Crowd Navigation with LiDAR via Reinforcement Learning

Rafael Cathomen
Perception and Learning for Robotics
ETH Zürich
Switzerland
carafael@student.ethz.ch

Hojune Kim
Perception and Learning for Robotics
ETH Zürich
Switzerland
hojkim@student.ethz.ch

Abstract: Navigating around dynamic obstacles such as human crowds poses a significant challenge for mobile robots. This project introduces a pipeline to train a hierarchical, LiDAR based perception-navigation policy end-to-end with reinforcement learning. The planner utilizes an encoding of a LiDAR point cloud to predict velocity commands that are passed to a low-level policy. Utilizing the student-teacher approach, we first train a teacher policy using privileged observations, which include the velocities of the obstacles and a sparse 2D point cloud. A LiDAR encoder, trained using self-supervision, embeds the raw point-cloud data for the student policy. To enable the planner to reason about the motion of the obstacles and prevent it from getting stuck in local minima, we include a gated recurrent unit in our policy network. We evaluate the teacher policy in both static and dynamic environments, demonstrating that observing the velocities of the obstacles enhances performance in dynamic scenarios. Additionally, our results reveal that the current implementation of the recurrent unit in combination with recurrent PPO does not improve the performance in the static case, necessitating further research.

Keywords: Navigation, Dynamic Obstacles, LiDAR

1 Introduction

Mobile robots are increasingly common, but mostly operate in controlled environments like warehouses or factories. Miki et al. [1] enabled legged robots to navigate unstructured terrain like forests or mountains. However, to integrate robots into daily life, advanced navigation skills are also a crucial component. Recent advancements allow autonomous agents to navigate complex environments, including avoiding undesirable areas like roads by utilizing visual semantic information [2]. Social-aware trajectory prediction methods, like [3], rely heavily on precise human information, often from advanced detection models applied to image data. LiDAR data, while occasionally used alongside image data, is seldom employed alone [4]. Effective end-to-end reinforcement learning approaches, as shown by Liu et al. [5], are promising, but directly implementing detection and tracking networks onto robots for real-world applications remains computationally challenging.

We believe that LiDAR data is sufficient to navigate around humans. The positional information is inherent to LiDAR and the temporal information can be inferred from multiple scans. By avoiding image data, we have the potential to have a considerably smaller end-to-end neural network that can be run directly on a computer on the robot. Moreover, it does not rely on lighting conditions. To address this problem, we first train a LiDAR encoder using supervised learning. In a second step we employ reinforcement learning to train a navigation model that predicts velocity commands given the LiDAR encoding and a target position. The combination of these models results in an end-to-end navigation model which does not rely on camera data.

Our contributions can be summarized as:

- We propose a pipeline that trains a navigation model via teacher-student policies using only LiDAR as the exteroceptive sensor.
- We trained the teacher policy in static and dynamic environments.
- We evaluated the effect of adding recurrence to the navigation pipeline.

This paper is organized as follows: We review related work in section 2, outline our methodology in section 3, discuss experiments in section 4, and conclude in section 5.

2 Related Work

Local path planning is a crucial task in robotics, traditionally relying on geometric approaches that utilize point clouds or meshes to determine traversability based on occupancy metrics [6, 7]. Sampling-based and optimization-based methods are commonly employed, however, these strategies can restrict the model's effectiveness, often resulting in failures when navigating paths that encounter geometric obstacles.

Recent advancements in mobile robot navigation have shown great success with learning based approaches. Some methods are based on supervised learning to replicate expert paths or ground truth trajectories [8, 9, 10]. However, their generalizability to diverse environments is hindered by the limited diversity and richness of the data, which can also compromise the optimality of the results.

Recently, non-supervised learning approaches have been developed, categorized into imperative learning and reinforcement learning. The imperative learning framework, as used in Yang et al. [11], utilizes depth images combined with a cost map. Meanwhile, Roth et al. [2] integrates depth images with semantic information for navigation in static environments. While these methods are training-efficient, they face challenges such as the need to pre-define cost maps and bridging the gap between simulated and real-world data.

Reinforcement learning-based planning has the advantage of incorporating the robots physical constraints by learning to directly map perception inputs to high-or low-level velocity commands through interacting with a simulated world at the cost of requiring a lot of data [12]. State of the art simulation frameworks, such as [13], which fully utilize the parallelization capabilities of modern GPUs alleviate this problem by simulating thousands of robots in parallel.

