**Collision Avoidance in Pedestrian-Rich Environments with Deep Reinforcement Learning**

**Abstract**

Collision avoidance algorithms are essential for safe and efficient robot operation among pedestrians. This work proposes using deep reinforcement (RL) learning as a framework to model the complex interactions and cooperation with nearby, decision-making agents, such as pedestrians and other robots. Existing RL-based works assume homogeneity of agent properties, use specific motion models over short timescales, or lack a principled method to handle a large, possibly varying number of agents. Therefore, this work develops an algorithm that learns collision avoidance among a variety of heterogeneous, noncommunicating, dynamic agents without assuming they follow any particular behavior rules. It extends our previous work by introducing a strategy using Long Short-Term Memory (LSTM) that enables the algorithm to use observations of an arbitrary number of other agents, instead of a small, fixed number of neighbors.

The proposed algorithm is shown to outperform a classical collision avoidance algorithm, another deep RL-based algorithm, and scales with the number of agents better (fewer collisions, shorter time to goal) than our previously published learning-based approach. Analysis of the LSTM provides insights into how observations of nearby agents affect the hidden state and quantifies the performance impact of various agent ordering heuristics. The learned policy generalizes to several applications beyond the training scenarios: formation control (arrangement into letters), demonstrations on a fleet of four multirotors and on a fully autonomous robotic vehicle capable of traveling at human walking speed among pedestrians.

**INDEX TERMS:** Collision Avoidance, Deep Reinforcement Learning, Motion Planning, Multiagent Systems, Decentralized Execution

**CHAPTER 1**

**INTRODUCTION**

A fundamental challenge in autonomous vehicle operation is to safely negotiate interactions with other dynamic agents in the environment. For example, it is important for self-driving cars to take other vehicles’ motion into account, and for delivery robots to avoid colliding with pedestrians. While there has been impressive progress in the past decade [1], fully autonomous navigation remains challenging, particularly in uncertain, dynamic environments cohabited by other mobile agents. The challenges arise because the other agents’ intents and policies (i.e., goals and desired paths) are typically not known to the planning system, and, furthermore, explicit communication of such hidden quantities is often impractical due to physical limitations.

These issues motivate the use of decentralized collision avoidance algorithms. Existing work on decentralized collision avoidance can be classified into cooperative and non-cooperative methods. Non-cooperative methods first predict the other agents’ motion and then plan a collision-free path for the vehicle with respect to the other agents’ predicted motion. However, this can lead to the freezing robot problem [2], where the vehicle fails to find any feasible path because the other agents’ predicted paths would occupy a large portion of the traversable space.

Cooperative methods address this issue by modeling interaction in the planner, such that the vehicle’s action can influence the other agent’s motion, thereby having all agents share the responsibility for avoiding collision. Cooperative methods include reaction-based methods [3], [4], [5], [6] and trajectory-based methods [7], [8], [9]. This work seeks to combine the best of both types of cooperative techniques – the computational efficiency of reaction-based methods and the smooth motion of trajectory based methods.

To this end, the work presents the collision avoidance with deep reinforcement learning (CADRL) algorithm, which tackles the aforementioned trade-off between computation time and smooth motion by using reinforcement learning (RL) to offload the expensive online computation to an offline learning procedure. Specifically, a computationally efficient (i.e., real-time implementable) interaction rule is developed by learning a policy that implicitly encodes cooperative behaviors.

Learning the collision avoidance policy for CADRL presents several challenges. A first key challenge is that the number of other agents in the environment can vary between timesteps or experiments, however the typical feedforward neural networks used in this domain require a fixed dimension input. Our prior work defines a maximum number of agents that the network can observe, and other approaches use raw sensor data as the input [10], [11].

This work instead uses an idea from Natural Language Processing [12], [13] to encode the varying size state of the world (e.g., positions of other agents) into a fixed-length vector, using long short-term memory (LSTM) [14] cells at the network input. This enables the algorithm to make decisions based on an arbitrary number of other agents in the robot’s vicinity. A second fundamental challenge is in finding a policy that makes realistic assumptions about other agents’ belief states, policies, and intents.

This work learns a collision avoidance policy without assuming that the other agents follow any particular behavior model and without explicit assumptions on homogeneity [10] (e.g., agents of the same size and nominal speed) or specific motion models (e.g., constant velocity) over short timescales [15], [16].

The main contributions of this work are:

• a new collision avoidance algorithm that greatly outperforms prior works as the number of agents in the environments is increased: a key factor in that improvement is to relax the assumptions on the other agents’ behavior models during training and inference,

• a novel use of LSTM in that it encodes spatial representations, rather than temporal, to address the challenge that the number of neighboring agents could be large and could vary in time, • simulation results that show significant improvement in solution quality compared with other recently published state-of-the-art methods (such as [5], [16], [10]), and

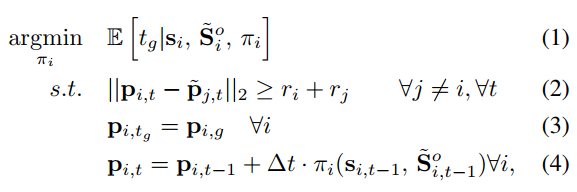
• hardware experiments with aerial and ground robots to demonstrate that the proposed algorithm can be deployed in real time on robots with real sensors. Open-source software based on this manuscript includes a pre-trained collision avoidance policy (as a ROS package) cadrl\_ros1, the GA3C-CADRL learning algorithm2, and a simulation/training environment with several implemented policies, gym\_collision\_avoidance3. Videos of the experimental results are posted at https://youtu.be/ Bjx4ZEov0yE.

This work is based on [15], [16], [17] and extends them as follows: (i) expanded discussion and example of the limitations of the prior work, (ii) further explanation of the proposed algorithm, including pseudo-code, (iii) analysis on the effect of sequence ordering in LSTM, which addresses a primary gap in the prior work, (iv) quantifying input gate activation to provide deeper intuition on why the proposed use of LSTM works, (v) additional comparisons to model and learning-based collision avoidance algorithms, (vi) ablation study of the proposed algorithm, and (vii) experiments with formation control and on real multirotors to demonstrate generalizability of the learned policy.

**A. PROBLEM FORMULATION**

The non-communicating, multiagent collision avoidance problem can be formulated as a sequential decision making problem [15], [16]. In an n-agent scenario (N≤n = f1; 2; : : : ; ng), denote the joint world state, sjn t , agent i’s state, si;t, and agent i’s action, ui;t, 8i 2 N≤n. Each agent’s state vector is composed of an observable and unobservable (hidden) portion, si;t = [so i;t; sh i;t].

In the global frame, observable states are the agent’s position, velocity, and radius, so = [px; py; vx; vy; r] 2 R5, and unobservable states are the goal position, preferred speed, and orientation4, sh = [pgx; pgy; vpref; ] 2 R4. The action is a speed and heading angle, ut = [vt; t] 2 R2. The observable states of all n−1 other agents is denoted, S~o i;t = f~so j;t : j 2 N≤n nig. A policy, π : s0:t; S~o 0:t 7! ut, is developed with the objective of minimizing expected time to goal E[tg] while avoiding collision with other agents,



where (2) is the collision avoidance constraint, (3) is the goal constraint, (4) is the agents’ kinematics, and the expectation in (1) is with respect to the other agent’s unobservable states (intents) and policies. Although it is difficult to solve for the optimal solution of (1)-(4), this problem formulation can be useful for understanding the limitations of the existing methods. In particular, it provides insights into the approximations/assumptions made by existing works.

**B. RELATED WORK**

Most approaches to collision avoidance with dynamic obstacles employ model-predictive control (MPC) [18] in which a planner selects a minimum cost action sequence, ui;t:t+T , using a prediction of the future world state, P (sjn t+1:t+T +1jsjn 0:t; ui;t:t+T ), conditioned on the world state history, sjn 0:t. While the first actions in the sequence are being implemented, the subsequent action sequence is updated by re-planning with the updated world state information (e.g., from new sensor measurements). The prediction of future world states is either prescribed using domain knowledge (model-based approaches) or learned from examples/experiences (learning-based approaches).

**1) Model-based approaches**

Early approaches model the world as a static entity, [vx; vy] = 0, but replan quickly to try to capture the motion through updated (px; py) measurements [19]. This leads to time-inefficient paths among dynamic obstacles, since the planner’s world model does not anticipate future changes in the environment due to the obstacles’ motion. To improve the predictive model, reaction-based methods use one-step interaction rules based on geometry or physics to ensure collision avoidance.

These methods [5], [4], [6] often specify a Markovian policy, π(sjn 0:t) = π(sjn t ), that optimizes a one-step cost while satisfying collision avoidance constraints. For instance, in velocity obstacle approaches [5], [6], an agent chooses a collision-free velocity that is closest to its preferred velocity (i.e., directed toward its goal). Given this one-step nature, reaction-based methods do account for current obstacle motion, but do not anticipate the other agents’ hidden intents – they instead rely on a fast update rate to react quickly to the other agents’ changes in motion.

Although computationally efficient given these simplifications, reaction-based methods are myopic in time, which can sometimes lead to generating unnatural trajectories [8], [15]. Trajectory-based methods compute plans on a longer timescale to produce smoother paths but are often computationally expensive or require knowledge of unobservable states. A subclass of non-cooperative approaches [20], [21] propagates the other agents’ dynamics forward in time and then plans a collision-free path with respect to the other agents’ predicted paths.

However, in crowded environments, the set of predicted paths could occupy a large portion of the space, which leads to the freezing robot problem [2]. A key to resolving this issue is to account for interactions, such that each agent’s motion can affect one another. Thereby, a subclass of cooperative approaches [7], [8], [9] has been proposed, which solve (1)-(4) in two steps. First, the other agents’ hidden states (i.e., goals) are inferred from their observed trajectories, S^ ~h t = f(S~o 0:t), where f(·) is a inference function. Second, a centralized path planning algorithm, π(s0:t; S~o 0:t) = πcentral(st; S~o t ; S^ ~h t ), is employed to find jointly feasible paths. By planning/anticipating complete paths, trajectory-based methods are no longer myopic.

However, both the inference and the planning steps are computationally expensive, and need to be carried out online at each new observation (sensor update S~o t).

**2) Learning-based approaches**

Our recent works [15], [16] proposed a third category that uses a reinforcement learning framework to solve (1)-(4). As in the reactive-based methods, we make a Markovian assumption: π(sjn 0:t) = π(sjn t ). The expensive operation of modeling the complex interactions is learned in an offline training step, whereas the learned policy can be queried quickly online, combining the benefits of both reactive- and trajectory-based methods.

Our prior methods pre-compute a value function, V (sjn), that estimates the expected time to the goal from a given configuration, which can be used to select actions using a one-step lookahead procedure described in those works. To avoid the lookahead procedure, this work directly optimizes a policy π(sjn) to select actions to minimize the expected time to the goal. The differences from other learning-based approaches will become more clear after a brief overview of reinforcement learning.

**C. REINFORCEMENT LEARNING**

RL [22] is a class of machine learning methods for solving sequential decision making problems with unknown statetransition dynamics. Typically, a sequential decision making problem can be formulated as a Markov decision process (MDP), which is defined by a tuple M = hS; A; P; R; γi, where S is the state space, A is the action space, P is the state-transition model, R is the reward function, and γ is a discount factor. By detailing each of these elements and relating to (1)-(4), the following provides a RL formulation of the n-agent collision avoidance problem.

**State space**

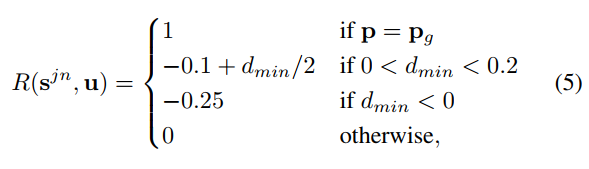
The joint world state, sjn, was defined in Section II-A.

**Action space**

The choice of action space depends on the vehicle model. A natural choice of action space for differential drive robots is a linear and angular speed (which can be converted into wheel speeds), that is, u = [s; !]. The action space is either discretized directly, or represented continuously by a function of discrete parameters.

**Reward function**

A sparse reward function is specified to award the agent for reaching its goal (3), and penalize the agent for getting too close or colliding with other agents (2),



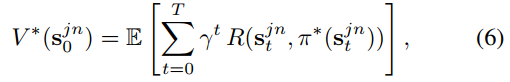
where dmin is the distance to the closest other agent. Optimizing the hyperparameters (e.g., -0.25) in Rcol is left for future work. Note that we use discount γ < 1 to encourage efficiency instead of a step penalty.

**State transition model:**

A probabilistic state transition model, P (sjn t+1jsjn t ; ut), is determined by the agents’ kinematics as defined in (4). Since the other agents’ actions also depend on their policies and hidden intents (e.g., goals), the system’s state transition model is unknown.

**Value function**

One method to find the optimal policy is to first find the optimal value function,



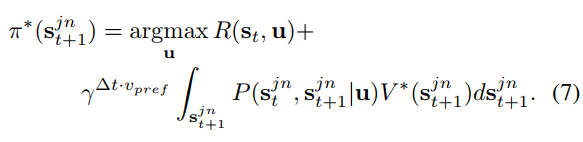
where γ 2 [0; 1) is a discount factor. Many methods exist to estimate the value function in an offline training process [22].

**Deep Reinforcement Learning**

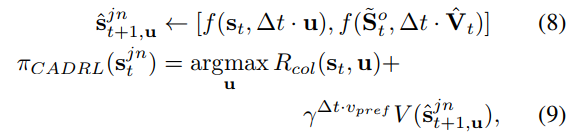
To estimate the high-dimensional, continuous value function (and/or associated policy), it is common to approximate with a deep neural network (DNN) parameterized by weights and biases, θ, as in [23]. This work’s notation drops the parameters except when possible, e.g., V (s; θ) = V (s).

