in the range $[0, 2\pi)$. The set $\mathcal{V}$ of speeds and set Φ of rotations are

$$\mathcal{V} = \left\{ \left(\frac{e^{1/5} - 1}{e - 1} \right) v_{pref}^{(r)}, \left(\frac{e^{2/5} - 1}{e - 1} \right) v_{pref}^{(r)}, \left(\frac{e^{3/5} - 1}{e - 1} \right) v_{pref}^{(r)}, \left(\frac{e^{4/5} - 1}{e - 1} \right) v_{pref}^{(r)}, v_{pref}^{(r)} \right\}$$

$$\Phi = \left\{ 0, \frac{\pi}{16}, \frac{\pi}{8}, \frac{3\pi}{16}, \frac{\pi}{4}, \frac{5\pi}{16}, \frac{3\pi}{8}, \frac{7\pi}{16}, \frac{\pi}{2}, \frac{9\pi}{16}, \frac{5\pi}{8}, \frac{11\pi}{16}, \frac{3\pi}{4}, \frac{13\pi}{16}, \frac{7\pi}{8}, \frac{15\pi}{16} \right\}$$

The robot is also able to receive a stop command, resulting in 81 possible discrete actions. Figure 3.1 depicts all 81 of the actions available in the discrete action space.

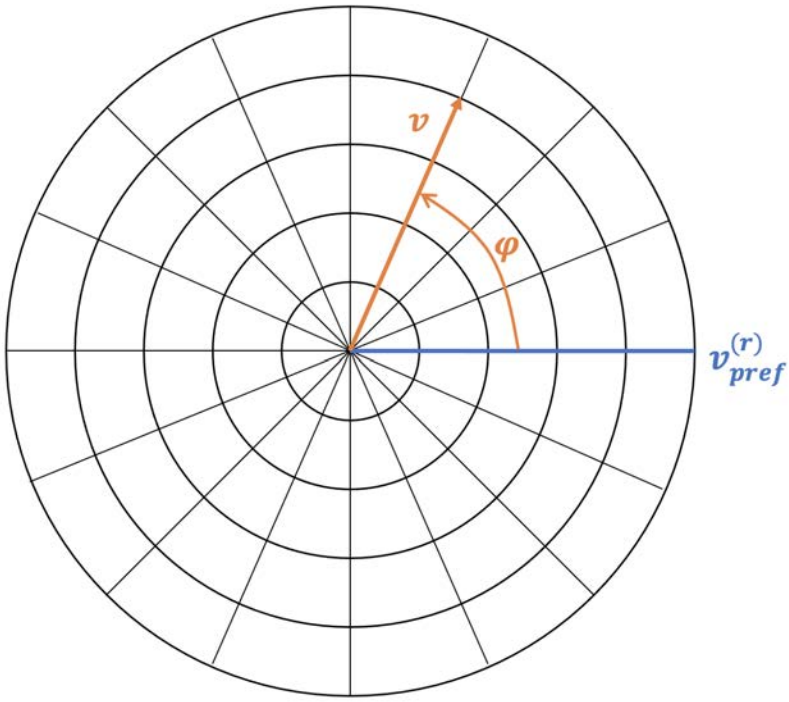

Figure 3.1: Aside from the stop command, each of the discrete actions in the action space is the combination of a speed $v \in \mathcal{V}$ and a rotation $\phi \in \Phi$. Each ring on the wheel shown represents an available speed, and each spoke is a possible rotation. Every point where a circle and line intersect is an available action in the discrete action space.

Simulation Environment

I designed a simulation environment built on OpenAI Gym, which is based on the CrowdNav simulation environment.[1] I modified this environment to allow for the distinct representation of randomized humans, stationary objects, and walls. An example is shown in Figure 3.2.

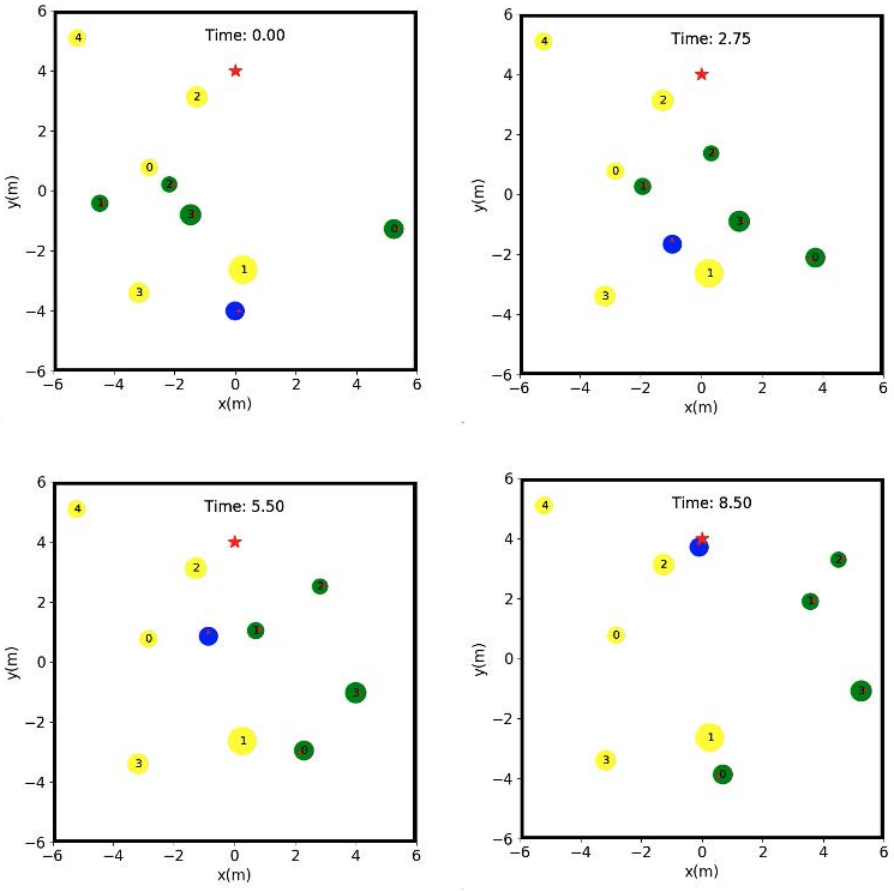

Figure 3.2: This is an example of the simulation environment used to train the robot navigation policy that only includes boundary walls shown in black. The goal of the robot (blue circle) is to move from its starting position at the bottom of the room to the goal (red star) at the opposite side of the room. In this example, there are 4 humans shown in green with random initial and goal positions, and there are 5 stationary objects shown in yellow with random positions. This example is shown across four different times along the robot's path towards the goal.

[1] `https://github.com/vita-epfl/CrowdNav`

3.2 Overview of Approach

As mentioned in Section 1.2, deep RL algorithms are effective for designing socially compliant robot navigation policies that allow robots to navigate simple, open spaces while following human social norms and avoiding collisions with humans. However, existing socially-aware RL policies cannot generalize to complex indoor environments with walls and other stationary objects. On the other hand, standard path planning algorithms are able to generate paths that the robot can take towards its goal in more complex environments, assuming a map of the environment is known. However, these algorithms treat humans as simple obstacles and are not designed to generate socially compliant behavior.

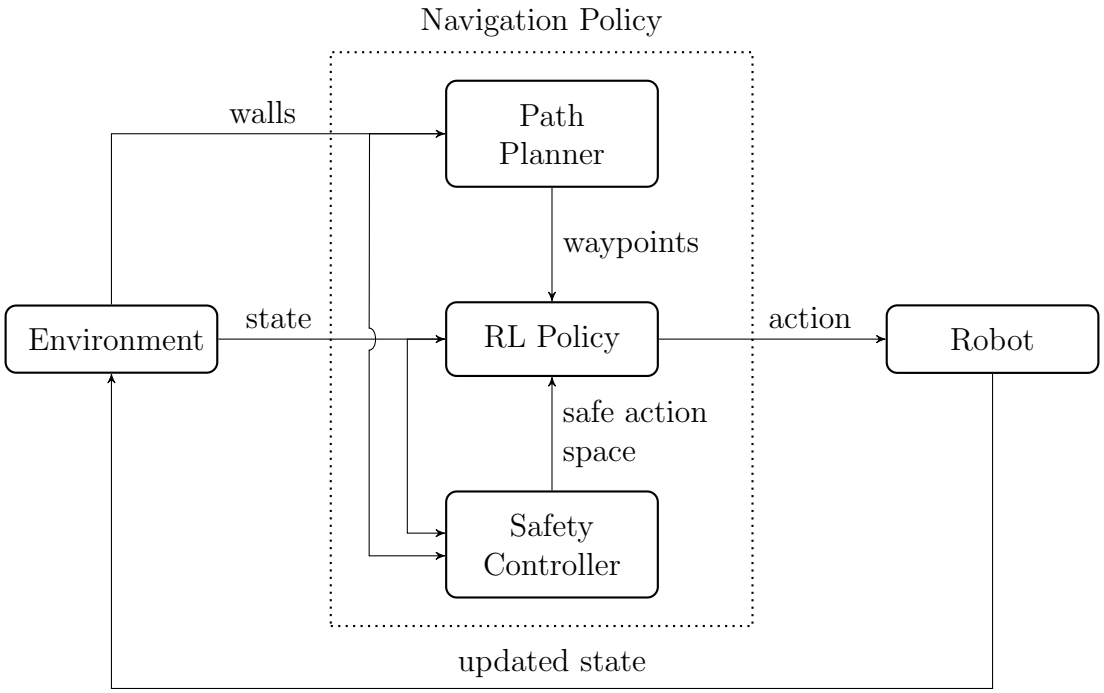

Figure 3.3: My policy is composed of three components: a socially attentive RL policy, a global path planning algorithm to choose waypoints, and a safety controller to determine the safe action space.

I would like to combine the benefits of each of these approaches by designing a socially compliant robot navigation policy that can be employed in complex indoor environments. My goal is to train an RL policy that uses knowledge only of humans and small stationary objects in randomly generated open environments. At the same time, I explore path planning algorithms that use a map of the environment, which includes walls but does not contain humans or small stationary objects. I then use waypoints from the path planning algorithm to generate local goals for the RL policy. Because the RL policy does not receive information about walls, I then design a separate safety controller that uses a map of walls in the robot's

environment to determine a safe action space to be used by the RL policy. The general architecture of my policy is shown in Figure 3.3, and all of the components of this architecture are described in the following sections.

I choose to use this modular architecture to leverage the benefits of deep reinforcement learning for socially compliant navigation without overloading the capabilities of the neural network. The most effective socially compliant robot navigation algorithms that depend on reinforcement learning focus on near-term interactions with humans within close proximity of the robot. Conversely, global path planning algorithms seek to address longer-horizon considerations driven by the environment layout and the location of the robot's ultimate goal. Relying on a single RL policy to address both of these goals simultaneously would likely lead to an undesirable trade-off between these two objectives. For this reason, combining an RL policy with a traditional global path planner seems most reasonable. In addition, it is difficult to incorporate useful information about the environment layout into the RL policy effectively. This presents the need for a separate safety controller specifically designed to avoid collisions with walls. Therefore, I believe a modular framework that uses an RL policy as a single component would be the most effective approach.

3.3 Reinforcement Learning Policy

Reward Function

Recall that in reinforcement learning, the agent learns a new policy by maximizing a reward function. To design a policy that enables a robot to reach its goal, while avoiding collisions and maintaining social norms, I design a reward function that depends on the the following:

Position of the robot at time step t – $\mathbf{p}_t^{(r)} = \left(p_x^{(r)}, p_y^{(r)}\right)_t$

Position of human i at time step t – $\mathbf{p}_t^{(h_i)} = \left(p_x^{(h_i)}, p_y^{(h_i)}\right)_t$

Position of object j at time step t – $\mathbf{p}_t^{(o_j)} = \left(p_x^{(o_j)}, p_y^{(o_j)}\right)_t$

Goal of the robot at time step t – $\mathbf{g}_t^{(r)} = \left(g_x^{(r)}, g_y^{(r)}\right)_t$

Current distance of robot from goal – $d_{goal} = \left|\left|\mathbf{p}_t^{(r)} - \mathbf{g}_t^{(r)}\right|\right|_2 - r^{(r)}$

Distance of robot to closest human – $d_{hum} = \min_{i \in \{1,\ldots,N\}} \left(\left|\left|\mathbf{p}_t^{(r)} - \mathbf{p}_t^{(h_i)}\right|\right|_2 - r^{(r)} - r^{(h_i)}\right)$

Distance of robot to closest object – $d_{obj} = \min_{j \in \{1,\ldots,M\}} \left(\left|\left|\mathbf{p}_t^{(r)} - \mathbf{p}_t^{(o_j)}\right|\right|_2 - r^{(r)} - r^{(o_j)}\right)$

Using these definitions, the reward at time step t is given by

$$r_t = k_{succ} H(r_{goal} - d_{goal}) + k_{obj} H(-d_{obj}) + k_{hum} H(-d_{hum}) + k_{disc} d_{hum} H(d_{disc} - d_{hum})$$

Note that $H(x)$ is the Heaviside step function, or the unit step function, which is one if x is greater than zero and zero otherwise. The first term of the reward function rewards the robot for reaching the goal within some radius, the second term penalizes the robot for colliding with a stationary object, the third term penalizes the robot for colliding with a human, and the fourth term penalizes the robot for getting within an uncomfortable distance of a human. The constants used in this reward function are summarized in Table 3.1.

Parameter	Value
Success Reward (k_{succ})	1.0
Goal Radius (r_{goal})	0.3
Human Collision Penalty (k_{hum})	-0.25
Object Collision Penalty (k_{obj})	-0.15
Discomfort Penalty Factor (k_{disc})	0.5
Discomfort Distance (d_{disc})	0.1

Table 3.1: Reward Function Parameters

Data Processing

The algorithm used to generate the RL policy is a type of value network algorithm, which was described in Section 2.3. In this type of algorithm, states are fed into a neural network to generate a model of a value function, which is then used to design a policy expected to result in high rewards. In the algorithm I use, before the observable state data is passed into the neural network, it is first transformed to a coordinate frame that is robot-centric and considers the interactions between the robot and each human/object. Recall that if the environment contains N humans and M objects, the state that is observable to the robot is

$$o = \begin{bmatrix} o^{(r)} & o^{(h_1)} & \dots & o^{(h_N)} & o^{(o_1)} & \dots & o^{(o_M)} \end{bmatrix}$$

Rather than simply passing the observable state o into the neural network, the observation is first transformed into a matrix, where each row reflects the interaction of the robot and a single human or object. This first transformation step results in the matrix

$$\tilde{o} = \begin{bmatrix} o^{(r)} & o^{(h_1)} \\ \vdots & \vdots \\ o^{(r)} & o^{(h_N)} \\ o^{(r)} & o^{(o_1)} \\ \vdots & \vdots \\ o^{(r)} & o^{(o_M)} \end{bmatrix}$$

Once the observation is expressed in this form, it is then useful to rotate it to a robot-centric coordinate frame. Consider the first row of the matrix $\tilde{o}$ given below:

$$\tilde{o}_1 = \begin{bmatrix} p_x^{(r)} & p_y^{(r)} & v_x^{(r)} & v_y^{(r)} & r^{(r)} & g_x^{(r)} & g_y^{(r)} & v_{pref}^{(r)} & \theta^{(r)} & p_x^{(h_1)} & p_y^{(h_1)} & v_x^{(h_1)} & v_y^{(h_1)} & r^{(h_1)} & 1 \end{bmatrix}$$

Using this information, it is helpful to define the following variables:

Horizontal distance of robot to goal:	$d_{gx} = g_x^{(r)} - p_x^{(r)}$
Vertical distance of robot to goal:	$d_{gy} = g_y^{(r)} - p_y^{(r)}$
Total distance of robot to goal:	$d_g = \sqrt{d_{gx}^2 + d_{gy}^2}$
Angle between robot and goal position:	$\phi = \tan^{-1}\left(\frac{d_{gy}}{d_{gx}}\right)$
Transformed horizontal velocity of robot:	$\tilde{v}_x^{(r)} = v_x^{(r)}\cos(\phi) + v_y^{(r)}\sin(\phi)$
Transformed vertical velocity of robot:	$\tilde{v}_y^{(r)} = v_y^{(r)}\cos(\phi) - v_x^{(r)}\sin(\phi)$
Transformed horizontal position of human:	$\tilde{p}_x^{(h_1)} = \left(p_x^{(h_1)} - p_x^{(r)}\right)\cos(\phi) + \left(p_y^{(h_1)} - p_y^{(r)}\right)\sin(\phi)$
Transformed vertical position of human:	$\tilde{p}_y^{(h_1)} = \left(p_y^{(h_1)} - p_y^{(r)}\right)\cos(\phi) - \left(p_x^{(h_1)} - p_x^{(r)}\right)\sin(\phi)$
Transformed horizontal velocity of human:	$\tilde{v}_x^{(h_1)} = v_x^{(h_1)}\cos(\phi) + v_y^{(h_1)}\sin(\phi)$
Transformed vertical velocity of human:	$\tilde{v}_y^{(h_1)} = v_y^{(h_1)}\cos(\phi) - v_x^{(h_1)}\sin(\phi)$
Total distance of robot to human:	$d_{h_1} = \sqrt{\left(p_x^{(h_1)} - p_x^{(r)}\right)^2 + \left(p_y^{(h_1)} - p_y^{(r)}\right)^2}$
Combined radius of robot and human:	$r_{tot} = r^{(r)} + r^{(h_1)}$

With these definitions, the rotated version of the first row of the observation is

$$\tilde{o}_1^r = \begin{bmatrix} d_g & \tilde{v}_x^{(r)} & \tilde{v}_y^{(r)} & r^{(r)} & v_{pref}^{(r)} & \theta^{(r)} & \tilde{p}_x^{(h_1)} & \tilde{p}_y^{(h_1)} & \tilde{v}_x^{(h_1)} & \tilde{v}_y^{(h_1)} & r^{(h_1)} & d_{h_1} & r_{tot} & 1 \end{bmatrix}$$

After rotating each row of the observation matrix $\tilde{o}$ in this way, we are left with an observation $\tilde{o}^r$, which has been transformed and rotated into a robot-centric coordinate frame. This is the observation that is used as input to the neural networks described in the following section.