He et al. [14] jointly trains an agile locomotion policy with a recovery policy to prevent failures, collaboratively achieving high-speed and collision-free navigation in static and dynamic environments. For exteroceptive perception, cameras were used to predicted 11-d ray distances in a $90°$ field of view, providing a low dimensional input for policy training.

LiDAR-based path planning methods, like [15], utilize LiDAR sensors to build maps and plan collision-free paths, bypassing issues of image-based methods like sensitivity to lighting conditions. Nonetheless, challenges persist in processing LiDAR point cloud data effectively, especially in crowded environments with dynamic obstacles.

Liu et al. [16] utilized reinforcement learning to train a crowd-navigation policy with a stereo camera and LiDAR observations in combination with an attention based human-robot model introduced by the works of Chen et al. [3]. Albeit showing impressive results, they did not use a memory unit which can lead to the planner getting stuck in local minima [17].

Most of these methods navigate each frame independently, neglecting the temporal movement of dynamic objects. Liu et al. [5] employed decentralized structural RNNs with model-free reinforcement learning, a method that's effective but can be unstable during training and demands significant computational resources, especially in dynamic environments. Additionally, scalability issues and computational complexity may hinder onboard computation with real-time performance.

3 Method

3.1 Problem Definition

We define the environment in which the robot operates as $\mathcal{Q} \subset \mathbb{R}^3$ and the robots pose at time step t as $\boldsymbol{p}_t^R \in \mathcal{Q}$, referring to its 2D position and yaw orientation. Let the subset $\mathcal{T}_t \subset \mathcal{Q}$ represent the workspace that the robot can traverse at time step t. The environment is static if $\mathcal{T}_0 = \mathcal{T}_t \forall t \in \mathbb{N}$. Given a LiDAR point cloud $\mathcal{P}_t \in \mathbb{R}^{n\times 3}$ (a set of 3D points) at time step t and a 2D goal position $\boldsymbol{p}_t^G \in \mathbb{R}^2$ in robot frame, a velocity command $\boldsymbol{v}_t^{des} \in \mathbb{R}^3$ should be calculated that navigates the robot to the goal while avoiding obstacles, such that $\boldsymbol{p}_{t+1}^R \in \mathcal{T}_{t+1}$. The navigation is successful if the goal is reached in finite time such that $||\boldsymbol{p}_{T,xy}^R - \boldsymbol{p}^G||_2 < \epsilon_{goal}$, where $\epsilon_{goal} \in \mathbb{R}$ is a distance threshold, $T \in \mathbb{N}$ is a finite time step and the subscript xy refers to the positional component of the robot pose.

In simulation, we have access to privileged information which we use for the encoder training and the teacher policy. Let $\boldsymbol{D} \in \mathbb{R}^{k\times 2}$ be an ordered, sparse 2D point cloud, meaning that all points lie on a plane that is parallel to the ground and $k \ll n$. The point cloud is ordered according to the yaw rotation of the robot, meaning that the point in front of the robot is always in the same entry of $\boldsymbol{D}$. Let $\boldsymbol{D}_{dist} \in \mathbb{R}^k$ be the distances of each point of $\boldsymbol{D}$ to the robots origin and $\boldsymbol{D}_{vel} \in \mathbb{R}^{k\times 2}$ be the velocities of the obstacles corresponding to the points in $\boldsymbol{D}$ rotated to the robots frame. Note that in a static environment all entries of $\boldsymbol{D}_{vel}$ are 0. In fig. 1 the privileged data is visualized.

