Robust RL-Based Map-Less Local Planning: Using 2D Point Clouds as Observations

Francisco Leiva and Javier Ruiz-del-Solar

***Abstract*—In this letter, we propose a robust approach to train map-less navigation policies that rely on variable size 2D point clouds, using Deep Reinforcement Learning (Deep RL). The navigation policies are trained in simulations using the DDPG algorithm. Through experimental evaluations in simulated and real-world environments, we showcase the benefits of our approach when compared to more classical RL-based formulations: better performance, the possibility to interchange sensors at deployment time, and to easily augment the environment observability through sensor preprocessing and/or sensor fusion. Videos showing trajectories traversed by agents trained with the proposed approach can be found in https://youtu.be/AzvRJyN6rwQ.**

***Index Terms*—Reinforcement learning, reactive and sensor-based planning, map-less local planning.**

I. Introduction

AUTONOMOUS navigation is an essential skill for mobile robotics: traversing collision-free paths towards a target destination is an indispensable ability for a wide spectrum of robotic applications. Classical formulations to address the navigation problem often consist of different hand-engineered modules, each of them solving a particular sub-task of the navigation problem (e.g. mapping, localization, and planning), and interacting with the other modules. As a result, navigation systems following this approach are often strongly parameter-dependent, specially when deployed in previously unseen environments, or across different robotic platforms.

In recent years, a growing interest on developing learning-based solutions for navigation has arrived in tandem with the progress that deep learning has made in video games [1], computer vision, and robotics [2]. Possibly, learning-based solutions for robotic navigation might cope with the lack of generalization and sub-optimal performance that currently undermine classical, modular systems. In this regard, reinforcement learning (RL) has emerged as a natural framework for developing such solutions, as it aims to learn behaviors directly from interactions between an agent and its environment, avoiding the need for expert demonstrations that approaches such as imitation learning require, and potentially learning complex skills that would be hard to acquire otherwise.

Using RL, the navigation problem can be framed either in an end-to-end fashion, or by decomposing it and solving some of its sub-tasks. In this work, we focus on map-less RL-based local planning. While several local planners trained using RL have been proposed (e.g. [3]–[5]), some key design decisions, such as the agent's observations and its policy parameterization, have not been extensively discussed. We aim to fill this gap by analyzing the impact that these decisions have in the performance of RL-based local planning policies, and upon the study of existing formulations, to prove that great performance improvements may be achieved by addressing them. In this regard, we propose an extensible and robust approach for the design of observations and the parameterization of RL-based local planning policies. The proposed approach differs from existing solutions by using variable size 2D point clouds as the agent's observations, and by parameterizing its policy accordingly, using an ad-hoc feature extractor. We empirically show that this not only allows the trained agents to achieve better performance than alternative approaches, but also endows them with robustness to extreme perturbations on their observations, the possibility to interchange sensors at deployment time, and to augment the environment's obervability through sensor pre-processing and/or sensor fusion.

We hypothesize that using variable size point clouds as observations eases learning meaningful representations to characterize the environment. Furthermore, it decouples the sensor's characteristics (resolution and field of view) from its readings, which may endow the agent with robustness to extreme perturbations on its observations as it would learn sensor-invariant features. Through simulated experiments, the proposed approach is compared to common design trends for RL-based local planning controllers that use range measurements as inputs. Furthermore, we showcase the benefits of utilizing the proposed representation for the observations in terms of robustness and extensibility in the real-world.

The main contributions of this work are the following:

- A robust approach for the design of RL-based local planning controllers, using variable size 2D point clouds as inputs, and an ad-hoc parameterization.
- An analysis on the impact of observations and parameterization design for RL-based local planning controllers, when relying on range measurements.

Manuscript received February 24, 2020; accepted July 1, 2020. Date of publication July 21, 2020; date of current version July 29, 2020. This letter was recommended for publication by Associate Editor G. Neumann and Editor T. Asfour upon evaluation of the reviewers' comments. This work was supported by FONDECYT project 1201170, ANID-PIA project AFB180004 and CONICYT-PFCHA/Magíster Nacional/2018-22182130. *(Corresponding author: Francisco Leiva.)*

The authors are with the Department of Electrical Engineering & the Advanced Mining Technology Center (AMTC), Universidad de Chile, Santiago 8370451, Chile (e-mail: francisco.leiva@ing.uchile.cl; jruizd@ing.uchile.cl).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.3010732

- An analysis on the benefits of decoupling the range sensors' characteristics from their readings when training RL-based local planning policies, and how this can be harnessed to improve the policy's performance in the real-world.

