2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

# MuRoSim – A Fast and Efficient Multi-Robot Simulation for Learning-based Navigation

Christian Jestel, Karol Rösner, Niklas Dietz, Nicolas Bach, Julian Eßer, Jan Finke, and Oliver Urbann.

***Abstract*— Multi-robot navigation and dynamic obstacle avoidance are challenging problems in robot learning. Recent advancements in Deep Reinforcement Learning (DRL) have demonstrated great potential in this area. Nonetheless, they often face challenges related to low sample efficiency. To overcome this challenge, some research proposes simulators that incorporate hardware acceleration. Although these simulators improve efficiency, they often lack the flexibility to generate diverse learning scenarios as often needed in multi-robot scenarios, where the different environments have varying numbers of agents.**

**In this paper, we introduce MuRoSim, a multi-robot simulation for lidar-based navigation specifically designed for DRL applications. Due to its high level of abstraction, complete implementation in C++, and rigorous thread pool utilization, MuRoSim achieves high computational performance. We apply MuRoSim for training navigation policies for omnidirectional mobile robots equipped with lidar sensors using DRL. Finally, we conduct extensive Sim-to-Real experiments to confirm the realism of the simulator, by deploying the learned policy for dynamic navigation with up to six robots in numerous of real-world experiments.**

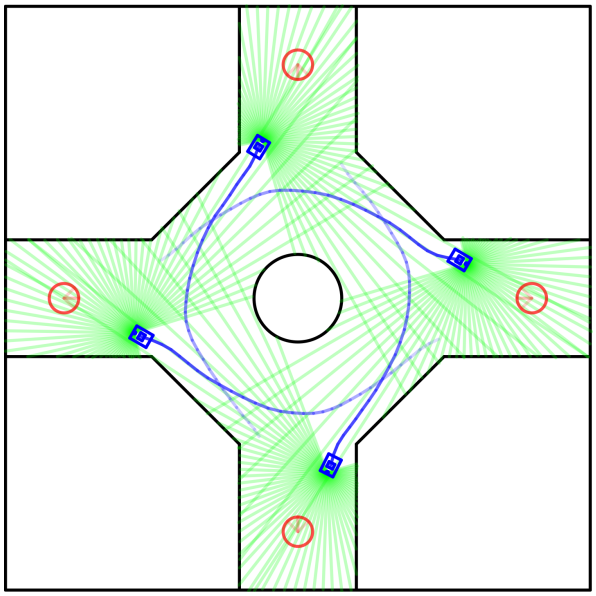

Fig. 1: Visualization of the 2D simulation MuRoSim.

## I. Introduction

In a warehouse filled with dynamic obstacles, it becomes crucial for a robot to possess the ability to navigate freely. The question arises: how can a robot achieve such autonomous navigation? With the ever-changing environment and numerous obstacles, the robot must possess advanced perception capabilities and intelligent decision-making. By constantly sensing its surroundings and analyzing the dynamic obstacles, the robot should adapt its navigation strategies in time, ensuring efficient and collision-free movement throughout the warehouse.

Recently, Deep Reinforcement Learning (DRL) has shown significant promise in the field of navigation [10], [19]. However, it is important to note that DRL can be sample-inefficient, requiring large amounts of data for training. To address these challenges, several simulators have been developed, some utilizing acceleration hardware to improve efficiency [7], [17], though, simulating a high number of parallel environments including perception-based sensors is still computationally expensive and currently available simulators have limitations in effectively representing multi-robot scenarios with varying numbers of actors in each environment.

An alternative approach to using these simulators involves abstracting the problem to an 2D space and instead of using complex physics engines, only representing the problem by its kinematic properties. In order to further leverage the 2D space, this approach can also be applied to the collision model. Calculating collisions with abstract shapes, such as lines and circles, rather than using an occupancy map, significantly reduces the computational complexity. It also allows high collision model parallelization. In summary, this simplifies the learning problem and allows for accelerated convergence during the training process.