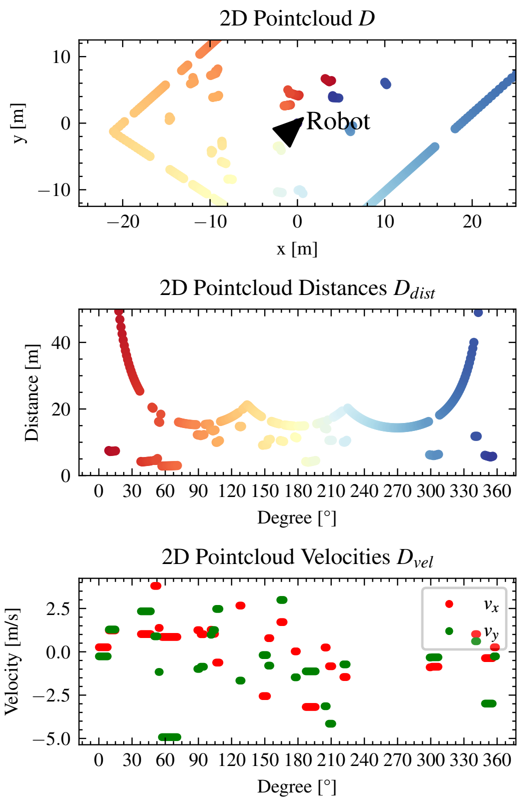

Figure 1: Privileged observations. The first plot shows the 2D point cloud, color coded by the ordering. The second one shows the unraveled distances of that point cloud. The third plot shows the non-zero velocities of the meshes corresponding to each point.

3.2 System Overview

The proposed training pipeline is illustrated in fig. 2. The navigation policy essentially consists of a perception and planning network. The perception network in the teacher policy takes as input the privileged information $[\boldsymbol{D}_{dist}, \boldsymbol{D}_{vel}] \in \mathbb{R}^{k\times 3}$ and transforms it as a three-channel 1D image with a convolutional neural network (CNN) into a spatio-temporal perception embedding. The previous position of the robot in the robot frame is first lifted into a higher dimension and then concatenated to the perception embedding. By providing this information we aim to prevent the policy from learning a registration of the spatial data. The concatenated embedding vector is passed through a multi layer perceptron (MLP) to get the privileged embedding $\mathcal{O}_{priv}$, before it enters a gated recurrent unit (GRU). Since the teacher has direct access to the velocities of the obstacles, the GRU is only required to prevent getting stuck in local minima (i.e., dead ends). The output of the GRU is concatenated with an embedding of the goal position $\boldsymbol{p}^G$ and a proprioceptive measurement $\boldsymbol{z} \in \mathbb{R}^8$ before it enters the planning network, which predicts a velocity command. The full point cloud contains $n = 28800$ points while the sparse, privileged point cloud contains $k = 360$ points. This means that a different perception head is required for the student policy. To process the full point

