3D GridWorld for Multi-Robot Systems: Integrating SLAM and Navigation in NVIDIA Isaac Sim

Kyler Witvoet
Computer Engineering, McMaster University
Hamilton ON, Canada
witvoetk@mcmaster.ca

***Abstract*—Multi-robot collaboration is essential for autonomous systems to work together efficiently. Reinforcement learning (RL) is increasingly being used for coordination and decision-making in multi-robot systems. However, many training and testing methods rely on simplistic 2D simulation environments, which, while easy to use, don't capture conditions adequately enough to effectively transfer the trained policy from simulation to the real world. To overcome these limitations, we propose a 3D simulation environment built with NVIDIA Isaac Sim, that integrates multi-robot simultaneous localization and mapping (SLAM), and parallel navigation nodes via robotic operating systems' SLAM Toolbox and Navigation2 (Nav2) packages. Our framework is scalable for multiple robots and reduces the need for custom-built simulation environments, allowing researchers to focus on refining and evaluating multi-robot control strategies.**
The source code of our 3D GridWorld can be found at: https://github.com/kylrw/3D-Gridworld-Isaac-Sim.

***Index Terms*—Multi-Robot Systems, 3D Simulation, Isaac Sim, SLAM**

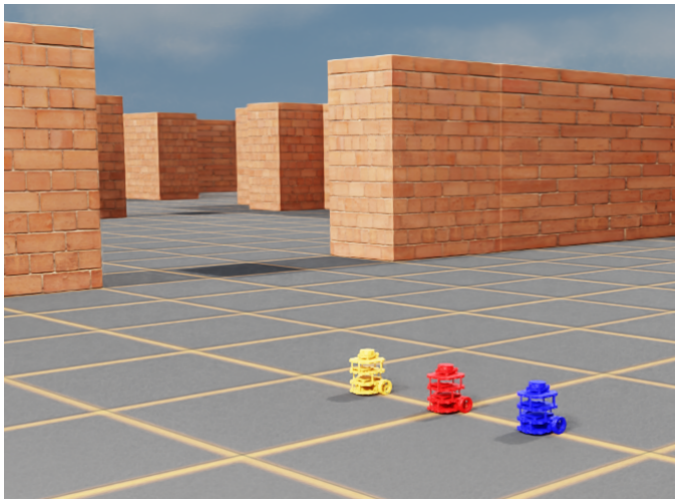

Fig. 1. Turtle Bots in Isaac Sim

I. Introduction

As autonomous robots become more advanced, collaboration between multiple robots becomes increasingly necessary to accomplish complex tasks. While reinforcement learning (RL) is largely used for robotic manipulation and locomotion, recent studies demonstrate its effectiveness in coordinating multi-robot systems [1].

RL is well-suited for robotic control due to its adaptability to various problem spaces. It enables an agent (a robot) to learn optimal decision-making through trial-and-error interactions with its environment by receiving feedback in the form of mathematical rewards. The data outlining the optimal decision-making behaviour is called a "policy". Training RL policies with the end goal of deploying them on a physical robot requires modern state-of-the-art simulators such as Gazebo [2], MuJoCo [3], or NVIDIA Isaac Sim [4]. These simulators provide environments with realistic physics and high-fidelity visuals, which enables the virtual training and testing of systems in real-world conditions. This allows engineers to train and test RL policies for robotic systems much easier and faster than they would in the real world while still operating in realistic conditions.

When training policies for multi-robot systems, high-fidelity physics simulations are often unnecessary, as the learned policies normally operate at a higher level of abstraction, such as coordination, task allocation, or path planning, rather than low-level motor control or precise robotic dynamics [5]. Simplified environments, such as 2D grid worlds, are commonly used for training these high-level decision-making policies [6]. In a 2D grid world environment, there is no need to worry about implementations of high-fidelity physics simulation, or low-level robotics control and perception algorithms, as the environment is simple enough that these features can be abstracted by simple path-planning algorithms and 2D windows of the grid world. However, when deploying multi-robot systems in the real world, implementing low-level control and perception systems with the higher-level policy, as well as accurate physics simulation, is essential to closing the gap between simulation and reality.

