# Crowd-Steer: Realtime Smooth and Collision-Free Robot Navigation in Densely Crowded Scenarios Trained using High-Fidelity Simulation

**Jing Liang**[1*], **Utsav Patel**[1*], **Adarsh Jagan Sathyamoorthy**[1] and **Dinesh Manocha**[1]
[1]University of Maryland, College Park, USA
{jliang92, Upatel22, asathyam}@umd.edu, dm@cs.umd.edu

## Abstract

We present a novel high fidelity 3-D simulator that significantly reduces the sim-to-real gap for collision avoidance in dense crowds using Deep Reinforcement Learning (DRL). Our simulator models realistic crowd and pedestrian behaviors, along with friction, sensor noise and delays in the simulated robot model. We also describe a technique to incrementally control the randomness and complexity of training scenarios to achieve better convergence and generalization capabilities. We demonstrate the effectiveness of our simulator by training a policy that fuses data from multiple perception sensors such as a 2-D lidar and a depth camera to detect pedestrians and computes smooth, collision-free velocities. Our novel reward function and multi-sensor formulation results in smooth and unobtrusive navigation. We have evaluated the learned policy on two differential drive robots and evaluate its performance in new dense crowd scenarios, narrow corridors, T and L-junctions, etc. We observe that our algorithm outperforms prior dynamic navigation techniques in terms of metrics such as success rate, trajectory length, mean time to goal, and smoothness.

## 1 Introduction

Mobile robots are frequently deployed in indoor and outdoor environments such as hospitals, hotels, malls, airports, warehouses, sidewalks, etc. These robots are used for surveillance, inspection, delivery, and cleaning, or as social robots. For such applications, robots must smoothly and reliably navigate in these scenarios by avoiding collisions with obstacles, including dynamic agents or pedestrians. The crowd density in these scenarios generally vary between 1-3 pedestrians per square meter.

In recent years, there has been significant work on learning-based collision avoidance for mobile robots operating in such dense scenarios. These include techniques based on end-to-end deep learning [Kim *et al.*, 2018; Pfeiffer *et al.*, 2017], generative adversarial imitation learning [Tai *et al.*, 2018], and Deep Reinforcement Learning (DRL) [Long *et al.*, 2017; Fan *et al.*, 2018b]. Specifically, DRL-based methods [Chen *et al.*, 2017; Everett *et al.*, 2018; Long *et al.*, 2017] have demonstrated superior collision avoidance rates, lower time-to-goal, and higher average speed of the agent when compared to traditional Velocity Obstacle (VO) methods. These methods typically use a single sensor such as a lidar, depth camera, or RGB camera for obstacle avoidance.

Figure 1: Turtlebot and Jackal robot using CrowdSteer to navigate in scenarios with pedestrians in a narrow corridor and areas with high occlusion. Our method uses data from multiple sensors such as a 2-D lidar and a depth camera to generate smooth collision avoidance maneuvers. We highlight the benefits over prior algorithms, such as DWA and DRL methods.

In practice, the DRL-based methods are trained in a simulator and the trained policy is transferred to a real robot. This strategy of DRL policy training suffers from the sim-to-real gap, i.e., the policy's navigation performance in real-life can be worse than its performance in the simulator. The main reasons for the sim-to-real gap are 1) not modeling noisy sensor data readings, friction and delays in robot hardware [Neunert *et al.*, 2016] and 2) difficulty in recreating real-life robot-pedestrian interactions with varying complexity in the simulator.

In addition to sim-to-real transfer issues, the choice of perception sensors that were used to train the policy also affects the performance of the navigation methods. For instance,

[*]Authors contributed equally.

algorithms that are trained using a single lidar [Long *et al.*, 2017] may perform well in dense scenarios but lack the ability to differentiate between animate and inanimate obstacles. Methods which use several perception sensors [Chen *et al.*, 2017; Everett *et al.*, 2018] exhibit good performance in moderately dense crowds but not in high density crowds, and are susceptible to perception errors. Their trajectories also tend to be jerky and oscillatory.

**Main Results.** We present a novel high-fidelity 3-D simulator for training learning-based navigation policies for densely crowded scenarios. In addition, we use our simulator to train CrowdSteer, a novel collision avoidance policy that fuses data from inexpensive perception sensors such as a 2-D lidar and a depth (RGB-D) camera to sense obstacle features. Our simulator helps the policy implicitly learn different kinds of robot interactions with the pedestrians, and significantly reduces the sim-to-real gap. The novel components of our work are:

- A high-fidelity 3-D simulator which accurately models real-world crowd behavior, sensor noise, frictions and delays of the robot, to close the sim-to-real gap while training DRL policies for dense crowd navigation. Our simulator also allows to incrementally vary the complexity and randomness of the training scenarios. The robot's performance in simulation and in real-world scenarios matches over 80% of the time, demonstrating our simulator's sim-to-real transfer capabilities. Previous methods perform well in simulations but exhibit oscillatory/jerky motions on real hardware.
- A DRL-based collision avoidance policy trained in our simulator that fuses inputs from a 2-D lidar and depth camera for implicitly characterizing the robot's interactions with pedestrians. The policy smoothly navigates through complex real-world scenarios.
- Reward function shaping for crowd navigation that leads to highly smooth robot trajectories which are less obtrusive to pedestrian motion. Our ablation study (Table 2) shows more than 4 times reduction in oscillations.