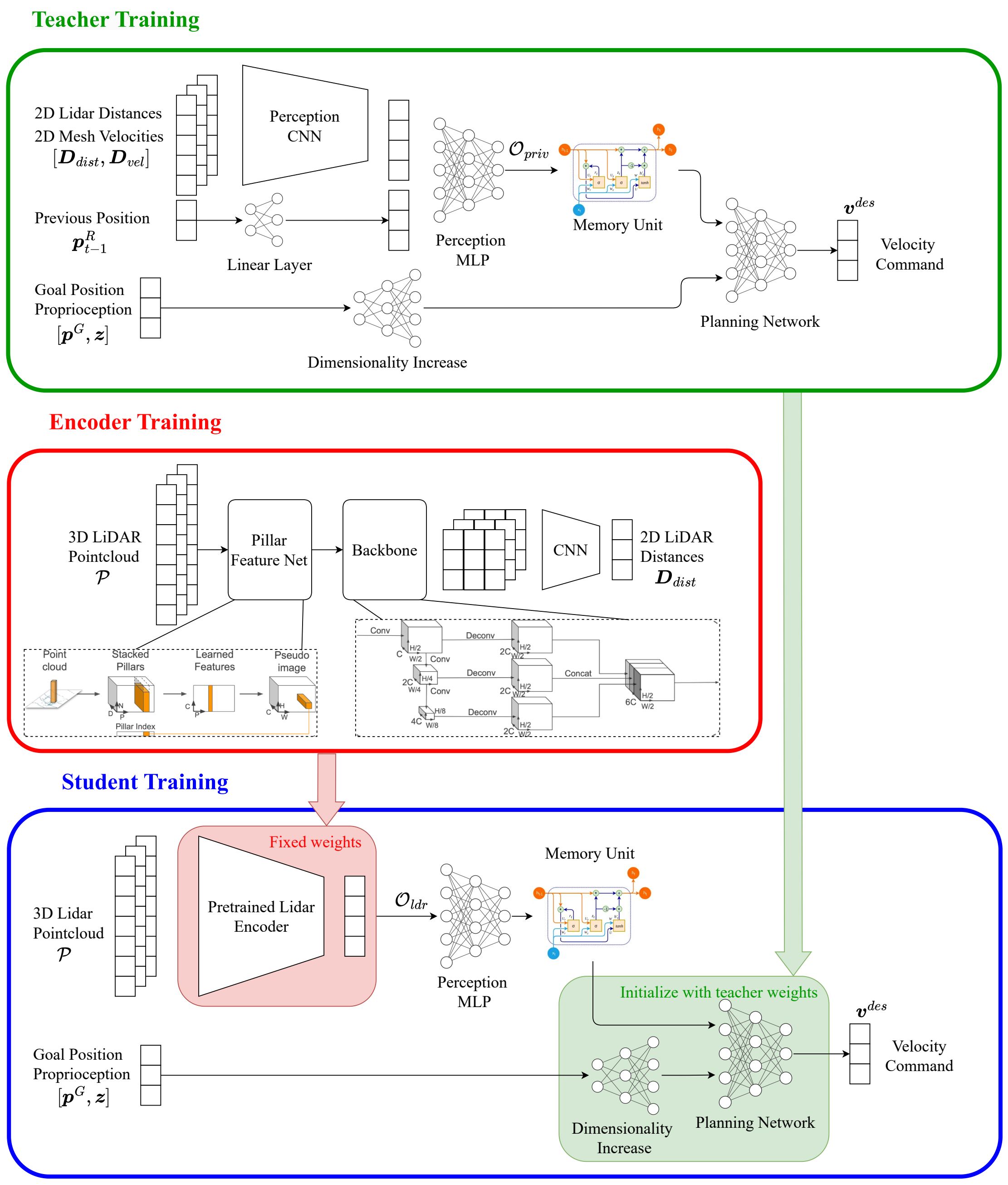

Figure 2: Overview of the proposed learning pipeline. The Teacher takes as input the privileged 2D point cloud, the previous position of the base, proprioceptive information, and a desired goal position and predicts a 2D velocity command. The encoder is based on the PointPillar architecture and is trained with the raw point cloud as features and the 2D point cloud distances as targets. The student policy takes as input the raw LiDAR point cloud, the goal position, and the proprioception and, like the teacher, predicts a velocity command.

cloud we employ the PointPillar [18] architecture which we first train supervised with a recorded dataset $\mathcal{D} = \{\mathcal{P}_i, \boldsymbol{D}_i | i = 1, 2, \ldots, n_{\mathcal{D}}\}$ of $n_{\mathcal{D}}$ pairs of full point clouds with sparse 2D point clouds to learn an efficient LiDAR encoding. The trained PointPillar network, now with its weights fixed, servers as the perception head of the student policy which takes the full point cloud $\mathcal{P}$ as input and returns a spatial embedding $\mathcal{O}_{ldr}$ which is then fed into an MLP before it enters a GRU. This recurrent unit should learn to extract the dynamic behavior of the moving obstacles while also alleviating the local minima problem. The planning network can be initialized with the weights of the teacher policy. In the following sections we will detail our pipeline.

3.3 LiDAR Encoder

As mentioned, the LiDAR encoder is implemented with the PointPillar architecture, which comprises a pillar feature network and a 2D CNN backbone. The pillar feature network converts 3D point clouds into stacked pillar tensors, creating a pseudo image. The primary advantage of PointPillar is its efficiency: it employs 2D convolution on the pseudo image rather than 3D convolution, which is more time-and memory-efficient for processing. Given the need for a streamlined encoder in an end-to-end model, this approach ensures the computation through the encoder weights is both simple and efficient. The 2D CNN backbone processes tensors at three different spatial resolutions to accurately detect obstacles of various sizes. We refined the model by replacing batch normalization with layer normalization, allowing training to be independent of batch size. The decoder was designed to downscale the latent vector to match the scale of privileged information through convolution, without using a pooling layer.