Value Network Architecture

The value network is composed of four different sets of multilayer perceptrons (MLPs), which I will refer to as MLP_1, MLP_2, MLP_3, and MLP_4 and whose layers are given in Table 3.2. I will use f_i to denote the function specified by MLP_i with weights W_i for $i = 1, 2, 3, 4$ and will use $n := N + M$ to denote the total number of humans and stationary objects in the robot's environment. Note that the weight vector includes the bias term for each MLP.

MLP	Layers
1	Linear(14, 150) → ReLU → Linear(150, 100) → ReLU
2	Linear(100, 100) → ReLU → Linear(100, 50)
3	Linear(200, 100) → ReLU → Linear(100, 100) → ReLU → Linear(100, 1)
4	Lin(56, 150) → ReLU → Lin(150, 100) → ReLU → Lin(100, 100) → ReLU → Lin(100, 1)

Table 3.2: The four MLPs in the value network are composed of Rectified Linear Unit (ReLU) activation functions and linear layers, whose input and output sizes are as shown.

The first goal of the value network is to explicitly model the pairwise interaction between the robot and each of the humans and objects. The first MLP is used to obtain an embedding vector e_i for each of the n humans and objects, which can be expressed as

$$e_i = f_1\big(\tilde{o}_i^r \ ; \ W_1\big)$$

This embedding vector is then passed through the second MLP to obtain the pairwise interaction feature for each of the humans and objects, which is given by

$$h_i = f_2\big(e_i \ ; \ W_2\big)$$

The first two MLPs in the value network are used to model the pairwise interaction between the robot and each of the humans and objects. While this is useful, the number of humans and objects in the robot's environment can vary dramatically, and the neural network needs to obtain a fixed size model of the value function from an arbitrary number of inputs. To handle this issue, a self-attention mechanism can be used to learn the relative importance of each human and object in the environment. The self-attention mechanism passes each of the embedding vectors e_i, along with their mean $\bar{e}$, into the third MLP to determine the attention score of each human and object as shown:

$$\alpha_i = f_3\big(e_i, \bar{e} \ ; \ W_3\big) \quad \text{where} \quad \bar{e} = \frac{1}{n}\sum_{i=1}^{n} e_i$$

The softmax function is then applied to each attention score to obtain a normalized set of weights. From the definition of the softmax function, each weight is given by

$$\omega_i = \sigma(\alpha)_i = \frac{e^{\alpha_i}}{\sum_{j=1}^{n} e^{\alpha_j}}$$

A compact representation of the entire set of humans and objects can then be obtained by computing a linear combination of the pairwise interaction features h_i as given below:

$$c = \sum_{i=1}^{n} \omega_i h_i$$

The representation c has a fixed size, regardless of the number of humans and objects, and can be passed into the fourth and final MLP, along with the robot's transformed state, to obtain an estimate of the value function for the given observation. From the full transformed observation, the robot's transformed state is simply

$$s = \begin{bmatrix} d_g & \tilde{v}_x^{(r)} & \tilde{v}_y^{(r)} & r^{(r)} & v_{pref}^{(r)} & \theta^{(r)} \end{bmatrix}$$

With both the representation of the robot s and the representation of the robot's environment c, the estimated value function for a given observation can be expressed as

$$\hat{V} = f_4\big(s, c\ ;\ W_4\big)$$

The structure of the neural network with all of these components is shown in Figure 3.4.

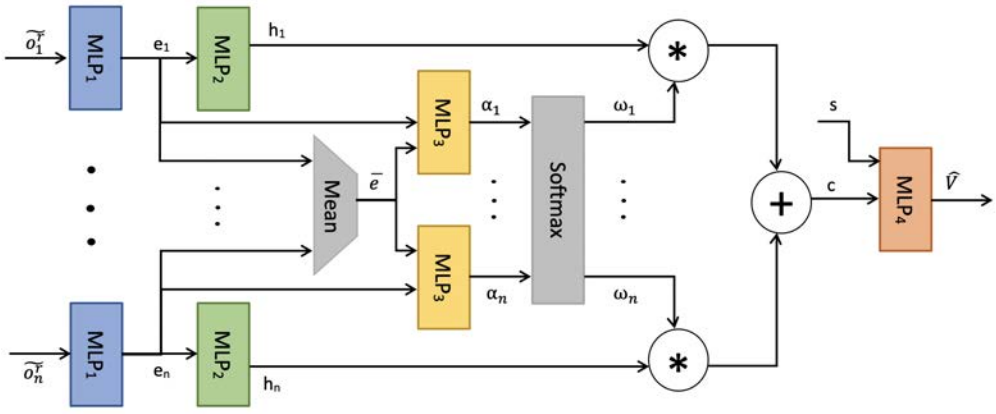

Figure 3.4: This is the architecture of the neural network used to train a model of the value function used in the socially-aware RL policy.

Optimal Policy

Recall from Section 2.3 that a popular policy to use during the training phase of value network algorithms is the epsilon-greedy policy. This policy chooses the action believed to be optimal with probability $1-\epsilon$ and randomly selects some other action in the action space with probability ϵ. During evaluation, the action believed to be optimal is always chosen.

The optimal action is estimated using the value function described by the neural network architecture discussed in the previous section. Given the current state s, the expected next state s' is determined for each action a in the action space $\mathcal{A}$ by using a simple model to approximate the motion of the robot. The next state is then used as input to the value network to determine the value $\hat{V}(s')$. This value is then multiplied by the discount factor

γ and added to the reward $r(s, a)$ for the current state and action. The action with the greatest corresponding value of $r(s, a) + \gamma \hat{V}(s')$ is the optimal action:

$$a^* = \arg\max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \hat{V}(s') \right)$$

Recall that the action space $\mathcal{A}$ is discretized into 5 speeds and 16 rotations, along with a stop command, resulting in 81 possible actions. Because the action space is discrete, the value of $r(s, a) + \gamma \hat{V}(s')$ can be explicitly computed for each of these 81 actions. The discrete action corresponding to the maximum value of $r(s, a) + \gamma \hat{V}(s')$ is then used to control the velocity of the robot. Figure 3.5 demonstrates how the optimal action is selected during testing.

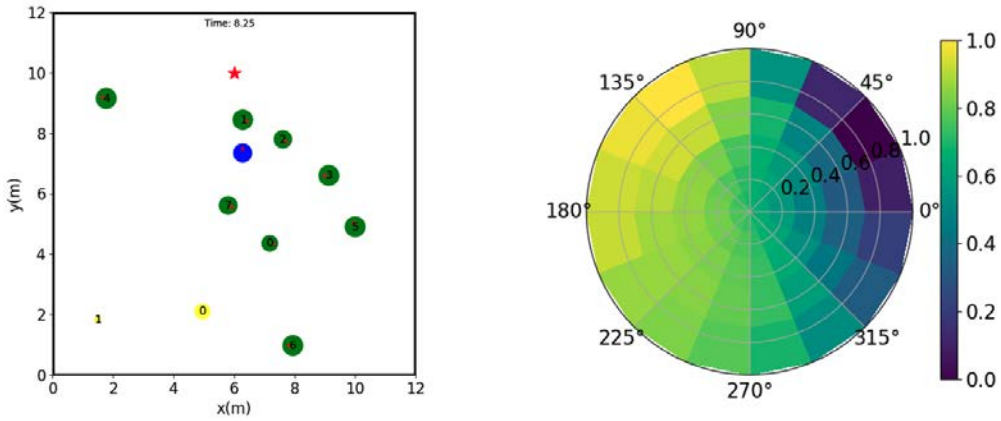

Figure 3.5: The image on the left depicts a robot crossing an open room populated with eight humans and two stationary objects. The image on the right depicts the value distribution of the action space. Each colored segment represents one of the 81 possible actions and its color reflects the relative value. The optimal action is shown in yellow, and the worst set of actions are shown in purple. Notice that the worst set of actions would likely result in a collision with pedestrian one, and the best action makes progress toward the goal by going behind this person.

Training Details

The RL policy was trained in the simulation environment described in Section 3.1. Each time a simulation is run, a simple square room is generated with a single robot at one edge of the room whose goal is to move to the opposite side of the room. The distance between the robot's initial position and goal position is sampled from a uniform distribution, whose parameters are given in Table 3.3. There is also a fixed number of humans and stationary objects with randomly chosen attributes that are sampled from uniform distributions, whose

parameters are specified in Table 3.3. The stationary objects are given random positions in the room, and the humans are given random initial positions and goal positions.

Each of the humans is given a goal within its field of view, and its velocities are controlled using the action specified by the Optimal Reciprocal Collision Avoidance (ORCA) policy [4] discussed in Section 1.3. Once the human has reached its goal, it is given a new goal, so the humans are perpetually moving. The robot is always visible to each human, assuming their view is not obstructed, and the simulated humans treat the robot as if it is another human operating under the same navigation policy. Note that this may not reflect the true reactions of real humans. All of the non-human objects are assumed to be stationary.

After initializing the simulation environment for training, at each time step, the robot chooses an action according to its policy and each human chooses an action according to its policy until the robot reaches its goal, runs out of time, or collides with a human, object, or wall. The parameters used to generate each of these training episodes are specified in Table 3.3.

Parameter	Value
Distance to Goal	$[5\text{m}, 10\text{m}]$
Number of Humans	8
Number of Objects	6
Radius of Robot	0.3m
Radius of Humans	$[0.25\text{m}, 0.35\text{m}]$
Radius of Objects	$[0.1\text{m}, 0.5\text{m}]$
Preferred Velocity of Robot	1m/s
Preferred Velocity of Humans	$[0.5\text{m/s}, 1.5\text{m/s}]$
Room Dimensions	$14\text{m} \times 14\text{m}$
Space Unoccupied by Walls	139.2m^2
Time Limit	30 sec
Length of Time Step	0.25 sec

Table 3.3: Training Simulation Environment Parameters

The RL policy was trained using the value network algorithm described in Section 2.3 with the mean squared error (MSE) loss function. Table 3.4 summarizes the parameters chosen for training. The discount factor (γ), learning rate (α), batch size (M), number of training episodes (N), target update interval (τ_{TU}), and epsilon (ϵ) are all discussed in Section 2.3. The momentum (β) has not yet been discussed but is commonly used in gradient descent to avoid shallow local minima by preserving some amount of momentum between iterations. Note that the value of epsilon is chosen such that

$$\epsilon = \begin{cases} \epsilon_{start} + \left(\frac{\epsilon_{end} - \epsilon_{start}}{\tau_\epsilon}\right) \text{episode} & \text{if episode} \leq \tau_\epsilon \\ \epsilon_{end} & \text{otherwise} \end{cases}$$

Hyperparameter	Value
Discount Factor (γ)	0.9
Learning Rate (α)	0.001
Batch Size (M)	100
Number of Training Episodes (N)	10000
Target Update Interval (τ_{TU})	50
Initial Epsilon (ϵ_{start})	0.5
Final Epsilon (ϵ_{end})	0.1
Epsilon Decay (τ_{ϵ})	4000
SGD Momentum (β)	0.9

Table 3.4: Reinforcement Learning Training Parameters

During training, the RL policy is validated after every 1,000 training episodes using 100 previously unseen validation episodes. The success rate across each set of validation episodes is plotted in Figure 3.6. This figure also shows the training success rate over all 10,000 training episodes, where the success rate is averaged over sets of 200 training episodes.

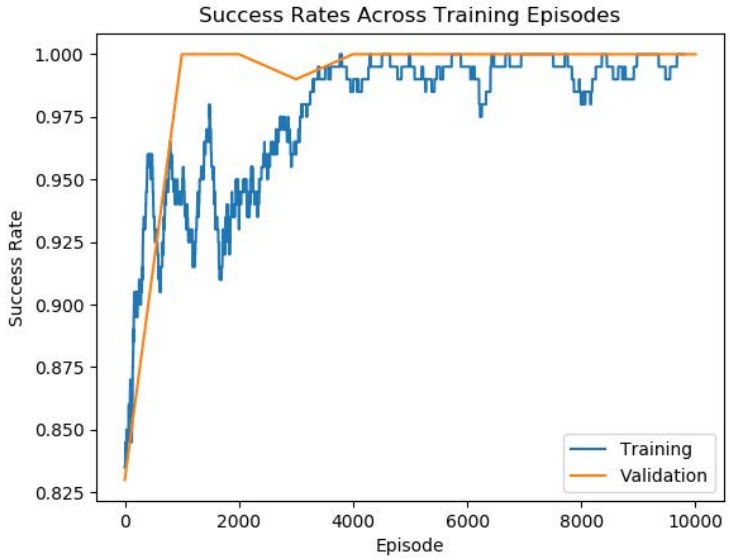

Figure 3.6: This training curve depicts the success rate during both training and validation across 10,000 training episodes. Validation is performed using 100 previously unseen episodes after every 1,000 training episodes, and the success rate is computed for each set of 100 episodes. For training data, the success rate is computed using sets of 200 training episodes.

3.4 Global Path Planner

Probabilistic Roadmap (PRM) Planner

The probabilistic roadmap (PRM) planner is used to generate a path from the robot's starting position to its goal position, while avoiding walls specified by the map of the environment. This path planning algorithm relies on a data structure called a roadmap, which is a network graph whose nodes V are unoccupied points in the configuration space and whose edges E are collision-free paths between these points. The configuration space is defined as all possible positions the robot can attain. Each of these positions may either be occupied or free, depending on whether a wall is present at that location or not. The PRM planning algorithm is given the initial position of the robot, the goal position of the robot, the radius of the robot, and a list of obstacle positions. The algorithm uses this information to generate a list of coordinates connecting the initial position to the goal. The PRM planning algorithm follows the general procedure described below and visualized in Figure 3.7.

1. Generate a sample of unoccupied positions.
    a) Initialize the set of nodes V with the initial and goal positions of the robot.
    b) Sample a random position from the configuration space.
    c) If the position is free and sufficiently far from other nodes in V, add it to V.
    d) Repeat steps (b) and (c) until the desired number of nodes has been attained.
2. Generate a roadmap from the sampled points.
    a) Use the sampled positions in V to generate a kd-tree.
    b) For the first sampled point in V, query the kd-tree to find its nearest neighbors.
    c) For each of its nearest neighbors, if the path from the neighbor to the given point does not result in a collision, add this path to the set of edges E.
    d) Repeat steps (b) and (c) for each of the sampled points in V.
3. Find the optimal path using the roadmap.
    a) Use Dijkstra's algorithm to find the shortest path between the nodes corresponding to the initial and goal positions in the roadmap.
    b) Return the list of nodes in the shortest path from the initial to goal position.

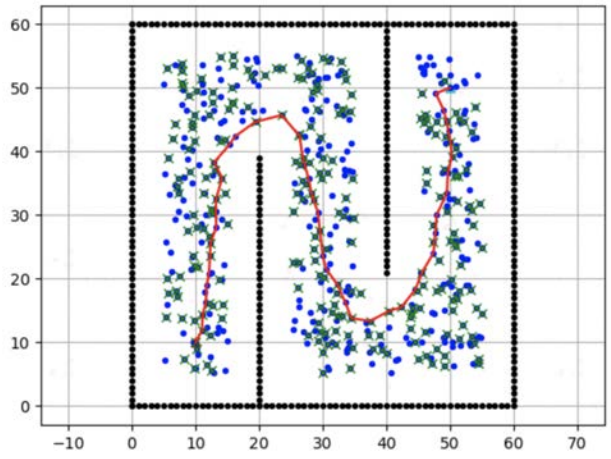

Figure 3.7: This figure demonstrates how the probabilistic roadmap (PRM) planning algorithm is used. The blue points were sampled in the first step of the PRM planning algorithm. Of these points, the ones with a green X were searched using the Dijkstra algorithm. The red line is the shortest path along these searched points.