II. Related Work

Learning-based map-less navigation has grown in popularity in the recent years. Several works have presented deployable, target-driven navigation policies, which are typically trained to behave as local planners [3], [5]–[7]. The training procedure of such planners can be framed as a supervised learning problem, where the mapping between observations and actions is learned from labeled examples generated by an expert demonstrator [8], [9].

RL-based local planners, on the other hand, learn directly from interactions between the agent and its environment. Due to the large number of interactions required to obtain proficient policies, these local planners are usually trained in simulations, and then deployed in the real world. Although several works on learning visual navigation policies exist (e.g. [6], [10], [11]), the mismatch between high dimensional observations coming from real and simulated sensory information, makes the deployment of these policies challenging.

More pragmatic approaches tend to rely on range sensory data, as the reality gap between simulated and real range measurements (e.g. coming from a LiDAR) is practically non-existent. In [3], [4] and [12], successful implementations of RL-based local planners using sparse range measurements as observations are presented. The low dimensional vectors employed in these cases are used as inputs to simple parameterizations for the policies, which consist of multilayer perceptrons. Although these works demonstrate the effectiveness of this approach, such observations are not suitable for fine-grained control in complex environments.

In [7], [5], [13] and [14], higher dimensional observations are constructed by stacking sensor readings to tackle the partial observability of the environment. While [7] uses fully connected layers to process these inputs, in [5], [13] and [14], convolutional layers are used. A similar approach is proposed in [15], however, a recurrent multi-modal architecture that has Long Short-Term Memory (LSTM) [16] layers is employed.

In [17], we explored the idea of using aggregated point clouds as part of the inputs for a multi-modal neural network parameterizing a collision avoidance policy trained using RL. In that case, however, most of the required information to avoid collisions was obtained from depth maps. In [18], formulating the observations of swarming agents as variable sized sets for RL-based policies is proposed.

Although the body of work on RL-based local planners that rely on fixed-size range information is closely related to our work, we aim to showcase the benefits of replacing such representations, to non-fixed size, unordered, point cloud-like representations. In this regard, our approach relates to [17] and [18], however, aims to validate the benefits of using this representation to train RL-based local-planning policies.

III. Proposed Approach

A. Problem Formulation

The agent-environment interaction is modeled as a Partially Observable Markov Decision Process (POMDP) described by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a scalar reward function $\mathcal{R}(s,a)$, a stochastic state transition function $T(s,a,s') = p(s'|s,a)$, a stochastic observation function $\mathcal{O}(s,'a,o) = p(o|s,'a)$, a set of observations Ω, and a discount factor $\gamma \in [0,1]$. At each time step t the agent observes o_t, executes an action a_t according to a policy $\pi(a_t|o_t)$, receives a scalar reward r_t, and transitions to a new state s_{t+1}. The RL objective is to learn a policy that maximizes the expected discounted reward: $J(\pi) = \mathbb{E}_{a_t\sim\pi(a_t|o_t)}[\sum_{t=1}^{T}\gamma^{t-1}r_t]$.

In this work, we use continuous control commands for a differential velocity controller, as continuous actions allow fine-grained control over the agent. The agent's actions are defined as $a = (v_x, v_\theta)$, where $v_x \in [0, v_x^{\max}]$ corresponds to the agent's linear velocity (being $v_x^{\max}$ its maximum value), and $v_\theta \in [v_\theta^{\min}, v_\theta^{\max}]$ to the agent's angular velocity (being $v_\theta^{\min}$ and $v_\theta^{\max}$ the minimum and maximum values of v_θ).

The agent's observations are defined by o_{pcl}, o_{odom}, and o_{target}. The first term, o_{pcl}, denotes a non-fixed size 2D point cloud representation of range measurements. The second term, $o_{\text{odom}} = (\hat{v}_x, \hat{v}_\theta)$, corresponds to odometry-based estimations of the agent's linear and angular speeds. Finally, $o_{\text{target}} = (\rho_{\text{target}}, \theta_{\text{target}})$ corresponds to the local target destination in polar coordinates.