The trained LiDAR encoder serves as the perception head of the student policy, extracting the spatial embedding $\mathcal{O}_{ldr}$ from raw point clouds, as illustrated in fig. 2. To train the encoder, we first simulated the LiDAR sensor in simulation to collect the training dataset $\mathcal{D}$ in a dynamic environment. Utilizing privileged information from the simulation, we amassed a total of $n_{\mathcal{D}} = 8000$ data points, totaling approximately $10\,\mathrm{GB}$. By removing the output layer, the encoder effectively transforms a LiDAR point cloud into the spatial embedding $\mathcal{O}_{ldr}$.

3.4 Planner

The planner takes as inputs the target position and exteroceptive and proprioceptive sensor-data, and returns a twist command $\boldsymbol{\tau} \in \mathbb{R}^3$ which is then passed to a robot-specific pretrained low-level policy. To constrain the velocity command to a valid range without clipping the output which could lead to vanishing gradients, we employ the beta distribution which has a support range $S = [0, 1]$. The beta distribution is parameterized by two values, α and β and its probability density function is defined as follows:

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha - 1}(1 - x)^{\beta - 1} \tag{1}$$

where Γ is the Gamma function. Instead of directly predicting the individual entries of the twist command, we instead predict the α and β values of a beta distribution, sample from it and linearly transform it to a robot-specific target range to get the actual prediction. This can be simply achieved by doubling the size of the output layer.

3.5 Teacher Policy Training

3.5.1 Static Obstacles

We first train the navigation policy in a static environment with a curriculum terrain. The terrain is generated with randomized terrain-cells of size 8×8 m. In fig. 3 different terrain-cell types, such as passages with dead ends and randomly placed cuboid obstacles are shown. We increase the difficulty simply by increasing the number of random obstacles on a cell from 0 to 16. When starting the train run, the robot spawns on the simplest terrain-cell. A goal position is sampled random in a neighboring cell. The episode is terminated when the robot reaches the goal, when it collides with an obstacle or when a time limit is reached. To determine when a robot moves to a more difficult or a simpler terrain, we count how many episodes terminated due to reaching the target $n_{success}$ and due to colliding with the environment n_{fail}. If $n_{success} - n_{fail} \geq 4$ the robot moves to a more difficult terrain and if $n_{fail} - n_{success} \geq 8$ the robot moves to a simpler terrain. After each terrain change, we reset the termination counters. We also increase the goal distance $d_{goal} \in \mathbb{N}$ linearly with respect to the training iteration. The goal is sampled in a square, centered in the spawn position, with a side length $l = 16 d_{goal}$ m, such that the goal is in the center of a terrain-cell which is guaranteed to be not blocked. We increase d_{goal} up to 5, resulting in a maximum possible goal distance of 56 m.

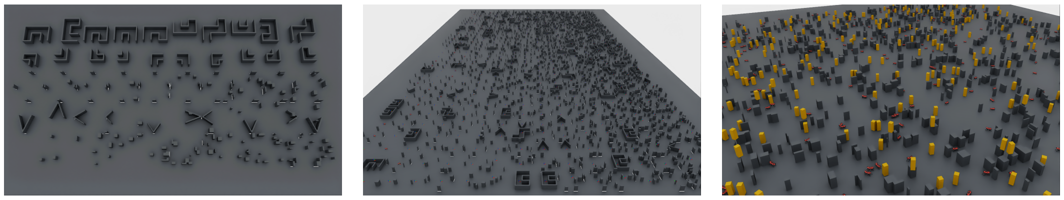

Figure 3: Different terrain types (left), static curriculum terrain with increasing difficulty from left to right (middle) and terrain with dynamic obstacles in yellow (right).

We designed the reward such that the robot should reach the goal as fast as possible, while staying far a way from obstacles. To discourage the robot from walking towards obstacles, we added a penalty if an obstacle is in front of the robot. The detailed rewards and penalties can be seen in table 1.