This paper introduces a 2D simulator, Multi-Robot Simulation (MuRoSim) depicted in Fig. 1, that embodies all the previously mentioned principles. Our work involves designing an efficient simulation by simplifying the complexity in 2D space and employing a thread pool for the concurrent processing of multiple environments with varying numbers of mobile robots. Additionally, we implement a fast collision detection model by utilizing Single-Instruction-Multiple-Data (SIMD) intrinsics for optimization. We have also demonstrated the feasibility of our simulator, MuRoSim, by training a lidar-based navigation policy for an omnidirec-

This research has been funded by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence, Dortmund, Germany and the Ministry of Economic Affairs, Industry, Climate Action and Energy of the State of North Rhine-Westphalia, Germany through the 5G NRW research competition.

All authors are with the department of AI and Autonomous Systems at the Fraunhofer Institute for Material Flow and Logistics (IML), Dortmund, Germany. Corresponding author: christian.jestel@iml.fraunhofer.de

tional robot. This policy has been successfully applied in real-world scenarios, allowing the robot to navigate through challenging environments. These achievements underscore the effectiveness and practicality of our approach. In summary, the contributions of this paper are:

1) MuRoSim, a fast and efficient 2D simulation for lidar-based navigation, specifically designed for multi-robot DRL. MuRoSim will be released as open-source under `https://github.com/iml130/MuRoSim`.
2) An application of MuRoSim for training navigation policies for an omnidirectional mobile robot equipped with lidar sensors based on DRL methods.
3) Extensive Sim-to-Real experiments with up to six robots to evaluate the effectiveness of navigation policies trained in MuRoSim for various driving scenarios.

The paper is structured as follows: Section II provides an overview of related work and Section III introduces MuRoSim. Following up, Section IV outlines the process of training the DRL policy, while Section V discusses the extensive Sim-to-Real experiments. Lastly, Section VI concludes with an outlook on future research aspects.

## II. Related Work

In this section, we provide an overview of different simulators used in the context of DRL as well as perception-based and multi-robot navigation. A survey on DRL for navigation of various mobile robot platforms is provided by [25].

### A. Simulators for DRL

Simulation-based training is an important part of DRL as it allows for accelerated training and enables agents to learn complex tasks through trial and error without causing damage to a real system during training [5]. Gazebo [13] is a very popular 3D simulator that is frequently used in conjunction with ROS [16]. Zamora et al. [24] have developed a gym-like environment for Gazebo ROS, specifically designed for navigation tasks using a turtlebot. It is worth mentioning MuJoCo [23], which is commonly used for manipulation and locomotion tasks. For urban driving and navigation, the 3D simulator CARLA [3] is highly popular. Another approach is to leverage GPU for highly parallel simulations, as demonstrated by ISAAC Gym [17] and Brax [7], which enables GPU-to-GPU DRL training. In addition to the 3D simulators mentioned earlier, it is worth noting that Stage [2] is a 2D simulator specifically designed for mobile robot navigation. In comparison, our approach directly targets lidar-based navigation for DRL and eliminates the need for a complex physics engine. We optimize the computationally intensive collision calculations and provide an easy-to-use integration with deep learning frameworks.

### B. Obstacle Avoidance with DRL

Vision-based obstacle avoidance using DRL has made significant advancements. Quadruped robots have demonstrated the ability to navigate various terrains successfully [10], [11], [19]. In the drone domain, fast and agile maneuvers are crucial for safe navigation as collisions can be fatal. Recent studies [15], [18] have shown that their approaches are effective in densely occupied obstacle scenarios, such as flying through forests. Other works, such as [8], [9], [22] deploys vision-based obstacle avoidance and navigation algorithms for wheeled robots. In contrast to these lines of work, our DRL experiments illustrate the capability of MuRoSim and specifically targets lidar-based obstacle avoidance for omnidirectional robots. These robots offer greater flexibility in movement, which also increases the potential for collisions. Therefore, we aim to deploy effective robot modeling and DRL techniques, allowing these robots to move flexibly without increasing the risk of collisions.