Waypoint Generation

After using the PRM planner to find a collision-free path from the robot's initial position to its final goal position, I use this path to generate waypoints for the robot. Waypoints are chosen to be more dense around corners and doorways and less dense in open spaces to give the robot more guidance in challenging maneuvers and more freedom when the navigation task is simple. This is an important consideration because if waypoints are too close, there may not be enough space to allow for social compliant behavior. However, if waypoints are too far apart, the global navigation performance may degrade as the robot deviates from the desired global trajectory. In considering the distance between waypoints, it is also important to remember the goal distances the robot saw during training because choosing waypoints very far away from the robot could lead to distributional shift.

To vary the spacing of waypoints, I first conduct a cubic spline interpolation of the points given by the PRM planner to construct a set of points that are all relatively close together, beginning at the robot's initial position and ending at its goal position. From the robot's initial position, I construct a small triangle originating from this point with an altitude of three meters and corresponding angle of eight degrees. I then choose the point in this cone that is furthest from the starting position to be the first waypoint in the sequence. From the first waypoint, I choose the second waypoint to be the furthest point in the triangle originating from the current point. I continue this way until the last waypoint is the robot's final goal position. Figure 3.8 helps demonstrate the waypoint selection process.

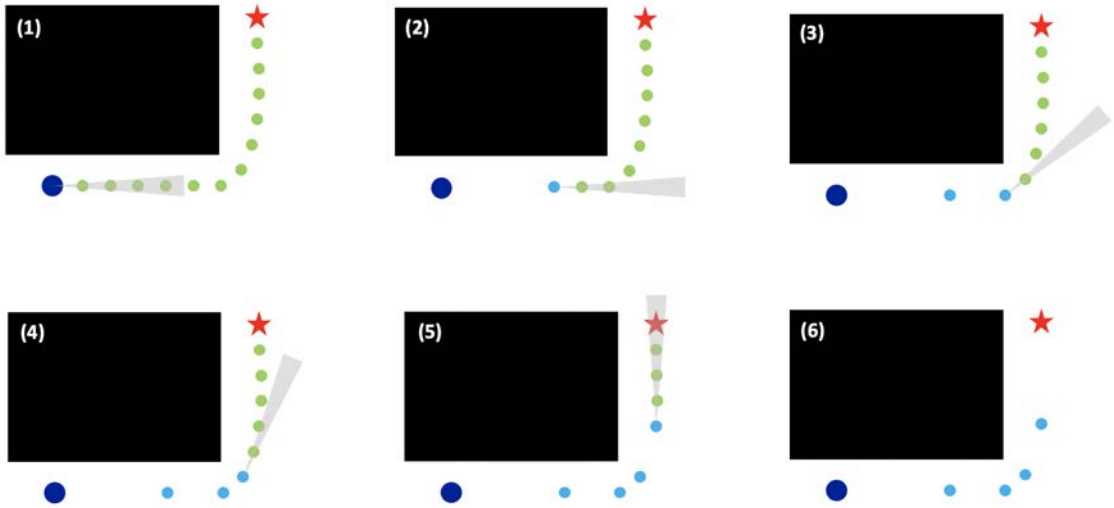

Figure 3.8: The dark blue dot represents the robot's initial position, the red star represents its goal position, and the black box represents a corner the robot must navigate around. The equally spaced green dots are obtained by conducting a cubic spline interpolation of the points obtained from the PRM planner, and the light blue dots are the robot's waypoints. From the robot's initial position, the first waypoint is the furthest point from the initial position within the gray triangle. Each subsequent waypoint is chosen in a similar way. The final waypoints are then denser around the corner and less dense where the robot follows a straight path.

Local Goal Selection

Based on the robot's current position, its local goal is chosen from the waypoints generated in the previous section. Initially, the robot's local goal is set to be the first waypoint after its starting position. Once the robot has entered within a one meter radius of the first waypoint, its local goal is set to be the second waypoint in the sequence. After entering within a one meter radius of this waypoint, its local goal is set to be the third waypoint in the sequence. This process continues until the robot's local goal is its ultimate goal position.

As described, local goals are selected from the list of waypoints generated from the collision-free path provided by the path planning algorithm. The path planning algorithm uses knowledge of walls in the robot's environment but does not use knowledge of smaller objects that may not remain fixed for the robot's lifetime. For this reason, it is possible for a local goal to be placed in the same location as a stationary object. To deal with this issue, if the next waypoint in the robot's path is in the same location as an object, the robot's local goal is shifted to either the left or right side of the object with respect to the robot's current position. Figure 3.9 helps demonstrate how a local goal is chosen.

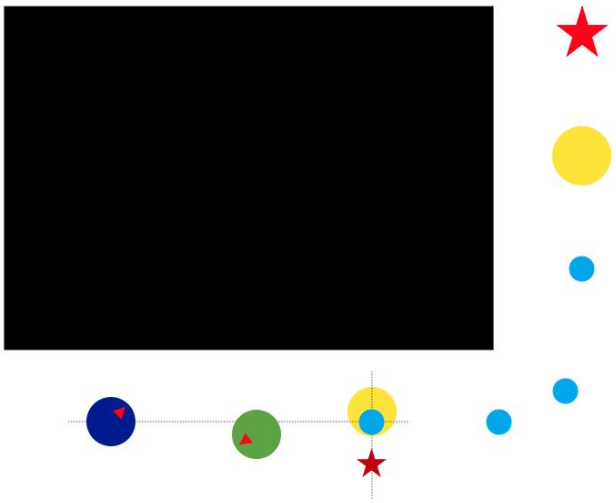

Figure 3.9: The dark blue dot represents the robot's current position, the light blue circles represent its waypoints, the small dark red star represents its local goal, and the large bright red star represents its final goal position. The black box represents a corner, the yellow circles represent stationary objects, and the green circle represents a human. Notice that the robot's first waypoint is in the same location as of one of the objects, so the local goal is shifted to the right side of the object with respect to the robot.

3.5 Safety Controller

Restricting the Action Space

Rather than generating a representation of walls that can be passed into the RL policy, wall avoidance is handled by limiting the action space of the policy. At each time step, before determining the optimal action from the RL policy as described in Section 3.3, the safety controller first determines which directions in the full action space are expected to result in a collision with a wall, based on the position of the robot and walls within its vicinity. The RL policy then computes the value of each action among the set of actions that are not expected to result in a wall collision, and the action with the largest value among these safe actions is provided to the robot. Figure 3.10 helps to demonstrate how the safety controller works to limit collisions with walls in the robot's environment.

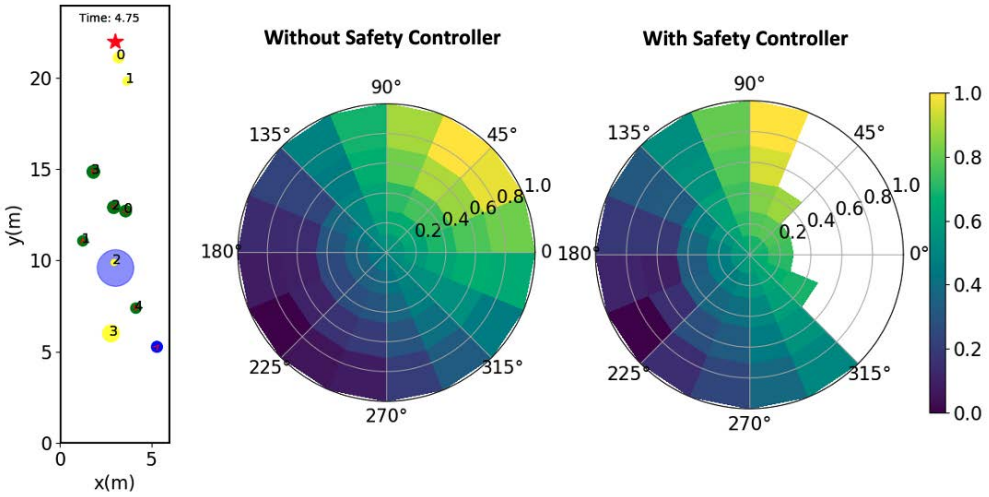

Figure 3.10: The leftmost image shows a situation in which the robot reaches its goal when the safety controller is employed but collides with a wall otherwise. The center image shows the value distribution of the action space without the safety controller, and the rightmost image shows the value distribution when the safety controller is employed. When the safety controller is not used, the optimal action shown in yellow results in a collision. When the safety controller is employed, this action is no longer valid, and a similar action that does not result in a collision is chosen.

Determining Safe Actions

Recall that the action space is discretized into a set of 5 speeds $\mathcal{V}$ and a set of 16 rotations Φ, which are shown in Figure 3.1. Consider the action (v, ϕ), which is composed of the speed $v \in \mathcal{V}$ and rotation $\phi \in \Phi$. Within a single time step of $\Delta t = 0.25$ seconds, the robot will move $(v * \Delta t)$ meters in the direction of ϕ from its initial position $(p_x^{(r)}, p_y^{(r)})$. If the robot has a radius of $r^{(r)}$, then a point on it will reach $(v * \Delta t + r^{(r)})$ meters away from its initial position. I am interested in whether the robot is expected to collide with a wall when moving in a straight line to this new point. To check this, I create a triangle originating at the robot's initial position with a corresponding angle of $\frac{\pi}{16}$. If this triangle intersects with a wall, then the associated action is unsafe. Otherwise, the action is assumed to be safe. To encourage the robot to maintain a comfortable distance away from the walls, I introduce a safety margin of $m = 3$ and consider the point $(v * \Delta t * m + r^{(r)})$ meters away from its initial position $(p_x^{(r)}, p_y^{(r)})$ in the direction ϕ. This is demonstrated in Figure 3.11.

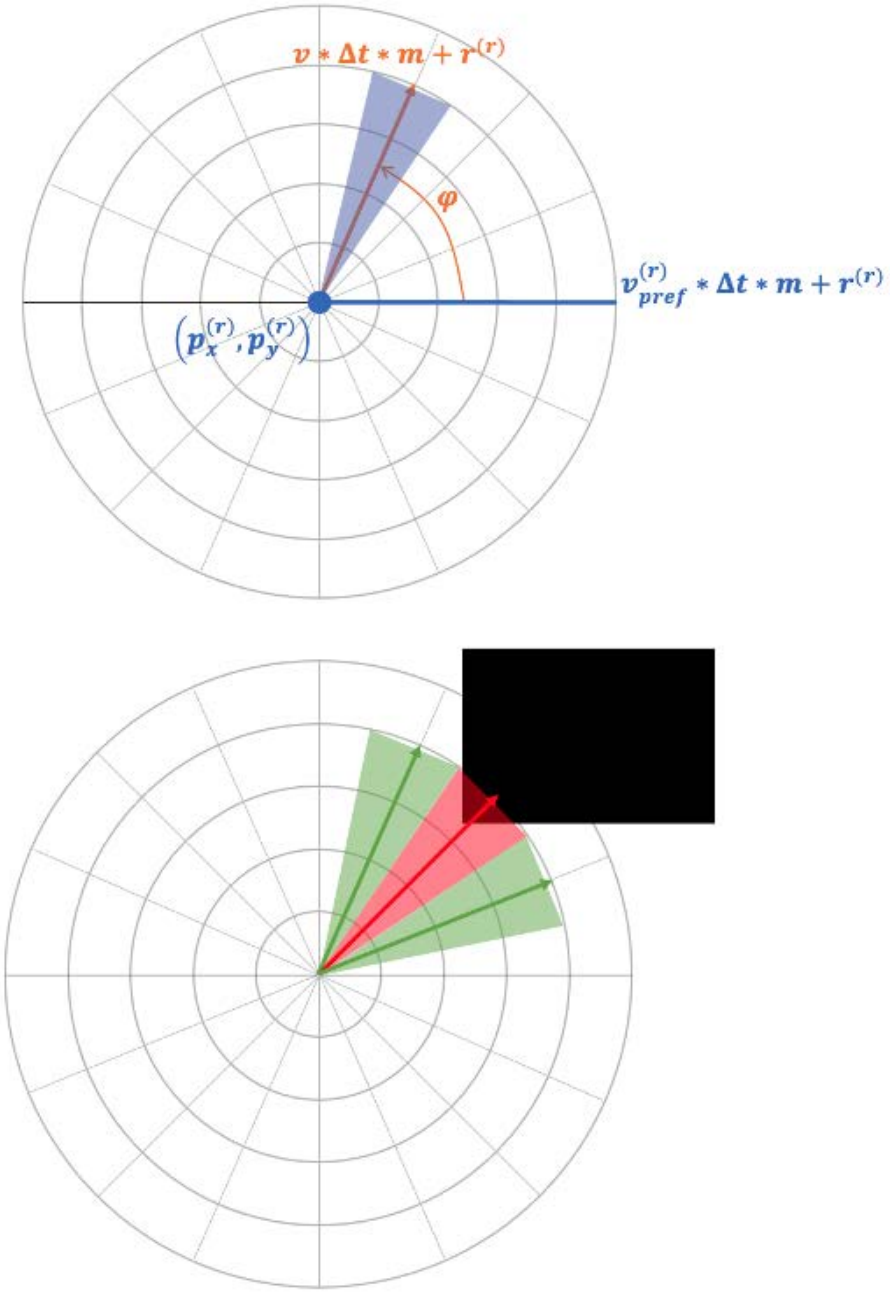

Figure 3.11: As shown in the top image, for each action (v, ϕ), a triangle with an altitude dependent on v is drawn pointing in the direction of ϕ. The associated action is deemed unsafe if this triangle intersects with a wall and safe otherwise. The bottom image shows a wall depicted by a black box and three example actions: one which is unsafe and two that are safe.

After determining which actions in the entire action space appear to be safe, the set of safe actions is refined to deal with the fact that the robot has a strictly positive radius and must navigate around sharp corners. To deal with this issue, actions that neighbor clearly unsafe actions actions are also deemed unsafe. This idea is demonstrated in Figure 3.12.

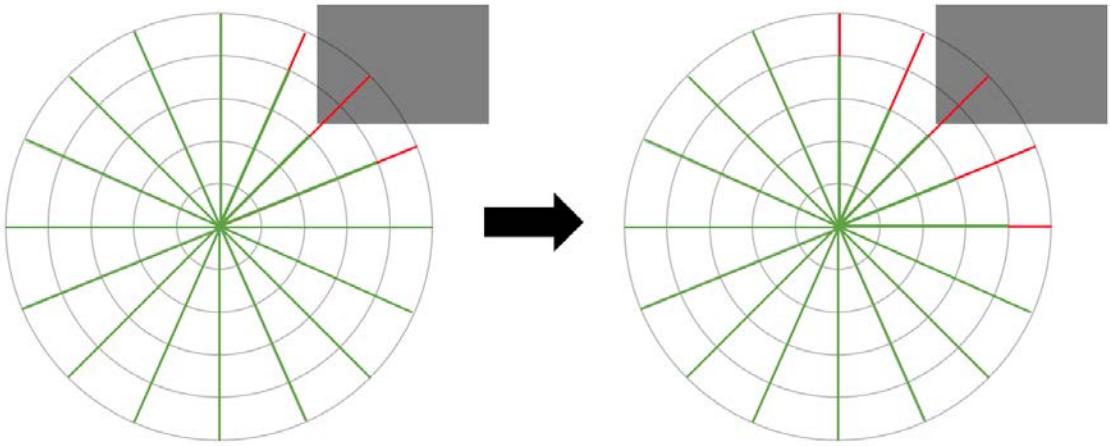

Figure 3.12: In this figure, each of the 80 segments is a discrete action composed of a speed and a rotation, and the dark box represents a wall next to the robot. In the image on the left, the red segments are actions that are deemed unsafe through the process described in Figure 3.11, and the green segments are the actions originally considered safe through this same process. The image on the right shows how the set of safe actions is refined such that any action that neighbors an unsafe action is now also deemed unsafe.

Chapter 4

Experimental Evaluation

4.1 Evaluation Overview

Environment Layouts

I designed several environment layouts that reflect the various hallways, corners, doorways, and intersections that a robot may encounter in its lifetime. There are seven different layouts with one to four sets of robot start and goal positions for each. Figure 4.1 depicts three different basic layouts: an open space, a single long hallway, and multiple hallways. Figure 4.2 and 4.4 each depict an environment with either a wide or narrow intersection and three different robot goal positions within each layout. Figure 4.3 demonstrates four start and goal positions for the robot in an environment with various hallways, and Figure 4.5 demonstrates four start and goal positions for the robot in an environment with various rooms. In total, there are seventeen environment configurations chosen to reflect the majority of possible scenarios a robot may encounter when navigating a typical indoor space.