**Decision-making Policy**

A value function of the current state can be implemented as a policy,



Our previous works avoid the complexity in explicitly modeling P (sjn t+1jsjn t ; u) by assuming that other agents continue their current velocities, V^ t, for a duration ∆t, meaning the policy can be extracted from the value function



under the simple kinematic model, f. However, the introduction of parameter ∆t leads to a difficult trade-off. Due to the approximation of the value function in a DNN, a sufficiently large ∆t is required such that each propagated ^sjn t+1;u is far enough apart, which ensures V (^sjn t+1;u) is not dominated by numerical noise in the network. The implication of large ∆t is that agents are assumed to follow a constant velocity for a significant amount of time, which neglects the effects of cooperation/reactions to an agent’s decisions.

As the number of agents in the environment increases, this constant velocity assumption is less likely to be valid. Agents do not actually reach their propagated states because of the multiagent interactions. The impact of separately querying the value function and performing collision checking is illustrated in Fig. 1. In (a), a red agent aims to reach its goal (star), and a purple agent is traveling at 1 m/s in the −y-direction. Because CADRL’s value function only encodes time-to-goal information, (b) depicts that the DNN appropriately recommends that the red agent should cut above the purple agent. However, there is a second term in (9) to convert the value function into a policy.

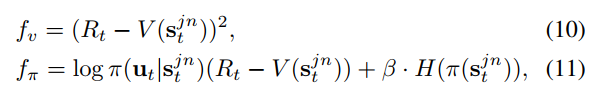
This second term, the collision cost, Rcol(st; u), shown in (c), penalizes actions that move toward the other agent’s predicted position (dashed circle). This model-based collision checking procedure requires an assumption about other agents’ behaviors, which is difficult to define ahead of time; the prior work assumed a constant-velocity model. When the value and collision costs are combined to produce πCADRL(sjn t ), the resulting objective-maximizing action is for the red agent to go straight, which will avoid a collision but be inefficient for both agents.

The challenge in defining a model for other agents’ behaviors was a primary motivation for learning a value function; even with an accurate value function, this example demonstrates an additional cause of inefficient paths: an inaccurate model used in the collision checking procedure. In addition to not capturing decision making behavior of other agents, our experiments suggest that ∆t is a crucial parameter to ensure convergence while training the DNNs in the previous algorithms. If ∆t is set too small or large, the training does not converge.

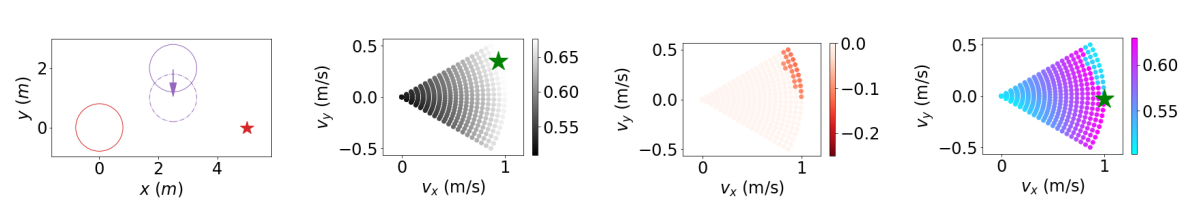
A value of ∆t = 1 sec was experimentally determined to enable convergence, though this number does not have much theoretical rationale. In summary, the challenges of converting a value function to a policy, choosing the ∆t hyperparameter, and our observation that the learning stability suffered with more than 4 agents in the environment each motivate the use of a different RL framework. To address the concerns raised about ∆T propagation, this work proposes a new algorithm that does not project agents forward during policy evaluation, thus eliminating the need for tuning the ∆t hyperparameter.

**Policy Learning**

Therefore, this work considers RL frameworks which generate a policy that an agent can execute directly, without any arbitrary assumptions about state transition dynamics. A recent actor-critic algorithm called A3C [24] uses a single DNN to approximate both the value (critic) and policy (actor) functions, and is trained with two loss terms



where (10) trains the network’s value output to match the future discounted reward estimate, Rt = Pk i=0 −1 γirt+i + γkV (sjn t+k), over the next k steps, just as in CADRL. For the



1. 2-Agent Scenario (b) Time-to-Goal Estimate, V (^sjn t+1;u) from DNN (c) Collision Cost, Rcol(st; u), using const. velocity model (d) Objective in πCADRL(sjn t )

FIGURE 1: Issue with checking collisions and state-value separately, as in (9). In (a), the red agent’s goal is at the star, and the purple agent’s current velocity is in the −y-direction. In (b), the CADRL algorithm propagates the other agent forward at its current velocity (dashed purple circle), then queries the DNN for candidate future states. The best action (green star) is one which cuts above the purple agent, which was learned correctly by the CADRL V-Learning procedure. However, the constant velocity model of other agents is also used for collision checking, causing penalties of Rcol(st; u), shown in (c). CADRL’s policy combines these terms (d), instead choosing to go straight (green star), which is a poor choice that ignores that a cooperative purple agent likely would adjust its own velocity as well. This fundamental issue of checking collisions and state-values separately is addressed in this work by learning a policy directly

policy output in (11), the first term penalizes actions which have high probability of occurring (log π) that lead to a lower return than predicted by the value function (R − V ), and the second term encourages exploration by penalizing π’s entropy with tunable constant β. In A3C, many threads of an agent interacting with an environment are simulated in parallel, and a policy is trained based on an intelligent fusion of all the agents’ experiences.

The algorithm was shown to learn a policy that achieves super-human performance on many video games. We specifically use GA3C [25], a hybrid GPU/CPU implementation that efficiently queues training experiences and action predictions. Our work builds on open-source GA3C implementations [25], [26]. Other choices for RL policy training algorithms (e.g., PPO [27], TD3 [28]) are architecturally similar to A3C. Thus, the challenges mentioned above (varying number of agents, assumptions about other agents’ behaviors) would map to future work that considers employing other RL algorithms or techniques [29] in this domain.

**D. RELATED WORKS USING LEARNING**

There are several concurrent and subsequent works which use learning to solve the collision avoidance problem, categorized as non-RL, RL, and agent-level RL approaches. Non-RL-based approaches to the collision avoidance problem include imitation learning, inverse RL, and supervised learning of prediction models. Imitation learning approaches [11] learn a policy that mimics what a human pedestrian or human teleoperator [30] would do in the same state but require data from an expert.

Inverse RL methods learn to estimate pedestrians’ cost functions, then use the cost function to inform robot plans [31], [7], but require real pedestrian trajectory data. Other approaches learn to predict pedestrian paths, which improves the world model used by the planner [32], but decoupling the prediction and planning steps could lead to the freezing robot problem (Section II-B1). A key advantage of RL over these methods is the ability to explore the state space through self-play, in which experiences generated in a low-fidelity simulation environment can reduce the need for expensive, real-world data collection efforts.

Within RL-based approaches, a key difference arises in the state representation: sensor-level and agent-level. Sensorlevel approaches learn to select actions directly from raw sensor readings (either 2D laser scans [10] or images [11]) with end-to-end training. This leads to a large state space (Rw×h×c for a camera with resolution w × h and c channels, e.g., 480 × 360 × 3 = 5184000), which makes training challenging. CNNs are often used to extract low-dimensional features from this giant state space, but training such a feature extractor in simulation requires an accurate sensor simulation model.

The sensor-level approach has the advantage that both static and dynamic obstacles (including walls) can be fed into the network with a single framework. In contrast, this work uses interpretable clustering, tracking, and multi-sensor fusion algorithms to extract an agent-level state representation from raw sensor readings. Advantages include a much smaller state space (R9+5(n−1)) enabling faster learning convergence; a sensor-agnostic collision avoidance policy, enabling sensor upgrades without re-training; and increased introspection into decision making, so that decisions can be traced back to the sensing, clustering, tracking, or planning modules.

Within agent-level RL, a key challenge is that of representing a variable number of nearby agents in the environment at any timestep. Typical feedforward networks used to represent the complex decision making policy for collision avoidance require a pre-determined input size.

The sensor-level methods do maintain a fixed size input (sensor resolution), but have the limitations mentioned above. Instead, our first work trained a 2-agent value network, and proposed a mini-max rule to scale up to n agents [15]. To account for multiagent interactions (instead of only pairwise), our next work defines a maximum number of agents that the network can handle, and pads the observation space if there are actually fewer agents in the environment [16]. However, this maximum number of agents is limited by the increased number of network parameters (and therefore training time) as more agents’ states are added.

This work uses a recurrent network to convert a sequence of agent states at a particular timestep into a fixedsize representation of the world state; that representation is fed into the input of a standard feedforward network. Beyond the scope of collision avoidance, recent work [33] introduced attention mechanisms, another tool popularized in NLP, as another method for embedding the variable number of other agents’ states. There are also differences in the reward functions used in RL-based collision avoidance approaches.

Generally, the non-zero feedback provided at each timestep by a dense reward function (e.g., [10]) makes learning easier, but reward shaping quickly becomes a difficult problem in itself. For example, balancing multiple objectives (proximity to goal, proximity to others) can introduce unexpected and undesired local minima in the reward function. On the other hand, sparse rewards are easy to specify but require a careful initialization/exploration procedure to ensure agents will receive some environment feedback to inform learning updates.

This work mainly uses sparse reward (arrival at goal, collision) with smooth reward function decay in near-collision states to encourage a minimum separation distance between agents. Additional terms in the reward function are shown to reliably induce higher-level preferences (social norms) in our previous work [16]. While learning-based methods have many potential advantages over model-based approaches, learning-based approaches typically lack the guarantees (e.g., avoiding deadlock, zero collisions) desired for safety-critical applications.

A key challenge in establishing guarantees in multiagent collision avoidance is what to assume about the world (e.g., policies and dynamics of other agents). Unrealistic or overly conservative assumptions about the world invalidate the guarantees or unnecessarily degrade the algorithm’s performance: striking this balance may be possible in some domains but is particularly challenging in pedestrian-rich environments. A survey of the active research area of Safe RL is found in [34].

Autonomy offers surface vehicles the opportunity to improve the efficiency of transportation while still cutting down on greenhouse emissions. However, for safe and reliable autonomous surface vehicles (ASV), effective path planning is a pre-requisite which should cater to the two important tasks of path following and collision avoidance (COLAV). In the literature, a distinction is typically made between reactive and deliberate COLAV methods [1].

In short, reactive approaches, most notably artificial potential field methods [2]–[4], dynamic window methods [5]–[7], velocity obstacle methods [8], [9] and optimal control-based methods [10]–[14], base their guidance decisions on sensor readings from the local environment, whereas deliberate methods, among them popular graph-search algorithms such as A\* [15] and Voronoi graphs [16], [17] as well as randomized approaches such as rapidly-exploring random tree [18] and probabilistic roadmap [19], exploit a priori known characteristics of the global environment in order to construct an optimal path in advance, which is to be followed using a low-level steering controller.

By utilizing more data than just the current perception of the local neighbourhood surrounding the agent, deliberate methods are generally more likely to converge to the intended goal, and less likely to suggest guidance strategies leading to dead ends, which is frequently observed with reactive methods due to local minima [20].

However, in the case where the environment is not perfectly known, as a result of either incomplete or uncertain mapping data or due to the environment having dynamic features, purely deliberate methods often fall short. To prevent this, such methods are often executed repeatably on a regular basis to adapt to discrepancies between recent sensor observations and the a priori belief state of the environment [20].

However, as this class of methods are computationally expensive by virtue of processing global environment data, this is sometimes rendered infeasible for real-world applications with limited processing power [21], especially as the problem of optimal path planning amid multiple obstacles is provably NP-hard [22]. Thus, a common approach is to utilize a reactive algorithm, which is activated whenever the presence of a nearby obstacle is detected, as a fallback option for the global, deliberate path planner.

Such hybrid architectures are intended to combine the strengths of reactive and deliberate approaches and have gained traction in recent years [23], [24]. The approach presented in this article is somewhat related to this; the existence of some a priori known nominal path is presumed, but following it strictly will invariantly lead to collisions with obstacles.

Unlike other approaches, there is, however, no switching mechanism that activates some reactive fallback algorithm in dangerous situations. To this end, a reinforcement learning (RL) agent is trained to exhibit rational behaviour under such circumstances, i.e. following the path strictly only when it is deemed safe.

Despite the vast amount of literature on the topic and the numerous different approaches, of which only a small subset has been mentioned here, it appears that, when applied to vehicles with non holonomic and real-time constraints such as autonomous surface vehicles, no existing method is without drawbacks, whether it is unrealistic assumptions about the vessel dynamics (if not an outright neglect thereof), problems with scalability in terms of environment complexity (including the degrees of freedom, the number of obstacles as well as their shapes and their velocities), excessive computation time requirements in general, unrealistic assumptions of availability of measurements, the disregard for desirable output path properties such as continuity, smoothness, feasibility or even safety, an incompatibility with external environmental forces, a lack of determinism (which may or may not be deemed problematic), stability issues due to singularities or local minima leading to sub-optimal guidance strategies [25], [26].

RL is an area of machine learning (ML) of particular interest for control applications, such as the guidance of surface vessels under consideration here. Fundamentally, this ML paradigm is concerned with estimating the optimal behaviour for an agent in an unknown, and potentially partly unobservable environment, relying on trial-and-error-like approaches in order to iteratively approximate the behaviour policy that maximizes the agent’s expected long-time reward in the environment.

The field of RL has seen rapid development over the last few years, leading to many impressive achievements, such as playing chess and various other games at a level that is not only superhuman, but also overshadows previous AI approaches by a wide margin [27]–[29]. The focus of this paper is to explore how RL, given the recent advances in the field, can be applied to the guidance and control of ASV.

Specifically, we look at the dual objectives of achieving the ability to follow a path constructed from a priori known way-points, while avoiding collision with obstacles along the way. In an end-to-end fashion, control signals for a simulated vessel are generated by a RL agent which, based on the readings from a rangefinder sensor suite which is attached to the vessel as well as rewards received from the environment, learns how to intelligently control the vessel in challenging obstacle avoidance scenarios.