This paper presents a framework for testing multi-robot systems in a 3D simulation environment built in NVIDIA Isaac Sim, integrating multi-robot simultaneous localization and mapping (SLAM), path planning navigation, and low-level robotic control. Our framework for creating the 3D environment was tested on three TurtleBot robots navigating a 3D scene inspired by the 2D Minigrid in [6]. An image of the three TurtleBots in their initial positions is in [Fig 1]. The system is designed to allow for easy adaptation from the 2D Minigrid environment and enables testing policy performance on different robotic platforms, teams, and environments.

The remainder of this paper is structured as follows: Section II reviews related works; Section III describes the theoretical foundations of SLAM, multi-robot SLAM, and navigation modules; Section IV details the system architecture and integration with Cooperative and Asynchronous Transformer-based Mission Planning (CATMiP); and Section V presents our conclusions and future work.

II. Related Works

In recent years, multi-robot systems have become a focal point of research due to their capacity to perform complex, collaborative tasks that would be challenging or impossible for single robots. Heterogeneous robot teams - teams composed of robots with different sensing, actuation, and computational capabilities - have shown superior efficiency and adaptability in various applications, including search and rescue, environmental monitoring, and industrial inspections [1].

RL, especially Multi-Agent Reinforcement Learning (MARL), has emerged as an effective approach for developing autonomous coordination strategies. [1] introduces the CATMiP framework which utilizes MARL to enable decentralized decision-making for heterogeneous robot teams. CATMiP provides asynchronous decision-making through macro-actions, allowing for efficient collaboration even in communication-constrained environments.

Similarly, [7] proposed a cooperative multi-robot exploration method using deep RL designed explicitly for communication-constrained scenarios. They employed an improved soft actor-critic algorithm to manage collaborative exploration and efficient task execution, highlighting MARL's adaptability to real-world constraints.

For multi-robot systems, the decision space of a high-level planning policy is less complex, as actions consist of providing robots with navigation goals and tasks to complete, as opposed to complex manipulation or locomotion. This means that MARL algorithms for robot teams do not necessarily require accurate real-world conditions and physics during training and can be trained in simple, abstract environments reducing the computational complexity and cost. To this end [6] introduced Minigrid and Miniworld, two abstract RL environments providing customizable, goal-oriented tasks. In these environments the "world" is represented by a grid of squares and robots, obstacles, and goals are represented as points within the grid. These platforms enable rapid experimentation and custom environment creation. However, effective RL training for *individual* robotic control requires robust simulation environments capable of accurately simulating real-world conditions and physics, which the abstract environments fall short of capturing. When MARL policies are combined with a team of robots trained using RL, a more accurate simulation environment is required.

To bridge the gap between simulation and complex real-world interactions, researchers have turned to more realistic simulators like NVIDIA's Isaac Sim. [4] highlights Isaac Sim's high-fidelity physics simulation and photorealistic environments, which position it as a leading tool for training and testing robotic systems, including multi-robot systems. NVIDIA Isaac Sim notably provides the environment for developing and testing RL policies effectively before transferring to real-world scenarios.

The capacity of NVIDIA Isaac Sim to simulate high-fidelity robotics environments has been extensively validated in the recent literature. For example, [8]–[10] demonstrated a successful sim-to-real transfer in mobile robots, validating navigation and obstacle avoidance policies trained in Isaac Sim. The relevance of Isaac Sim extends to MARL scenarios, where its support for scalable simulations and robust integration with robotic operating systems (ROS) [11] is critical. It provides realistic sensory data streams, accurate dynamics modelling, and seamless ROS integration, allowing researchers to conduct experiments closely aligned with real-world implementations.