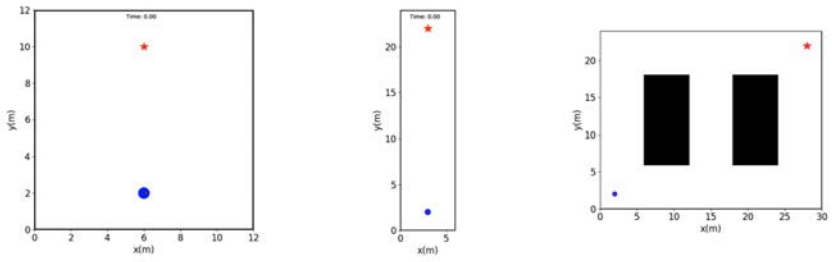

Figure 4.1: The figure on the left demonstrates a situation where the robot navigates to the opposite side of an open space. In the figure in the center, the robot navigates to the opposite side of a single hallway. In the figure on the right, the robot navigates through multiple hallways, where there are multiple paths the robot can take.

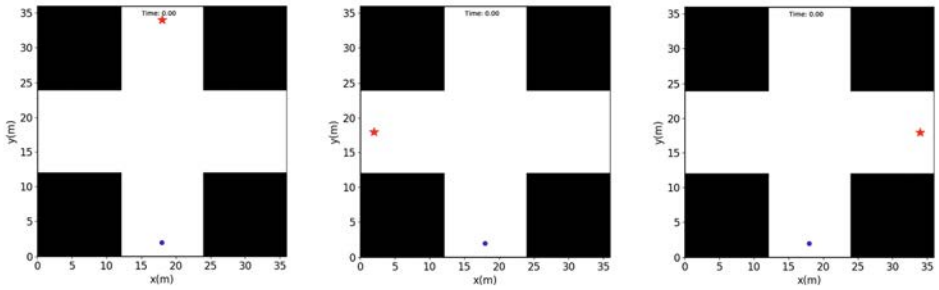

Figure 4.2: These figures depict an environment containing an intersection with wide hallways. In each figure, the robot starts at the bottom edge of the space and navigates to either the top, left, or right edge of the environment.

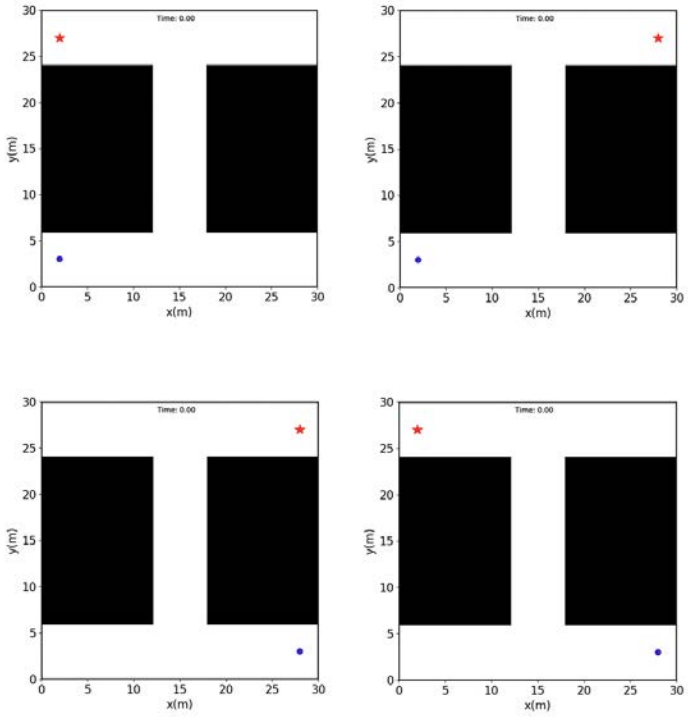

Figure 4.3: These figures depict an environment with various hallways, where the robot must navigate around humans and stationary obstacles in tight spaces. In each figure, the robot is given one of two starting positions and one of two goals.

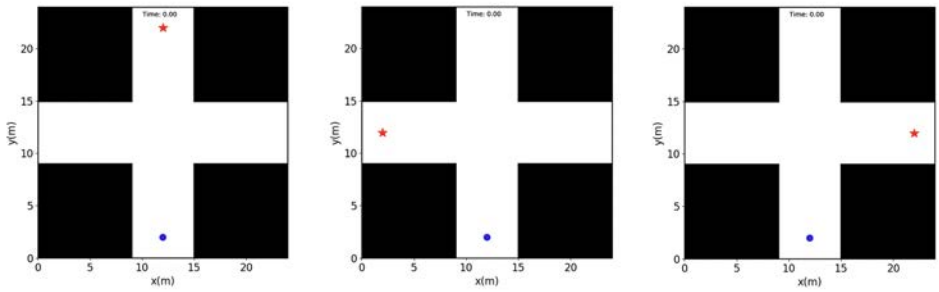

Figure 4.4: These figures depict an environment containing an intersection with narrow hallways. In each figure, the robot starts at the bottom edge of the space and navigates to either the top, left, or right edge of the environment.

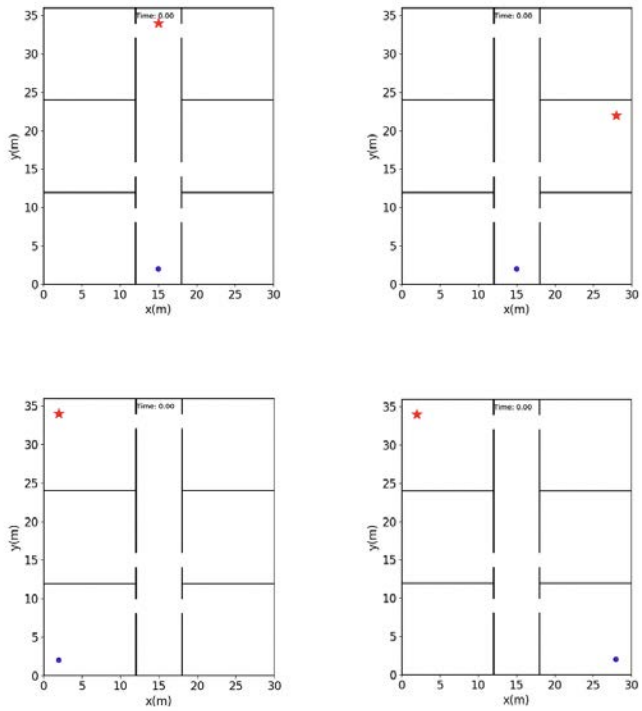

Figure 4.5: These figures depict an environment containing various rooms, where the robot encounters several hallways and doorways. In each figure, the robot is given a different combination of start and goal positions.

In my analysis, I grouped the seventeen environment configurations into five qualitatively distinct groups to better evaluate each component of my modular approach and better illustrate the utility of my complete architecture (Figure 4.6). These five groups are:

1. **Open Space**: This group consists only of the open space shown in Figure 4.1, which is most similar to the environment in which the RL policy was trained.
2. **Hallways**: This group consists of configurations that are more complex than the open space but allow the robot to travel in a straight line to the goal. This includes the single long hallway shown in Figure 4.1, the first wide intersection configuration shown in Figure 4.2, and the first narrow intersection configuration shown in Figure 4.4.
3. **Intersections**: This group consists of configurations in which the robot must navigate around a single corner at a four-way intersection to reach the goal. This includes the second two configurations shown in Figure 4.2 and the second two shown in Figure 4.4.
4. **Doorways**: This group consists of configurations containing rooms with doorways the robot must navigate through, which are all shown in Figure 4.5.
5. **Corners**: This group consists of configurations that require the robot to navigate around multiple corners. This includes all four configurations shown in Figure 4.3, as well as the last configuration with multiple hallways shown in Figure 4.1.

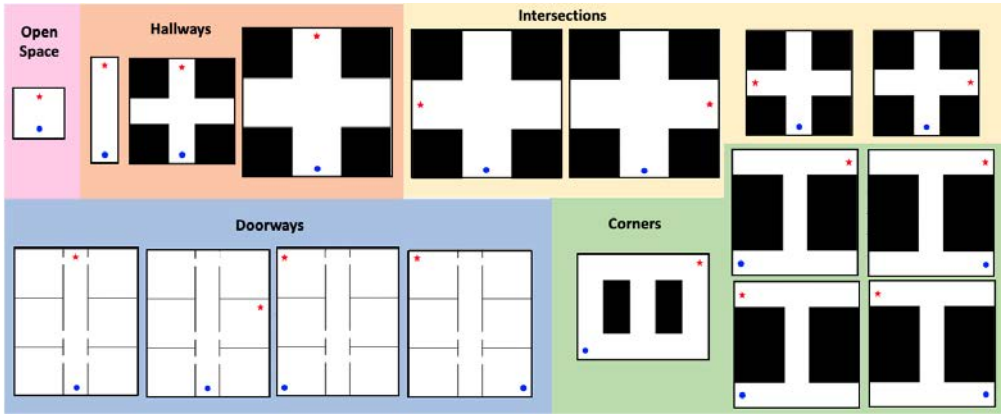

Figure 4.6: The 17 environment configurations are grouped into five categories: open space, hallways, intersections, doorways, and corners. For each configuration, walls are shown in black, the blue circle is the robot's initial position, and the red star is the robot's goal.

Experimental Setup

One hundred trials were performed for each of the 17 environment configurations. For each trial, the density of humans and density of objects were sampled from a uniform distribution, whose parameters are specified in Table 4.1. Each circular human was given a random radius sampled from a uniform distribution, a random preferred speed sampled from a uniform distribution, and a random starting position. Similarly, each stationary object was given a random radius sampled from a uniform distribution and a random fixed position. As in the training environment, all of the non-human objects are stationary. Each human is given a goal within its field of view, and its velocities are controlled using the action specified by the Optimal Reciprocal Collision Avoidance (ORCA) policy. Once the human has reached its goal, it is given a new goal, so the humans are perpetually moving. The robot is always visible to the human, assuming the human's view is not obstructed, and the simulated humans treat the robot as if it is another human operating under the same navigation policy.

After initializing the simulation environment for evaluation, the robot chooses an action according to its policy at each time step and each human chooses an action until the robot reaches its goal, runs out of time, or collides with a human, object, or wall. The parameters used to generate each of these evaluation trials are specified in Table 4.1.

Parameter	Value
Density of Humans (humans/m^2)	$[0.02, 0.05]$
Density of Objects (objects/m^2)	$[0.01, 0.04]$
Radius of Robot (meters)	0.3
Radius of Humans (meters)	$[0.25, 0.35]$
Radius of Objects (meters)	$[0.1, 0.5]$
Preferred Velocity of Robot (m/s)	1
Preferred Velocity of Humans (m/s)	$[0.5, 1.5]$
Time Limit (seconds)	100
Length of Time Step (seconds)	0.25

Table 4.1: Evaluation Simulation Environment Parameters

4.2 Performance of Modular Architecture

I first evaluate the utility of each component of my modular architecture by conducting an ablation study and analyzing the overall performance of the controller with certain components removed. I considered the performance of the RL policy on its own, the RL policy with just the safety controller, the RL policy with just the global path planner, and the complete architecture with all three components. In performing this analysis, I considered the following metrics, which reflect the overall performance of the navigation strategy:

- **Success rate**: Percentage of the trials in which the robot reached the goal.
- **Human collision rate**: Percentage of the trials the robot collided with a human.
- **Object collision rate**: Percentage of the trials the robot collided with an object.
- **Wall collision rate**: Percentage of the trials the robot collided with a wall.
- **Timeout rate**: Percentage of the trials the robot did not reach the goal in time.

As described in Section 3.2, the robot controller I designed is composed of an RL policy, a global path planner used to generate waypoints, and a safety controller designed to limit collisions with walls. Table 4.2 helps demonstrate how each of these components impacts the overall performance of the simulated robot on average.

Path Planner	Safety Controller	Success Rate (%)	Collision Rate (%)			Timeout Rate (%)
			Human	Object	Wall	
No	No	12.3	0.0	0.0	87.5	0.2
No	Yes	29.8	0.1	0.0	6.1	64.0
Yes	No	97.8	0.2	0.1	1.5	0.5
Yes	Yes	99.2	0.1	0.0	0.0	0.7

Table 4.2: This table illustrates the average success rate, collision rates, and timeout rate across all trials for the RL policy with and without the path planner, as well as with and without the safety controller.

From Table 4.2, it is clear that, regardless of whether the safety controller is employed, including a global path planner is necessary for successful robot navigation. As expected, the RL policy is best used as a local planner in combination with a separate global planner. From this table, it is also clear that the safety controller increases the success rate of the robot as well, but this increase is not nearly as great as the increase obtained by employing the global path planner. The highest success rate, among each combination of controller components, is over 99% and is obtained by using the RL policy in combination with both the global path planner and safety controller, indicating that the modular approach proposed in Section 3.2 is effective and each component of the controller plays an important role.

Rather than simply looking at the average performance across all 1700 trials, it is helpful to consider the performance within different environment configuration categories. Table 4.3 shows the impact of the controller components on performance across various categories.

Environment Layout	Path Planner	Safety Controller	Success Rate (%)	Collision Rate (%)			Timeout Rate (%)
				Human	Object	Wall	
Open Space	No	No	100.0	0.0	0.0	0.0	0.0
	No	Yes	100.0	0.0	0.0	0.0	0.0
	Yes	No	100.0	0.0	0.0	0.0	0.0
	Yes	Yes	100.0	0.0	0.0	0.0	0.0
Hallways	No	No	33.3	0.0	0.0	66.3	0.3
	No	Yes	93.0	0.3	0.0	0.0	6.7
	Yes	No	99.7	0.0	0.0	0.0	0.3
	Yes	Yes	99.7	0.0	0.0	0.0	0.3
Intersections	No	No	1.8	0.0	0.0	98.0	0.2
	No	Yes	14.2	0.0	0.0	0.2	85.5
	Yes	No	99.8	0.0	0.0	0.2	0.0
	Yes	Yes	99.8	0.0	0.0	0.0	0.2
Doorways	No	No	0.5	0.0	0.0	99.5	0.0
	No	Yes	16.8	0.0	0.0	1.8	81.5
	Yes	No	96.8	0.5	0.0	2.0	0.8
	Yes	Yes	99.0	0.0	0.0	0.0	1.0
Corners	No	No	0.0	0.0	0.0	99.8	0.2
	No	Yes	0.8	0.0	0.0	19.2	80.0
	Yes	No	95.4	0.2	0.2	3.2	1.0
	Yes	Yes	98.6	0.2	0.0	0.0	1.2

Table 4.3: This table breaks down the overall performance of the robot controller across the open space environment used for training and various other categories of environment layouts. Within each category, performance rates are averaged across the trials run for each set of configurations in that category and is analyzed with and without the global path planner, as well as with and without the safety controller.

From Table 4.3, it is evident that the RL policy is 100% successful in open space environments, regardless of whether a global path planner or safety controller is employed. This demonstrates that the RL policy is able to generalize to cases where the basic environment layout is very similar to the one in which it was trained. However, the RL policy on its own does not generalize well to navigation tasks in more complex environment layouts, resulting in a low success rate in unseen environments when the global path planner and safety controller are not employed. By introducing just the safety controller for wall avoidance, the wall collision rate decreases across all of the more complex environment layouts, causing the success rate of the robot to increase. However, with only the safety controller employed, the success rate is still relatively low and the timeout rate is quite high. By introducing just the global path planner, without incorporating the safety controller, the success rate increases to

over 99% for both the hallway and intersection environments, and the success rate increases to over 95% for the environments with doorways and multiple corners. Incorporating the safety controller in addition to the path planner causes the wall collision rate to decrease to 0%, further increasing the success rate of the modular controller to nearly 100% for all environment configurations. This indicates that the global path planner is critical for enabling robot navigation in more complex indoor spaces and demonstrates the effectiveness of the safety controller in limiting collisions with walls in the environment. In addition, it is clear that the safety controller is most useful in environments with many doorways and corners.

I ranked the complexity of environment configurations based on the category of the layout and the density of humans and objects in the space. The simplest configuration is the open space with a below average density of obstacles, followed by the open space with an above average density. The complexity of the spaces increases with the order of environment layouts shown in Table 4.3. The performance of my modular framework with certain components removed for varying levels of environment complexity is shown in Figure 4.7. This graph emphasizes the utility of my full modular framework in more complex environments.

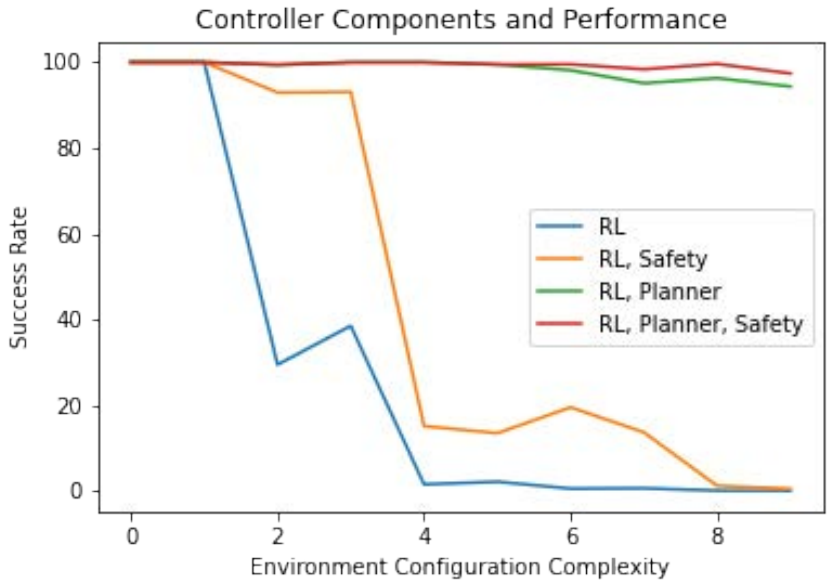

Figure 4.7: This graph displays the success rates of the robot controller across increasing levels of environment complexity with and without the global path planner, as well as with and without the safety controller.