The resulting interplay between the environment, which incorporates the dynamics of the vessel itself, and the autonomous RL agent is illustrated in Figure 1. For simplicity, we limit the scope of this work to non-moving obstacles of circular shapes. As RL methods are, model-free approaches, by their very nature, a positive result can bring significant value to the robotics and autonomous system field, where implementing guidance system typically requires knowledge of the vessel dynamics, in the form of non-linear first-principle models with parameters that can only be determined experimentally at great cost.

**CHAPTER 2**

**LITERATURE SURVEY**

**[1] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, “A navigation system for robots operating in crowded urban environments,” in 2013 IEEE International Conference on Robotics and Automation. IEEE, 2013, pp. 3225–3232.**

Over the past years, there has been a tremendous progress in the area of robot navigation. Most of the systems developed thus far, however, are restricted to indoor scenarios, non-urban outdoor environments, or road usage with cars. Urban areas introduce numerous challenges to autonomous mobile robots as they are highly complex and in addition to that dynamic. In this paper, we present a navigation system for pedestrian-like autonomous navigation with mobile robots in city environments.

We describe different components including a SLAM system for dealing with huge maps of city centers, a planning approach for inferring feasible paths taking also into account the traversability and type of terrain, and a method for accurate localization in dynamic environments. The navigation system has been implemented and tested in several large-scale field tests in which the robot Obelix managed to autonomously navigate from our university campus over a 3.3 km long route to the city center of Freiburg.

In this paper, we presented a navigation system that enables a mobile robot to autonomously navigate through city centers. To accomplish its task, this navigation system uses an extended SLAM routine that deals with the outliers generated by the partially GPS-denied environments, a localization routine that utilizes a special data structure for large-scale maps, dedicated terrain analysis methods for also dealing with negative obstacles, and a trajectory planning system that considers dynamic objects.

The system has been implemented and demonstrated in a large-scale field test, during which the robot Obelix autonomously navigated over a path of more than three kilometers through the crowded city center of Freiburg thereby negotiating with several potential hazards.

**[2] P. Trautman and A. Krause, “Unfreezing the robot: Navigation in dense, interacting crowds,” in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2010, pp. 797–803.**

In this paper, we study the safe navigation of a mobile robot through crowds of dynamic agents with uncertain trajectories. Existing algorithms suffer from the “freezing robot” problem: once the environment surpasses a certain level of complexity, the planner decides that all forward paths are unsafe, and the robot freezes in place (or performs unnecessary maneuvers) to avoid collisions.

Since a feasible path typically exists, this behavior is suboptimal. Existing approaches have focused on reducing the predictive uncertainty for individual agents by employing more informed models or heuristically limiting the predictive covariance to prevent this overcautious behavior. In this work, we demonstrate that both the individual prediction and the predictive uncertainty have little to do with the frozen robot problem. Our key insight is that dynamic agents solve the frozen robot problem by engaging in “joint collision avoidance”:

They cooperatively make room to create feasible trajectories. We develop IGP, a nonparametric statistical model based on dependent output Gaussian processes that can estimate crowd interaction from data. Our model naturally captures the non-Markov nature of agent trajectories, as well as their goal-driven navigation. We then show how planning in this model can be efficiently implemented using particle based inference.

Lastly, we evaluate our model on a dataset of pedestrians entering and leaving a building, first comparing the model with actual pedestrians, and find that the algorithm either outperforms human pedestrians or performs very similarly to the pedestrians. We also present an experiment where a covariance reduction method results in highly overcautious behavior, while our model performs desirably.

In this paper, we studied the Freezing Robot Problem (FRP), a phenomenon where planning algorithms exhibit overcautious or evasive behavior due to anticipated collisions with stochastically moving agents. While most existing techniques for dealing with the FRP focus on more informed (or less uncertain) models, we show that the FRP can occur even with perfect prediction, and that the key to safely navigating through dense crowds is to capture the cooperative collision avoidance inherent in real world behavior. We develop IGP, a nonparametric statistical model based on dependent output Gaussian processes, coupled through a nonlinear interaction potential.

We show how navigation in this model is naturally cast as an inference task, which can be approximately solved using importance sampling. Lastly, we demonstrated the efficacy of this algorithm on real world pedestrian data. Our results show that IGP leads to paths which are both safer and shorter than those taken by actual pedestrians and existing state of the art path planning algorithms.

**[3] J. Snape, J. Van den Berg, S. J. Guy, and D. Manocha, “The hybrid reciprocal velocity obstacle,” IEEE Transactions on Robotics, vol. 27, no. 4, pp. 696–706, Aug. 2011.**

We present the hybrid reciprocal velocity obstacle for collision-free and oscillation-free navigation of multiple mobile robots or virtual agents. Each robot senses its surroundings and acts independently without central coordination or communication with other robots. Our approach uses both the current position and the velocity of other robots to compute their future trajectories in order to avoid collisions.

Moreover, our approach is reciprocal and avoids oscillations by explicitly taking into account that the other robots sense their surroundings as well and change their trajectories accordingly. We apply hybrid reciprocal velocity obstacles to iRobot Create mobile robots and demonstrate direct, collision-free, and oscillation-free navigation.

In this paper, we have introduced the hybrid reciprocal velocity obstacle for navigation of multiple mobile robots or virtual agents sharing an environment. We take into account obstacles in the environment, uncertainty in radius, position, and velocity. We also consider the dynamics and kinematics of the robots, thereby allowing us to implement our approach on iRobot Create mobile robots.

Our formulation explicitly considers reciprocity, such that each robot can assume that other robots are cooperating to avoid collisions, but each of the robots acts completely independently without central coordination and does not communicate with other robots.

We show direct, collision-free, and oscillation-free navigation. In the future, we would like to develop a more sophisticated and less conservative model of uncertainty, that takes into account more than uncertainty in position and velocity originating from the sensors of the robot, and apply it to the hybrid reciprocal velocity obstacle formulation. In our current implementation, each of the robots currently receives their sensor readings from an overhead video camera.

As a next step, we would like to equip each robot with purely localized sensing and computing, as in [30], which uses odometry, orientation sensors, and relative positions to estimate global positions. Our approach can be applied without adaptation if data are gathered locally, and the hybrid reciprocal velocity obstacles are defined just as well, using only the relative positions and velocities of the robots.

**[4] G. Ferrer, A. Garrell, and A. Sanfeliu, “Social-aware robot navigation in urban environments,” in 2013 European Conference on Mobile Robots (ECMR), Sep. 2013, pp. 331–336.**

In this paper we present a novel robot navigation approach based on the so-called Social Force Model (SFM). First, we construct a graph map with a set of destinations that completely describe the navigation environment. Second, we propose a robot navigation algorithm, called social-aware navigation, which is mainly driven by the social-forces centered at the robot.

Third, we use a MCMC Metropolis-Hastings algorithm in order to learn the parameters values of the method. Finally, the validation of the model is accomplished throughout an extensive set of simulations and real-life experiments.

We have presented a novel robot navigation approach based on the so called Social-Forces Model. The validation of the model has been demonstrated throughout an extensive set of simulations and real-life experiments in a urban area. In contrast to other existing approaches, our method can handle realistic situations, such as dealing with large environments with obstacles and highly crowded scenes.

For that reason, this work can be applied to certain specific real robot applications, for instance, guiding tourists or accompanying professional visitors. In future work, we aim to obtain more sophisticated robot behavior, for instance, solving the oscillatory problems observed and in general making the robot-aware navigation much more robust.

**[5] J. Van den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal nbody collision avoidance,” in Robotics Research, ser. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2011, no. 70, pp. 3–19.**

In this paper, we present a formal approach to reciprocal n-body collision avoidance, where multiple mobile robots need to avoid collisions with each other while moving in a common workspace. In our formulation, each robot acts fully independently, and does not communicate with other robots. Based on the definition of velocity obstacles [5], we derive sufficient conditions for collision-free motion by reducing the problem to solving a low-dimensional linear program.

We test our approach on several dense and complex simulation scenarios involving thousands of robots and compute collision-free actions for all of them in only a few milliseconds. To the best of our knowledge, this method is the first that can guarantee local collision-free motion for a large number of robots in a cluttered workspace.

In this paper, we have presented an efﬁcient method that provides a sufﬁcient con-dition for multiple robots to select an action that avoids collisions with other robots,though each acts independently without communication with others. Our approachto reciprocal n-body collision avoidance exhibits fast running times and smooth,convincing behavior in our experiments.

We have used a simple robot model, in which kinematics and dynamics are ig-nored. An important extension for future work is to take such constraints into ac-count. We can either do this as a post-processing step, in which the computed newvelocity is ‘clamped’ to what the kinematic and dynamic constraints allow.

Thiswould not strictly guarantee avoiding collisions anymore, but it may work well inpractice [24]. A more thorough solution would be to take these factors intrinsicallyinto account in the derivation of the permitted velocities for the robots. [26] and [19]provide some interesting ideas in this direction.

**[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, “Optimal reciprocal collision avoidance for multiple non-holonomic robots,” in Distributed Autonomous Robotic Systems. Springer, 2013, pp. 203–216.**

In this paper a method for distributed reciprocal collision avoidance among multiple non-holonomic robots with bike kinematics is presented. The proposed algorithm, bicycle reciprocal collision avoidance (B-ORCA), builds on the concept of optimal reciprocal collision avoidance (ORCA) for holonomic robots but furthermore guarantees collision-free motions under the kinematic constraints of car-like vehicles.

The underlying principle of the B-ORCA algorithm applies more generally to other kinematic models, as it combines velocity obstacles with generic tracking control. The theoretical results on collision avoidance are validated by several simulation experiments between multiple car-like robots.

In this paper, a novel collision avoidance strategy for a group of car-like robots is presented. Various application areas throughout research and industry have seen an evergrowing interest in mobile robots. Industrial and service robots are mostly non-holonomic, and often designed as car-like vehicles.

A particular example of car-like vehicles deployed in an industrial setting are the MagneBikes [1], compact robots with bicycle kinematics designed for the collaborative inspection in power plants. This and all other applications, where multiple car-like robots interact in their workspaces, require reciprocal collision avoidance methods.

In this work, a distributed method for reciprocal local collision avoidance among bicycle or car-like robots, socalled B-ORCA, is presented, where each individual robot does not need information about the kinematics of other robots. The method guarantees collision-free motions and achieves smooth trajectories as shown in simulated experiments with ten MagneBike and ten car robots.

The method relies on the ORCA algorithm that computes a collisionfree velocity as if the robots were holonomic. The method further relies on a trajectory tracking controller for car-like vehicles, which could essentially be substituted by any other tracking controller for kinematic constraints different than those presented in this paper.

**[7] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, “Socially compliant mobile robot navigation via inverse reinforcement learning,” The International Journal of Robotics Research, Jan. 2016.**

Mobile robots are increasingly populating our human environments. To interact with humans in a socially compliant way, these robots need to understand and comply with mutually accepted rules. In this paper, we present a novel approach to model the cooperative navigation behavior of humans. We model their behavior in terms of a mixture distribution that captures both the discrete navigation decisions, such as going left or going right, as well as the natural variance of human trajectories.

Our approach learns the model parameters of this distribution that match, in expectation, the observed behavior in terms of user-defined features. To compute the feature expectations over the resulting high-dimensional continuous distributions, we use Hamiltonian Markov chain Monte Carlo sampling.

Furthermore, we rely on a Voronoi graph of the environment to efficiently explore the space of trajectories from the robot’s current position to its target position. Using the proposed model, our method is able to imitate the behavior of pedestrians or, alternatively, to replicate a specific behavior that was taught by tele-operation in the target environment of the robot.

We implemented our approach on a real mobile robot and demonstrated that it is able to successfully navigate in an office environment in the presence of humans. An extensive set of experiments suggests that our technique outperforms state-of-the-art methods to model the behavior of pedestrians, which also makes it applicable to fields such as behavioral science or computer graphics.

We presented a novel approach that allows a mobile robot to learn a model of the navigation behavior of cooperatively navigating agents such as pedestrians. Based on observations of their continuous trajectories, our method infers a model of the underlying decision-making process. To cope with the discrete and continuous aspects of this process, our model uses a joint mixture distribution that captures the discrete decisions regarding the homotopy classes of the composite trajectories as well as continuous properties of the trajectories such as higher-order dynamics.

To compute the feature expectations with respect to the continuous, high-dimensional probability distributions, our method uses Hamiltonian MCMC sampling. To efficiently explore the space of trajectories, we use a Voronoi graph of the environment. The learned model enables socially compliant mobile robot navigation since it allows the robot to predict the navigation behavior of pedestrians in terms of a probability distribution over their trajectories.

**[8] P. Trautman, J. Ma, R. M. Murray, and A. Krause, “Robot navigation in dense human crowds: the case for cooperation,” in Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA), May 2013, pp. 2153–2160.**

We consider mobile robot navigation in dense human crowds. In particular, we explore two questions. Can we design a navigation algorithm that encourages humans to cooperate with a robot? Would such cooperation improve navigation performance? We address the first question by developing a probabilistic predictive model of cooperative collision avoidance and goal-oriented behavior by extending the interacting Gaussian processes approach to include multiple goals and stochastic movement duration.

We answer the second question with an extensive quantitative study of robot navigation in dense human crowds (488 runs completed), specifically testing how cooperation models effect navigation performance. We find that the “multiple goal” interacting Gaussian processes algorithm performs comparably with human teleoperators in crowd densities near 1 person/m 2 , while a state of the art noncooperative planner exhibits unsafe behavior more than 3 times as often as this multiple goal extension, and more than twice as often as the basic interacting Gaussian processes.

Furthermore, a reactive planner based on the widely used “dynamic window” approach fails for crowd densities above 0.55 people/m 2 . Based on these experimental results, and previous theoretical observations, we conclude that a cooperation model is important for safe and efficient robot navigation in dense human crowds.

We posed two questions: how should human-robot cooperation be modeled? And would such cooperation improve navigation in dense crowds? We answered the first question by introducing mgIGP, and treating that density as a prediction of how the robot should act in order to be cooperative. We answered the second question empirically: the mgIGP algorithm was shown to perform comparably with human teleoperators in crowd densities nearing 1 person/m2 , while a state of the art noncooperative planner exhibited unsafe behavior more than 3 times as often as our planner and twice as often as the basic IGP planner.