In addition to an accurate simulation environment, algorithms for individual robotic perception, planning, and control are necessary for multi-robot systems. Implementing these algorithms requires minimal changes between a single-robot system and a multi-robot system. However, SLAM, which is essential for autonomous robot navigation especially in unknown environments, introduces additional challenges and potential benefits when considered for multi-robot systems. Collaborative SLAM enables teams of robots to share spatial information, effectively allowing a single system to be in multiple places at once. Methods like Swarm-SLAM in [12] or the DCL-SLAM system outlined in [13] address the challenges associated with collaborative communication by effectively merging local occupancy grids into globally consistent maps. These methods significantly enhance robots' ability to navigate unknown environments collaboratively.

The works mentioned here reinforce the pivotal role of accurate simulation for effective policy transfer in multi-robot systems. Although abstract environments such as Minigrid and Miniworld provide foundational frameworks suitable for theoretical experimentation and initial training, NVIDIA Isaac Sim's realism and integration capabilities make it ideal for advanced testing of robot systems before real-world deployments.

III. Theory

Our system uses Isaac Sim to host the simulation environment, but relies on the second version of the Robot Operating System (ROS2), which is an open-source framework for robot software development that enables real-time communication between different components of the system, such as perception, planning, and control modules.

A. Simultaneous Localization and Mapping (SLAM)

SLAM enables robots to build a map of an unknown environment while simultaneously keeping track of their location within it. SLAM is crucial for autonomous navigation in dynamic and unknown spaces, especially for mobile robotics deployed in real-world environments.

Our system utilizes the open-source SLAM Toolbox [14], which integrates directly with ROS2 and is particularly suitable for robotic applications involving large-scale, unknown

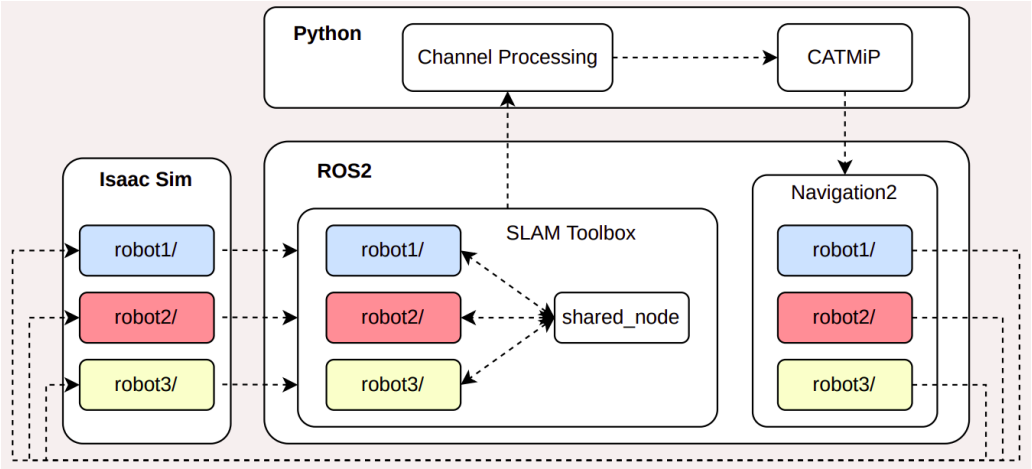

Fig. 2. High-level System Architecture

and dynamic environments. The mapping process begins by collecting sensor data from a 2D LiDAR scanner and wheel odometry (or IMU-based dead reckoning). The LiDAR generates a continuous stream of 2D point clouds representing the robot's immediate surroundings. These scans are grouped into sub maps which are small local representations of the environment. The resolution of each submap determines the level of detail captured, with higher resolutions providing finer environmental features with the cost of increased computational load; the default pixel resolution of SLAM Toolbox is 5 cm. Submaps are stitched together based on the robot's estimated trajectory, forming a global map over time. In SLAM Toolbox's visualizations of the global map, it is rendered with black for occupied space, white for free space, and grey for unknown areas.