We evaluate the generalization and sim-to-real capabilities of the trained DRL policy on two real robots, Turtlebot and ClearPath Jackal in indoor environments such as corridors, L and T-junctions with varying pedestrian density (1-3 humans/$m^2$). We also compare its performance with prior traditional methods such as the Dynamic Window Approach (DWA) [Fox *et al.*, 1997] and a state-of-the-art learning-based crowd navigation algorithm [Long *et al.*, 2017]. We observe that our approach surpasses these methods in terms of success rates, smoothness and shows a reduction of up to **68.16%** in time to goal, and **6.12%** reduction in trajectory length when compared to the current state of the art in DRL-based solutions [Long *et al.*, 2017].

## 2 Related Work

In this section, we give a brief overview of the prior work on crowd simulation, and traditional and learning-based collision avoidance algorithms.

### 2.1 Sim-to-Real and Crowd Simulation

Reasons for the sim-to-real gap and methods to mitigate them [Neunert *et al.*, 2016; Yu *et al.*, 2019] have been discussed in several works. For training a policy for crowd navigation in simulation, the crowd's behavior must mimic real-world behavior. Works such as [Narang *et al.*, 2015; Pelechano *et al.*, 2007; Narain *et al.*, 2009; Braun *et al.*, 2003] discuss several dynamics, physiological and psychological factors that should be accounted to make crowd simulation more realistic. [Curtis *et al.*, 2016] demonstrates a modular structure for developing a crowd simulator. We extend the concepts from these works for our crowd navigation policy training.

### 2.2 Navigation in Dynamic Scenes

There is extensive work on collision avoidance in dynamic scenes for robots. These include techniques based on potential-field methods [Tilove, 1990], social-forces [Helbing and Molnar, 1995], velocity obstacles and its extensions [van den Berg *et al.*, 2009; Alonso-Mora *et al.*, 2010; Bareiss and van den Berg, 2015; Alonso-Mora *et al.*, 2018], etc. In terms of real-world scenarios, these methods require accurate sensing of obstacles' positions and velocities and parameter tuning that is scenario-dependent. These requirements make it difficult to directly apply them for navigation in dense crowds. The Dynamic Window Approach (DWA) [Fox *et al.*, 1997] is another widely used method which calculates reachable dynamically-constrained velocities for collision avoidance within a short time interval. However, it does not scale well to large numbers of dynamic obstacles (Section 5.5).

### 2.3 Learning-based Collision Avoidance

Several works have used a single perception sensor such as 2-D lidar or raw data from a depth camera to train collision avoidance behaviors in a robot using expert demonstrations [Pfeiffer *et al.*, 2016] and imitation learning [Tai *et al.*, 2018]. End-to-end methods [Kim *et al.*, 2018], and methods using deep double-Q networks [Xie *et al.*, 2017] use RGB image data for training.

**DRL-based Methods.** A decentralized collision avoidance method was trained with Proximal Policy Optimization (PPO) [Schulman *et al.*, 2017] using a 2-D lidar in [Long *et al.*, 2017]. This approach was extended to a hybrid control architecture [Fan *et al.*, 2018b], which switched between different policies to optimize the navigation. This approach works well in open spaces, but exhibits oscillatory and jerky motions in dense scenarios since the 2-D lidar only senses the proximity data and fails to sense more complex interactions. [Chen *et al.*, 2017] presents a decentralized agent-level policy that utilizes a trained value network that models the cooperative behaviors in multi-agent systems. An LSTM-based strategy that uses observations of arbitrary numbers of neighboring agents during the training phase is described in [Everett *et al.*, 2018]. Other methods [Chen *et al.*, 2019] explicitly model robot-human interactions in a crowd for robot navigation. These algorithms use a 2-D or 3-D lidar along with several RGB cameras for pedestrian classification and have been tested in moderately dense crowds.

These methods use either basic 2.5D simulators or trajectories generated by methods such as ORCA [van den Berg *et al.*, 2009] for policy training, which may not translate well to real-world scenarios.

# 3 Overview

In this section, we provide a brief overview on the multi-sensor based navigation policy trained in our simulator.

## 3.1 Multiple Sensors

To test the capabilities of our simulator, we train a policy that exploits data from two sensors (2-D lidar and depth camera) simultaneously. Our simulator is also extendable to other sensor/robot models. The lidar's raw data does not provide sufficient information to differentiate between animate and inanimate obstacles or sudden changes in the orientation of obstacles which makes it difficult to infer the direction in which obstacles are moving. Therefore, we also use images from a depth camera to capture such interactions that are useful for a robot's navigation.