### C. Multi-Robot Navigation with DRL

In the multi-robot navigation setting, coordination among multiple robots is necessary to resolve conflicts in narrow areas and intersections. This coordination can be achieved either through a central system [20] or through decentralized methods where the robots organize themselves. In some cases, a central motion capture system can be used for fast pose estimation when employing decentralized navigation systems [1]. Other approaches rely on communication between robots. In the approach of [14] robots can operate in groups as long as they maintain their communication radius [14]. The work closest to ours is presented by [6]. Their approach involves combining classic control methods with DRL techniques in a hybrid control system for collision avoidance. In comparison, we are using a pure end-to-end DRL approach on an omnidirectional-driving mobile robotic platform and train the policy on a diverse set of environments.

## III. MuRoSim

In this section, we introduce the multi-robot simulation MuRoSim and provide an overview of its underlying software architecture, adopted optimization techniques, and differences to other simulators.

### A. Motivation

The development of MuRoSim is motivated by the need of a fast, efficient, and easy-to-use simulation for navigation via DRL. Robotic simulations like Gazebo [13] use client-server architecture for interaction, which makes accessibility and synchronization more challenging. ISAAC Gym [17] is designed for highly parallel DRL for robotics by utilizing the GPU. However, while ISAAC Gym requires a powerful GPU, MuRoSim can also be used on resource-constrained embedded systems like an NVIDIA Jetson Board, allowing the GPU memory to be fully utilized for training.

### B. Software Architecture

Building upon previous work [12], MuRoSim is a 2D simulation that uses shapes such as lines and circles for the structure of the world and the robots. The core of MuRoSim is implemented in C++, and the accessibility to the deep learning frameworks is achieved through Python bindings.

One MuRoSim simulation instance consists of a world that defines the static structure of the environment (see Fig. 2).

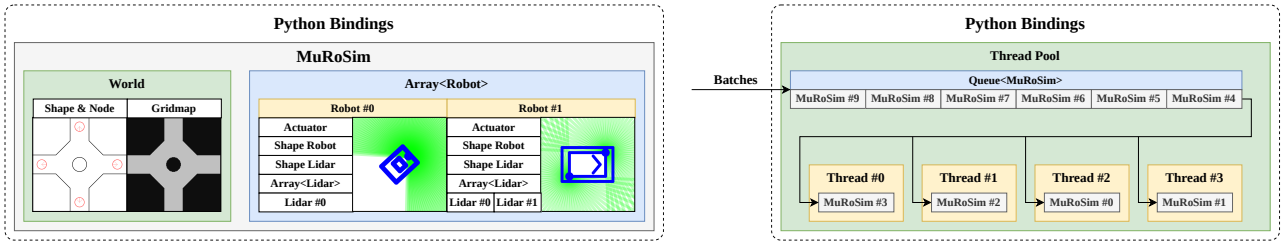


Fig. 2: Visualization of the MuRoSim software architecture: On the left side is the structure of a MuRoSim simulation instance, which includes the world with two different robots. On the right side is the thread pool that distributes the MuRoSim instances across the threads by retrieving them from a queue.

Each world contains a set of predefined nodes represented as circles, which serve as potential starting points and goals for the robots. The nodes are also used to generate a grid map of the world that describes the free space, using a flood-fill algorithm that originates from the nodes' locations. The free space is utilized to generate random nodes during the training phase to enhance generalization. Thus, the grid map is not employed for collision detection purposes.

Furthermore, a MuRoSim simulation instance can include multiple types of robots. The actuator class determines the movement of each robot and encapsulates the physical properties of the actual robot, such as differential or omni-directional drive. Additionally, each robot is designed with a two-layered shape: the overall robot shape as the collision body, and the lidar shape for lidar collision detection. This design choice was made because lidar sensors often fail to detect the entire shape of the robot owing to their placement. Finally, a robot can be outfitted with various lidars, each with different properties.