Reward	**Formula**	**Weight**
Goal reached	$\|\|\boldsymbol{p}^R_{t,xy} - \boldsymbol{p}^G\|\|_2 < \epsilon_{goal}$	1000
Goal progress	$\text{clip}(\frac{\boldsymbol{p}^R_{t,xy} - \boldsymbol{p}^G}{\|\|\boldsymbol{p}^R_{t,xy} - \boldsymbol{p}^G\|\|_2} \cdot \dot{\boldsymbol{p}}^R_{xy}, 0, 1)$	1
Lateral movement	$\min((\dot{\boldsymbol{p}}^R_y)^2, 1)$	-0.05
Backward movement	$\min(\min(\dot{\boldsymbol{p}}^R_x, 0)^2, 1)$	-0.05
Termination	if episode terminates without timeout	-500
Close to obstacle	$1 - \tanh \frac{d_{min,360} - 0.75}{0.2}$, $d_{min,360}$ = distance to closest obstacle	-2
Obstacle in front wide	$1 - \tanh \frac{d_{min,60} - 1}{0.5}$, $d_{min,60}$ = distance to closest obstacle in a 60° field of view in front of the robot	-1
Obstacle in front narrow	$1 - \tanh \frac{d_{min,30} - 2}{1.5}$, $d_{min,30}$ = distance to closest obstacle in a 30° field of view in front of the robot	-1
No movement	$\exp(-10 * \sum_{k=0}^{20} \|\|\boldsymbol{p}^R_{t-k,xy} - \boldsymbol{p}^R_{t-k-1,xy}\|\|_2)$	-5
Action rate	$\|\|\boldsymbol{\tau}_t - \boldsymbol{\tau}_{t-1}\|\|_2$	-0.1
Waste time	1	-0.001

Table 1: Rewards and penalties used during teacher training with static obstacles.

To estimate the performance of the navigation policy we use the success rate η which we calculate as the ratio of terminations due to reaching the targeted to total terminations over the last 10 terminations of all robots. A success rate of $\eta = 1$ would mean that all robots have terminated the last 10 episodes successfully.

3.5.2 Dynamic Obstacles

Upon achieving a satisfactory success rate in combination with a sufficiently high terrain difficulty, we transition to an environment with dynamic obstacles. To adapt to the new environment, we include the obstacle velocities to the observations, which are not required in the case of static obstacles, and replace the perception head accordingly.

To learn to navigate around dynamic obstacles, we designed a new environment and added a simplified crowd simulation. In fig. 3 dynamic obstacles are shown in yellow. For the static obstacle, we randomly place cuboid obstacles in the scene with an uniform density, i.e., a terrain without curriculum. To simulate a crowd, we created dynamic obstacles represented as cuboids of size $0.75 \times 0.75 \times 2$ m. A simple pd-controller steers these obstacles to a randomly sampled target position with a maximum velocity $v_{obs,max} \in \mathbb{R}^+$. We sample the obstacle goal position uniformly on a disk with radius 10 m around the current obstacle position in uniformly sampled time intervals between 5 and 20 s. To ensure that the moving obstacles do not leave the scene, we clip any goal position that lies outside the environment to its border. To increase the difficulty for the robots, we implement a curriculum that linearly increases $v_{obs,max}$ from 0 to 0.5 $\mathrm{m\,s^{-1}}$. To encourage the rl-agent to not collide with an obstacle, thus terminating the episode, we replace the "waste time" penalty with a "stay alive" reward with the same formula, but with a positive weight of 0.1.

3.6 Implementation Details

The teacher policy is trained with an actor-critic structure. In fig. 2 the actor structure is shown. The critic architecture is the same, except for the output dimension which is 1. The teacher perception head is a three layer CNN, each with 16 channels, a kernel size of 5 and a stride of 2 for the first and last layer and 1 for the middle layer. We used circular padding for all layers. This CNN converts the 360×1 dimensional input to a 90×16 dimensional output. This is flattened to 1440 dimension and passed through a two layer MLP with 256 and 128 layers. To this, we concatenate the embedded previous position of dimension 16 resulting in a 144 dimensional vector which is then passed through an additional layer of dimension 128, before it enters the GRU with a hidden dimension of 512. The concatenation of the goal position and the proprioception have dimension 10 and are passed through a single nonlinear layer with 128 dimensions. This output is concatenated with the output of the GRU and passed through the planning network which is designed as an MLP with 256 dimensions for the first two layers and 128 for the third layer before it enters the output layer which has dimension 6 for the actor and 1 for the critic. We used the exponential linear unit (ELU) as our activation function. This architecture is optimized using Proximal Policy Optimization (PPO) [19] within the ORBIT [13] framework based on NVIDIA ISAAC SIM.