Also, a state of the art reactive planner was insufficient for crowd densities above 0.55 people/m2 . These experimental results, along with previous theoretical results in [10], provide strong evidence that a human-robot cooperation model is important for safe and efficient dense crowd navigation.

**[9] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, “Featurebased prediction of trajectories for socially compliant navigation,” in Robotics:Science and Systems, 2012.**

Mobile robots that operate in a shared environment with humans need the ability to predict the movements of people to better plan their navigation actions. In this paper, we present a novel approach to predict the movements of pedestrians. Our method reasons about entire trajectories that arise from interactions between people in navigation tasks. It applies a maximum entropy learning method based on features that capture relevant aspects of the trajectories to determine the probability distribution that underlies human navigation behavior.

Hence, our approach can be used by mobile robots to predict forthcoming interactions with pedestrians and thus react in a socially compliant way. In extensive experiments, we evaluate the capability and accuracy of our approach and demonstrate that our algorithm outperforms the popular social forces method, a state-of-the-art approach. Furthermore, we show how our algorithm can be used for autonomous robot navigation using a real robot.

As the scope of application of robots is expanding from industrial labor to domestic services, mobile robots are more and more expected to share their environment with people. Asa result, new challenges for their navigation systems arise. The ability to perceive the intention and to predict the behavior of people is of pivotal importance for efﬁcient and socially compliant navigation in human environments.

In this paper, we presented a novel approach to predicting socially compliant trajectories in populated environments. Our approach is based on features that capture relevant aspects of the trajectories. It applies the principle of maximum entropy to elicit the probability distribution that governs the navigation behavior of people from real-world observations.

We have implemented and tested our method with extensive data sets and real robots. The experimental results demonstrate the efﬁcacy of our method and suggest that it leads to natural, socially compliant trajectories. In particular, our algorithm seems to better capture characteristics of human trajectories than the social forces method, a state-of-the-art approach.

**[10] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 6252–6259.**

Developing a safe and efficient collision avoidance policy for multiple robots is challenging in the decentralized scenarios where each robot generates its paths without observing other robots’ states and intents. While other distributed multi-robot collision avoidance systems exist, they often require extracting agent-level features to plan a local collision-free action, which can be computationally prohibitive and not robust.

More importantly, in practice the performance of these methods are much lower than their centralized counterparts. We present a decentralized sensor-level collision avoidance policy for multi-robot systems, which directly maps raw sensor measurements to an agent’s steering commands in terms of movement velocity.

As a first step toward reducing the performance gap between decentralized and centralized methods, we present a multi-scenario multi-stage training framework to learn an optimal policy. The policy is trained over a large number of robots on rich, complex environments simultaneously using a policy gradient based reinforcement learning algorithm.

We validate the learned sensor-level collision avoidance policy in a variety of simulated scenarios with thorough performance evaluations and show that the final learned policy is able to find time efficient, collision-free paths for a large-scale robot system. We also demonstrate that the learned policy can be well generalized to new scenarios that do not appear in the entire training period, including navigating a heterogeneous group of robots and a large-scale scenario with 100 robots.

In this paper, we present a multi-scenario multi-stage training framework to optimize a fully decentralized sensorlevel collision avoidance policy with a robust policy gradient algorithm. The learned policy has demonstrated several advantages on an extensive evaluation of the state-of-theart NH-ORCA policy in terms of success rate, collision avoidance performance, and generalization capability.

Our work can serve as a first step towards reducing the navigation performance gap between the centralized and decentralized methods, though we are fully aware that the learned policy focusing on local collision avoidance cannot replace a global path planner when scheduling many robots to navigate through complex environments with dense obstacles.

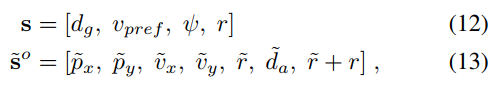
**CHAPTER 3**

**APPROACH**

**A. GA3C-CADRL**

Recall the RL training process seeks to find the optimal policy, π : st; S~o t 7! ut, which maps from an agent’s observation of the environment to a probability distribution across actions and executes the action with highest probability. We use a local coordinate frame (rotation-invariant) as in [15], [16] and separate the state of the world in two pieces: information about the agent itself, and everything else in the world. Information about the agent can be represented in a small, fixed number of variables.

The world, on the other hand, can be full of any number of other objects or even completely empty. Specifically, there is one s vector about the agent itself and one ~so vector per other agent in the vicinity:



where d g = jjpg − pjj2 is the agent’s distance to goal, and ~da = jjp − p~jj2 is the distance to the other agent. The agent’s action space is composed of a speed and change in heading angle. It is discretized into 11 actions: with a speed of vpref there are 6 headings evenly spaced between ±π=6, and for speeds of 12vpref and 0 the heading choices are [−π=6; 0; π=6]. These actions are chosen to mimic real turning constraints of robotic vehicles.

This multiagent RL problem formulation is solved with GA3C in a process we call GA3C-CADRL (GPU/CPU Asynchronous Advantage Actor-Critic for Collision Avoidance with Deep RL). Since experience generation is one of the time-intensive parts of training, this work extends GA3C to learn from multiple agents’ experiences each episode.

Training batches are filled with a mix of agents’ experiences (fsjn t ; ut; rtg tuples) to encourage policy gradients that improve the joint expected reward of all agents. Our multiagent implementation of GA3C accounts for agents reaching their goals at different times and ignores experiences of agents running other policies (e.g., non-cooperative agents).

**B. HANDLING A VARIABLE NUMBER OF AGENTS**

Recall that one key limitation of many learning-based collision avoidance methods is that the feedforward NNs typically used require a fixed-size input. Convolutional and max-pooling layers are useful for feature extraction and can modify the input size but still convert a fixed-size input into a fixed-size output. Recurrent NNs, where the output is produced from a combination of a stored cell state and an input, accept an arbitrary-length sequence to produce a fixed-size output.

Long short-term memory (LSTM) [14] is recurrent architecture with advantageous properties for training5. Although LSTMs are often applied to time sequence data (e.g., pedestrian motion prediction [35]), this paper leverages their ability to encode a sequence of information that is not time-dependent (see [36] for a thorough explanation of LSTM calculations). LSTM is parameterized by its weights, fWi; Wf; Wog, and biases, fbi; bf; bog, where fi; f; og correspond to the input, forget, and output gates.

The variable number of ~so i vectors is a sequence of inputs that encompass everything the agent knows about the rest of the world. As depicted in Fig. 2, each LSTM cell has three inputs: the state of agent j at time t, the previous hidden state, and the previous cell state, which are denoted ~so j;t, hj, Cj, respectively. Thus, at each decision step, the agent feeds each ~so i (observation of ith other agent’s state) into a LSTM cell sequentially. That is, the LSTM initially has empty states (h0; C0 set to zeros) and uses f~so 1; h0; C0g to generate fh1; C1g, then feeds

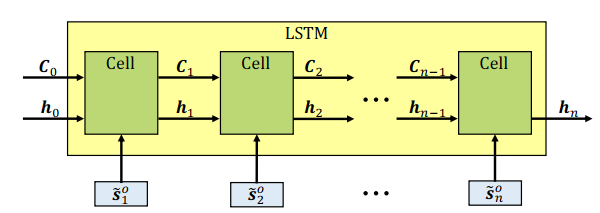


FIGURE 2: LSTM unrolled to show each input. At each decision step, the agent feeds one observable state vector, ~so i , for each nearby agent, into a LSTM cell sequentially. LSTM cells store the pertinent information in the hidden states, hi. The final hidden state, hn, encodes the entire state of the other agents in a fixed-length vector, and is then fed to the feedforward portion of the network. The order of agents is sorted by decreasing distance to the ego agent, so that the closest agent has the most recent effect on hn.

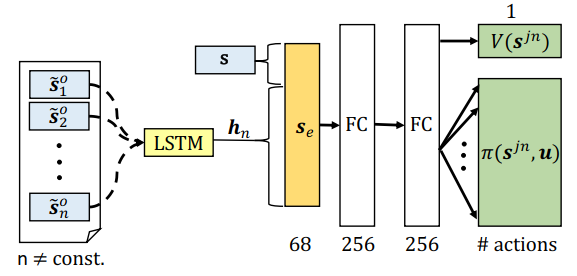


FIGURE 3: Network Architecture. Observable states of nearby agents, ~so i , are fed sequentially into the LSTM, as unrolled in Fig. 2. The final hidden state is concatenated with the agent’s own state, s, to form the vector, se. For any number of agents, se contains the agent’s knowledge of its own state and the state of the environment. The encoded state is fed into two fully-connected layers (FC). The outputs are a scalar value function (top, right) and policy represented as a discrete probability distribution over actions (bottom, right).

f~so 2; h1; C1g to produce fh2; C2g, and so on. As agents’ states are fed in, the LSTM “remembers” the pertinent information in its hidden/cell states, and “forgets” the less important parts of the input (where the notion of memory is parameterized by the trainable LSTM weights/biases).

After inputting the final agent’s state, we can interpret the LSTM’s final hidden state, hn as a fixed-length, encoded state of the world, for that decision step. The LSTM contains n cells, so the entire module receives inputs fS~o t ; ht−1; Ct−1g and produces outputs fhn; Cng, and hn is passed to the next network layer for decision making. Given a sufficiently large hidden state vector, there is enough space to encode a large number of agents’ states without the LSTM having to forget anything relevant.

In the case of a large number of agent states, to mitigate the impact of the agent forgetting the early states, the states are fed in reverse order of distance to the agent, meaning the closest agents (fed last) should have the biggest effect on the final hidden state, hn. Because the list of agents needs to be ordered in some manner, reverse distance is one possible ordering heuristic – we empirically compare to other possibilities in Section IV-D. Another interpretation of the LSTM objective is that it must learn to combine an observation of a new agent with a representation of other agents (as opposed to the architectural objective of producing a fixed-length encoding of a varying size input).

This interpretation provides intuition on how an LSTM trained in 4-agent scenarios can generalize reasonably well to cases with 10 agents. The addition of LSTM to a standard actor-critic network is visualized in Fig. 3, where the box labeled s is the agent’s own state, and the group of boxes is the n other agents’ observable states, ~so i . After passing the n other agents’ observable states into the LSTM, the agent’s own state is concatenated with hn to produce the encoded representation of the joint world state, se.

Then, se is passed to a typical feedforward DNN with 2 fully-connected layers (256 hidden units each with ReLU activation). The network produces two output types: a scalar state value (critic) and a policy composed of a probability for each action in the discrete action space (actor). During training, the policy and value are used for Equations (10) and (11); during execution, only the policy is used. During the training process, the LSTM’s weights are updated to learn how to represent the variable number of other agents in a fixed length h vector. The whole network is trained end-to-end with backpropagation.

**C. TRAINING THE POLICY**

The original CADRL and SA-CADRL (Socially Aware CADRL) algorithms used several clever tricks to enable convergence when training the networks. Specifically, forward propagation of other agent states for ∆t seconds was a critical component that required tuning, but does not represent agents’ true behaviors. Other details include separating experiences into successful/unsuccessful sets to focus the training on cases where the agent could improve.

The new GA3C-CADRL formulation is more general, and does not require such assumptions or modifications. The training algorithm is described in Algorithm 1. In this work, to train the model, the network weights are first initialized in a supervised learning phase, which converges in less than five minutes. The initial training is done on a large, publicly released set of state-action-value tuples, fsjn t ; ut; V (sjn t ; φCADRL)g, from an existing CADRL solution.

The network loss combines square-error loss on the value output and softmax cross-entropy loss between the policy output and the one-hot encoding of the closest discrete action to the one in D, described in Lines 1-6 of Algorithm 1. The initialization step is necessary to enable any possibility of later generating useful RL experiences (noninitialized agents wander randomly and probabilistically almost never obtain positive reward).

Agents running the initialized GA3C-CADRL policy reach their goals reliably when there are no interactions with other agents. However, the policy after this supervised learning process still performs poorly in collision avoidance. This observation contrasts with CADRL, in which the initialization step was sufficient to learn a policy that performs comparably to existing reaction based methods, due to relatively-low dimension value function combined with manual propagation of states.

Key reasons behind this contrast are the reduced structure in the GA3C-CADRL formulation (no forward propagation), and that the algorithm is now learning both a policy and value function (as opposed to just a value function), since the policy has an order of magnitude higher dimensionality than a scalar value function. To improve the solution with RL, parallel simulation environments produce training experiences, described in Lines 8-24 of Algorithm 1.

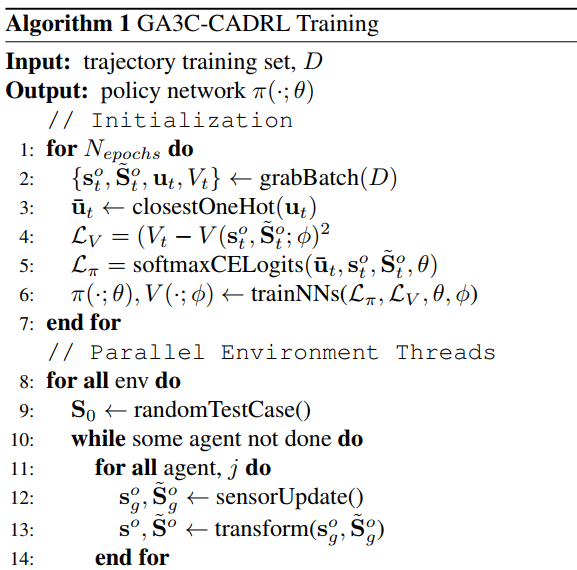
Each episode consists of 2-10 agents, with random start and goal positions, running a random assortment of policies (Non-Cooperative, Zero Velocity, or the learned GA3C-CADRL policy at that iteration) (Line 9). Agent parameters vary between r 2 [0:2; 0:8]m and vpref 2 [0:5; 2:0]m/s, chosen to be near pedestrian values. Agents sense the environment and transform measurements to their ego frame to produce the observation vector (Lines 12, 13).