### C. MuRoSim's Performance

One of the computationally intensive tasks in simulations is collision detection between objects, such as the laser rays of the lidar and other entities. The collision detection is a brute-force method that can be efficiently parallelized by using SIMD instructions and adopting the Structure of Arrays (SoA) memory layout, eliminating the need for data shuffling. For the x86 architecture, the Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) instruction sets are implemented, while the NEON instruction set is implemented for the ARM architecture. This makes the use of MuRoSim particularly interesting for DRL on embedded systems such as the NVIDIA Jetson series. We also provide a thread pool that takes a batch of MuRoSim simulation instances and distributes them via a queue to multiple threads to execute the step and reset function of the environments simultaneously (see Fig. 2). The thread pool is implemented in C++ and, being accessible via Python bindings, is therefore independent of the Python Global Interpreter Lock (GIL). We evaluated the optimization on an Intel Core i9-12900H and an NVIDIA Jetson Xavier NX with a set of three different world sizes, each multiplied by 500 (see Table I). Overall, we found that utilizing these SIMD instructions, in comparison to the scalar method, leads to a speedup factor of up to 5x in the overall simulation steps per second, depending on the CPU architecture.

TABLE I: Simulation steps per second for a set of three different world sizes with SIMD optimization and thread pool utilization for two different hardware platforms.

| Intel Core i9-12900H | | | | NVIDIA Jetson Xavier NX | | |
|---|---|---|---|---|---|---|
| **Threads** | **Scalar** | **SSE** | **AVX** | **Threads** | **Scalar** | **NEON** |
| 1 | 207.2 | 707.2 | 1023.4 | 1 | 45.0 | 217.2 |
| 2 | 384.1 | 1334.8 | 1887.8 | 2 | 90.2 | 437.5 |
| 4 | 711.7 | 2453.3 | 3502.2 | 4 | 183.6 | 861.2 |
| 8 | 1109.9 | 3935.8 | 5416.5 | 5 | 216.8 | 1039.6 |
| 16 | 1351.0 | 5176.0 | 7029.4 | 6 | 250.9 | 1154.9 |

## IV. Deep Reinforcement Learning

In this section, we present an application of MuRoSim for learning-based multi-robot navigation and provide details about the learning environment, the neural network architecture, and the overall training setup.

### A. Learning Environment

The multi-robot navigation problem involves navigating omnidirectionally driven robots in a planar environment towards a goal position. This problem can be formulated as a partially observable Markov decision process (POMDP). In this context, each environment consists of $N$ robots, where each robot is treated as an independent decision-making agent with no knowledge of the states of other robots.

The environment is modeled as a world in MuRoSim, where the DJI RoboMaster's collision shape is a rectangle. For the lidar collision, the lidar itself is modeled as a circle, and the computing unit housing is represented by a smaller rectangle. An episode starts with a random starting pose for the agent and a random goal. It ends immediately for an agent if a collision occurs or if the goal is reached. Additionally, an episode ends for all agents if the maximum number of steps per episode is reached.

An agent is considered to have reached the goal when it is within the goal's radius and remains there for a specified time period. This requirement is intended to encourage the agent to decelerate upon approaching the goal, preventing the potential for wheel slippage that could cause a physical robot to overshoot the goal after Sim-to-Real transfer.

TABLE II: Hyperparameters for the reward scaling, the learning algorithm, and MuRoSim's learning environment.