4 Experiments

4.1 Teacher Policy

The teacher policy was trained for 10000 steps in the static environment with 512 robots in parallel on a single NVIDIA RTX 2080 GPU. To show the effect of the recurrent unit, we evaluated the teacher policy on three different static and one dynamic environment. In fig. 4 the three terrains are visualized. During testing, we deterministically set the goal position but spawned the robot with random yaw orientations. A run is considered successful if the goal is reached within 60 s and failed if the robot collides or the time is up. We let 128 robots terminate 50 times and calculated the success rate as the ratio of terminations due to reaching the goal to the total number of terminations. In fig. 5 the success rate for the three different terrains and policies with and without recurrence are visualized. When removing the GRU, we replaced it with a linear layer of the same dimension as the hidden state. Surprisingly the presence of a recurrent unit did not increase, but slightly decrease the success rate, i.e., it did not help to prevent the robots getting stuck in local minima. We hypothesize that this is caused by the way the recurrence works. The GRU treats each sequential data point as equally spaced, i.e., there is no notion of spacial relation. We tried to mitigate this by adding the previous position to the input of the recurrence but this does not seem to help. Additionally, there might be a fundamental error in the implementation of recurrent PPO that we used.

Since the addition of the recurrence does not improve the performance in the static setting, we ignored it in the dynamic setting, because the teacher does not need to infer the motion of the obstacles as this is given to the observations. The policy trained in the static environment was trained for another 5000 steps in the dynamic environment, also with 512 robots and the same amount of dynamic obstacles. The size of the training terrain is 96×96 m, resulting in an average density of one obstacle per 18 m^2. We evaluated this policy the same way we did for the static case and with the same obstacle density that we used during training. We measured a success rate of 0.745. To show the effect of adding the velocities to the observations, we also trained a policy with the same observations as in the static case in the dynamic environment, resulting in a success rate of 0.631.

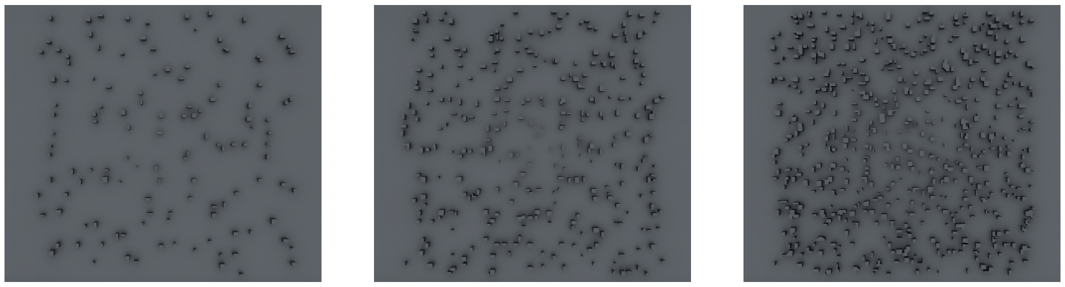

Figure 4: Static test terrains with increasing obstacle density from left to right.

5 Conclusion, Limitations and Future Work

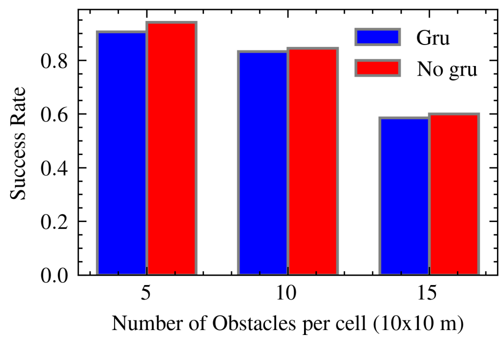

Figure 5: Success rate in a static environment with and without recurrence

In this work we propose a teacher-student pipeline to train a LiDAR based navigation policy end-to-end in simulation. We show that sparse 2D distances are sufficient exteroceptive observations to navigate to a goal position in a static environment. To train a navigation policy in a dynamic environment we simulated a crowd with cuboids that move with a limited velocity to a random sampled position. In this environment we, train a policy with privileged information, namely the mesh velocities, and we show that the performance improves. We argue that without the privileged velocity observations, recurrence is required to extract the dynamics of the scene from a series of static spatial embeddings. We added a gated recurrent unit to our network and trained it with a recurrent version of PPO, however, we noticed that adding the recurrence results in a slightly worse performance than without it. To this end, we are not sure what caused this performance decrease. Our hypothesis is, that the current implementations of recurrent units are missing a spatial metric between the data points. Additionally, the recurrent PPO implementation might not be optimal.