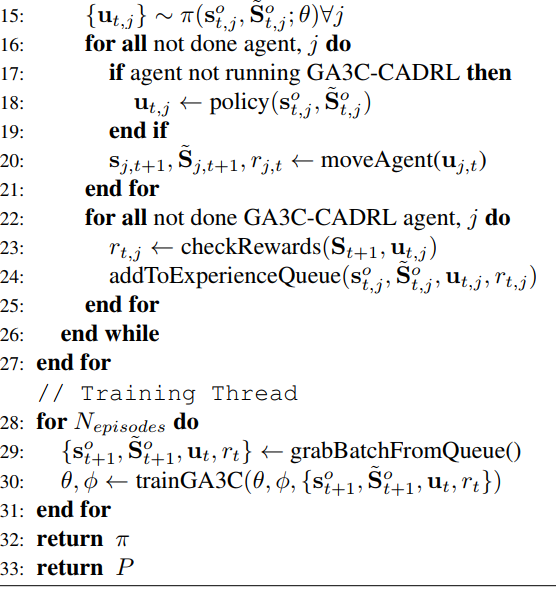
Each agent sends its observation vector to the policy queue and receives an action sampled from the current iteration of the GA3C-CADRL policy (Line 15). Agents that are not running the GA3C-CADRL policy use their own policy to overwrite ut;j (Line 18). Then, all agents that have not reached a terminal condition (collision, at goal, timed out), simultaneously move according to ut;j (Line 20). After all agents have moved, the environment evaluates R(sjn; u) for each agent, and experiences from GA3C-CADRL agents are sent to the training queue (Lines 23,24). In another thread, experiences are popped from the queue to produce training batches (Line 29).

These experience batches are used to train a single GA3C-CADRL policy (Line 30) as in [25]. An important benefit of the new framework is that the policy can be trained on scenarios involving any number of agents, whereas the maximum number of agents had to be defined ahead of time with CADRL/SA-CADRL6. This work begins the RL phase with 2-4 agents in the environment, so that the policy learns the idea of collision avoidance in reasonably simple domains. Upon convergence, a second RL phase begins with 2-10 agents in the environment.

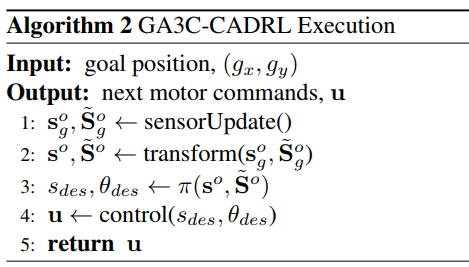
**D. POLICY INFERENCE**

Inference of the trained policy for a single timestep is described in Algorithm 2. As in training, GA3C-CADRL agents sense the environment, transfer to the ego frame, and select an action according to the policy (Lines 1-3). Like many RL algorithms, actions are sampled from the stochastic policy during training (exploration), but the action with highest probability mass is selected during inference (exploitation). A necessary addition for hardware is a low-level controller to track the desired speed and heading angle (Line 4). Note that the value function is not used during inference; it is only





learned to stabilize estimates of the policy gradients during training.



The architecture of the training and inference steps for the simulated and real robot system are shown in Fig. 4.

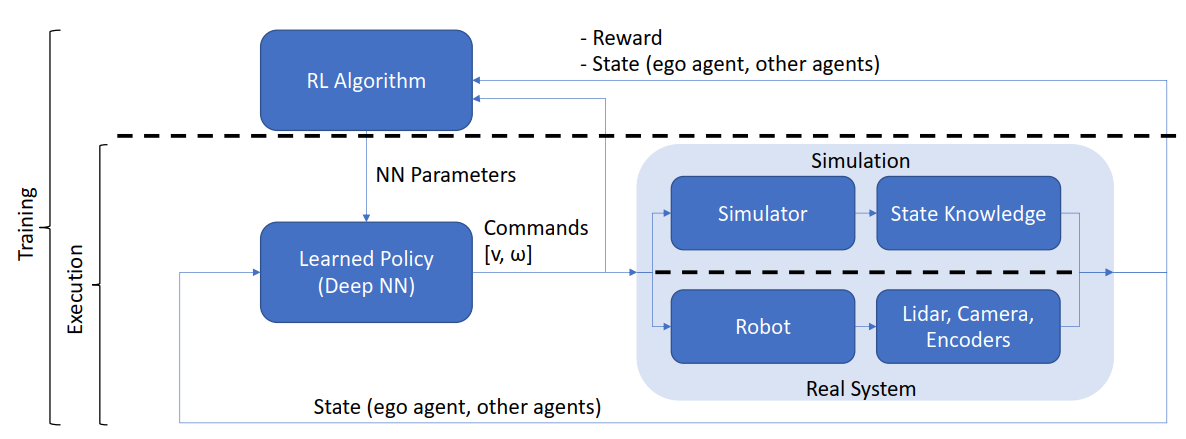


FIGURE 4: System Architecture. During training, the policy receives state measurements to compute robot commands, and the environment returns next states and rewards. Collections of (state, action, reward) tuples enable an RL algorithm to update the parameters of the learned policy. During execution, only the blocks below the upper dashed line run (NN parameters are fixed). The key difference between executing in simulation vs. the real robot is that the robot’s onboard sensors (lidar, cameras, encoders) estimate the state of the environment (e.g., agents’ positions, velocities).

**B. REINFORCEMENT LEARNING**

In this section, we will briefly review the RL paradigm and introduce the specific technique that our method builds on. For a more comprehensive coverage, the reader is advised to consult the book by Sutton and Barto [32]. Fundamentally, RL is an approach to let autonomous agents learn how to behave optimally in their environments.

Using the phrase ‘‘let learn’’ instead of ‘‘teach’’ is not accidental; a defining feature of RL is that the learning is not instructive, as opposed to the related field of supervised learning. Instead, learning is achieved through a combination of exploration and evaluative feedback, which bears a close resemblance to the way in which humans and other animals learn [32]; they become gradually wiser by virtue of trial and error.

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and punishing undesired ones. In general, a reinforcement learning agent -- the entity being trained -- is able to perceive and interpret its environment, take actions and learn through trial and error. Reinforcement learning is one of several approaches developers use to train machine learning systems. What makes this approach important is that it empowers an agent, whether it's a feature in a video game or a robot in an industrial setting, to learn to navigate the complexities of the environment it was created for. Over time, through a feedback system that typically includes rewards and punishments, the agent learns from its environment and optimizes its behaviors.

**How does reinforcement learning work?**

In reinforcement learning, developers devise a method of rewarding desired behaviors and punishing negative behaviors. This method assigns positive values to the desired actions to encourage the agent to use them, while negative values are assigned to undesired behaviors to discourage them. This programs the agent to seek long-term and maximum overall rewards to achieve an optimal solution. These long-term goals help prevent the agent from getting stuck on less important goals. With time, the agent learns to avoid the negative and seek the positive. This learning method has been adopted in artificial intelligence (AI) as a way of directing unsupervised machine learning through rewards or positive reinforcement and penalties or negative reinforcement.

The Markov decision process serves as the basis for reinforcement learning systems. In this process, an agent exists in a specific state inside an environment; it must select the best possible action from multiple potential actions it can perform in its current state. Certain actions offer rewards for motivation. When in its next state, new rewarding actions are available to it. Over time, the cumulative reward is the sum of rewards the agent receives from the actions it chooses to perform.

**Applications and examples of reinforcement learning**

While reinforcement learning has been a topic of much interest in the field of AI, its widespread, real-world adoption and application remain limited. Noting this, however, research papers abound on theoretical applications, and there have been some successful use cases. Current uses include but are not limited to the following:

Gaming.

Resource management.

Personalized recommendations.

Robotics.

A learning algorithm playing Pac-Man might have the ability to move in one of four possible directions, barring obstruction. From pixel data, an agent might be given a numeric reward for the result of a unit of travel: 0 for empty spaces, 1 for pellets, 2 for fruit, 3 for power pellets, 4 for ghost post-power pellets, 5 for collecting all pellets to complete a level, and a 5-point deduction for collision with a ghost.

The agent starts from randomized play and moves to more sophisticated play, learning the goal of getting all pellets to complete the level. Given time, an agent might even learn tactics such as conserving power pellets until needed for self-defense. Reinforcement learning can operate in a situation as long as a clear reward can be applied. In enterprise resource management, reinforcement algorithms allocate limited resources to different tasks as long as there's an overall goal it is trying to achieve. A goal in this circumstance would be to save time or conserve resources.

**CHAPTER 4**

**RESULT**

Collision Avoidance in Pedestrian-Rich Environments with Deep Reinforcement Learning

Now we are entering into an ERA of automation where everything is going to be automated of human daily life activities such as Automatic self-driving cars, home cleaning robots, drone based parcel delivery system and many more. All this sensors based automation require expert knowledge to move freely without collision with any other object. In Robotic automation all robots has to move freely without collision and to avoid collision many existing techniques such as Trajectory based algorithm, cooperative or non-cooperative collision avoidance algorithms and many more.

Trajectory based algorithm will continuously inspect trajectory data to avoid collision but inspecting trajectory require heavy computation. Cooperative or Non-Cooperative algorithms will predict motion of other object and based on others motions algorithm will predict its motion and this algorithm will fall to freezing problem where sometime it fail to predict feasible path.

To overcome from above issues author of this paper employing Deep Reinforcement Learning based LSTM (long short term memory) called Collision avoidance with deep reinforcement learning (CADRL) algorithm, which tackles the aforementioned trade-off between computation time and smooth motion by using reinforcement learning (RL) to offload the expensive online computation to an offline learning procedure. Specifically, a computationally efficient (i.e., real-time implementable) interaction rule is developed by learning a policy that implicitly encodes cooperative behaviours.

Reinforcement learning (RL) helps in taking each robot as Agent and then encode all his movements as vector. RL consists of Agent, State, Environment and action.

Agent refers to an object which is moving such as Robot

Environment is the platform where RL get executed

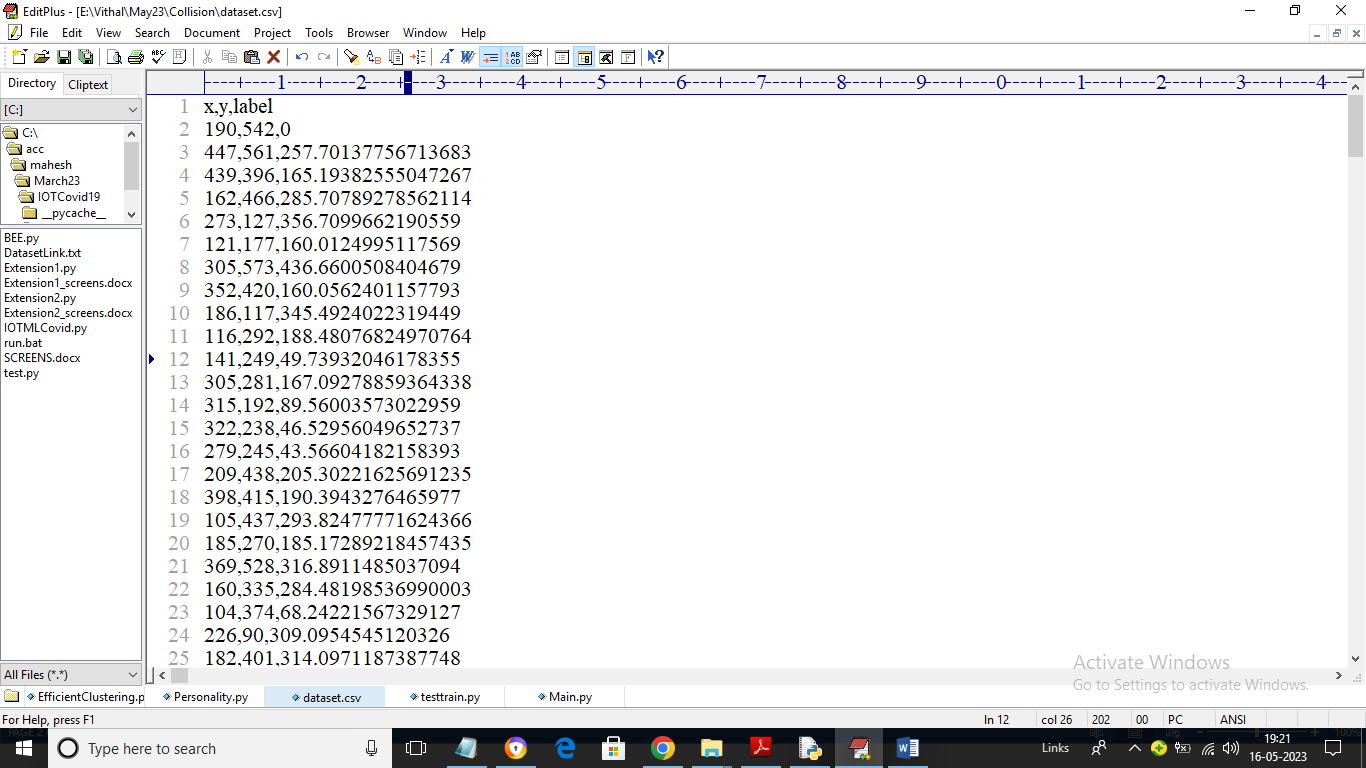
Action refers to speed and velocity of the robot

State refers to prediction which is based on action and if LSTM predict action as Non-Collision then state will change to non-collision and RL will get rewarded and if predicted value is collision then state will change to collide and RL will get penalty.

LSTM get trained on robot movement data and based on TEST data it will predict state of the robot.

Author has implemented propose CADRL algorithm on GA3C environment but that environment is not getting installed so we design our own simulation where RL will be applied to calculate state based on LSTM prediction. All input to LSTM will be from sensors but we don’t have any sensor we are using randomly generated data as Robot X and Y location.