| **MuRoSim** | | **Reward** | | | | **Proximal Policy Optimization (PPO)** | |
|---|---|---|---|---|---|---|---|
| Steps per episode | 256 | $\omega_{\text{delta\_pos}}$ | 0.05 | $\omega_{\text{laser\_neg}}$ | -0.003 | Horizon | 256 |
| Frequency | 10 Hz | $\omega_{\text{delta\_neg}}$ | -0.001 | $\omega_{\text{laser\_threshold}}$ | 0.5 | Mini-batch size | 4096 |
| Goal radius | 0.25 m | $\omega_{\text{action\_rate}}$ | -0.005 | $\omega_{\text{reached\_goal}}$ | 3.0 | Clip range | 0.2 |
| Time on goal | 1.0 s | $\omega_{\text{on\_goal}}$ | 0.2 | $\omega_{\text{collision\_world}}$ | -1.5 | Discount factor $\gamma$ | 0.99 |
| World bias scale | [1.0,1.5] | $\omega_{\text{forward\_driving}}$ | 0.02 | $\omega_{\text{collision\_robot}}$ | -2.0 | GAE discount factor $\lambda$ | 0.95 |
| | | | | | | Learning rate $\alpha$ | 0.0003 |

### B. Observation Space

The observations of the agent are split into perceptual and positional observations. A perceptual observation, denoted as $o_{\text{per}}$, is obtained from a single laser scan of the lidar sensor. Each laser scan includes 1200 distance measurements with a 270-degree field of view. Furthermore, the information about the target goal is relevant for navigation, which is defined as positional observation $o_{\text{pos}}$ and consists of the following components:

- *orientation to the goal ($o_g$)*: It is described as a unit vector pointing to the goal expressed in the robot's coordinate system.
- *distance to the goal ($o_d$)*: This represents the Euclidean distance between the robot's pose and the goal's pose.
- *velocities ($o_v$)*: These include the robot's linear x velocity, linear y velocity, and angular velocity.

To enable the agent to perceive its own motion as well as the motion of other agents, we utilize frame stacking by considering the three most recent observations. Furthermore the distance-based observations ($o_{\text{d}}$, $o_{\text{per}}$) are normalized by the maximum range of the lidar sensor, which is limited to 30 m.

### C. Action Space

The agent's action space is continuous and consists of linear x, linear y, and angular command velocities. Despite the lidar sensor's perceptual blind spot, we have restricted the robot's ability to drive backward by setting the range for the linear x command velocity to [-0.5, 2.5] m/s. The small restriction for negative linear x velocity is intentional to allow the robot to maneuver linearly while rotating. The linear y command velocity range is set to [-1.0, 1.0] m/s, and the angular command velocity range is set to [-2.0, 2.0] rad/s. The agent's actions are sampled from a Gaussian distribution with a variable standard deviation, using the mean as the sampled value.

### D. Reward Formulation

The reward function is applied to each robot independently at every time step. Each term is scaled linearly depending on the severity of the action, e.g. the faster the agent approaches the goal, the more reward it gets. The final reward is a weighted sum of the following components (see Table II for scaling details):

- *delta distance to goal*: The agent receives a positive reward $r_{\text{delta\_pos}}$ when the delta of the Euclidean distance between the robot pose and the goal pose gets smaller; otherwise, the reward is negative $r_{\text{delta\_neg}}$.
- *action rate*: This limits the agent from making large changes in actions. The agent receives a penalty $r_{\text{action\_rate}}$ when the selected action significantly differs from the subsequent possible velocities.
- *on goal*: The agent receives a positive reward when the agent is within the goal's radius $r_{\text{on\_goal}}$.
- *forward driving*: The agent is rewarded with $r_{\text{forward\_driving}}$ when it maintains a high linear x velocity and a low linear y velocity, discouraging diagonal driving for increased speed.
- *minimum laser distance*: The agent receives a penalty $r_{\text{laser\_neg}}$, when the shortest measured laser distance is under a given threshold $r_{\text{laser\_threshold}}$.
- *episode ending*: Upon successfully staying within the goal radius until the time expires, the agent is awarded a positive sparse reward $r_{\text{reached\_goal}}$. Additionally, the agent incurs a penalty for collisions with the environment $r_{\text{collision\_world}}$ or with another robot $r_{\text{collision\_robot}}$.