**2-D Lidar.** Each scan/frame from a 2-D lidar consists of a list of distance values on the plane of sight of the lidar (See left scenario in Fig. 2). With its high accuracy, Field Of View (FOV), and low dimensional output data, the 2-D lidar allows us to detect clusters of closest points in the robot's surroundings.

**Depth Camera.** Depth images possess an additional dimension over and above a 2-D lidar. Therefore, features such as obstacle contours and changes in obstacles' poses are more prominently recorded even in low resolution images. If we consider several consecutive frames from the camera, the approximate positions, orientations and velocities of all obstacles/pedestrians in the frame can be extracted.

## 3.2 Robot Navigation

The dynamics of the robot we train is formulated using non-holonomic constraints [Alonso-Mora *et al.*, 2010]. The robot knows its environment only through local observations, and it has no global knowledge of the environment, and cannot access hidden parameters of pedestrians, such as their goals. At each time step $t$, the robot has access to an observation vector $o^t$ which it uses to compute a collision-free action that drives it towards its goal. We also assume that the pedestrians cooperate with the robot to avoid collisions, and that data from the lidar and depth camera are synchronized.

We split the robot's *observation space* into four components, $\mathbf{o}^t = [\mathbf{o}^t_{lid}, \mathbf{o}^t_{cam}, \mathbf{o}^t_g, \mathbf{o}^t_v]$, where $\mathbf{o}^t_{lid}$ denotes raw noisy 2-D lidar measurements, $\mathbf{o}^t_{cam}$ denotes the raw image data from a depth camera, $\mathbf{o}^t_g$ refers to the relative goal location with respect to the robot, and $\mathbf{o}^t_v$ denotes the current velocity of the robot.

The *action space* of the robot is composed of its linear and angular velocities $\mathbf{a}^t = [v^t, \omega^t]$. At each instant, the trained navigation policy $\pi_\theta$ selects an action $\mathbf{a}^t$ to drive the robot towards its goal while avoiding collisions with pedestrians and static obstacles, until a new observation $\mathbf{o}^{t+1}$ is measured. During training, we optimize the policy $\pi_\theta$ to minimize the arrival time of the robot to its goal. We choose Proximal Policy Optimization [Schulman *et al.*, 2017], a policy gradient based method to train our policy in simulation. Since the current state-of-the-art DRL-based collision avoidance policy uses PPO, it helps us compare our policy against it.

# 4 Our Simulation Approach and CrowdSteer

In this section, we discuss the development of our simulator and our multi-sensor based collision avoidance method that is trained in it.

## 4.1 Pedestrian Behavior Modeling

Pedestrian velocities in crowds are dictated by several complex factors such as crowd density, stride length of a person, and need for personal space. The fundamental diagram [Narang *et al.*, 2015] provides an inverse relationship between individual pedestrian velocities and the crowd density. To create realistic crowd navigation training scenarios, we need to account for the density-dependent behaviors based on the above said factors.

Each pedestrian in our simulation is modeled as a disk in 2-D plane with the following state space: [r $\vec{p}$ $\vec{v}^{curr}$ $\vec{v}^{pref}$] $\in \mathbb{R}^7$. Here, r denotes the disc radius. $\vec{p}, \vec{v}^{curr}$ and $\vec{v}^{pref}$ represent the current 2-D position, current velocity and the preferred velocities of the pedestrians respectively. We relate a pedestrian's natural walking velocity (V) with physiological (pedestrian's height and stride length), and psychological (need for personal space) factors using the following equation:

$$V = min\left(||\vec{v}^{pref}||, \left(\frac{S\alpha}{H(1+\beta)}\right)^2\right) \quad (1)$$

where S is the available space in front of the pedestrian, H (height/1.72) is a height normalization factor, $\alpha$ and $\beta$ are constants that account for stride of the pedestrian. Accounting for these factors during policy training along with appropriate reward function shaping lead to navigation that is more pedestrian-friendly.

## 4.2 Scenario and Robot Model Development

An important challenge when designing a simulator and scenario for training navigation policies is to strike a balance between: (i) the generality of the scenario such that overfitting is avoided, (ii) maintaining the complexity of the training scenarios such that the policy converges. We achieve this by training the policy starting from simple to complex scenarios. We vary each scenario's complexity and randomness by using a *complexity factor* ($\mathfrak{c}$) for the different scenarios, and by correlating it with a certain attribute of the scenario.

We classify our scenarios into 3 broad categories and explain the complexity factor for each category.

**Static Scenario.** This category of scenarios contains only static obstacles in an enclosed space of constant area and the policy learns goal-reaching and static obstacle avoidance behaviors in these scenarios. The complexity factor $\mathfrak{c}$ controls the number of obstacles in the scenario. As $\mathfrak{c}$ is increased, obstacles are randomly placed in the scenario, which reduces the free space available for the robot to traverse through.