To train LSTM we are using below Robot movement data which we generate randomly



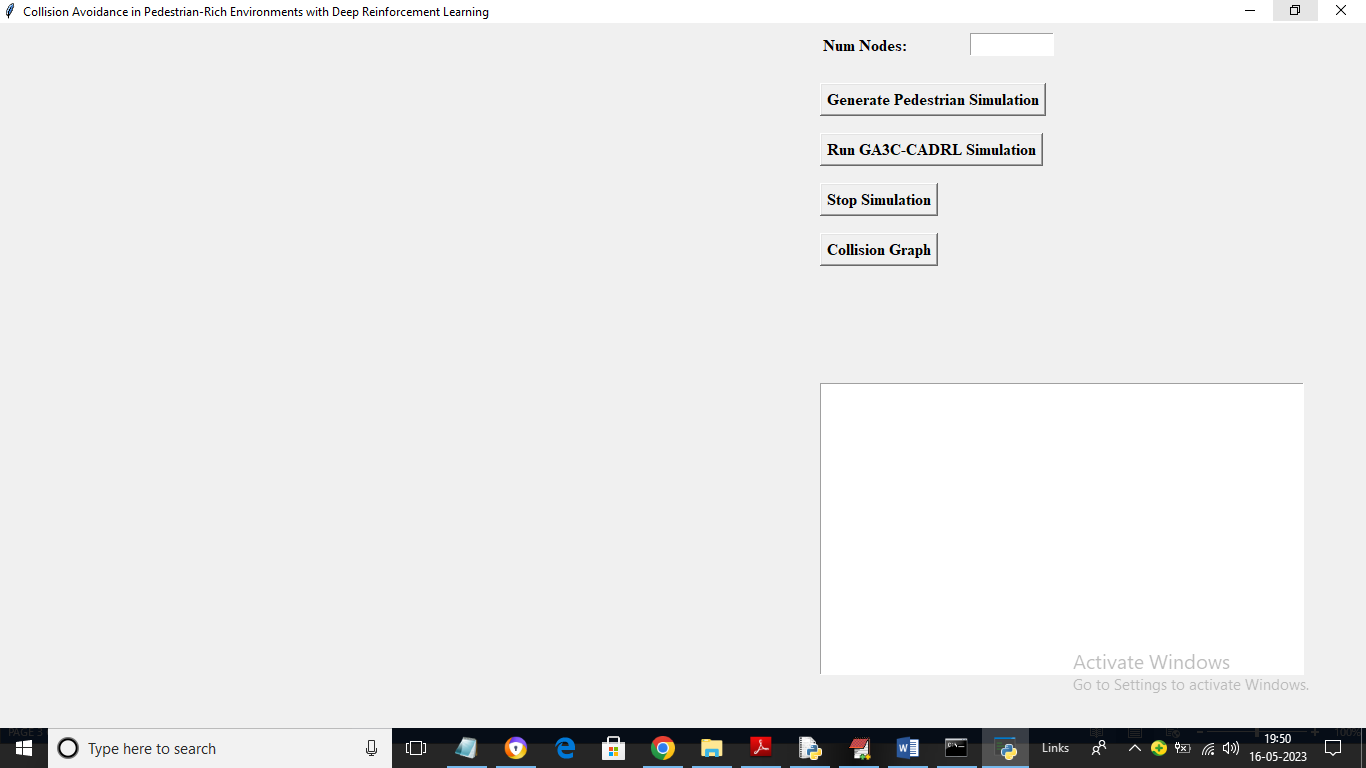
In above dataset we have X and Y values and label is consider as the distance so by using above dataset will train LSTM algorithm

To implement this project we have designed following modules

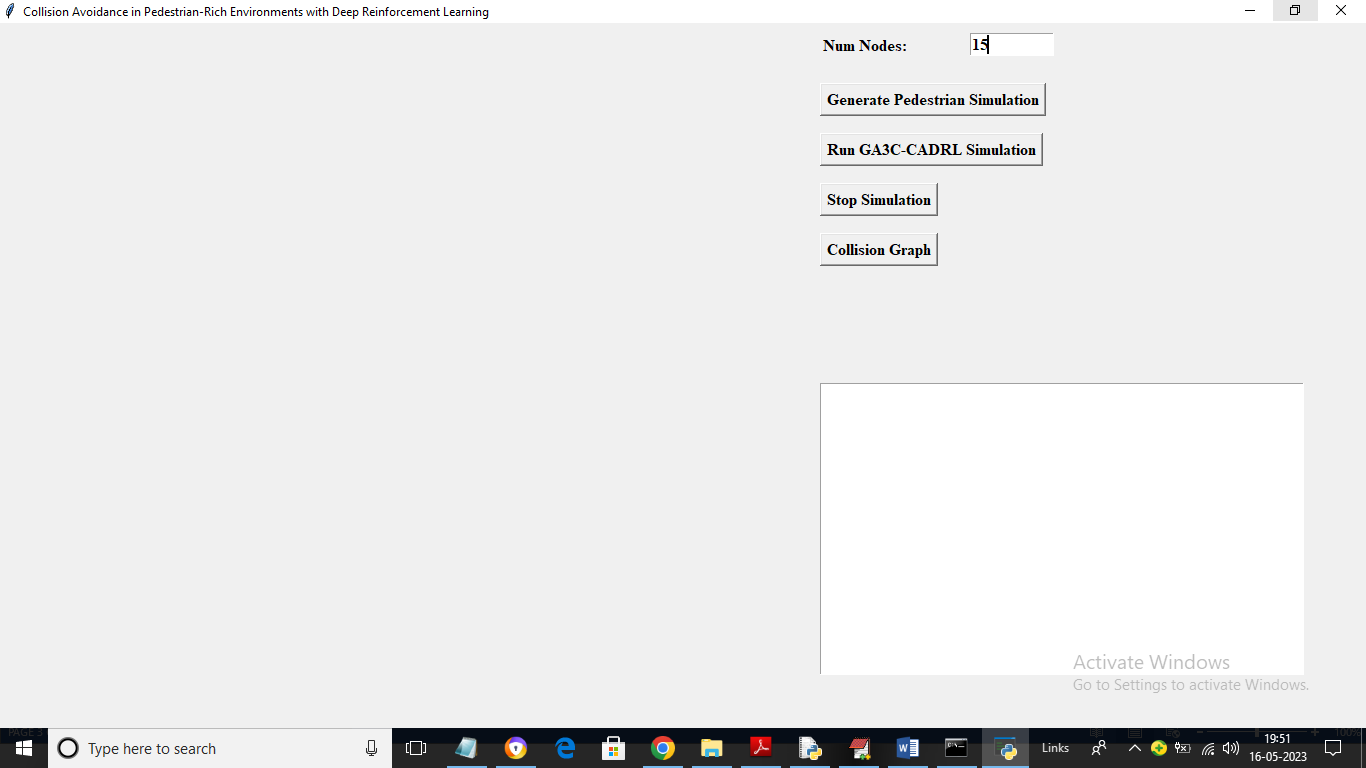
1. Generate Pedestrian Simulation: using this module we will create pedestrian simulation as CIRCLES
2. Run GA3C-CADRL Simulation: using this module we will apply CADRL algorithm on pedestrian simulation to predict states such as collision and avoiding collision. In this simulation all circles will move at random location and if two circles about to collide then they will avoid collision and choose random location. For each collision algorithm will get penalty and for avoiding collision and for free movement algorithm will get awarded
3. Stop Simulation: using this module we will stop simulation
4. Collision Graph: using this module we will plot collision penalty and award earned graph

SCREEN SHOTS

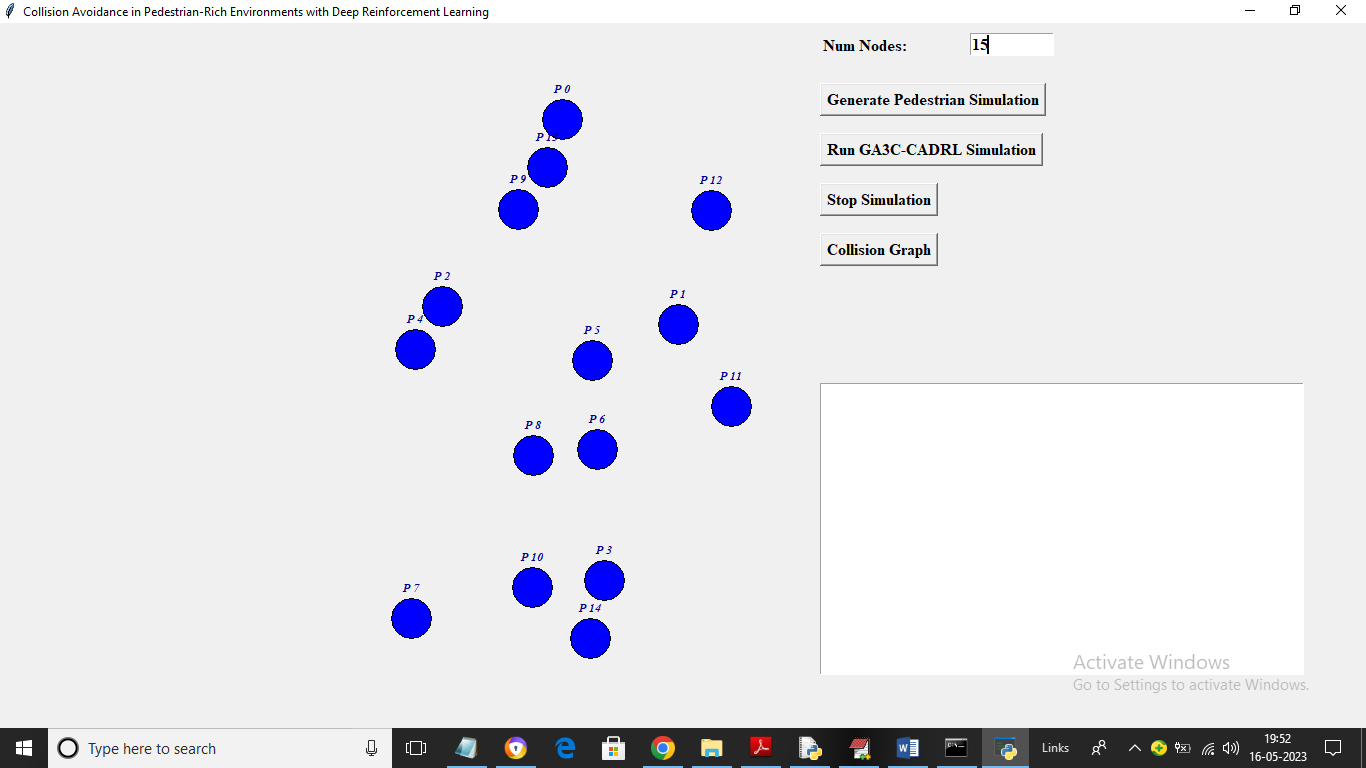
To run project double click on ‘run.bat’ file to get below screen



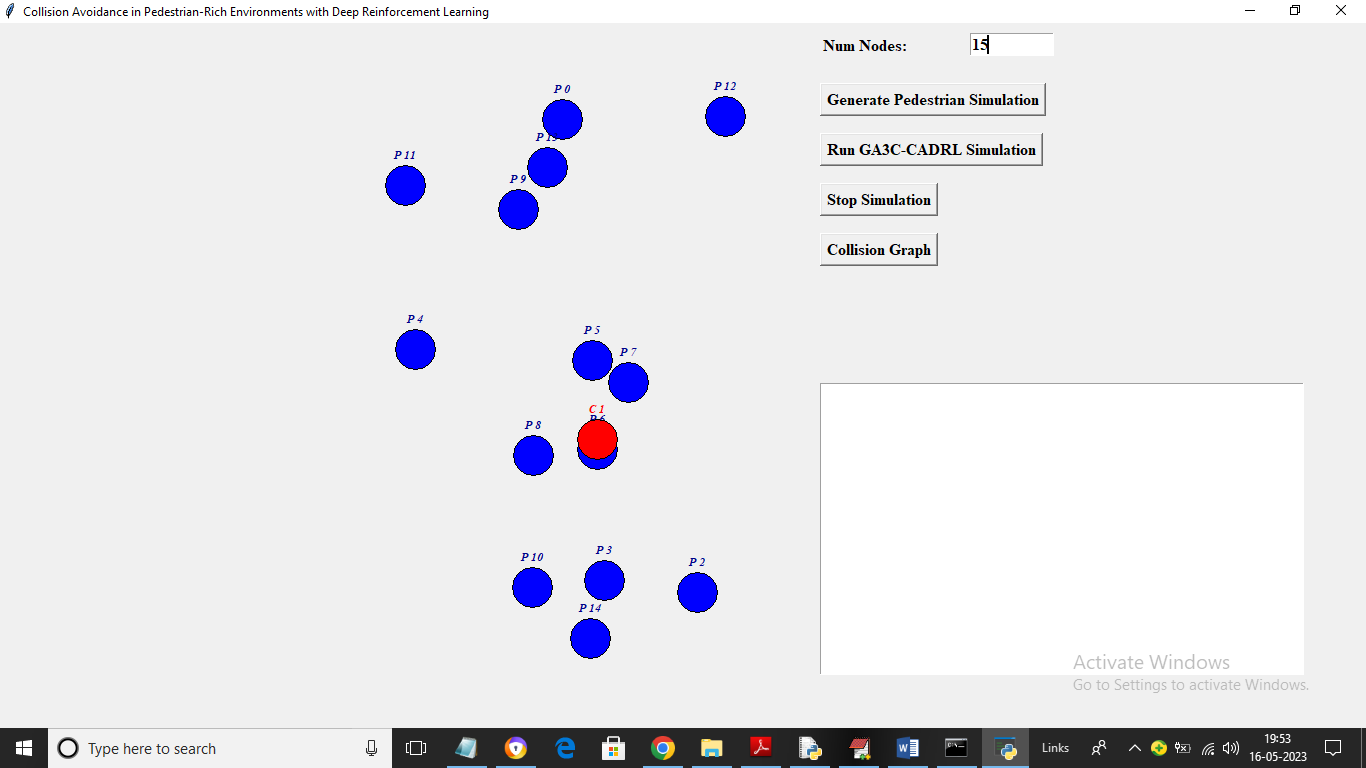
In above screen enter number of nodes as simulation and then click on ‘Generate Pedestrian Simulation’ button to generate pedestrian simulation and then will get below output