To see the results of this ablation study for all 17 environment configurations individually, please refer to Table A.1 and Table A.2 in Appendix A.

4.3 Safety Controller Analysis

Because the safety controller restricts the action space of the RL policy and prevents it from choosing the action is believes to be optimal, it is interesting to consider how aggressive this behavior is. To further analyze how the safety controller interacts with the RL policy in the modular architecture to prevent wall collisions, I collect data on the following:

- **Safety active**: Percentage of time the safety controller is active, as determined from the number of times the safety controller adjusted the action that the RL policy would have chosen from the unrestricted action space.
- **Safety speed**: Average speed of the robot when the safety controller is active.

Table 4.4 demonstrates how often the safety controller is active, both when the global path planner is and is not employed. This table also indicates the speed of the robot when the safety controller is actively restricting the action space of the RL policy.

Environment Layout	Path Planner	Safety Active (%) (avg ± std)	Safety Speed (m/s) (avg ± std)
Open Space	No	0.00 ± 0.00	–
	Yes	0.00 ± 0.00	–
Hallways	No	33.70 ± 30.83	0.24 ± 0.18
	Yes	0.04 ± 0.56	0.76 ± 0.22
Intersections	No	83.14 ± 20.95	0.18 ± 0.13
	Yes	1.51 ± 6.64	0.39 ± 0.15
Doorways	No	77.69 ± 22.67	0.20 ± 0.15
	Yes	3.97 ± 3.41	0.80 ± 0.19
Corners	No	80.96 ± 24.37	0.21 ± 0.16
	Yes	2.13 ± 5.60	0.47 ± 0.20

Table 4.4: This table demonstrates the aggressiveness of the safety controller in restricting the action space of the RL policy to prevent wall collisions. For each environment layout category, every performance metric is averaged across the trials ran for each set of configurations within that category and is analyzed with and without the global path planner.

From Table 4.4, it is apparent that the safety controller is relied on very heavily when the global path planner is not employed but is not used frequently within the complete architecture. As a component of the complete framework with the global path planner, the safety controller is never used in open spaces and almost never used in simple hallways. It is

active around 1.5% of the time in intersection environments, 4.0% of the time in environments with doorways, and 2.1% of the time in environments with multiple corners. These values are relatively low, indicating that the RL policy is able to take the socially optimal action the majority of the time with some limitations when necessary to avoid collisions with walls. From Table 4.4, it is also clear that the robot moves slower when the safety controller is active, indicating that the robot slows down around walls to avoid collisions. However, it is still able to maintain a fair amount of speed in many situations.

To view additional data on the safety controller that considers all 17 environment configurations individually, please refer to Table A.3 in Appendix A.

4.4 Performance of Learning Component

As described in Section 3.3, a key component of my robotic controller is the RL policy, which is designed to enable socially compliant behavior in simple, open spaces. It is interesting to consider how learning can enable more effective robotic behavior around crowds of humans and stationary objects in complex indoor spaces. To demonstrate the impact of learning, I compared my full modular approach to a navigation strategy that uses ORCA as a local planner and the probabilistic roadmap (PRM) planner as the global planner. This controller is used as a baseline to analyze the impact of learning because ORCA is a local planner designed the navigate safely around humans that does not rely on deep learning. To compare my modular architecture, which includes a learning component, to a standard navigation strategy that does not employ learning, I consider the following performance rates:

- **Success rate**: Percentage of the trials in which the robot reached the goal.
- **Human collision rate**: Percentage of the trials the robot collided with a human.
- **Object collision rate**: Percentage of the trials the robot collided with an object.
- **Wall collision rate**: Percentage of the trials the robot collided with a wall.
- **Timeout rate**: Percentage of the trials the robot did not reach the goal in time.

I also evaluate both strategies based on robot navigation performance and social compliance. In terms of robot navigation performance, I assess the quality of the robot's ability to navigate to the goal efficiently. More specifically, I consider the following metrics:

- **Navigation time**: Average time needed by the robot to reach its final goal position (within some radius) in successful trials.
- **Path length**: Average length of path taken by robot in successful trials.

- **Average speed**: Average speed of robot across all time steps in successful trials.

To assess social compliance, I quantified how well the robot maintained social distance among humans. Specifically, I considered the following metrics:

- **Average distance**: Average distance between the robot and the closest human averaged across all time steps in successful trials.
- **Minimum distance**: The closest the robot gets to any human in successful trials.
- **Discomfort time**: Percentage of time spent in the discomfort zone (0.1m-radius circle around each human) of any human in successful trials.

Environment Layout	Navigation Strategy	Success Rate (%)	Collision Rate (%)			Timeout Rate (%)
			Human	Object	Wall	
Open Space	Mine	100.00	0.00	0.00	0.00	0.00
	Baseline	99.00	0.00	1.00	0.00	0.00
Hallways	Mine	99.67	0.00	0.00	0.00	0.33
	Baseline	97.33	0.00	1.33	0.00	1.33
Intersections	Mine	99.75	0.00	0.00	0.00	0.25
	Baseline	97.25	0.00	1.75	0.00	1.00
Doorways	Mine	99.00	0.00	0.00	0.00	1.00
	Baseline	93.00	0.50	3.50	0.75	2.25
Corners	Mine	98.60	0.20	0.00	0.00	1.20
	Baseline	92.20	0.00	4.00	0.00	3.80

Table 4.5: This table compares the overall performance of my navigation approach to that of a standard navigation strategy. The performance rates of my complete modular architecture, which includes a learning component, and those of a standard navigation strategy that does not employ learning are shown across various environment layout categories.

From Table 4.5, it is clear that my navigation strategy generates higher success rates than the baseline strategy across all environment layout categories. The higher rates of success seen by my approach are primarily due to fewer collisions with objects and fewer instances in which the robot is not able to reach the goal in the allotted time. This difference in success rate becomes more apparent as the complexity of the simulation environment increases and becomes more similar to the types of environments a robot might encounter in the real world. This seems to indicate that including a learning component as an element of the robot navigation strategy can improve the robot's ability to navigate more complex environments.

I ranked the complexity of environment configurations based on the category of the layout and the density of humans and objects. The simplest configuration is the open space with a below average density of obstacles, followed by the hallways with below average density, intersections with below average density, doorways with below average density, and corners with below average density. The next levels of complexity have an above average density of obstacles and follow the same environment layout order. The performance of my approach is compared to the baseline in Figure 4.8. This graph emphasizes the utility of incorporating learning into my modular framework in more complex environments.

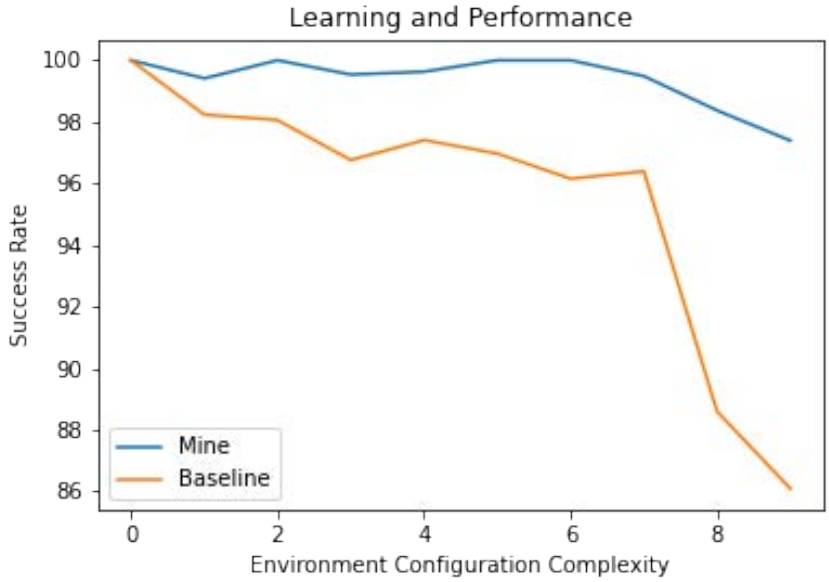

Figure 4.8: This graph displays the success rates of my modular approach, which utilizes a learning component, and a baseline policy that does not utilize learning across increasing levels of environment complexity

Upon analyzing the videos of various successful and unsuccessful trials, I found that this improvement is generally seen because the RL policy is able to learn more difficult maneuvers that ORCA is not capable of. In the majority of cases where the learning-based controller is successful and the non-learning-based controller is not, the ORCA policy is unable to maneuver around clumps of objects. In all of these cases, the robot controlled by the baseline controller either tries to move around the group of objects and hits one, or it gets stuck at the group of objects before timing out. Figure 4.9 helps demonstrate what commonly occurs in these situations. These cases indicate that a controller that employs a learning component can enable successful navigation when more complicated maneuvers are required.

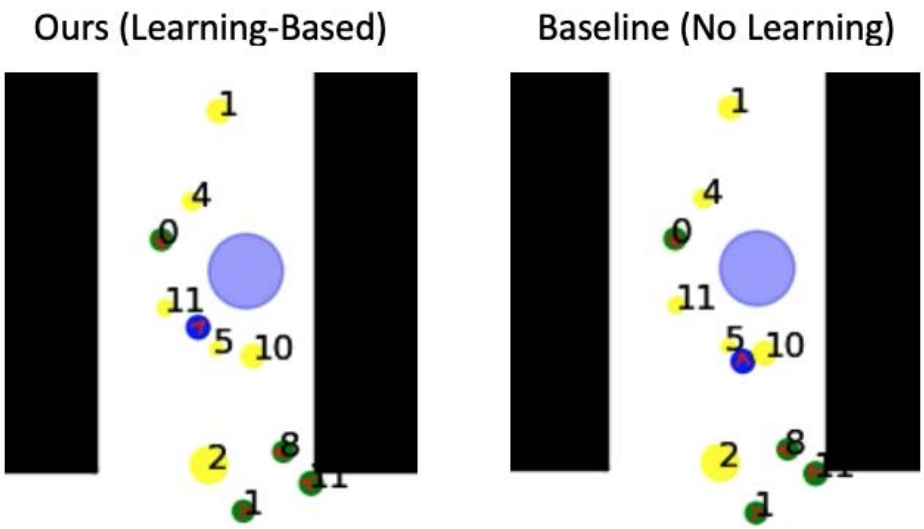

Figure 4.9: This figure shows a situation where my controller, which utilizes an RL policy, is successful and the baseline controller, which has no learning component, is unsuccessful. The image on the left shows that the robot controlled by my learning-based controller is able to successfully maneuver around a small clump of objects. The image on the right shows that the robot controlled by the baseline controller gets stuck at this clump of objects, where it stays for the remainder of the trial.

While my learning-based controller is generally more successful than the baseline controller, it is important to note that this increase in success comes with slightly longer navigation times and path lengths, as seen in Table 4.6. However, this difference in navigation time and path length is relatively small, compared to the difference in success rate, collision rates, and timeout rate between the two navigation strategies. This suggests that my approach provides a worthwhile trade-off between navigation efficiency and overall performance.

Table 4.7 compares the social compliance of my approach to that of the baseline. From this table, it is clear that my navigation strategy generally achieves a larger average distance to nearby humans, a smaller percentage of time spent in a human's discomfort zone, and a comparable minimum distance to the closest human. Therefore, my navigation approach performs better in terms of social compliance, compared to the baseline. However, this difference in social compliance is not very large. Both navigation strategies generally maintain a reasonable distance from humans, intruding in their discomfort zones very infrequently.

To see a comparison of my navigation strategy to the baseline for all 17 environment configurations individually, please reference Tables A.4, A.5, and A.6 in Appendix A.

Environment Layout	Navigation Strategy	Navigation Time (s) (avg ± std)	Path Length (m) (avg ± std)	Speed (m/s) (avg ± std)
Open Space	Mine	7.65 ± 0.27	7.65 ± 0.27	1.00 ± 0.00
	Baseline	7.61 ± 0.17	7.47 ± 0.06	0.98 ± 0.06
Hallways	Mine	24.19 ± 5.68	24.18 ± 5.68	1.00 ± 0.00
	Baseline	23.86 ± 5.68	23.57 ± 5.67	0.99 ± 0.05
Intersections	Mine	21.29 ± 4.32	21.19 ± 4.31	1.00 ± 0.01
	Baseline	20.87 ± 4.26	20.62 ± 4.23	0.99 ± 0.05
Doorways	Mine	40.70 ± 10.94	40.35 ± 10.72	0.99 ± 0.05
	Baseline	39.66 ± 10.14	39.07 ± 10.13	0.99 ± 0.06
Corners	Mine	45.38 ± 5.16	44.89 ± 4.44	0.99 ± 0.03
	Baseline	43.71 ± 2.22	43.11 ± 2.02	0.99 ± 0.05

Table 4.6: This table compares the navigation performance of my navigation approach to that of a standard navigation strategy that does not utilize learning across various environment layout categories.

Environment Layout	Navigation Strategy	Avg. Distance (m) (avg ± std)	Minimum Distance (m)	Discomfort Time (%) (avg ± std)
Open Space	Mine	1.58 ± 0.74	0.10	0.00 ± 0.00
	Baseline	1.57 ± 0.73	0.04	0.21 ± 2.11
Hallways	Mine	1.97 ± 1.40	0.00	0.08 ± 0.52
	Baseline	1.96 ± 1.41	0.02	0.17 ± 0.92
Intersections	Mine	2.21 ± 1.46	0.04	0.04 ± 0.31
	Baseline	2.18 ± 1.44	0.03	0.15 ± 1.41
Doorways	Mine	2.05 ± 1.41	0.02	0.07 ± 0.32
	Baseline	2.07 ± 1.44	0.01	0.18 ± 0.88
Corners	Mine	2.22 ± 1.75	0.00	0.21 ± 1.59
	Baseline	2.18 ± 1.72	0.03	0.19 ± 0.75

Table 4.7: This table compares the social compliance of my navigation approach to that of a standard navigation strategy that does not utilize learning across various environment layout categories.

4.5 Real-World Demonstration

Goals of Demonstration

To demonstrate the applicability of my modular controller to real-world systems, I implemented my algorithm on a physical robot. Because real-world robotic systems exhibit many physical limitations, and assumptions made in simulation may not hold in the real world, I find it valuable to demonstrate my algorithm on an actual robot. It is important to consider that physical robotic systems exhibit many sensing limitations that are generally ignored in simulation. For example, I assumed that the robot has perfect odometry measurements and that the robot could determine the exact location of humans and objects within its detection range. However, the robot may not know the exact location of itself and nearby obstacles in the real world. Physical robotic systems also exhibit control limitations that are ignored in simulation. For example, I assume that the robot can instantaneously achieve the exact velocity command, which is not realistic in the real world. In addition, assumptions made in simulation about the environment the robot could encounter may not hold true in the real world. For example, I assumed that all of the humans and objects could be represented using a circle of some radius and that all humans navigated according to the same collision-avoidance policy. However, this may not necessarily be true for all humans and objects the robot will encounter in its lifetime. For these reasons, algorithms that work well in simulation may not perform well in the real world. Therefore, to demonstrate that my algorithm could truly be applied to socially assistive robots navigating around real people and objects in the physical world, it is valuable to implement my algorithm on a real robot.

Robot Platform & Interface

For the real-world demonstrations, I used a TurtleBot kit, which includes a Kobuki mobile base, an Orbbec Astra camera, and a Gigabyte laptop computer. Figure 4.10 shows the robot platform I used for all the hardware demonstrations with all of the components labeled.

To control the TurtleBot, I designed a controller that receives odometry measurements from the Kobuki mobile base and receives RGB color images and depth images from the Orbbec Astra camera. The controller then uses this input to determine the position, velocity, angle, and radius of the robot and obstacles. This information about the state of the environment is fed into the modular policy described in Section 3.2. My policy uses this state information, along with information about the walls surrounding the robot, to determine the desired velocity of the robot. The TurtleBot controller receives this velocity and sends the appropriate command to the mobile base to move the robot. This process is carried out using the Robot Operating System (ROS) and is summarized in Figure 4.11.