**Random Static and Dynamic Obstacles.** In this category of scenarios, $\mathfrak{c}$ controls the number of static obstacles, and the magnitude of velocity of dynamic pedestrians. The simulated pedestrians use the velocity directions computed as mentioned in Section 4.1. As ($\mathfrak{c}$) is increased, the training scenario has more randomly placed static obstacles (less traversible space for the robot) and robot needs to handle dynamic pedestrians.

**Scenario with Occluded Obstacles.** In this category of scenarios, the robot learns to perform several sharp turns and avoid pedestrians who can only be observed in close proximity. This trains the robot to perform quick maneuvers in real-world scenarios. Here $\mathfrak{c}$ is correlated with the occlusion % in the environment, and the number of dynamic obstacles. Depending on the FOV of the depth-camera, the occlusion % is defined as $\left(\frac{\text{Angle not visible}}{\text{Camera's FOV}}\right) \times 100\%$. As $\mathfrak{c}$ increases, the occlusions become more severe and the robot faces more dynamic obstacles in random intervals.

The complexity factor $\mathfrak{c}$ gives more control on incrementally changing the randomness and complexity of the scenarios. This aids in training convergence, and since all scenarios include some randomness, overfitting is prevented.

**Robot Model.** We use the model of a Turtlebot in simulation for training. We compare the velocities of a real Turtlebot mounted with a laptop and sensors with a simulated robot, and use system identification techniques to deduce additional motor friction and inertia parameters that the simulated robot model must include. We also add Gaussian noise $\mathcal{N}(0, 0.2)$ to the depth images, and the goal location in the simulation and increased the delay in subscribing to current velocity observations. The mounting position of the depth camera is also varied during training to improve generalization. These modifications help more accurately recreate hardware phenomenon in simulation.

### 4.3 Reward Function Shaping

To train the basic objectives of navigation, the robot is rewarded for heading towards and reaching its goal, and penalized for moving too close or colliding with an obstacle. In addition, the robot is penalized for oscillatory velocities, and rewarded for reaching intermediate waypoints ($r_{wp}$) provided relative to the robot. No global information is available.

Formally, the total reward collected by a robot *i* in the simulation at time instant *t* can be given as:

$$r_i^t = (r_g)_i^t + (r_c)_i^t + (r_{osc})_i^t + (r_{safedist})_i^t \tag{2}$$

where the reward for reaching the goal $(r_g)_i^t$ or an intermediate waypoint is given as:

$$(r_g)_i^t = \begin{cases} r_{wp} & if\ ||\mathbf{p}_i^t - \mathbf{p}_{wp}|| < 0.1, \\ r_{goal} & if\ ||\mathbf{p}_i^t - \mathbf{g}_i|| < 0.1, \\ 2.5(||p_i^{t-1} - g_i|| - ||p_i^t - g_i||) & otherwise. \end{cases} \tag{3}$$

Here, $\mathbf{p}_i^t$ and $\mathbf{p}_{wp}$ denotes the position of the $i^{th}$ robot at time *t* and the relative position of its next waypoint respectively, and $\mathbf{g}_i$ denotes its final goal location. The collision penalty $(r_c)_i^t$ is given as:

$$(r_c)_i^t = \begin{cases} r_{collision} & if\ ||\mathbf{p}_i^t - \mathbf{p}_{obs}|| < 0.3, \\ 0 & otherwise. \end{cases} \tag{4}$$

The oscillatory behaviors (choosing sudden large angular velocities $\omega_i^t$) are penalized as:

$$(r_{osc})_i^t = -0.1|\omega_i^t| \qquad if\ |\omega_i^t| > 0.3. \tag{5}$$

Although the penalty for collision teaches the robot not to collide, it does not specifically result in the robot maintaining a safe distance from pedestrians. The penalty for moving too close to an obstacle is given by:

$$\begin{cases} (r_{safedist})^t = -0.1|d_{thresh} - d_{rob}| \ \text{ if } \ d_{thresh} > d_{rob} \\ (r_{safedist})_i^t = -0.1||S - d_{rob}|| \ \text{ if } \ S > d_{rob} \end{cases} \tag{6}$$

where $d_{thresh}$ and $d_{rob}$ denote the *threshold* distance that the robot needs to maintain from an inanimate obstacle at any time, and the actual distance that the robot maintains from an obstacle. S is the space needed for a pedestrian to walk without changing his/her velocity, calculated from equation 1 for a given V, $\alpha$ and $\beta$. This reward term is dynamic, as the value of S varies for different pedestrians. This novel addition trains the policy to navigate among humans while causing minimum disturbance to them, which previous methods have not accounted for in their training.

### 4.4 Network Architecture

The network architecture used for training our DRL policy is shown in Fig.3. The network has 4 branches to process each observation of the robot. We use a deeper branch with several 2-D convolutional layers to process depth images.

## 5 Results and Evaluations

In this section, we describe our implementation and highlight its performance in different scenarios, compare it with prior methods and perform ablation studies to highlight the benefits of our simulator, using multiple sensors and our reward function.