### E. Neural Network Architecture

As shown in Fig. 3, the neural network uses two inputs, one is the perceptual observation $o_{\text{per}}$ and the other is the positional observation $o_{\text{pos}}$. The perceptual observation is passed through a 1D Min Pooling layer with a kernel size of 5, two IMPALA Blocks, and one MLP layer with 256 units. On the other hand, the positional observation is connected to one MLP layer with 96 units. One IMPALA block consists of one convolutional layer followed by a 1D Max Pooling layer and two residual blocks. It has a similar structure to the one shown in [4]. The convolution layer in the IMPALA block has the following configuration (from the first to the second layer): filters $[24,16]$, kernel size $[7,5]$, stride $[4,3]$, and padding $[0,0]$. After that, both observation outputs are concatenated and passed to the last MLP layer with 256 neurons. The rectified linear unit is used as the

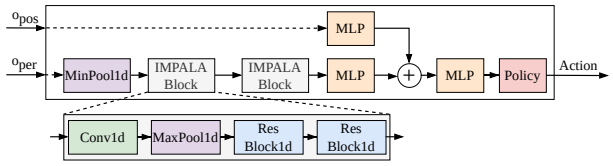


Fig. 3: Neural network architecture with the perceptual ($o_{\text{per}}$) and positional ($o_{\text{pos}}$) observation as input and the action as output. Each each color represents a different layer type.

activation function for all layers. For our approach we decided to use the gradient policy actor-critic algorithm, Proximal Policy Optimization (PPO) [21]. Furthermore, we use separated networks for the actor and the critic. The actor's action output is the three command speeds with the tanh as the activation function, and the critic has a single value output with no activation function.

### F. Policy Training

For the training, we choose a set of 37 different worlds, a small selection is shown in Fig. 4. Each world is duplicated five times with an additional bias scale as domain randomization. Overall 185 worlds with 790 robots are used for the training. During the training, the data of the agents are collected into a buffer until the episode of an agent ends or the horizon is reached. The training data is then shuffled into mini-batches and the policy is updated via PPO. As there are multiple agents in a single environment, the batch varies per PPO cycle. During the optimization process, the generated training data by each agent is used to train a single policy that is distributed to all agents. For every millionth update, the policy is evaluated in the training world set without Gaussian distribution of the action. After 220 million update steps, the policy reached a success rate of over 99%, where the success rate defines that the agent reached the goal in one episode. The hyperparameters for the training are listed in Table II.

## V. Real-World Experiments

In this section, we present the Sim-to-Real transfer of the trained navigation policy from MuRoSim. We begin by introducing the experimental setup, followed by various multi-robot and special case experiments.

### A. Experimental Setup

Our target platform is an omnidirectional-driving robot with high accelerations, and thus we chose the DJI RoboMaster EP (see Fig. 5). It is equipped with Mecanum wheels and can achieve a linear velocity of over 5 m/s when driving forward. To deploy our algorithms on the system, we connected an Nvidia Jetson NX Xavier directly to the motor controller using the CAN-Bus protocol. For perception, we used an RPLIDAR S2 with a 270-degree field of view, operating at 10 Hz and with a maximum range of 30 m. Additionally, the DJI RoboMaster is equipped with an Intel RealSense D455, although it was not utilized in the experiments.

For the real-world experiments, the velocity information is obtained from the robot's odometry, while a Vicon motion-capturing system is utilized to estimate the robot's pose.

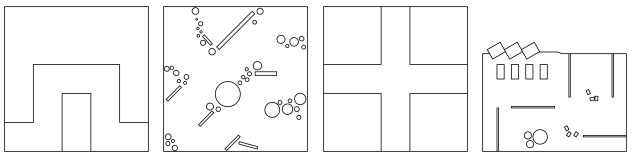

Fig. 4: Examples of various worlds present in MuRoSim as part of the training set for learning multi-robot navigation.

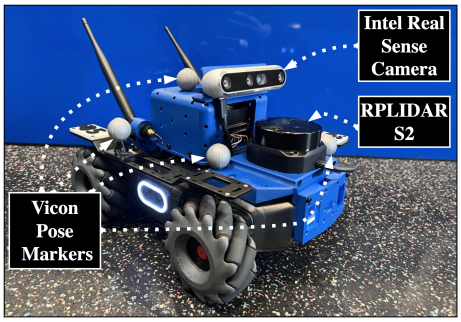


Fig. 5: The extended DJI RoboMaster experimental platform.