The odometry data provides information on the robot's current trajectory; by accumulating these measurements we can create a history of the robot's locations. However, there are small errors in the odometry measurements due to noise and drift that cause pose estimation errors to accumulate over time. This means that when trying to stitch together the point cloud data, there may be errors due to incorrect positioning. This is where pose-graph optimization comes in. To align each scan with the current submap, SLAM Toolbox employs pose graph optimization and scan matching, using a nonlinear least squares optimization (via Ceres Solver from Google) to minimize the difference between the new scan points and the existing sub map [14]. When the robot revisits previously mapped areas the SLAM system identifies this and creates a new constraint between the current pose and the previous one, effectively "closing" the loop the robot travelled. These loop closures help reduce accumulated drift errors and minimize the overall error by adjusting the poses within the graph and allowing the graph optimizer to retroactively adjust all connected poses, improving the global map's consistency and accuracy.

Once a map is created, SLAM activates its localization phase, using the generated map for real-time positioning. SLAM Toolbox has this localization built in, and matches incoming LiDAR scans with the existing pose graph, providing the necessary information to determine the robot's location within the global map. Localization includes adaptive filtering, so old scan data gradually expires from a buffer, allowing the system to adapt to environmental changes, such as a door opening or two robots crossing paths, while maintaining global consistency. This approach, called elastic pose graph deformation, supports accurate positioning even in partially dynamic environments [14].

B. Multi-robot SLAM

Recent developments have extended the SLAM Toolbox framework to support decentralized multi-robot SLAM, addressing the increasing need for scalable, bandwidth-efficient mapping in multi-robot systems. One such extension, developed as part of research at the Singapore University of Technology and Design, adds collaborative SLAM capabilities while preserving the original functionalities of the core SLAM Toolbox nodes [15].

In this version of multi-robot SLAM - which is used in our system - each robot runs an individual instance of SLAM Toolbox under a unique namespace. To synchronize map data, the robots share localized laser scans via a ROS2 node. A diagram of this can be seen in [Fig 2]. This minimal data-sharing strategy dramatically reduces network bandwidth usage, which is a critical factor in real-world deployments. Robots do not exchange full laser scans, maps, or transformations. This approach performs distributed loop closure and merges position graphs across all participating robots, i.e., loop closures between robots won't occur until robots cross paths. In deployment, robots must start near each other and in similar orientations, allowing the scan matcher to detect overlap and initialize the relative transformation [15].

C. Navigation

With a map generated by SLAM or multi-robot SLAM and accurate localization, a robot can plan and execute paths

Fig. 3. Isaac Sim Map and CATMiP Channels Pre and Post Processing

through its environment using ROS2's Navigation2 (Nav2) stack. Nav2 creates a multi-layered cost map to represent the environment and integrates static obstacle data from the SLAM-generated map with real-time sensor readings (e.g., from LiDAR) to account for dynamic obstacles. The costmap includes several layers, such as the static layer, obstacle layer, and inflation layer. The static layer incorporates information from the SLAM map, while the obstacle layer updates the cost map with real-time obstacle data detected by sensors. The inflation layer adds a safety buffer around obstacles, expanding their size on the map to ensure the robot maintains a safe distance, accounting for the robot's footprint and navigation uncertainties.

Nav2 employs a two-tiered planning approach:

1) Global Planner: This planner computes an initial, collision-free path from the robot's current position to the goal, based on the global costmap. Nav2 supports various global planner plugins, which use popular path planning algorithms such as Dijkstra's Algorithm or A*.
2) Local Planner: This module generates low-level motion commands to follow the global path while dynamically responding to immediate obstacles and the robot's kinematic constraints. Nav2 provides several controller plugins, including the Dynamic Window Approach (DWB) controller and the Timed Elastic Band (TEB) controller. The DWB controller evaluates a set of possible trajectories and selects the one that optimally balances progress toward the goal, obstacle avoidance, and adherence to the robot's dynamics. The TEB controller optimizes the robot's trajectory based on execution time, distance from obstacles, and feasibility concerning the robot's size/footprint.

Each robot runs an individual instance of Nav2 on a separate thread under a unique namespace, ensuring that path planning and execution are handled independently. This architecture allows for scalable and efficient navigation in multi-robot systems, as each robot can autonomously compute and follow its path while sharing relevant information, such as obstacle data, to enhance overall system performance [16].