### 5.1 Implementation

We train our model in simulations that were created using ROS Kinetic and Gazebo 8.6 on a workstation with an Intel Xeon 3.6GHz processor and an Nvidia GeForce RTX 2080Ti GPU. We use Tensorflow, Keras and Tensorlayer for implementing our network. We use models of the Hokuyo 2-D lidar and the Orbbec Astra depth camera in Gazebo to simulate sensor data during training and evaluation. We mount these sensors on a Turtlebot 2 and a Clearpath Jackal robot to test our policy in real-world scenarios such as crowded corridors and occluded scenes.

### 5.2 Training Convergence and Data Efficiency

The convergence of our reward function versus the number of iterations for different training scenarios is shown in Fig.4. We gradually increase the complexity factor of the training

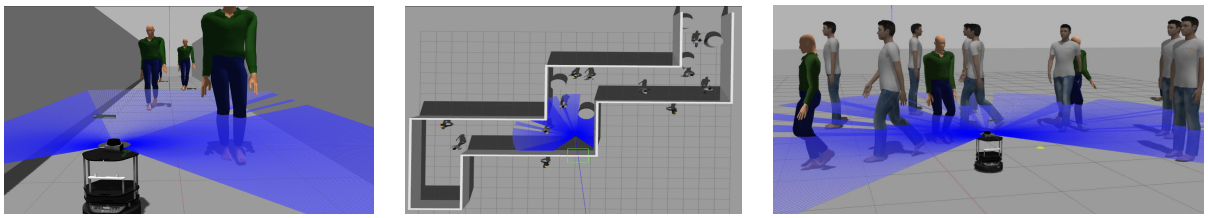

Figure 2: Left to right: Some of the training scenarios created in our simulator. Left: Corridor with dynamic pedestrians. Middle: Scenario with high occlusions, and static and dynamic obstacles. Right: Scenario with random number of standing and walking pedestrians.

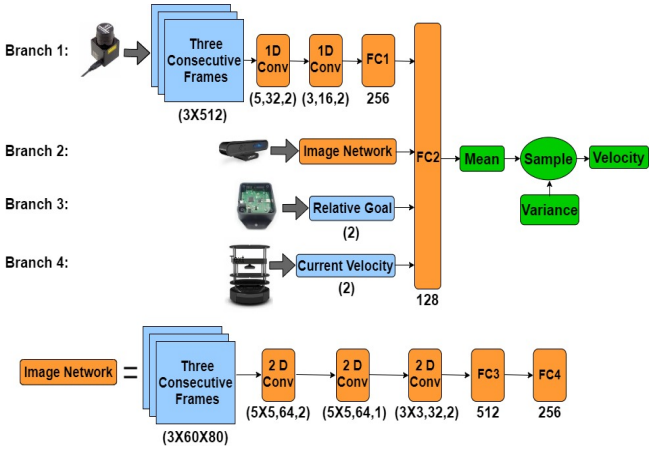


Figure 3: Architecture of our multi-sensor based navigation network with four branches for different observations. The input layer is marked in blue, the hidden layers are marked in orange. Fully connected layers in the network are marked as FCn. The second branch extracts features from three consecutive image frames, which are fused with features extracted from three frames of the lidar in FC2 layer. The three values underneath each hidden layer denote the kernel size, number of filters, and stride length respectively.

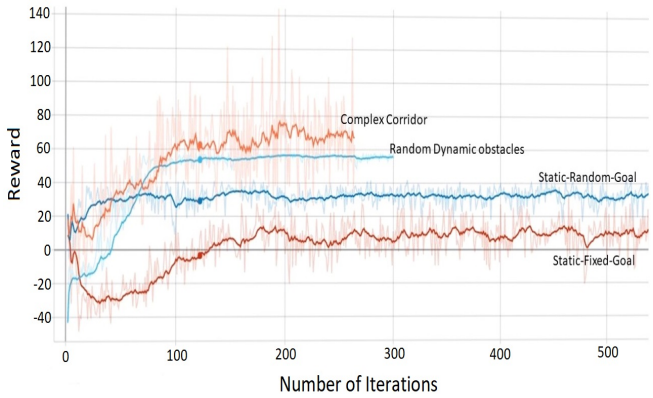


Figure 4: Convergence of the total reward earned per episode vs the number of iterations (training episodes) for different training scenarios. Each iteration consists of utmost 500 training steps. Training in all scenarios converges within 200 iterations. This shows the benefits of our training scenario design.

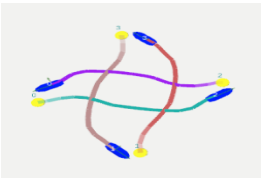

Figure 5: Testing the generalization of our training in a scenario with four agents (in blue) moving towards antipodal points on a circle (in yellow).

scenarios and the training starts to converge at 100 iterations, and stabilizes at 200 iterations. The total process completes in six days. This increase in training time when compared with previous methods is due to the complexity and number of scenarios, and the dimensionality of the 3-D depth data used during training. However, the training time does not affect our run-time performance, which can be observed from our real-world tests.