For the reward signal, we consider the function defined by Eq. (1), where ρ^t_{target} denotes the euclidean distance between the agent and the target destination at time t, whereas ρ_{thresh} is a predefined threshold distance used to decide whether the agent has reached the target or not.

$$r_t = \begin{cases} r^t_{\text{navigation}} & \text{if } \rho^t_{\text{target}} \geq \rho_{\text{thresh}} \\ r^t_{\text{success}} & \text{if } \rho^t_{\text{target}} < \rho_{\text{thresh}} \\ r^t_{\text{collision}} & \text{if the agent collides}\,. \end{cases} \tag{1}$$

The term $r^t_{\text{navigation}}$ is designed to guide the agent to the target while penalizing undesirable behaviors. It is defined as $r^t_{\text{navigation}} = r^t_{\text{target}} + r^t_{\text{fov}} + r^t_{v_\theta} + r^t_{\text{danger}}$. The term r^t_{target} encourages the agent to move towards the target (Eq. (2)).

$$r^t_{\text{target}} = \frac{\hat{v}^t_x}{v^{\max}_x}\cos(\theta^t_{\text{target}}) + 5\cdot\mathbb{1}_{\{\rho^t_{\text{target}}:\rho^t_{\text{target}}<\rho^{t-1}_{\text{target}}\}}(\rho^t_{\text{target}}) - 6 \tag{2}$$

The term r^t_{fov} provides a penalization signal to the agent when θ^t_{target} exceeds 120°, that is, when the target lies outside the projection of a 240° FoV with respect to the agent's local frame (Eq. (3)). By adding this term to the reward function, we assume that targets queried to the agent do not require complex long-term planning capabilities to be reached.

$$r^t_{\text{fov}} = (3\cos(\theta^t_{\text{target}}) - 5)\cdot\mathbb{1}_{\{\theta^t_{\text{target}}:|\theta^t_{\text{target}}|>120^\circ\}}(\theta^t_{\text{target}}) \tag{3}$$

The term $r^t_{v_\theta}$ penalizes large angular velocities and accelerations. It is defined as $r^t_{v_\theta} = -2\,K^t_{v_\theta}\cdot\mathbb{1}_{\{K^t_{v_\theta}:K^t_{v_\theta}>0.5\}}(K^t_{v_\theta})$,

where $K^t_{v_\theta}$ is given by Eq. (4).

$$K^t_{v_\theta} = \frac{1}{2v^{\max}_\theta} \max\{2|\hat{v}^t_\theta|, |\hat{v}^t_\theta - \hat{v}^{t-1}_\theta|\} \tag{4}$$

The term r^t_{danger} provides a strong penalization signal whenever the agent gets too close to an obstacle, thus, discouraging hazardous maneuvers. The value of r^t_{danger} depends on l^t_{min}, the minimum range measurement observed by the agent (Eq. (5)). Finally, r^t_{success} is set to 100, while $r^t_{\text{collision}}$ to -200.

$$r^t_{\text{danger}} = (60 \max\{l^t_{\text{min}} - 0.35, 0\} - 5) \cdot \mathbb{1}_{\{l^t_{\text{min}} : l^t_{\text{min}} < 0.4\}}(l^t_{\text{min}}) \tag{5}$$

To encourage the agent to reach the target as fast as possible, all the terms that define $r^t_{\text{navigation}}$ have a maximum value of zero. Whenever undesirable behaviors are detected, these terms provide strong penalization signals. The constants defining each of these terms have been adjusted so these penalizations have comparable minimum values: -5 for r^t_{target}, -6.5 for r^t_{fov}, -1 for $r^t_{v_\theta}$, and -2 for r^t_{danger}.

B. Non-Fixed Size Point Cloud Observations

We use 2D range sensors as the agent's primary source of information on the environment state. The measurements provided by these sensors contain geometric information regarding the agent's surroundings, which is essential for effective collision avoidance. Vectors containing range measurements have been extensively used as observations in learning-based approaches for robotic navigation (e.g. [3], [8], [12]). Using these observations, however, imposes some sensible constraints when following traditional approaches for feature extraction using neural networks: the observations must maintain a fixed dimension, and as a consequence, out-of-range measurements should be encoded.

As range measurements vectors do not explicitly provide a relationship between ranges and their corresponding angles (with respect to the sensor's origin), a constant geometric distribution of these measurements becomes critical to maintain consistency across observations. This is true even for sub-sampling strategies applied over range readings as a pre-processing stage, such as those proposed in [8] and [12], as they assume a relatively stable sensor's FoV, and always produce fixed size outputs. Moreover, there is a direct relationship between the observations' dimension and its resolution, which generates a trade-off between the environment's observability, and the complexity of learning a useful representation from the observation.