IV. System Architecture

Our system is designed as a modular, scalable framework that integrates high-fidelity simulation with modern ROS2-based SLAM and navigation stacks with NVIDIA Isaac Sim at the project's core. Every process including the simulation is initiated by the main launch script (launch multiSLAM.py), which launches each process concurrently on a separate thread. Standard output and error streams from each process are redirected to a structured logging directory, ensuring that each component's feedback is stored separately for debugging and evaluation. Each robot in the system launches instances of SLAM and Nav2 under a unique namespace, this is important as ROS2 needs to be able to separate the data given and received from both Isaac Sim and CATMiP for each robot. Separate namespaces ensure no conflicts or overlaps in topic names, service calls, or parameter servers. A high-level overview of our system's architecture is provided in [Fig 2], which showcases a simplified diagram of all the individual processes in our system and how they communicate.

To test our system, we integrate it with the CATMiP as a control algorithm [1] which is a high-level algorithm trained using RL for search and rescue applications. CATMiP provides directions to a team of robots, which consists of "explorers" and "rescuers" to navigate an unknown map and locate the "target". In this scenario, explorer robots move at twice the speed of rescuer robots which represents how a search and rescue team would consist of fast robots such as drones or quadrupeds to explore the area initially before calling over a heavier robot to interact with the target. Once the target has been located by a robot CATMiP navigates a rescuer robot towards it and ends the scenario when the rescuer robot reaches the target.

CATMiP requires several "channels" as input that must to be extracted from the SLAM and odometry instances in ROS2 before being processed. Once the input channels are received CATMiP will output high-level positional commands for each robot in the team. CATMiP requires input channels for a global map which matches the size of the simulation map where each pixel represents a 1 m x 1 m section of the map. As well as smaller local maps for each robot which are 7 x 7 pixels (meters) centered on the robot's current position. Explanations behind the rationale of choosing the following channels is provided in [1]:

- Occupancy Map
- Exploration Map
- Location of Target
- Locations of Explorer Robots
- Locations of Rescuer Robots

- Location of All Robots
- History of Navigation Goal Locations

These channels are encoded as binary masks where the relevant regions are encoded as white and everything else is black.

Since the expected resolution of the input channel images is expected to be 1 m x 1 m the resolution of the SLAM Toolbox was reduced from 5 cm to 1 m, this means that Nav2 has less information when planning paths and sometimes results in convoluted paths, but makes for much easier communication with CATMiP. To process the individual channels the extracted SLAM maps are cropped to the explored area and resized to 30x30 pixels. In the Occupancy Map, all of the grey or unknown cells are turned white, and in the Exploration Map all of the black cells are turned white and all of the grey cells are turned black. The other channels are various combinations of robot coordinates against a black 30x30 pixel map. These coordinate maps can be generated from extracting individual robot coordinates via ROS2 and transferring to the relevant channels. [Fig 3] displays the raw maps and various processed channels.

V. Conclusion

In this paper, we presented a 3D simulation framework for collaborative SLAM and navigation in heterogeneous multi-robot teams, implemented using NVIDIA Isaac Sim and ROS2. By integrating SLAM Toolbox's decentralized posegraph merging with Nav2's layered cost map planners, our system enables multiple robots to concurrently build and share globally consistent maps while autonomously navigating complex 3D environments. Unlike existing 2D grid-world platforms, our framework, called 3D GridWorld, provides realistic sensor data streams, accurate dynamics modeling, and seamless process management, narrowing the sim-to-real gap and reducing the engineering overhead associated with custom simulation environments. Through experiments with three TurtleBot robots in a grid-based 3D scene, we demonstrated robust multi-robot map convergence, efficient loop closure, and clean interoperability between high-level decision-making and low-level control stacks. Collectively, these contributions establish a flexible foundation for evaluating MARL algorithms in settings that better reflect real-world robotics deployments.