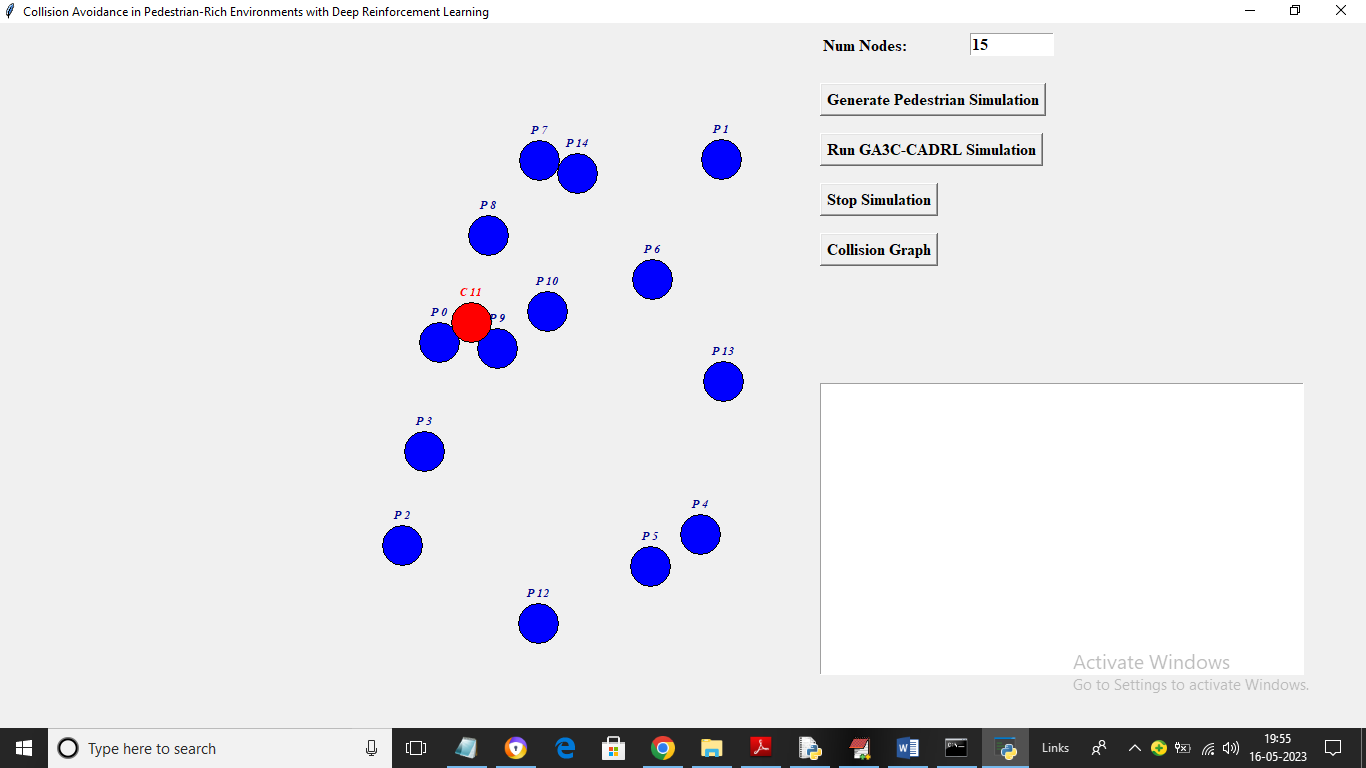
In above screen I entered number of nodes as 15 and then press ‘Generate Pedestrian Simulation’ button to get below output

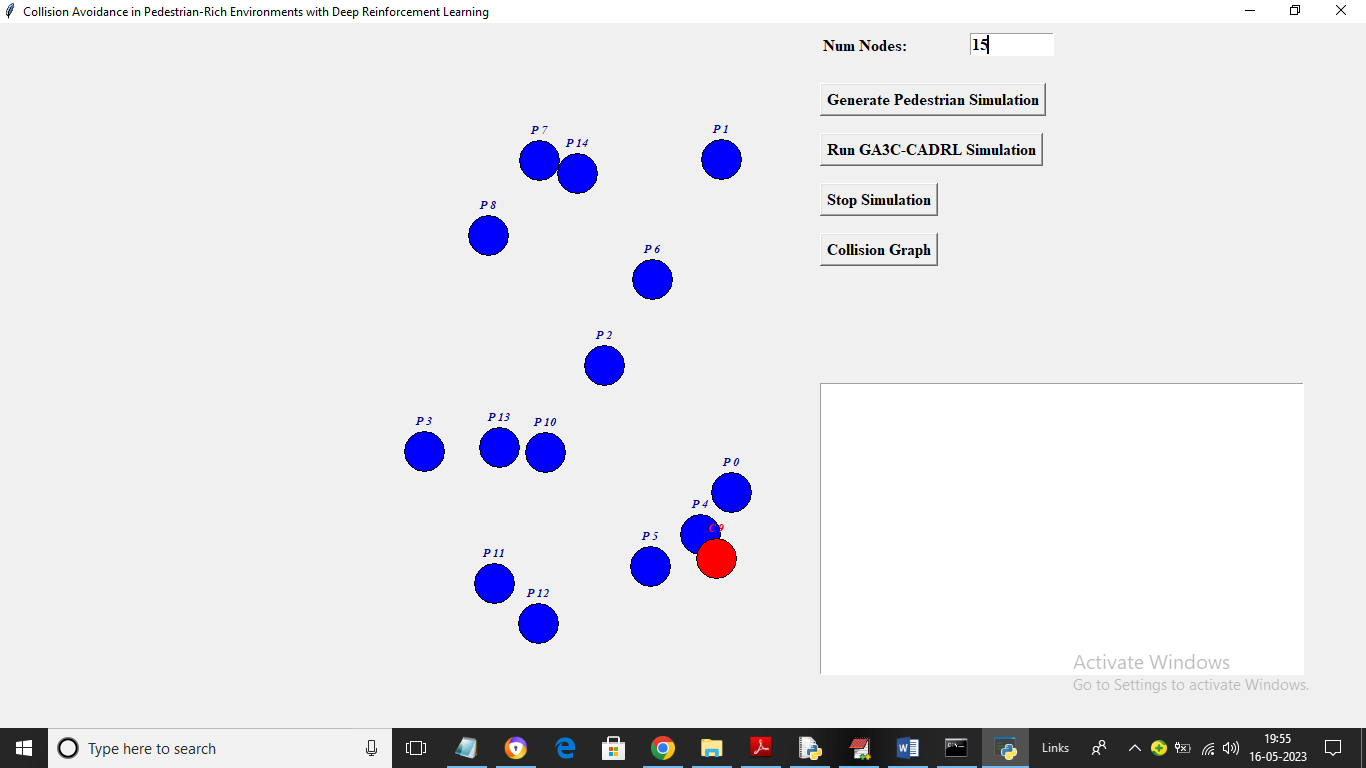


In above screen each blue circle is represented as one pedestrian and now click on ‘Run GA3C-CADRL Simulation’ button to start simulation and get below output

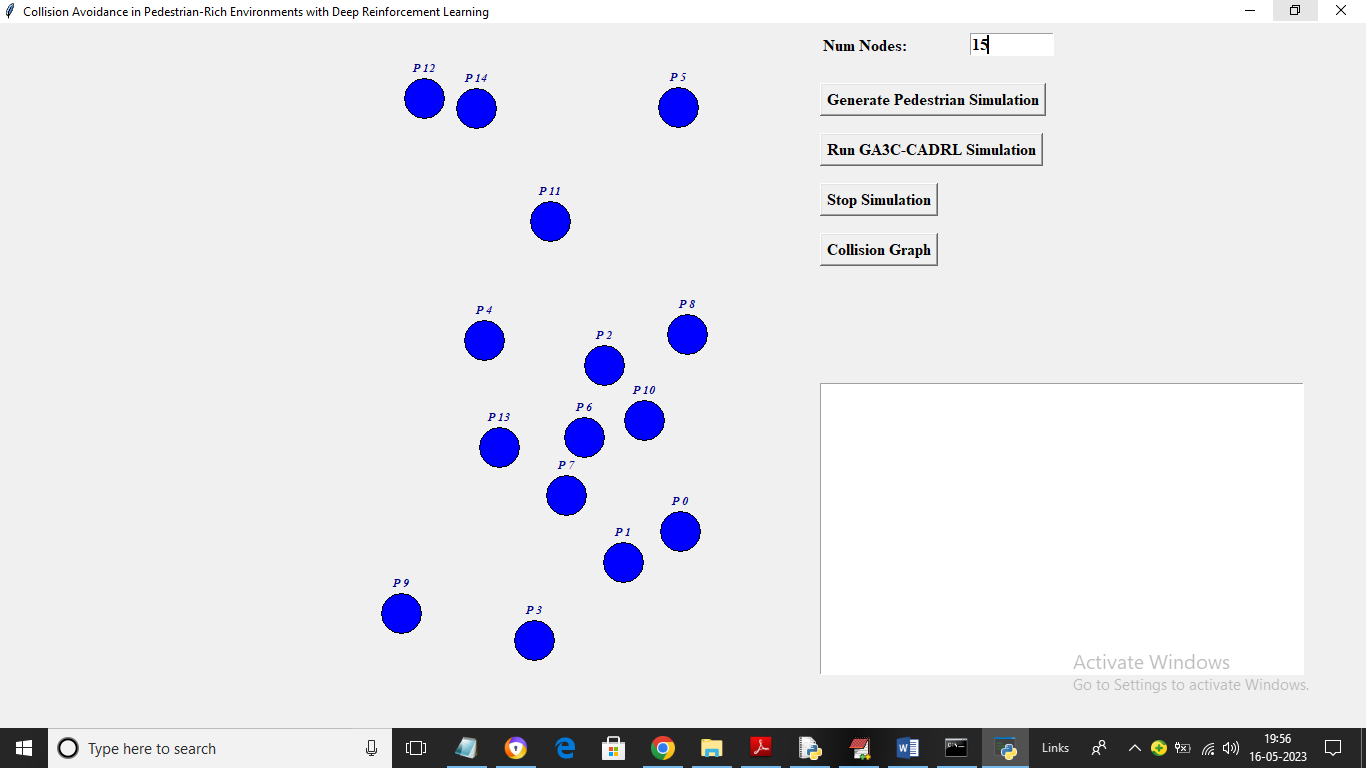


In above two screens you can see all circles are moving as their locations get changing randomly and in above screen if node is predicted as collision then algorithm will change its state to RED COLOR and then algorithm will instruct node to take another location

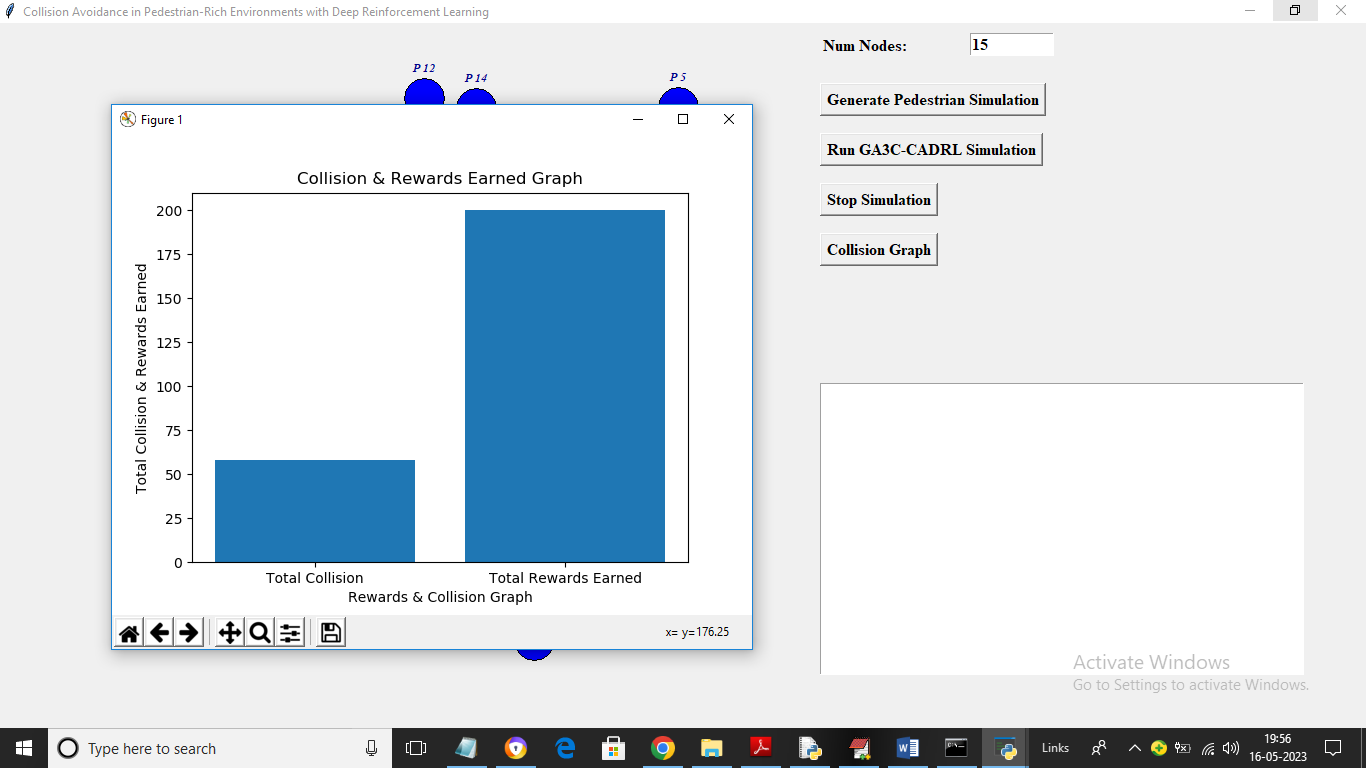




In above screens we can see all nodes are moving and now click on ‘Stop Simulation’ button to get below output



In above screen simulation get stopped and now click on ‘Collision Graph’ button to get below output



In above graph x-axis represents Collision or Rewards and y-axis represents count and we can see total rewards earned by the algorithm and total collisions occurred and predicted by the algorithm.