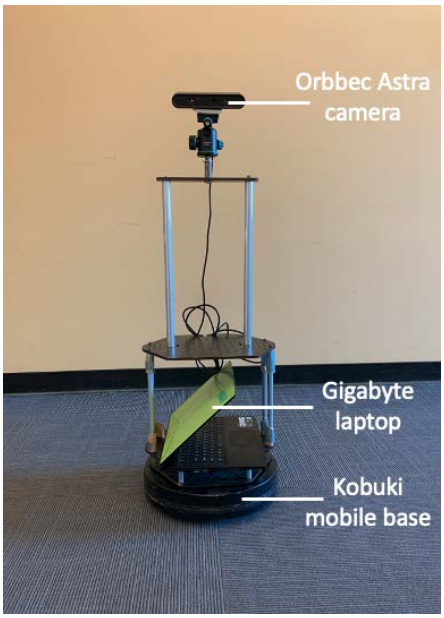

Figure 4.10: For all of my hardware demonstrations, I used a TurtleBot as the robot platform, which is composed of an Orbecc Astra camera, a Gigabyte laptop computer, and a Kobuki mobile base.

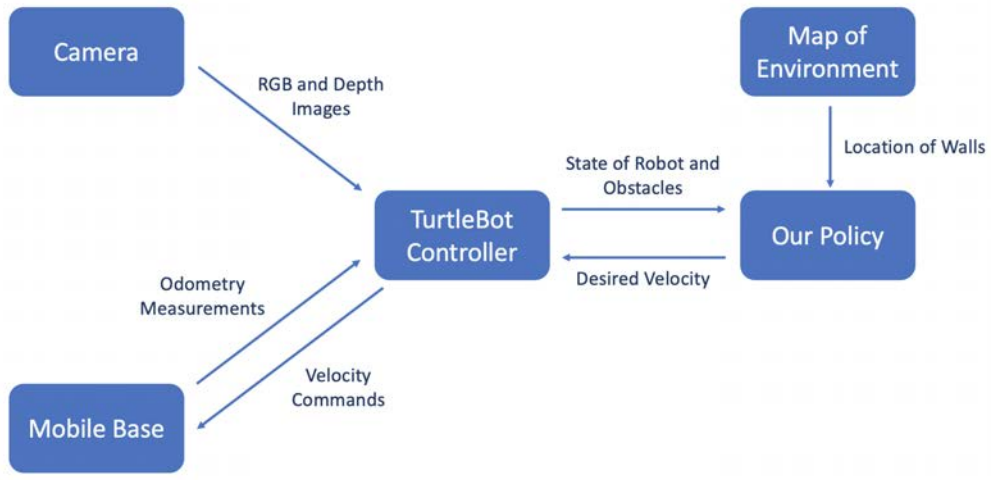

Figure 4.11: The TurtleBot controller communicates with the camera, mobile base, and my policy to control the robot in real-world environments around people and objects. This figure depicts the messages that are sent between these components during the hardware demonstrations.

Camera Calibration

Before performing the real-world demonstrations, the camera needed to be calibrated to accurately detect objects and humans within the robot's field of view. Keeping the robot in a fixed position, I mapped out its field of view as shown in Figure 4.12. I then measured various distances (d_l, d_r, d_f, and d_{offset}), which are labeled in Figure 4.12, and used these measurements to calculate the field of view angle:

$$\theta_{fov} = \cos^{-1}\left(\frac{d_l^2 + d_r^2 - d_f^2}{2d_l d_r}\right)$$

The offset distance d_{offset} and field of view angle θ_{fov} are fixed parameters used to determine the position, velocity, angle, and radius of humans and stationary objects surrounding the robot. The obstacle detection and tracking processes are described in the proceeding sections.

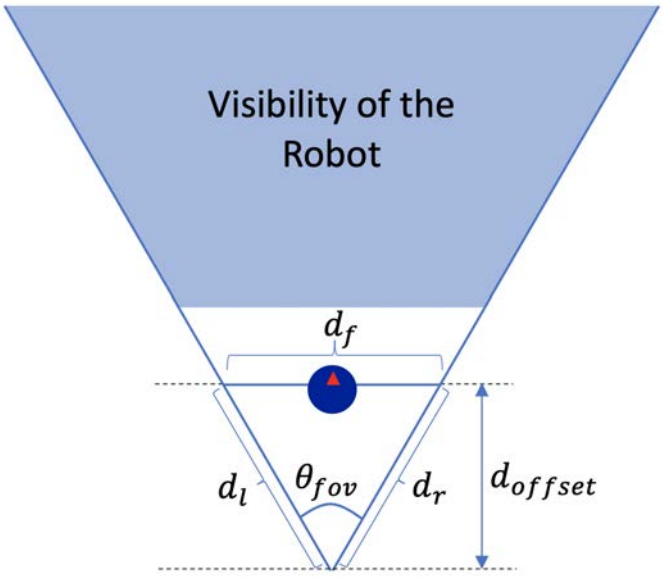

Figure 4.12: To calibrate the camera, the robot (blue circle) is placed in a fixed position pointing forward and its field of view (blue shaded region) is used to determine the offset distance d_{offset} and field of view angle θ_{fov}.

Obstacle Detection

To determine the position of a human or an object in the robot's field of view, I use the "You Only Look Once" (YOLO) real-time object detection system. This system passes an RGB color image through a single neural network to generate bounding boxes for each human and object in the image. An example bounding box is shown in Figure 4.13. For more information about the YOLO detection system, please refer to [36].

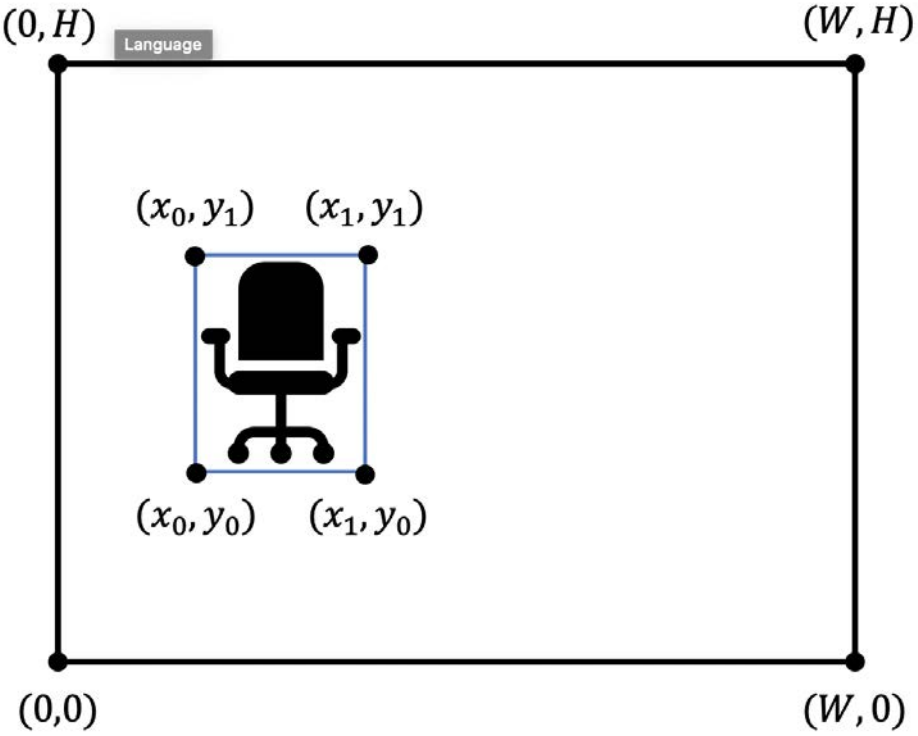

Figure 4.13: Given an RGB color image with a width of W and height of H, the YOLO detection system is used to generate a bounding box around each object in the image. An example bounding box for a chair in the image is shown with its corners labeled using the pixel positions.

After using the YOLO detection system to generate a bounding box for a given obstacle in the robot's field of view, I want to determine the angle ϕ shown in Figure 4.14. To do so, I first determine the horizontal position of the obstacle in terms of pixels:

$$x_c = x_0 + \frac{x_1 - x_0}{2}$$

I then use the horizontal pixel position of the obstacle in the image and the field of view angle calculated in the previous section to compute the angle of the obstacle:

$$\phi = \left(\frac{x_c}{W}\right)\theta_{fov} + \frac{\theta_{fov}}{2}$$

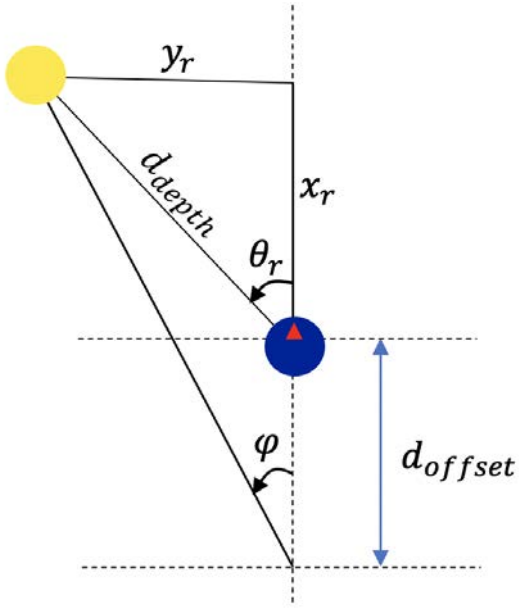

Figure 4.14: This figure shows the variables used to compute the position of an object (yellow circle) with respect to the robot (blue circle). In this diagram, d_{offset} is the offset distance determined during camera calibration, ϕ is the camera angle, d_{depth} is the distance of the object from the robot, θ_r is the angle between the object and robot, and (x_r, y_r) is the position of the object with respect to the robot.

I then use the bounding box obtained from YOLO in combination with the depth image to determine the distance of the object from the robot. More specifically, I use the pixels in the RGB image corresponding to bounding box of the obstacle to determine the depth at each associated position in the depth image. Ignoring regions where the view of the obstacle is obstructed and where the depth sensor cannot get a good measurement, I compute the median of the depth measurements of the obstacle to estimate its distance from the robot. This distance is then adjusted to account for inaccuracies in the depth camera, which depend on the angle ϕ. This distance is labeled as d_{depth} in Figure 4.14.

The next step is to find the angle of the obstacle with respect to the robot, which is labeled θ_r in Figure 4.14. Using properties of triangles, this angle can be calculated as

$$\theta_r = \phi + \sin^{-1}\left(\frac{d_{offset}\sin(\phi)}{d_{depth}}\right)$$

Using the depth measurement and the angle of the obstacle with respect to the robot, I can then determine the position of the obstacle with respect to the robot. The horizontal

position x_r and vertical position y_r (shown in Figure 4.14) are computed as

$$x_r = d_{depth} * \cos(\theta_r)$$

$$y_r = d_{depth} * \sin(\theta_r)$$

The position and angle of the robot with respect to the world frame are determined from the odometry measurements provided by the mobile base. Let $\left(p_x^{(r)}, p_y^{(r)}\right)$ be the position of the robot with respect to the global origin and $\theta^{(r)}$ be the angle of the robot. Using these values, the position of the obstacle with respect to the world frame is given by

$$p_x^{(o)} = x_r * \cos\left(\theta^{(r)}\right) - y_r * \sin\left(\theta^{(r)}\right) + p_x^{(r)}$$

$$p_y^{(o)} = x_r * \sin\left(\theta^{(r)}\right) + y_r * \cos\left(\theta^{(r)}\right) + p_y^{(r)}$$

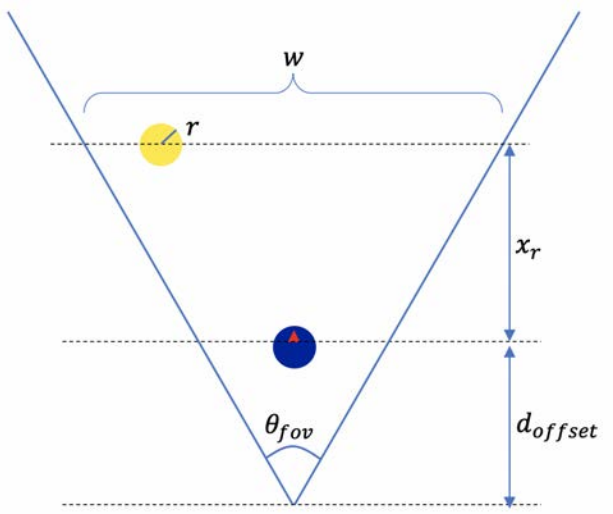

Figure 4.15: This figure shows the variables used to compute the radius of an object (yellow circle). In this diagram, d_{offset} is the offset distance determined during camera calibration, θ_{fov} is the field of view angle determined during camera calibration, x_r is the horizontal position of the object with respect to the robot, w is the width of the visible region at the given distance, and r is the radius of the object.

Finally, we need to determine the radius r of the obstacle as shown in Figure 4.15. Using knowledge of the amount of space the obstacle takes up in the image and the distance of the obstacle from the robot, the radius of the obstacle can be determined. First, notice that the width of the visible region at the depth of the obstacle, which I denote w, is given by

$$w = 2\left(x_r + d_{offset}\right) * \tan\left(\frac{\theta_{fov}}{2}\right)$$

Now we can use the width of the bounding box in pixels ($\Delta x := x_1 - x_0$) and the width of the image in pixels (W), along with the width of the visible region in meters (w), to compute the width of the bounding box in meters. The width of the bounding box will be considered the diameter of the obstacle. Therefore, the radius of the obstacle is given by

$$r = \frac{(x_1 - x_0)w}{2W}$$

Obstacle Tracking

Because the robot has a limited field of view, obstacles the robot detects are expected to remain nearby after they are no longer visible to the robot. To account for this, I developed a method to track detected obstacles. Because humans and stationary objects are expected to behave differently, each time an obstacle is detected, I first check whether it is a human or an object. I then check whether the human or object was previously seen.

If a detected object intersects with an already existing object, these objects are assumed to be the same. Otherwise, they are assumed to be two distinct objects. Every time the robot observes a new object, it stores the state of this object in memory. When the robot detects an object it has already seen, I update the state of the existing object using a weighted average of the state stored in memory and the position and radius determined from the current image. Based on the number of times an object was seen, the object remains in memory for some length of time after the robot last detected the object.

Similar to as is done for objects, each time the robot sees a new human, the current position and radius of the human are stored in memory. The next time this human is detected, its change in position over time is used to estimate the human's velocity. At each time step, the current position and velocity of the human are used to predict the position of the human at the next time step. The expected next position of previously seen humans is then used to predict whether a recently detected human is one of the existing humans. If a detected human is believed to be the same as an already existing human, the human's position is updated to be the position determined from the current image. The radius of the human is updated using a weighted average of the radius stored in memory and the radius determined by the current image. The velocity is also updated using the expected position and the position determined by the current image. Like objects, humans are expected to continue to exist after they exit the robot's field of view. However, unlike stationary objects, humans are expected to continue moving at their last seen velocity, and they remain in memory for a shorter time after they were last detected by the robot.

Action Selection

Obstacle detection and tracking are used to determine the states of humans and stationary objects, which are used by my modular policy to determine the best action for the robot to take. Recall that during simulation, I assumed the robot receives x and y velocity commands

as actions. The TurtleBot receives linear and angular velocity commands, so the velocity command sent to the mobile base needs to be adjusted accordingly. Because my controller was designed based on a simulated robot that does not face many real-world constraints, I modify the command produced by my controller before sending it to the robot. The TurtleBot cannot turn instantaneously before moving forward, as it does in simulation, so I slow the robot's linear velocity when the angular velocity is high. To reduce strain on the motors, which did not need to be considered in simulation, I also cap both the linear and angular acceleration rates. The TurtleBot controller continues to send commands generated by my modular controller until the robot reaches its final goal position.

Demonstration Performance

A video of my hardware demonstration is available here: `https://youtu.be/zd5c6yyiNgE`. This video seeks to demonstrate the role of each component in my modular framework, as well as the effectiveness of my complete framework in the real world.

For the first part of my hardware demonstration, I set up stationary objects in a hallway such that all three components of my modular framework–the RL policy, global path planner, and safety controller–would be actively engaged at various points along the robot's path. I then conducted an ablation study to demonstrate the role of each component in the real world. The hardware demonstration shows how the robot behaves with and without the global path planner and safety controller. I find that the robot navigates most effectively when my full policy is employed. Using the RL policy alone, the robot quickly collides with a wall in every trial. Adding the global path planner, the robot makes more progress towards the goal but still ends up colliding with a wall. Including the safety controller in addition to the global path planner, the robot navigates to the goal in every trial. This shows that my modular framework enables the effective usage of socially-aware RL policies in real indoor spaces.

In the second part of my hardware demonstration, I demonstrate the effectiveness of my modular framework in more complex environments. I show the robot using my complete policy to navigate through hallways, around corners, and through doorways in various academic buildings and office spaces. Using my policy, the robot is able to navigate effectively in the presence of static and dynamic people and objects.