### 5.3 Testing Scenario

We consider five different test scenarios that have narrower or different sections, as compared to our training scenarios. The scenarios demand tight maneuvers from the robot to reach the goal and are more challenging than existing simulation benchmarks and help to better test CrowdSteer's sim-to-real, and generalization capabilities. We define *Least Passage Space* (LPS) as the minimum space available to the robot in-between obstacles when moving towards the goal. The scenarios we consider are:

- **Narrow-Static**: Scenario with only static obstacles where the LPS is $< 0.7$ meters.
- **Narrow-Ped**: Scenario with 8 pedestrians walking in the opposite direction to the robot's motion in a narrow corridor, with an LPS of $< 1.5$ meters.
- **Occluded-Ped**: Scenario with sharp turns with static obstacles and pedestrians that are occluded by walls. The LPS is $< 1$ meter.
- **Dense-Ped**: Scenario with 18 pedestrians in a corridor of width 6 meters. Pedestrians may walk together as pairs, which require a robot to make sharp turns in the presence of multiple dynamic obstacles. The LPS is $< 1$ meter.
- **Circle**: To test the generalization of our method for multi-robot collision avoidance, we make 4 robots move towards antipodal positions on a circle. The trajectories of the 4 robots is shown in Fig. 5.

| Metrics | Method | Narrow-Static | Narrow-Ped | Occluded-Ped | Dense-Ped |
|---|---|---|---|---|---|
| Success Rate (higher is better) | DWA | 1.00 | 0.10 | 0.50 | 0.00 |
| | Depth Camera | 0.85 | 0.55 | 0.20 | 0.20 |
| | Long et al. | 1.00 | 0.00 | 0.00 | 0.00 |
| | CrowdSteer | 1.00 | 1.00 | 0.90 | 0.67 |
| Avg Trajectory Length (lower is better) | DWA | 6.33 | 14.86 | 27.20 | 7.46 |
| | Depth Camera | 6.18 | 16.30 | 25.70 | 14.95 |
| | Long et al. | 6.86 | 6.16 | 13.63 | 9.17 |
| | CrowdSteer | 6.44 | 15.51 | 27.18 | 16.58 |
| Mean Time (lower is better) | DWA | 20.90 | 44.10 | 60.40 | 25.72 |
| | Depth Camera | 58.70 | 43.60 | 78.20 | 90.73 |
| | Long et al. | 106.93 | 13.76 | 28.02 | 38.05 |
| | CrowdSteer | 34.04 | 41.48 | 70.54 | 64.90 |
| Avg Velocity (higher is better) | DWA | 0.30 | 0.34 | 0.45 | 0.29 |
| | Depth Camera | 0.11 | 0.37 | 0.33 | 0.16 |
| | Long et al. | 0.06 | 0.44 | 0.48 | 0.23 |
| | CrowdSteer | 0.20 | 0.37 | 0.39 | 0.26 |

Table 1: We compare the relative performance of CrowdSteer that uses multiple sensors (depth camera + lidar) with other learning methods that use a single sensor, and a traditional method DWA in challenging scenarios. Note: The numbers in grey are values recorded until the robot collided or started oscillating indefinitely and represent poor performance. These results clearly highlight the benefit of our novel deep reinforcement learning algorithm (CrowdSteer) that uses multiple sensors over prior methods.

### 5.4 Performance Benchmarks and Metrics

We compare the benefits of our multi-sensor policy with three prior algorithms: (i) Dynamic Window Approach [Fox *et al.*, 1997] along with a global planner which requires a map of the environment.(ii) An implementation that uses a single depth camera that was trained using PPO and our reward functions; (iii) Long et al. [Long *et al.*, 2017], the current state-of-the-art DRL-based collision avoidance policy for dense crowd navigation, trained in a 2.5D simulator. Its real-world implementation is shown in [Fan *et al.*, 2018a]. We use the following metrics to evaluate the performance of different navigation algorithms:

- **Success Rate** - The number of times that the robot reached its goal without colliding with an obstacle over the total number of attempts.
- **Average Trajectory Length** - The trajectory length traversed by the robot until the goal is reached, averaged over the total number of attempts. In cases where the robot never reached the goal, we report the trajectory length *until it collided or started oscillating* indefinitely.
- **Mean Time** - Average time taken to reach the goal over all attempts. If the goal is never reached in all attempts, we report the mean time until a collision or indefinite oscillation.
- **Average Velocity** - The average velocity of the robot until a collision occurs or the goal is reached over all attempts.

The values for the trajectory length, mean time and velocity reported when Long et al's method always failed, are *until a collision or indefinite oscillation* to give a sense of how much the robot traversed towards the goal before failing.

### 5.5 Analysis and Comparison

The results of our experiments and our ablation study to check the benefits of using multiple sensors (both lidar and depth camera) versus using one sensor (only depth camera) are shown in Table 1.