Future research on this project should focus on:

- Develop a user-friendly interface for dynamically spawning different robot models with different kinematic constraints.
- Implement procedurally generated environments (e.g., randomized obstacle and wall layouts) to support robust learning and testing.
- Automate orchestration through containerization.
- Accommodate robots with disparate sensor modalities (e.g., 3d LiDAR, RGB-D, stereo vision).
- Integrate sensor noise and communication latency models to more accurately emulate field deployments.

By pursuing these next steps, our 3D GridWorld could transform into a comprehensive research platform that accelerates the development, evaluation, and deployment of advanced multi-robot collaboration strategies.

References

[1] M. Farjadnasab and S. Sirouspour, "Cooperative and asynchronous transformer-based mission planning for heterogeneous teams of mobile robots," *arXiv preprint arXiv:2410.06372*, 2025. [Online]. Available: https://arxiv.org/abs/2410.06372

[2] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Sendai, Japan, 2004, vol. 3, pp. 2149–2154, doi: https://doi.org/10.1109/IROS.2004.1389727.

[3] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Vilamoura-Algarve, Portugal, 2012, pp. 5026–5033, doi: https://doi.org/10.1109/IROS.2012.6386109.

[4] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High performance GPU-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021. [Online]. Available: https://arxiv.org/abs/2108.10470

[5] A. Labiosa and J. P. Hanna, "Multi-robot collaboration through reinforcement learning and abstract simulation," *arXiv preprint arXiv:2503.05092*, Mar. 2025. [Online]. Available: https://arxiv.org/abs/2503.05092

[6] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry, "Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks," *arXiv preprint arXiv:2306.13831*, 2023. [Online]. Available: https://arxiv.org/abs/2306.13831

[7] R. Wang, M. Lyu, and J. Zhang, "A multi-robot collaborative exploration method based on deep reinforcement learning and knowledge distillation," *Mathematics*, vol. 13, no. 1, p. 173, 2025. [Online]. Available: https://doi.org/10.3390/math13010173

[8] S. Salimpour, J. Peña Queralta, D. Paez-Granados, J. Heikkonen, and T. Westerlund, "Sim-to-real transfer for mobile robots with reinforcement learning: From NVIDIA Isaac Sim to Gazebo and real ROS 2 robots," *arXiv preprint arXiv:2501.02902*, Jan. 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.02902

[9] M. Yang, H. Cao, L. Zhao, C. Zhang, and Y. Chen, "Robotic sim-to-real transfer for long-horizon pick-and-place tasks in the Robotic Sim2Real Competition," *arXiv preprint arXiv:2503.11012*, 2025. [Online]. Available: https://arxiv.org/abs/2503.11012

[10] Y. Jiang, C. Wang, R. Zhang, J. Wu, and L. Fei-Fei, "TRANSIC: Sim-to-real policy transfer by learning from online correction," *arXiv preprint arXiv:2405.10315*, 2024. [Online]. Available: https://arxiv.org/abs/2405.10315

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, 2009.

[12] P.-Y. Lajoie and G. Beltrame, "Swarm-SLAM: Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 475–482, Jan. 2024. [Online]. Available: https://doi.org/10.1109/LRA.2023.3333742

[13] S. Zhong, Y. Qi, Z. Chen, J. Wu, H. Chen, and M. Liu, "DCL-SLAM: A distributed collaborative LiDAR SLAM framework for a robotic swarm," *IEEE Sensors Journal*, vol. 24, no. 4, pp. 4786–4797, Feb. 2024. [Online]. Available: https://doi.org/10.1109/JSEN.2023.3345541

[14] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021. [Online]. Available: https://doi.org/10.21105/joss.02783

[15] A. C. Chathuranga *et al.*, "Multirobot SLAM for ROS2," GitHub Pull Request #592, https://github.com/SteveMacenski/slam_toolbox/pull/592, Accessed: 2025.

[16] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The Marathon 2: A navigation system," in *Proc. 2020 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2718–2725. [Online]. Available: https://doi.org/10.1109/IROS45743.2020.9341207