Chapter 5

Conclusions & Future Work

5.1 Summary of Results

Through this project, I demonstrated the utility of standard robot navigation methods in enabling the effective usage of socially-aware RL policies for socially acceptable robot navigation in complex indoor spaces. In particular, I combined an RL policy with a path planning algorithm and an additional safety controller in a modular framework. I used the probabilistic roadmap (PRM) planning algorithm to generate waypoints from the robot's initial position to its goal, then used these waypoints to select local goals used by the RL policy. Rather than generating a representation of walls that can be passed into the RL policy network, wall avoidance is handled by a separate safety controller that limits the action space of the policy to only actions that are not expected to result in a collision with a wall. By combining all these elements in a modular framework, I enable a simulated robot to reach its goal from an arbitrary starting position, while limiting close encounters with humans and avoiding collisions with humans, stationary objects, and walls.

I found that my modular learning-based controller is able to achieve an overall success rate of over 99% when tested in various simulation environments designed to reflect the majority of scenarios a robot may encounter when navigating typical indoor spaces. When compared against a baseline navigation policy that uses the optimal reciprocal collision avoidance (ORCA) planner, along with the PRM planner, I found that my approach results in higher success rates in guiding the robot to the target destination and a fewer number of collisions with walls and stationary objects. This demonstrates the value of combining learning-based methods with more robust navigation components to develop socially compliant robot navigation policies that are safe and effective in real-world settings. I also employed my controller on a physical robot to demonstrate its utility in the real world. As research into deep RL policies increases, such a modular approach could help expedite the adoption of cutting edge RL policies in real-world applications to increase the use of socially assistive robots.

5.2 Future Directions

Impact of Learning

There are various avenues of future work related to the ideas presented in this paper. One potential direction for future work is to continue to explore the impact of learning on socially compliant robot navigation. By comparing my modular learning-based controller to a traditional controller that does not employ learning, I demonstrated the benefit of including a learning component to enable successful robot navigation around crowds of people and objects in indoor spaces. I also suspect that using an RL policy as a single component of a larger controller architecture would be more effective than training a more complex RL policy that could be used independently. It would be interesting to compare my modular controller to an end-to-end RL policy. Doing so may demonstrate that reinforcement learning is best used as a component of a larger system for certain robotic applications.

Optimal Path Selection & Adaptation

Another area for future work would be to explore even more complex indoor spaces, where there may be multiple potential paths between the robot and its goal. Within these more complex spaces, it would be interesting to consider ways to select the optimal path when multiple are available. This may allow the robot to select shorter or less crowded paths, or the robot may be able to choose a path that requires the most simple maneuvers. If the robot is operating in a familiar environment, it may also use some aspect of memory to select the optimal path in a given scenario. In environments where multiple paths are available, the robot may also be able to adapt if the first path it chooses is blocked. It would be interesting to consider ways to enable backtracking and alternative routing in such scenarios.

Safety Guarantees

Another important direction to explore would concern safety in socially compliant robot navigation. Before employing assistive robots in real-world applications around groups of humans, it is important to guarantee the safety and comfort of humans operating in these shared spaces. With this in mind, it would be interesting to consider worst-case robotic behavior and incorporate probabilistic safety guarantees. Within the general area of safety, it is also important to be aware that the robot may not behave as expected when it encounters new scenarios that were not seen during training or validation. It would be interesting to study the impact of distributional shift and explore ways to mitigate these effects. Along these same lines, research could be done into uncertainty estimation. If the robot encounters a very unfamiliar scenario, where it is not confident in its optimal action, it may be beneficial for the robot to default to a provably safe action or to ask for human intervention.

Additional Social Norms

Work could also be done to consider additional aspects of socially compliant robot behavior, beyond limiting close encounters with humans. It would be interesting to consider additional methods to maintain comfort around humans by encouraging robots to slow down around humans or avoid approaching humans from behind. This area of research may also benefit from user studies to determine which behaviors cause the greatest discomfort in humans and which controllers lead to the most comfortable robot behavior. Beyond the comfort aspect of socially compliant behavior, it would also be interesting to explore different types of social norms. For example, in certain regions, humans choose to walk on the right side of the hallway, so it may be beneficial for robots to follow these same norms. Some high-level social norms are also domain-specific, so it may be beneficial to consider specific applications. It would be interesting to collect data of humans operating in particular application domains to generate better human models and learn domain-specific social norms. With this data, one may be able to generate better robotic controllers for socially compliant navigation.

Task Planning

Another potential area for future work is in task planning. I considered cases in which the robot was tasked with traveling to a single goal. However, in real world applications, a robot may have multiple tasks, which lead to several goals it must travel to. It would be interesting to consider what changes need to be made to a robot navigation controller such as mine to enable effective task planning. This could also lead to interesting work in human-robot teaming, through which humans could give a robot a high level task to execute. Enabling a robot to receive high-level tasks would be useful to design robot guides for elderly people or people with disabilities and could be useful in the design of automated wheel chairs.

Interpretability

A final direction for future work that would be interesting to explore is interpretability in robot navigation. In the simulation environment I used, the simulated humans treat the robot as if it is another human and behave in a way that is consistent whether it is around a robot or other humans. Similarly, in the hardware demonstration, the people walking around the robot are comfortable with the robot, so they also move in a way that is generally predictable. However, outside of simulation and controlled experiments, humans are not necessarily comfortable around robots or familiar with how they operate. This may cause them to behave in a way that is not predictable to the robot. For this reason, it would be helpful to ensure that the robot's intent is clear to nearby humans so that these people may predict the robot's actions and move in a way that is consistent with how humans typically behave. It would be interesting to study methods to improve interpretability in robot navigation and to perform user studies to determine which methods are most effective.

Bibliography

[1] Neziha Akalin and Amy Loutfi. "Reinforcement Learning Approaches in Social Robotics". In: *arXiv:2009.09689 [cs]* (Feb. 2021). arXiv: 2009.09689. URL: http://arxiv.org/abs/2009.09689 (visited on 09/23/2021).

[2] Bobak H. Baghi and Gregory Dudek. "Sample Efficient Social Navigation Using Inverse Reinforcement Learning". In: *arXiv:2106.10318 [cs]* (June 2021). arXiv: 2106.10318. URL: http://arxiv.org/abs/2106.10318 (visited on 09/23/2021).

[3] Jur van den Berg, Ming Lin, and Dinesh Manocha. "Reciprocal Velocity Obstacles for real-time multi-agent navigation". In: *2008 IEEE International Conference on Robotics and Automation*. Pasadena, CA, USA: IEEE, May 2008, pp. 1928–1935. ISBN: 978-1-4244-1646-2. DOI: 10.1109/ROBOT.2008.4543489. URL: http://ieeexplore.ieee.org/document/4543489/ (visited on 10/02/2021).

[4] Jur van den Berg et al. "Reciprocal collision avoidance with acceleration-velocity obstacles". In: *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 3475–3482. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5980408. URL: http://ieeexplore.ieee.org/document/5980408/ (visited on 10/02/2021).

[5] Priyadarshi Bhattacharya and Marina Gavrilova. "Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path". In: *IEEE Robotics & Automation Magazine* 15.2 (June 2008), pp. 58–66. ISSN: 1070-9932. DOI: 10.1109/MRA.2008.921540. URL: http://ieeexplore.ieee.org/document/4539723/ (visited on 10/02/2021).

[6] Kuanqi Cai et al. "Mobile Robot Path Planning in Dynamic Environments: A Survey". In: *arXiv:2006.14195 [cs]* (Mar. 2021). arXiv: 2006.14195. DOI: 10.15878/j.cnki.instrumentation.2019.02.010. URL: http://arxiv.org/abs/2006.14195 (visited on 10/02/2021).

[7] Changan Chen et al. "Crowd-Robot Interaction: Crowd-aware Robot Navigation with Attention-based Deep Reinforcement Learning". In: *arXiv:1809.08835 [cs]* (Feb. 2019). arXiv: 1809.08835. URL: http://arxiv.org/abs/1809.08835 (visited on 09/20/2021).

[8] Yu Fan Chen et al. "Decentralized Non-communicating Multiagent Collision Avoidance with Deep Reinforcement Learning". In: *arXiv:1609.07845 [cs]* (Sept. 2016). arXiv: 1609.07845. URL: http://arxiv.org/abs/1609.07845 (visited on 09/23/2021).

[9] Yu Fan Chen et al. "Socially Aware Motion Planning with Deep Reinforcement Learning". In: *arXiv:1703.08862 [cs]* (May 2018). arXiv: 1703.08862. URL: http://arxiv.org/abs/1703.08862 (visited on 09/23/2021).

[10] Yuying Chen et al. "Robot Navigation in Crowds by Graph Convolutional Networks with Attention Learned from Human Gaze". In: *arXiv:1909.10400 [cs]* (Sept. 2019). arXiv: 1909.10400. URL: http://arxiv.org/abs/1909.10400 (visited on 09/20/2021).

[11] Eftychios G. Christoforou et al. "An Overview of Assistive Robotics and Technologies for Elderly Care". en. In: *XV Mediterranean Conference on Medical and Biological Engineering and Computing – MEDICON 2019*. Ed. by Jorge Henriques, Nuno Neves, and Paulo de Carvalho. Vol. 76. Series Title: IFMBE Proceedings. Cham: Springer International Publishing, 2020, pp. 971–976. ISBN: 978-3-030-31634-1 978-3-030-31635-8. DOI: 10.1007/978-3-030-31635-8_118. URL: http://link.springer.com/10.1007/978-3-030-31635-8_118 (visited on 09/20/2021).

[12] Grazia D'Onofrio et al. "Assistive robots for socialization in elderly people: results pertaining to the needs of the users". en. In: *Aging Clinical and Experimental Research* 31.9 (Sept. 2019), pp. 1313–1329. ISSN: 1720-8319. DOI: 10.1007/s40520-018-1073-z. URL: http://link.springer.com/10.1007/s40520-018-1073-z (visited on 09/20/2021).

[13] E. W. Dijkstra. "A note on two problems in connexion with graphs". en. In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0029-599X, 0945-3245. DOI: 10.1007/BF01386390. URL: http://link.springer.com/10.1007/BF01386390 (visited on 10/02/2021).

[14] Michael Everett, Yu Fan Chen, and Jonathan P. How. "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning". In: *arXiv:1805.01956 [cs]* (May 2018). arXiv: 1805.01956. URL: http://arxiv.org/abs/1805.01956 (visited on 09/23/2021).

[15] Muhammad Fahad, Zhuo Chen, and Yi Guo. "Learning How Pedestrians Navigate: A Deep Inverse Reinforcement Learning Approach". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, Oct. 2018, pp. 819–826. ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8593438. URL: https://ieeexplore.ieee.org/document/8593438/ (visited on 09/20/2021).

[16] D. Feil-Seifer and M.J. Mataric. "Socially Assistive Robotics". In: *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.* Chicago, IL, USA: IEEE, 2005, pp. 465–468. ISBN: 978-0-7803-9003-4. DOI: 10.1109/ICORR.2005.1501143. URL: http://ieeexplore.ieee.org/document/1501143/ (visited on 09/20/2021).

[17] Laura Fiorini et al. "Assistive robots to improve the independent living of older persons: results from a needs study". en. In: *Disability and Rehabilitation: Assistive Technology* 16.1 (Jan. 2021), pp. 92–102. ISSN: 1748-3107, 1748-3115. DOI: 10.1080/17483107.2019.1642392. URL: https://www.tandfonline.com/doi/full/10.1080/17483107.2019.1642392 (visited on 09/20/2021).

[18] Paolo Fiorini and Zvi Shiller. "Motion Planning in Dynamic Environments Using Velocity Obstacles". en. In: *The International Journal of Robotics Research* 17.7 (July 1998), pp. 760–772. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836499801700706. URL: http://journals.sagepub.com/doi/10.1177/027836499801700706 (visited on 10/02/2021).

[19] Cristina Getson and Goldie Nejat. "Socially Assistive Robots Helping Older Adults through the Pandemic and Life after COVID-19". en. In: *Robotics* 10.3 (Sept. 2021), p. 106. ISSN: 2218-6581. DOI: 10.3390/robotics10030106. URL: https://www.mdpi.com/2218-6581/10/3/106 (visited on 09/20/2021).

[20] Peter Hart, Nils Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136. URL: http://ieeexplore.ieee.org/document/4082128/ (visited on 10/02/2021).

[21] Juana Valeria Hurtado, Laura Londoño, and Abhinav Valada. "From Learning to Relearning: A Framework for Diminishing Bias in Social Robot Navigation". In: *Frontiers in Robotics and AI* 8 (Mar. 2021), p. 650325. ISSN: 2296-9144. DOI: 10.3389/frobt.2021.650325. URL: https://www.frontiersin.org/articles/10.3389/frobt.2021.650325/full (visited on 09/23/2021).

[22] M. Shamim Kaiser et al. "Healthcare Robots to Combat COVID-19". en. In: *COVID-19: Prediction, Decision-Making, and its Impacts*. Ed. by K.C. Santosh and Amit Joshi. Vol. 60. Series Title: Lecture Notes on Data Engineering and Communications Technologies. Singapore: Springer Singapore, 2021, pp. 83–97. ISBN: 9789811596810 9789811596827. DOI: 10.1007/978-981-15-9682-7_10. URL: https://link.springer.com/10.1007/978-981-15-9682-7_10 (visited on 09/20/2021).

[23] Karthik Karur et al. "A Survey of Path Planning Algorithms for Mobile Robots". en. In: *Vehicles* 3.3 (Aug. 2021), pp. 448–468. ISSN: 2624-8921. DOI: 10.3390/vehicles3030027. URL: https://www.mdpi.com/2624-8921/3/3/27 (visited on 10/02/2021).

[24] Kapil Katyal et al. "Group-Aware Robot Navigation in Crowded Environments". In: *arXiv:2012.12291 [cs]* (Dec. 2020). arXiv: 2012.12291. URL: http://arxiv.org/abs/2012.12291 (visited on 09/20/2021).

[25] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (Aug. 1996), pp. 566–580. ISSN: 1042296X. DOI: 10.1109/70.508439. URL: http://ieeexplore.ieee.org/document/508439/ (visited on 10/02/2021).

[26] Dimitrios Koutentakis, Alexander Pilozzi, and Xudong Huang. "Designing Socially Assistive Robots for Alzheimer's Disease and Related Dementia Patients and Their Caregivers: Where We Are and Where We Are Headed". en. In: *Healthcare* 8.2 (Mar. 2020), p. 73. ISSN: 2227-9032. DOI: 10.3390/healthcare8020073. URL: https://www.mdpi.com/2227-9032/8/2/73 (visited on 09/20/2021).

[27] Henrik Kretzschmar et al. "Socially compliant mobile robot navigation via inverse reinforcement learning". en. In: *The International Journal of Robotics Research* 35.11 (Sept. 2016), pp. 1289–1307. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364915619772. URL: http://journals.sagepub.com/doi/10.1177/0278364915619772 (visited on 09/20/2021).

[28] Thibault Kruse et al. "Human-aware robot navigation: A survey". en. In: *Robotics and Autonomous Systems* 61.12 (Dec. 2013), pp. 1726–1743. ISSN: 09218890. DOI: 10.1016/j.robot.2013.05.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889013001048 (visited on 09/23/2021).

[29] Markus Kuderer et al. "Feature-Based Prediction of Trajectories for Socially Compliant Navigation". In: *Robotics: Science and Systems VIII*. Robotics: Science and Systems Foundation, July 2012. ISBN: 978-0-262-51968-7. DOI: 10.15607/RSS.2012.VIII.025. URL: http://www.roboticsproceedings.org/rss08/p25.pdf (visited on 09/20/2021).

[30] Maria Kyrarini et al. "A Survey of Robots in Healthcare". en. In: *Technologies* 9.1 (Jan. 2021), p. 8. ISSN: 2227-7080. DOI: 10.3390/technologies9010008. URL: https://www.mdpi.com/2227-7080/9/1/8 (visited on 09/20/2021).

[31] Lucia Liu et al. "Robot Navigation in Crowded Environments Using Deep Reinforcement Learning". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 5671–5677. ISBN: 978-1-72816-212-6. DOI: 10.1109/IROS45743.2020.9341540. URL: https://ieeexplore.ieee.org/document/9341540/ (visited on 09/20/2021).

[32] Pinxin Long et al. "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning". In: *arXiv:1709.10082 [cs]* (May 2018). arXiv: 1709.10082. URL: http://arxiv.org/abs/1709.10082 (visited on 09/23/2021).

[33] Billy Okal and Kai O. Arras. "Learning socially normative robot navigation behaviors with Bayesian inverse reinforcement learning". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden: IEEE, May 2016, pp. 2889–2895. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487452. URL: http://ieeexplore.ieee.org/document/7487452/ (visited on 09/20/2021).

[34] Mark Pfeiffer et al. "Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Daejeon, South Korea: IEEE, Oct. 2016, pp. 2096–2101. ISBN: 978-1-5090-3762-9. DOI: 10.1109/IROS.2016.7759329. URL: http://ieeexplore.ieee.org/document/7759329/ (visited on 09/23/2021).