**Comparisons with DWA and [Long *et al.*, 2017].** All methods perform well in static scenarios. However, as the number and density of dynamic agents increases, both DWA and Long et al's method's performances drop considerably. This is mainly due to the high replanning time in DWA and the robot freezing problem and oscillations in the case of Long at al. DWA performs well in occluded areas due to it using a global map, while CrowdSteer has comparable performance using only local knowledge. Apart from having a much better success rate, CrowdSteer has similar performance as prior methods in terms of average velocities, better trajectory lengths and mean time to Long et al.'s method. Therefore, our method outperforms the current state-of-the-art in dense and occluded scenarios. This is a testament to the benefits of our training scenarios, reward function and use of multiple sensors.

**Comparing Smoothness.** We observe that CrowdSteer has significantly less oscillations (sudden changes in the angular velocity of the robot) than Long et al's method [Long *et al.*, 2017] resulting in smoother robot trajectories. CrowdSteer's novel reward structure trains the policy to maintain a safe and comfortable distance from pedestrians and avoids oscillations.

**Ablation Study for Multi-Sensor Fusion.** We compare the effects of using fused data from two sensors (CrowdSteer) versus a policy which uses one sensor (Depth Camera) trained

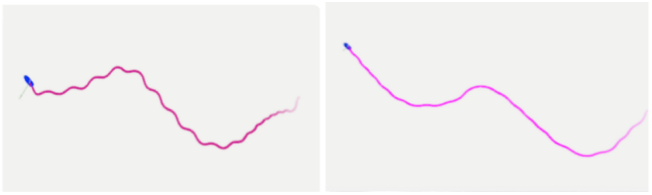

Figure 6: Trajectories generated by Long et al.'s method (left) and CrowdSteer (right) in the Occluded-ped scenario. Our realistic scenarios and reward function have led to much fewer oscillations.

| Scenario | Without Penalty | With Penalty |
|---|---|---|
| Empty world | 9.8 | 2.0 |
| Static obstacle | 9.0 | 2.0 |

Table 2: Ablation Study for smoothness: The average number of oscillations in two scenarios for two models trained without and with the oscillations penalty term (Section 4.3). We see a reduction of up to 4 times in the number of oscillations in our model that is trained with the penalty.

with our training scenarios and reward function. Due to limited depth camera FOV, the depth camera model does not have a 100% success rate in any scenario. This also reflects in the drop in success rate in the Occluded-Ped and Dense-Ped benchmarks. However, it still manages to succeed in the dynamic testing scenarios due to our realistic training scenarios and reward function. CrowdSteer takes advantage of both the high accuracy of the lidar and the complex features extracted from the depth images and demonstrates significantly better success rates, lower mean time, and higher average velocities.

**Ablation Study for Smoothness.** We study the effect of our reward function on the robot's trajectory smoothness. We trained two policies, one including the penalty for oscillations in the reward function (Eqn. 5), and the other without it. The average number of oscillations in the robot's trajectory are summarized in Table 2. The models are evaluated in two scenarios: (i) Scenario without any obstacles, and the robot moves in a straight line for 10 meters towards the goal, (ii) Scenario where the robot must maneuver to avoid a static obstacle before reaching the goal. Empty/sparse scenarios are used so that turns during dynamic obstacle avoidance do not affect the number of oscillations. We observe that there is a significant reduction in the number of oscillations when the penalty is included.

**Real-World Scenarios.** We use CrowdSteer to navigate a Turtlebot and a Clearpath Jackal robot in crowds with varying densities (1-3 person/$m^2$), as shown in the video.[1] The robots face high randomness in terms of the direction and velocities of pedestrians, which was not encountered during training. We compare the motion of CrowdSteer with Long et al.[Long *et al.*, 2017] method in similar scenarios. Compared to Long et al., we observe that CrowdSteer has smoother trajectories in both robots and avoids all collisions with the obstacles. In occluded spaces such as corridors, CrowdSteer was able to avoid sudden obstacles which appear in places such as T and L junctions. These tests again highlight the benefits of our extensive training in occluded places, and implicit sensor fusion. Our real-world tests also demonstrate our simulator's strong sim-to-real and generalization capabilities.

**Failure Cases.** CrowdSteer may not work well certain cases. It might fail in highly acute angled turns and in environments with reflective or transparent surfaces, and high interference from infrared light in the surroundings. In crowds with density $> 4$ people/$m^2$ or scenarios with very minimal or narrow space for navigation, the robot may not find a collision-free path.

### 5.6 Improved Sim-to-Real Transfer

We compared the linear and angular velocities between a simulated and real robot when both are made to navigate in the similar scenarios with static obstacles. We observed that the velocity changes in both robots correlated over 80 % of the times. This demonstrates the accurate modeling of real-world interactions, and hardware parameters such as friction, inertia and delays. Previous DRL policies, when implemented in real hardware exhibit unnatural oscillatory behaviors since their training could not model such phenomenon due to the simplicity of their simulators and datasets. We also note that the simulator can be used to extensively collect trajectory/sensing datasets.