Similarly by following above screens you can run the project

**CHAPETER 5**

**SOFTWARE AND HARDWARE REQUIRMENT SPECIFICATION**

**5.1 Hardware Requirement:**

• Processor Type: Pentium -IV

• RAM: 512 MB RAM

• Hard disk: 20 GB

**5.2 Software Requirement:**

• Operating System: Windows 2007

• Script: python: Jupyter notebook

**CONCLUSION**

This work presented a collision avoidance algorithm, GA3CCADRL, that is trained in simulation with deep RL without requiring any knowledge of other agents’ dynamics. It also proposed a strategy to enable the algorithm to select actions based on observations of a large (possibly varying) number of nearby agents, using LSTM at the network’s input. The new approach was shown to outperform a classical method, another deep RL-based method, and scales better than our previous deep RL-based method as the number of agents in the environment increased. These results support the use of LSTMs to encode a varying number of agent states into a fixed-length representation of the world. Analysis of the trained LSTM provides deeper introspection into the effect of agent observations on the hidden state vector, and quantifies the effect of agent ordering heuristics on performance throughout training. The work provided an application of the algorithm for formation control, and the algorithm was implemented on two hardware platforms: a fleet of 4 fully autonomous multirotors successfully avoided collisions across multiple scenarios, and a small ground robot was shown to navigate at human walking speed among pedestrians. Combined with the numerical comparisons to prior works, the hardware experiments provide evidence of an algorithm that exceeds the state of the art and can be deployed on real robots.

**REFERENCES**

[1] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, “A navigation system for robots operating in crowded urban environments,” in 2013 IEEE International Conference on Robotics and Automation. IEEE, 2013, pp. 3225–3232.

[2] P. Trautman and A. Krause, “Unfreezing the robot: Navigation in dense, interacting crowds,” in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2010, pp. 797–803.

[3] J. Snape, J. Van den Berg, S. J. Guy, and D. Manocha, “The hybrid reciprocal velocity obstacle,” IEEE Transactions on Robotics, vol. 27, no. 4, pp. 696–706, Aug. 2011.

[4] G. Ferrer, A. Garrell, and A. Sanfeliu, “Social-aware robot navigation in urban environments,” in 2013 European Conference on Mobile Robots (ECMR), Sep. 2013, pp. 331–336.

[5] J. Van den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal nbody collision avoidance,” in Robotics Research, ser. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2011, no. 70, pp. 3–19.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, “Optimal reciprocal collision avoidance for multiple non-holonomic robots,” in Distributed Autonomous Robotic Systems. Springer, 2013, pp. 203–216.

[7] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, “Socially compliant mobile robot navigation via inverse reinforcement learning,” The International Journal of Robotics Research, Jan. 2016.

[8] P. Trautman, J. Ma, R. M. Murray, and A. Krause, “Robot navigation in dense human crowds: the case for cooperation,” in Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA), May 2013, pp. 2153–2160.

[9] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, “Featurebased prediction of trajectories for socially compliant navigation,” in Robotics:Science and Systems, 2012.

[10] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 6252–6259.

[11] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. IEEE, 2017, pp. 31–36.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in Advances in neural information processing systems, 2014, pp. 3104–3112.

[13] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” arXiv preprint arXiv:1406.1078, 2014.

[14] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] Y. Chen, M. Liu, M. Everett, and J. P. How, “Decentralized, noncommunicating multiagent collision avoidance with deep reinforcement learning,” in Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017.

[16] Y. F. Chen, M. Everett, M. Liu, and J. P. How, “Socially aware motion planning with deep reinforcement learning,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, September 2017.

[17] M. Everett, Y. F. Chen, and J. P. How, “Motion planning among dynamic, decision-making agents with deep reinforcement learning,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, September 2018.

[18] J. B. Rawlings, “Tutorial overview of model predictive control,” IEEE control systems magazine, vol. 20, no. 3, pp. 38–52, 2000.

[19] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” IEEE Robotics Automation Magazine, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[20] M. Phillips and M. Likhachev, “SIPP: Safe interval path planning for dynamic environments,” in 2011 IEEE International Conference on Robotics and Automation (ICRA), May 2011, pp. 5628–5635.

[21] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, “Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns,” Autonomous Robots, vol. 35, no. 1, pp. 51–76, May 2013.

[22] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in International Conference on Machine Learning, 2016, pp. 1928–1937.

[25] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, “Reinforcement learning thorugh asynchronous advantage actor-critic on a gpu,” in ICLR, 2017.

[26] S. Omidshafiei, D.-K. Kim, J. Pazis, and J. P. How, “Crossmodal attentive skill learner,” arXiv preprint arXiv:1711.10314, 2017.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.

[28] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” arXiv preprint arXiv:1802.09477, 2018.

[29] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[30] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.

[31] B. Kim and J. Pineau, “Socially adaptive path planning in human environments using inverse reinforcement learning,” International Journal of Social Robotics, vol. 8, no. 1, pp. 51–66, Jun. 2015.

[32] M. Pfeiffer, U. Schwesinger, H. Sommer, E. Galceran, and R. Siegwart, “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). IEEE, 2016, pp. 2096– 2101.

[33] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, “Emergent tool use from multi-agent autocurricula,” in International Conference on Learning Representations, 2020. [Online]. Available: <https://openreview.net/forum?id=SkxpxJBKwS>

[34] J. Garcıa and F. Fernández, “A comprehensive survey on safe reinforcement learning,” Journal of Machine Learning Research, vol. 16, no. 1, pp. 1437–1480, 2015.

[35] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, “Social lstm: Human trajectory prediction in crowded spaces,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 961–971.

**APPENDIX**

**PYTHON**

\* One of the most popular languages is Python. Guido van Rossum released this language in 1991. Python is available on the Mac, Windows, and Raspberry Pi operating systems. The syntax of Python is simple and identical to that of English. When compared to Python, it was seen that the other language requires a few extra lines.

\*It is an interpreter-based language because code may be run line by line after it has been written. This implies that rapid prototyping is possible across all platforms. Python is a big language with a free, binary-distributed interpreter standard library.

\* It is inferior to maintenance that is conducted and is straightforward to learn. It is an object-oriented, interpreted programming language. It supports several different programming paradigms in addition to object-oriented programming, including functional and procedural programming.

\* It supports several different programming paradigms in addition to object-oriented programming, including practical and procedural programming. Python is mighty while maintaining a relatively straightforward syntax. Classes, highly dynamic data types, modules, and exceptions are covered. Python can also be utilised by programmes that require programmable interfaces as an external language.

**Python Features:**

**1) Easy:** Because Python is a more accessible and straightforward language, Python programming is easier to learn.

**2) Interpreted language:** Python is an interpreted language, therefore it can be used to examine the code line by line and provide results.

**3) Open Source:** Python is a free online programming language since it is open-source.

**4) Portable:** Python is portable because the same code may be used on several computer standard **libraries:** Python offers a sizable library that we may utilize to create applications quickly.

**6) GUI:** It stands for GUI (Graphical User Interface)

**7) Dynamical typed:** Python is a dynamically typed language, therefore the type of the value will be determined at runtime.

**Python GUI (Tkinter)**

\* Python provides a wide range of options for GUI development (Graphical User Interfaces).

\* Tkinter, the most widely used GUI technique, is used for all of them.

\* The Tk GUI toolkit offered by Python is used with the conventional Python interface.

\* Tkinter is the easiest and quickest way to write Python GUI programs.

\* Using Tkinter, creating a GUI is simple.

\* A part of Python's built-in library is Tkinter. The GUI programs were created.

\* Python and Tkinter together give a straightforward and quick way. The Tk GUI toolkit's object-oriented user interface is called Tkinter.

\* Making a GUI application is easy using Tkinter. Following are the steps:

1) Install the Tkinter module in place.

2) The GUI applicatioMakeske the primary window

3) Include one or more of the widgets mentioned above in the GUI application.

4) Set up the main event loop such that it reacts to each user-initiated event.

\*Although Tkinter is the only GUI framework included in the Python standard library, Python includes a GUI framework. The default library for Python is called Tkinter. Tk is a scripting language often used in designing, testing, and developing GUIs. Tk is a free, open-source widget toolkit that may be used to build GUI applications in a wide range of computer languages.

**Machine Learning**

\*Artificial intelligence (AI), which includes machine learning, enables computer systems to learn without being explicitly programmed. It has to do with statistics and applied mathematics. Mike Robert's definition of machine learning. As a computer gathers and learns from the data it provides, it may operate more correctly via machine learning.

\*For large classes of machine learning, many algorithms are used. We must provide algorithms with more precise data for them to complete certain jobs. In some circumstances, a computer will utilize data to gather information, check its output against the desired outcome, and make necessary corrections.

\*For instance, when someone texts on a phone, the phone learns about spelling errors and either autocorrects the offending word or suggests a replacement. For many top organizations, machine learning is a critical component of the creation of new products.

\*ML is an important factor in the operations of many companies, like Facebook and Google. Data science uses machine learning in many different ways. Data scientists rely on ML approaches to carry out their modeling. Regression and classification are of utmost relevance in data science; hence, the main tool utilized in ML is to accomplish such objectives.

\* ML applies applicable to practically all phases of data science and is most often associated with the data modeling phase. Python has been the primary computer language used for data processing. Several Python packages are used in ML settings. The three sections of Python are huge data, optimizing your code, and data files in memory.

**1.6 Types of Machine Learning**

There are three fundamental forms of machine learning: -supervising, semi-supervised, and machine learning

**a) Supervised Machine Learning**

\* That method looks for patterns in the labeled data set to obtain results. Data labeling in supervised learning requires human intervention. To train the algorithm with labeled inputs and the intended output, supervised ML requires human participation. ML under supervision is good for a task like;

**I.** Classify the data using a binary system into two groups.

**II. Multi-classification:** The division of data into more than two categories,

**III.** Modeling imaging continuous value using regression.

**IV. Assembling:** Compiling the estimates from many ML models to provide a precise estimate.

**b) Unsupervised Machine Learning**

\*This method searches for patterns in the data collection without relying on labeled data or human interaction. Data labeling is not necessary for this strategy. ML Unsupervised is effective for tasks like;

**I. Dimensionality reduction:** Reduce the number of variables in the data collection.

**II. Clustering:** Grouping the dataset based on similarities.

**III.** Association mining identifies the item or group of items that commonly appear together in data.

**IV.** Data point identification for anomaly detection in the data set

**c) Semi-supervised Learning**:

\*For this method, you require labeled data. As a consequence, human interaction is also necessary, but the process still moves forward. In this kind of learning, the algorithm is given a tiny quantity of labeled data by data scientists, and as a result, the algorithm gains knowledge about the data set's dimension, which it may then apply to mother del, unlabeled data.

\* There are several contexts in which semi-supervised machine learning (ML) may be used.

**I. Machine translation:** Language conversion using a learning system.

**II. Data labeling:** An algorithm trained on modest amounts of data will automatically apply data labels to enormous collections.

**1.7 Uses of Machine Learning**

\*Machine learning is used in many areas nowadays. The most well-known example is the machine learning recommendation engine that drives a book's news feed. This engine makes an effort to reinforce established patterns in a user's online activity inside a certain Facebook group.

\* The news is appropriately adjusted if a user alters the design and doesn't read anything from that particular group the following week. Applications of machine learning (ML) include business intelligence, human resource information systems, autonomous vehicles, and virtual assistants.

**Advantages:**

• ML helps enterprises in comprehending their clients. ML assists in improving goods in response to client demand by gathering the necessary user data and associating it with shifting behavior. Some companies' business models are heavily reliant on machine learning, such as Uber, which uses an algorithm to connect drivers and customers. To surface the advertising in searches, Google employs ML.

**Disadvantage:**

• ML might be expensive. High wages for machine learning are a result of data emotions command on the project. These initiatives also often demand expensive software infrastructure.

• In addition to that, when an algorithm is trained on a data set, ML bias might develop. That has flaws in it that might provide erroneous results.

**Steps to choosing the suitable ML model**

The issue is solved by selecting the best ML model, which might take some time. The steps are as follows:

1) For the difficulty with the pure date alignment, the input should be thought about.

2) Gather, label, and prepare the data as appropriate.

3) To put the right algorithms to use and test them to determine how well they perform.

**Libraries Used**

**Pandas:**

\* Pandas is a Python computer language library for data analysis and manipulation. It offers a specific operation and data format for handling time series and numerical tables. It differs significantly from the release3-clause of the BSD license. It is a well-liked open-source of opinion that is utilized in machine learning and data analysis.

**NumPy:**

\* The NumPy Python library for multi-dimensional, big-scale matrices adds a huge number of high-level mathematical functions. It is possible to modify NumPy by utilizing a Python library. Along with line, algebra, and the Fourier transform operations, it also contains several matrices-related functions.

**Matplotlib:**

\* It is a multi-platform, array-based data visualization framework built to interact with the whole SciPy stack. MATLAB is proposed as an open-source alternative. Matplotlib is a Python extension and a cross-platform toolkit for graphical plotting and visualization.

**Scikit-learn:**

\* The most stable and practical machine learning library for Python is scikit-learn. Regression, dimensionality reduction, classification, and clustering are just a few of the helpful tools it provides through the Python interface for statistical modeling and machine learning. It is an essential part of the Python machine learning toolbox used by JP Morgan. It is frequently used in various machine learning applications, including classification and predictive analysis.

**Keras:**

\* Google's Keras is a cutting-edge deep learning API for creating neural networks. It is created in Python and is designed to simplify the development of neural networks. Additionally, it enables the use of various neural networks for computation. Deep learning models are developed and tested using the free and open-source Python software known as Keras.

**h5py:**

\* The h5py Python module offers an interface for the binary HDF5 data format. Thanks to p5py, the top can quickly halt the vast amount of numerical data and alter it using the NumPy library. It employs common syntax for Python, NumPy, and dictionary array