TABLE III: Multi-robot performance in terms of success rate and number of collisions with static and dynamic obstacles.

| Szenario | Number of Robots | Number of Collision | Success Rate |
|---|---|---|---|
| Swap | 4 | 2 | 90 % |
| Intersection | 4 | 1 | 95 % |
| Random | 6 | 0 | 100 % |

This motion-capturing system eliminates the localization problem and is beneficial in multi-robot scenarios. We access all hardware components using the Robot Operating System 2 (ROS2) framework [16]. The navigation policy's inference model is executed with TensorRT and is triggered after the completion of each laser scan. However, despite using the Vicon system for the experiments, the trained navigation policy is capable of guiding the robot to a relative goal using only odometry.

In previous experiments, navigation policies were successfully trained using MuRoSim for various robot platforms, including the industrial mobile robot ShuttleMove from Fraunhofer IML, the RB-1 BASE from Robotnik, and a swarm of TurtleBot 2s. In the present work, the focus is specifically on robots with omnidirectional drive systems, in contrast to the differentially driven platforms mentioned above.

### B. Multi-Robot Experiments

We evaluate the trained navigation policy in three real-world multi-robot scenarios: position swap, narrow intersections, and randomly distributed obstacles (see Fig. 6). In the position swap scenario, two robots navigate through a narrow corridor to exchange positions. In the narrow intersection scenario, each robot is assigned a goal on the opposite side of the crossing. Finally, in the scenario with randomly distributed obstacles, various objects are placed in the environment, with the robots' goals consistently located near the periphery.

Each scenario was evaluated a total of five times, and the results are presented in Table III. Unlike in MuRoSim, where an episode terminates immediately after a collision occurs, in these real-world experiments, the robots are able to continue navigating. In all three scenarios, the policy successfully guided the robots to their goals; however, some

Fig. 6: Multi-robot experiments (from left to right): position swap, narrow intersection, and randomly distributed obstacles.

Fig. 7: Special case experiments (from left to right): bottleneck transition, dynamic obstacle avoidance, human interaction, and dead-end recovery.

collisions were recorded. Collisions are not counted as successful episodes. Specifically, in the position swap scenario, two robots collided with the walls while trying to avoid an oncoming robot. In the intersection scenario, one robot displaced the other, resulting in a collision at the border. In the randomly distributed obstacles scenario, no collisions were reported due to the ample space available.

### C. Special Case Experiments

We also evaluate the behavior of the navigation policy across four unique scenarios: bottleneck transition, dynamic obstacle avoidance, human interaction, and dead-end recovery (see Fig. 7).

In the bottleneck scenario, the behavior of two robots needing to swap positions through a narrow bottleneck is analyzed. Observations indicate that one robot proceeds directly through the bottleneck, while the other temporarily yields and gets displaced. Subsequently, sufficient space becomes available for both robots to pass each other and proceed through the bottleneck.

Next, the evasion capabilities of the navigation policy are tested in a corridor scenario where the robot must dodge rolling balls as it passes through. In one trial, the robot successfully avoids all rolling balls provided they are not moving too quickly. The policy employs a combination of linear y velocity and angular velocity for lateral evasion. However, due to the forward driving reward function, the use of linear y velocity is minimized.

In analyzing the policy's interaction with humans in a corridor, we observed that when humans briefly attempt to block the robot's path, the policy adapts by altering its course to navigate around the person. However, it was found that there exists a threshold gap width below which the policy struggles to proceed, even when sufficient space is ostensibly available. This phenomenon can be explained by the penalization associated with the minimum lidar distance reward.