## 6 Conclusion, Limitations and Future Work

We present a novel 3-D, high-fidelity simulator for training learning-based collision avoidance policies, with systematically increased complexity. We trained CrowdSteer, a policy that simultaneously uses multiple sensors such as 2-D lidars and cameras. In practice, our approach works well in complex, occluded scenarios and results in smoother trajectories. Our approach has some limitations and failure cases. It is susceptible to freezing robot problem in very dense settings and the computed trajectories are not globally optimal. Furthermore, the current sensors may not accurately handle glass or non-planar surfaces. These are avenues for future work. In addition to overcoming these limitations, we need to evaluate its performance in other scenarios and outdoor settings.

## Acknowledgments

This work was supported in part by ARO Grants W911NF1910069 and W911NF1910315, and Intel.

## References

[Alonso-Mora *et al.*, 2010] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul A. Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *DARS*, 2010.

[Alonso-Mora *et al.*, 2018] J. Alonso-Mora, P. Beardsley, and R. Siegwart. Cooperative collision avoidance for nonholonomic robots. *IEEE Transactions on Robotics*, 34(2):404–420, 2018.

[1] https://www.youtube.com/watch?v=3NrhuQDsAoc

[Bareiss and van den Berg, 2015] Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[Braun *et al.*, 2003] A. Braun, S. R. Musse, L. P. L. de Oliveira, and B. E. J. Bodmann. Modeling individual behaviors in crowd simulation. In *Proceedings 11th IEEE International Workshop on Program Comprehension*, pages 143–148, May 2003.

[Chen *et al.*, 2017] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *ICRA*, pages 285–292. IEEE, 2017.

[Chen *et al.*, 2019] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *ICRA*, pages 6015–6022. IEEE, 2019.

[Curtis *et al.*, 2016] Sean Curtis, Andrew Best, and Dinesh Manocha. Menge: A modular framework for simulating crowd movement. *Collective Dynamics*, 1:1–40, 2016.

[Everett *et al.*, 2018] Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *IROS*, pages 3052–3059. IEEE, 2018.

[Fan *et al.*, 2018a] Tingxiang Fan, Xinjing Cheng, Jia Pan, Dinesh Manocha, and Ruigang Yang. CrowdMove: Autonomous Mapless Navigation in Crowded Scenarios. *arXiv e-prints*, page arXiv:1807.07870, Jul 2018.

[Fan *et al.*, 2018b] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Fully Distributed Multi-Robot Collision Avoidance via Deep Reinforcement Learning for Safe and Efficient Navigation in Complex Scenarios. *arXiv e-prints*, page arXiv:1808.03841, Aug 2018.

[Fox *et al.*, 1997] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, March 1997.

[Helbing and Molnar, 1995] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[Kim *et al.*, 2018] Y. Kim, J. Jang, and S. Yun. End-to-end deep learning for autonomous navigation of mobile robot. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, Jan 2018.

[Long *et al.*, 2017] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1709.10082, Sep 2017.

[Narain *et al.*, 2009] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. *ACM Trans. Graph.*, 28(5):1–8, December 2009.

[Narang *et al.*, 2015] Sahil Narang, Andrew Best, Sean Curtis, and Dinesh Manocha. Generating pedestrian trajectories consistent with the fundamental diagram based on physiological and psychological factors. *PLOS ONE*, 10:e0117856, 04 2015.

[Neunert *et al.*, 2016] M. Neunert, T. Boaventura, and J. Buchli. *Why Off-The-Shelf Physics Simulators Fail In Evaluating Feedback Controller Performance - A Case Study For Quadrupedal Robots: Proceedings of the 19th International Conference on CLAWAR 2016*, pages 464–472. 10 2016.

[Pelechano *et al.*, 2007] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, page 99–108, Goslar, DEU, 2007. Eurographics Association.

[Pfeiffer *et al.*, 2016] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots. *arXiv e-prints*, page arXiv:1609.07910, Sep 2016.

[Pfeiffer *et al.*, 2017] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1527–1533, May 2017.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv e-prints*, page arXiv:1707.06347, Jul 2017.

[Tai *et al.*, 2018] L. Tai, J. Zhang, M. Liu, and W. Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *ICRA*, pages 1111–1117, May 2018.

[Tilove, 1990] R. B. Tilove. Local obstacle avoidance for mobile robots based on the method of artificial potentials. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 566–571 vol.1, May 1990.

[van den Berg *et al.*, 2009] Jur P. van den Berg, Stephen J. Guy, M. Chiao Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *ISRR*, 2009.

[Xie *et al.*, 2017] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1706.09829, Jun 2017.

[Yu *et al.*, 2019] Wenhao Yu, Visak C. V. Kumar, Greg Turk, and C. Karen Liu. Sim-to-real transfer for biped locomotion. *ArXiv*, abs/1903.01390, 2019.