The LiDAR encoder was trained using a dataset obtained from our simulation setup, which includes dynamic obstacles. The encoder model contained 5,500,000 parameters, and training was conducted on 8,000 non-continuous frames. We observed that the training loss converged, but the validation loss diverged. We believe the encoder's high complexity and the insufficient dataset size led to overfitting. Due to these limitations, we were unable to train the student policy.

For future work, a spacial recurrent unit should be developed in combination with a proper recurrent PPO implementation. Additionally, the LiDAR encoder should be trained online, i.e., while simulating a trained teacher policy to collect data on the fly. This practically leads to an infinite amount of training data which should prevent the encoder from overfitting. Due to the sim-to-real gap, the LiDAR encoder should also be fine tuned on real data. Furthermore, Our basic crowd simulation setup does not reflect the behavior of real crowds, necessitating the development of a more realistic implementation, consisting of better humanoid obstacles and crowd dynamics. Finally, the full pipeline should get evaluated on a real robot with a real crowd.

Acknowledgments

We would like to thank Pascal Roth, Fan Yang and Takahiro Miki for their contributions and insightful guidance throughout this project. We also appreciate Cesar Cadena for his foundational lectures, as well as Jonas Frey, Kaixian Qu, and Fan Shi for their introductory presentations that helped deepen our understanding in robot learning and perception.

References

[1] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), Jan. 2022. ISSN 2470-9476. doi:10.1126/scirobotics.abk2822. URL `http://dx.doi.org/10.1126/scirobotics.abk2822`.

[2] P. Roth, J. Nubert, F. Yang, M. Mittal, and M. Hutter. Viplanner: Visual semantic imperative learning for local navigation, 2023.

[3] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 international conference on robotics and automation (ICRA)*, pages 6015–6022. IEEE, 2019.

[4] Z. Liu, A. Amini, S. Zhu, S. Karaman, S. Han, and D. L. Rus. Efficient and robust lidar-based end-to-end navigation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13247–13254, 2021. doi:10.1109/ICRA48506.2021.9561299.

[5] S. Liu, P. Chang, W. Liang, N. Chakraborty, and K. Driggs-Campbell. Decentralized structural-rnn for robot crowd navigation with deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3517–3524, 2021. doi:10.1109/ICRA48506.2021.9561595.

[6] C. Cao, H. Zhu, F. Yang, Y. Xia, H. Choset, J. Oh, and J. Zhang. Autonomous exploration development environment and the planning algorithms. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8921–8928. IEEE, 2022.

[7] N. Hudson, F. Talbot, M. Cox, J. Williams, T. Hines, A. Pitt, B. Wood, D. Frousheger, K. L. Surdo, T. Molnar, et al. Heterogeneous ground and air platforms, homogeneous sensing: Team csiro data61's approach to the darpa subterranean challenge. *arXiv preprint arXiv:2104.09053*, 2021.

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

[9] A. Sadat, S. Casas, M. Ren, X. Wu, P. Dhawan, and R. Urtasun. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*, pages 414–430. Springer, 2020.

[10] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[11] F. Yang, C. Wang, C. Cadena, and M. Hutter. iplanner: Imperative path planning, 2023.

[12] D. Hoeller, L. Wellhausen, F. Farshidian, and M. Hutter. Learning a state representation and navigation in cluttered and dynamic environments, 2021.

[13] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi:10.1109/LRA.2023.3270034.

[14] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi. Agile but safe: Learning collision-free high-speed legged locomotion, 2024.

[15] Y. Jiang, P. Peng, L. Wang, J. Wang, J. Wu, and Y. Liu. Lidar-based local path planning method for reactive navigation in underground mines. *Remote Sensing*, 15(2):309, 2023.

[16] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé. Robot navigation in crowded environments using deep reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5671–5677, 2020. doi:10.1109/IROS45743.2020.9341540.

[17] E. Wijmans, M. Savva, I. Essa, S. Lee, A. S. Morcos, and D. Batra. Emergence of maps in the memories of blind navigation agents, 2023.

[18] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2019.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.