[35] Natasha Randall et al. "More than just friends: in-home use and design recommendations for sensing socially assistive robots (SARs) by older adults with depression". In: *Paladyn, Journal of Behavioral Robotics* 10.1 (June 2019), pp. 237–255. ISSN: 2081-4836. DOI: 10.1515/pjbr-2019-0020. URL: https://www.degruyter.com/document/doi/10.1515/pjbr-2019-0020/html (visited on 09/20/2021).

[36] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv:1506.02640 [cs]* (May 2016). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640 (visited on 02/17/2022).

[37] Sunil Srivatsav Samsani and Mannan Saeed Muhammad. "Socially Compliant Robot Navigation in Crowded Environment by Human Behavior Resemblance Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 6.3 (July 2021), pp. 5223–5230. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3071954. URL: https://ieeexplore.ieee.org/document/9399789/ (visited on 09/20/2021).

[38] A. Stentz. "Optimal and efficient path planning for partially-known environments". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. San Diego, CA, USA: IEEE Comput. Soc. Press, 1994, pp. 3310–3317. ISBN: 978-0-8186-5330-8. DOI: 10.1109/ROBOT.1994.351061. URL: http://ieeexplore.ieee.org/document/351061/ (visited on 10/02/2021).

[39] Steven LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998). URL: https://www.semanticscholar.org/paper/Rapidly-exploring-random-trees-%3A-a-new-tool-for-LaValle/d967d9550f831a8b3f5cb00f8835a4c866da60ad (visited on 10/02/2021).

[40] Lei Tai et al. "Socially Compliant Navigation through Raw Depth Inputs with Generative Adversarial Imitation Learning". In: *arXiv:1710.02543 [cs]* (Feb. 2018). arXiv: 1710.02543. URL: http://arxiv.org/abs/1710.02543 (visited on 09/20/2021).

[41] Chieh-En Tsai and Jean Oh. "NaviGAN: A Generative Approach for Socially Compliant Navigation". In: *arXiv:2007.05616 [cs]* (July 2020). arXiv: 2007.05616. URL: http://arxiv.org/abs/2007.05616 (visited on 09/20/2021).

Appendix A

Additional Data

Environment Layout	Path Planner	Safety Controller	Success Rate (%)	Collision Rate (%)			Timeout Rate (%)
				Human	Object	Wall	
Open Space	No	No	100	0	0	0	0
	No	Yes	100	0	0	0	0
	Yes	No	100	0	0	0	0
	Yes	Yes	100	0	0	0	0
Single Hallway	No	No	8	0	0	92	0
	No	Yes	92	0	0	0	8
	Yes	No	99	0	0	0	1
	Yes	Yes	99	0	0	0	1
Multiple Hallways	No	No	0	0	0	99	1
	No	Yes	2	0	0	7	91
	Yes	No	98	0	0	1	1
	Yes	Yes	98	0	0	0	2
Wide Intersection (1)	No	No	70	0	0	30	0
	No	Yes	93	1	0	0	6
	Yes	No	100	0	0	0	0
	Yes	Yes	100	0	0	0	0
Wide Intersection (2)	No	No	2	0	0	97	1
	No	Yes	12	0	0	0	88
	Yes	No	100	0	0	0	0
	Yes	Yes	100	0	0	0	0
Wide Intersection (3)	No	No	3	0	0	97	0
	No	Yes	12	0	0	0	88
	Yes	No	100	0	0	0	0
	Yes	Yes	100	0	0	0	0
Narrow Intersection (1)	No	No	22	0	0	77	1
	No	Yes	94	0	0	0	6
	Yes	No	100	0	0	0	0
	Yes	Yes	100	0	0	0	0
Narrow Intersection (2)	No	No	1	0	0	99	0
	No	Yes	11	0	0	0	89
	Yes	No	99	0	0	1	0
	Yes	Yes	99	0	0	0	1
Narrow Intersection (3)	No	No	1	0	0	99	0
	No	Yes	22	0	0	1	77
	Yes	No	100	0	0	0	0
	Yes	Yes	100	0	0	0	0

Table A.1: Controller Components and Performance Across All Layouts (1-9)

Environment Layout	Path Planner	Safety Controller	Success Rate (%)	Collision Rate (%) Human	Object	Wall	Timeout Rate (%)
Various Hallways (1)	No	No	0	0	0	100	0
	No	Yes	0	0	0	0	100
	Yes	No	93	0	0	6	1
	Yes	Yes	98	0	0	0	2
Various Hallways (2)	No	No	0	0	0	100	0
	No	Yes	0	0	0	2	98
	Yes	No	96	0	1	3	0
	Yes	Yes	100	0	0	0	0
Various Hallways (3)	No	No	0	0	0	100	0
	No	Yes	2	0	0	0	98
	Yes	No	95	0	0	5	0
	Yes	Yes	100	0	0	0	0
Various Hallways (4)	No	No	0	0	0	100	0
	No	Yes	0	0	0	87	13
	Yes	No	95	1	0	1	3
	Yes	Yes	97	1	0	0	2
Various Rooms (1)	No	No	2	0	0	98	0
	No	Yes	60	0	0	3	37
	Yes	No	98	1	0	0	1
	Yes	Yes	99	0	0	0	1
Various Rooms (2)	No	No	0	0	0	100	0
	No	Yes	7	0	0	1	92
	Yes	No	98	0	0	1	1
	Yes	Yes	99	0	0	0	1
Various Rooms (3)	No	No	0	0	0	100	0
	No	Yes	0	0	0	3	97
	Yes	No	95	0	0	5	0
	Yes	Yes	99	0	0	0	1
Various Rooms (4)	No	No	0	0	0	100	0
	No	Yes	0	0	0	0	100
	Yes	No	96	1	0	2	1
	Yes	Yes	99	0	0	0	1

Table A.2: Controller Components and Performance Across All Layouts (10-17)

Environment Layout	Path Planner	Safety Active (%) (avg ± std)	Safety Speed (m/s) (avg ± std)
Open Space	No	0.00 ± 0.00	–
	Yes	0.00 ± 0.00	–
Single Hallway	No	57.08 ± 26.70	0.23 ± 0.18
	Yes	0.11 ± 0.96	0.76 ± 0.22
Multiple Hallways	No	71.96 ± 19.85	0.26 ± 0.20
	Yes	2.52 ± 8.88	0.30 ± 0.16
Wide Intersection (1)	No	8.48 ± 16.99	0.27 ± 0.20
	Yes	0.00 ± 0.00	–
Wide Intersection (2)	No	80.08 ± 20.65	0.20 ± 0.14
	Yes	0.09 ± 0.88	0.58 ± 0.28
Wide Intersection (3)	No	73.01 ± 23.70	0.22 ± 0.18
	Yes	0.03 ± 0.28	0.90 ± 0.14
Narrow Intersection (1)	No	35.55 ± 25.75	0.24 ± 0.18
	Yes	0.00 ± 0.00	–
Narrow Intersection (2)	No	91.20 ± 16.23	0.16 ± 0.10
	Yes	1.54 ± 9.34	0.21 ± 0.10
Narrow Intersection (3)	No	88.26 ± 17.34	0.16 ± 0.10
	Yes	4.38 ± 8.72	0.58 ± 0.21
Various Hallways (1)	No	76.15 ± 25.81	0.23 ± 0.19
	Yes	2.66 ± 5.31	0.53 ± 0.21
Various Hallways (2)	No	92.05 ± 9.74	0.19 ± 0.14
	Yes	2.34 ± 4.27	0.59 ± 0.22
Various Hallways (3)	No	91.75 ± 11.33	0.17 ± 0.12
	Yes	2.04 ± 4.28	0.54 ± 0.22
Various Hallways (4)	No	72.90 ± 35.72	0.22 ± 0.15
	Yes	1.10 ± 3.44	0.47 ± 0.21
Various Rooms (1)	No	59.69 ± 22.60	0.28 ± 0.23
	Yes	0.11 ± 0.72	0.73 ± 0.16
Various Rooms (2)	No	88.17 ± 18.99	0.17 ± 0.12
	Yes	6.46 ± 2.64	0.85 ± 0.17
Various Rooms (3)	No	81.51 ± 24.99	0.20 ± 0.15
	Yes	3.89 ± 2.20	0.79 ± 0.20
Various Rooms (4)	No	81.37 ± 9.95	0.19 ± 0.15
	Yes	5.43 ± 3.32	0.78 ± 0.20

Table A.3: Safety Controller Analysis Across All Layouts

Environment Layout	Navigation Strategy	Success Rate (%)	Collision Rate (%)			Timeout Rate (%)
			Human	Object	Wall	
Open Space	Mine	100	0	0	0	0
	Baseline	99	0	1	0	0
Single Hallway	Mine	99	0	0	0	1
	Baseline	96	0	2	0	2
Multiple Hallways	Mine	98	0	0	0	2
	Baseline	97	0	1	0	2
Wide Intersection (1)	Mine	100	0	0	0	0
	Baseline	98	0	1	0	1
Wide Intersection (2)	Mine	100	0	0	0	0
	Baseline	97	0	3	0	0
Wide Intersection (3)	Mine	100	0	0	0	0
	Baseline	98	0	1	0	1
Narrow Intersection (1)	Mine	100	0	0	0	0
	Baseline	98	0	1	0	1
Narrow Intersection (2)	Mine	99	0	0	0	1
	Baseline	98	0	0	0	2
Narrow Intersection (3)	Mine	100	0	0	0	0
	Baseline	96	0	3	0	1
Various Hallways (1)	Mine	98	0	0	0	2
	Baseline	89	0	6	0	5
Various Hallways (2)	Mine	100	0	0	0	0
	Baseline	92	0	5	0	3
Various Hallways (3)	Mine	100	0	0	0	0
	Baseline	94	0	2	0	4
Various Hallways (4)	Mine	97	1	0	0	2
	Baseline	89	0	6	0	5
Various Rooms (1)	Mine	99	0	0	0	1
	Baseline	95	0	1	0	4
Various Rooms (2)	Mine	99	0	0	0	1
	Baseline	93	2	1	1	3
Various Rooms (3)	Mine	99	0	0	0	1
	Baseline	92	0	6	1	1
Various Rooms (4)	Mine	99	0	0	0	1
	Baseline	92	0	6	1	1

Table A.4: Learning and Performance Rates Across All Layouts

Environment Layout	Navigation Strategy	Navigation Time (s) (avg ± std)	Path Length (m) (avg ± std)	Speed (m/s) (avg ± std)
Open Space	Mine	7.65 ± 0.27	7.65 ± 0.27	1.00 ± 0.00
	Baseline	7.61 ± 0.17	7.47 ± 0.06	0.98 ± 0.06
Single Hallway	Mine	20.25 ± 0.91	20.24 ± 0.88	1.00 ± 0.00
	Baseline	19.88 ± 0.58	19.55 ± 0.12	0.98 ± 0.06
Multiple Hallways	Mine	41.01 ± 2.34	40.59 ± 0.96	0.99 ± 0.02
	Baseline	39.93 ± 0.65	39.43 ± 0.33	0.99 ± 0.05
Wide Intersection (1)	Mine	32.09 ± 0.44	32.09 ± 0.44	1.00 ± 0.00
	Baseline	31.83 ± 0.34	31.54 ± 0.11	0.99 ± 0.04
Wide Intersection (2)	Mine	25.48 ± 0.59	25.47 ± 0.55	1.00 ± 0.00
	Baseline	24.97 ± 0.29	24.76 ± 0.16	0.99 ± 0.04
Wide Intersection (3)	Mine	25.41 ± 0.59	25.40 ± 0.58	1.00 ± 0.00
	Baseline	25.22 ± 0.50	24.90 ± 0.20	0.99 ± 0.05
Narrow Intersection (1)	Mine	20.10 ± 0.76	20.09 ± 0.74	1.00 ± 0.00
	Baseline	19.79 ± 0.38	19.54 ± 0.11	0.99 ± 0.05
Narrow Intersection (2)	Mine	17.20 ± 0.47	17.18 ± 0.40	1.00 ± 0.01
	Baseline	16.94 ± 0.44	16.68 ± 0.17	0.98 ± 0.05
Narrow Intersection (3)	Mine	16.99 ± 2.04	16.61 ± 0.67	0.98 ± 0.04
	Baseline	16.31 ± 0.42	16.08 ± 0.22	0.99 ± 0.05
Various Hallways (1)	Mine	45.22 ± 2.58	44.66 ± 1.21	0.99 ± 0.04
	Baseline	43.80 ± 0.86	43.20 ± 0.33	0.99 ± 0.05
Various Hallways (2)	Mine	46.93 ± 2.88	46.44 ± 2.35	0.99 ± 0.04
	Baseline	45.49 ± 1.01	44.88 ± 0.40	0.99 ± 0.05
Various Hallways (3)	Mine	47.80 ± 7.59	47.25 ± 6.71	0.99 ± 0.04
	Baseline	45.18 ± 0.98	44.46 ± 0.35	0.98 ± 0.06
Various Hallways (4)	Mine	46.09 ± 5.07	45.62 ± 4.12	0.99 ± 0.03
	Baseline	44.33 ± 0.82	43.78 ± 0.42	0.99 ± 0.05
Various Rooms (1)	Mine	32.45 ± 0.79	32.43 ± 0.75	1.00 ± 0.00
	Baseline	31.91 ± 0.56	31.58 ± 0.15	0.99 ± 0.04
Various Rooms (2)	Mine	28.24 ± 4.95	27.93 ± 4.75	0.99 ± 0.06
	Baseline	27.60 ± 0.52	26.76 ± 0.24	0.97 ± 0.12
Various Rooms (3)	Mine	51.29 ± 2.52	50.78 ± 1.78	0.99 ± 0.05
	Baseline	49.74 ± 1.22	49.13 ± 0.33	0.99 ± 0.05
Various Rooms (4)	Mine	51.25 ± 1.14	50.70 ± 0.84	0.99 ± 0.06
	Baseline	49.78 ± 1.04	49.17 ± 0.30	0.99 ± 0.05

Table A.5: Learning and Navigation Performance Across All Layouts

Environment Layout	Navigation Strategy	Avg. Distance (m) (avg ± std)	Minimum Distance (m)	Discomfort Time (%) (avg ± std)
Open Space	Mine	1.58 ± 0.74	0.10	0.00 ± 0.00
	Baseline	1.57 ± 0.73	0.04	0.21 ± 2.11
Single Hallway	Mine	2.32 ± 1.63	0.06	0.06 ± 0.61
	Baseline	2.31 ± 1.64	0.02	0.17 ± 0.95
Multiple Hallways	Mine	2.29 ± 1.55	0.08	0.05 ± 0.26
	Baseline	2.20 ± 1.54	0.04	0.11 ± 0.49
Wide Intersection (1)	Mine	1.66 ± 1.19	0.00	0.09 ± 0.48
	Baseline	1.67 ± 1.22	0.03	0.18 ± 0.80
Wide Intersection (2)	Mine	2.00 ± 1.34	0.04	0.06 ± 0.38
	Baseline	2.07 ± 1.37	0.04	0.08 ± 0.57
Wide Intersection (3)	Mine	2.17 ± 1.42	0.09	0.01 ± 0.09
	Baseline	2.15 ± 1.41	0.08	0.03 ± 0.30
Narrow Intersection (1)	Mine	2.13 ± 1.52	0.06	0.09 ± 0.48
	Baseline	2.09 ± 1.48	0.04	0.15 ± 0.98
Narrow Intersection (2)	Mine	2.43 ± 1.62	0.12	0.00 ± 0.03
	Baseline	2.31 ± 1.51	0.12	0.00 ± 0.00
Narrow Intersection (3)	Mine	2.36 ± 1.54	0.06	0.09 ± 0.46
	Baseline	2.25 ± 1.54	0.03	0.49 ± 2.71
Various Hallways (1)	Mine	2.26 ± 1.86	0.01	0.19 ± 0.76
	Baseline	2.16 ± 1.83	0.03	0.21 ± 0.68
Various Hallways (2)	Mine	2.25 ± 1.81	0.00	0.14 ± 0.50
	Baseline	2.21 ± 1.76	0.03	0.24 ± 0.91
Various Hallways (3)	Mine	2.16 ± 1.76	0.04	0.13 ± 0.44
	Baseline	2.21 ± 1.83	0.03	0.21 ± 0.86
Various Hallways (4)	Mine	2.14 ± 1.76	0.01	0.55 ± 3.38
	Baseline	2.09 ± 1.64	0.03	0.20 ± 0.71
Various Rooms (1)	Mine	2.09 ± 1.21	0.06	0.05 ± 0.31
	Baseline	2.14 ± 1.29	0.03	0.12 ± 0.73
Various Rooms (2)	Mine	2.24 ± 1.54	0.06	0.04 ± 0.18
	Baseline	2.27 ± 1.58	0.01	0.29 ± 1.33
Various Rooms (3)	Mine	1.99 ± 1.46	0.02	0.11 ± 0.36
	Baseline	2.03 ± 1.48	0.03	0.13 ± 0.58
Various Rooms (4)	Mine	1.98 ± 1.42	0.03	0.07 ± 0.38
	Baseline	1.95 ± 1.42	0.04	0.16 ± 0.67

Table A.6: Learning and Social Compliance Across All Layouts