Lastly, the navigation policy's ability to recover in dead-end situations is evaluated. In this scenario, a robot begins at a fork, tasked with reaching a goal located on the right side, which is a dead end. The robot initially chooses to drive towards the dead end but is able to correct its course. Concurrently, another robot coming from the left side forces the first robot further into the dead end. Despite this, the first robot successfully recovers and navigates to the left, ultimately reaching its goal. The navigation policy demonstrates the ability to recover from smaller dead ends within the limited 0.3 s observation period.

## VI. Conclusions And Future Work

In this paper, we present MuRoSim, a fast and efficient 2D simulation designed for lidar-based multi-robot navigation. The simulation in combination with off-the-shelf DRL methods is applicable to a variety of robotic systems. To show its effectiveness, we have implemented a DRL pipeline and integrated it with MuRoSim for the omnidirectional-driven DJI Robomaster EP. Through various real-world experiments, we have demonstrated the zero-shot Sim-to-Real transfer of the trained navigation policy, proving the realism of the MuRoSim simulator for DRL applications.

As part of future work, we are planning to further improve the performance and realism of the simulator, e.g., by incorporating more realistic lidar models that also respect the corresponding motions of the robots. Moreover, we are planning to apply MuRoSim to even more complex robotic tasks. For instance, by also incorporating inter-robot communication in the training task, the robots could share for instance intentional movements for learning cooperative behaviors.

## REFERENCES


[1] Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav S. Sukhatme. Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 576–586. PMLR, 08–11 Nov 2022.

[2] Toby HJ Collett, Bruce A MacDonald, and Brian P Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian conference on robotics and automation (ACRA 2005)*, page 145. Citeseer Citeseer, 2005.

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[4] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[5] Julian Eßer, Nicolas Bach, Christian Jestel, Oliver Urbann, and Sören Kerner. Guided reinforcement learning: A review and evaluation for efficient and effective real-world robotics. *IEEE Robotics Automation Magazine*, pages 2–22, 2022.

[6] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, 39(7):856–892, 2020.

[7] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021.

[8] Lingping Gao, Jianchuan Ding, Wenxi Liu, Haiyin Piao, Yuxin Wang, Xin Yang, and Baocai Yin. A vision-based irregular obstacle avoidance framework via deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9262–9269. IEEE, 2021.

[9] Noriaki Hirose, Dhruv Shah, Ajay Sridhar, and Sergey Levine. Exaug: Robot-conditioned navigation policies via geometric experience augmentation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4077–4084. IEEE, 2023.

[10] David Hoeller, Lorenz Wellhausen, Farbod Farshidian, and Marco Hutter. Learning a state representation and navigation in cluttered and dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):5081–5088, 2021.

[11] Chieko Sarah Imai, Minghao Zhang, Yuchen Zhang, Marcin Kierebiński, Ruihan Yang, Yuzhe Qin, and Xiaolong Wang. Vision-guided quadrupedal locomotion in the wild with multi-modal delay randomization. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5556–5563, 2022.

[12] Christian Jestel, Hartmut Surmann, Jonas Stenzel, Oliver Urbann, and Marius Brehler. Obtaining robust control and navigation policies for multi-robot navigation via deep reinforcement learning. In *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, pages 48–54, 2021.

[13] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[14] Juntong Lin, Xuyun Yang, Peiwei Zheng, and Hui Cheng. Connectivity guaranteed multi-robot navigation via deep reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 661–670. PMLR, 30 Oct–01 Nov 2020.

[15] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[16] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[17] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[18] Robert Penicka, Yunlong Song, Elia Kaufmann, and Davide Scaramuzza. Learning minimum-time flight in cluttered environments. *IEEE Robotics and Automation Letters*, 7(3):7209–7216, 2022.

[19] Nikita Rudin, David Hoeller, Marko Bjelonic, and Marco Hutter. Advanced skills by learning locomotion and local navigation end-to-end. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2497–2503, 2022.

[20] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. GNM: A General Navigation Model to Drive Any Robot. In *International Conference on Robotics and Automation (ICRA)*, 2023.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[24] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.

[25] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.