Taking the above into account, we propose using range measurements as non-fixed size 2D point clouds. Let us consider a range measurement vector with n elements, $o_{\text{vect}} = [\rho_1, \ldots, \rho_n]$, where ρ_i corresponds to the i-th range measurement, and θ_i to its corresponding angle. The ranges are such that $\rho_i \in (\rho_{\text{min}}, \rho_{\text{max}}) \cup \{\overline{\rho}, \underline{\rho}\}$, where ρ_{min} and ρ_{max} are the minimum and maximum values for the measurements, whilst $\overline{\rho}$ and $\underline{\rho}$ are codifications for measurements which are above ρ_{max} or below ρ_{min}, that is, codifications for out-of-range measurements. Therefore, the proposed representation corresponds to $o_{\text{pcl}} = \{(\rho_j \cos(\theta_j), \rho_j \sin(\theta_j))\}_{j=1}^{k \leq n}$, where k is the number of in-range measurements. If k equals zero, o_{pcl} is arbitrarily set to the singleton $\{(\rho_{\text{max}}, 0)\}$.

The proposed representation has advantages compared to the classic fixed-size range measurements vector. It explicitly incorporates θ_i as part of the observation, which allows the learning of features that are independent of the readings' geometric distribution. Furthermore, these features are also independent of the sensor's resolution: the number of points conforming the observation is variable, as no explicit codification for out-of-range measurements is employed.

C. Algorithm and Policy Parameterization

For training, we use the Deep Deterministic Policy Gradient (DDPG) [19] algorithm, so two independent neural networks are used to parameterize the policy (actor) and the action-value function (critic). We chose this algorithm as it has been specially designed for continuous control, while being sample efficient due to its off-policy nature. Considering the agent's observations have varying dimensions because of o_{pcl}, the actor and critic networks cannot use fully connected nor convolutional layers to take o_{pcl} as input. Instead, the feature extractor proposed in the PointNet architecture [20] is employed. The feature extraction process performed over o_{pcl} can be described as a two-stages process. In the first stage, each point is normalized and independently fed to the same multilayer perceptron (MLP) to produce a set of fixed-size intermediate representations. In the second stage, these representations are aggregated by applying a max-pooling operation over them, producing a single fixed-size representation.

The other components of the agent's observations, o_{odom} and o_{target}, are normalized and fed to fully connected layers. This provides balance across the dimensions of the intermediate representations associated to the observations' components, so their influence on the agent's behavior becomes more even. This design decision follows [21] and [22], where the disadvantage of directly combining observations and intermediate representations is discussed. For the critic, the action inferred by the actor is concatenated to both o_{odom} and o_{target}, before feeding them to their respective feature extractors.

The representations obtained by these different modules are then concatenated and fed to a sequence of layers to get the networks' final outputs. This sequence is conformed by a fully connected layer, an LSTM layer, and then another fully connected layer. For the actor, the resulting representation is then fed to two independent fully connected layers, which separately output values for normalized linear and angular velocities (because of their sigmoid and tanh activation functions, respectively). For the critic, the aforementioned representation is fed to a single fully connected layer with a linear output activation to output the predicted state-action value.

We choose to include LSTM layers in our model as a means to integrate temporal information contained in the agent's observations. To learn meaningful hidden states, we follow the approach utilized in [6] and validated in [23], that is, instead of just performing uniform sampling, we sample traces of experiences from the replay buffer, and skip updates for the first elements of

(a) Stage world 1 (SW1) (b) Stage world 2 (SW2)

Fig. 1. Simulated worlds used to train the local planning policies, both are 8 m long by 8 m wide. The red polygon at the center of each world is consistent with the Pioneer 3DX footprint. The shaded blue regions represent the area covered by the laser range finder mounted on the robot.

the sampled sequences. A diagram of the proposed architecture is depicted in Fig. 2(c), including the specific number of units utilized for each layer. Notice that two variants are displayed: one that has an LSTM layer, named PCL-LSTM, and one that does not, named PCL.

IV. Experiments

A. Experimental Setup

Training was conducted using the Stage simulator [24] and the ROS [25] framework. The validation of the proposed approach is performed both in simulations using Gazebo [26], and in the real-world. The platform employed to perform the real-world validation corresponds to a differential drive Pioneer 3DX robot. Consequently, the agents utilized in simulations are designed to resemble the Pioneer 3DX. This robot is equipped with a Hokuyo URG-04LX-UG01 laser range finder, which has a max range of 5.6 m, an angular resolution of $\sim 0.352^\circ$, and a 240° FoV. Furthermore, we added an Asus Xtion RGB-D camera aligned with the x-axis of the laser range finder (looking forward), in order to improve the robot's observability in the real world.

For implementing DDPG and training the policies, TensorFlow is utilized. All the processing required (for both training and validation) is conducted on a laptop equipped with a Intel Core i7 7700 HQ CPU, and a Nvidia GeForce GTX 1060 GPU. For the real-world experiments, all processing is performed on-board, on a Raspberry Pi 3 Model B.

B. Training in Simulations

Training was conducted in the Stage worlds proposed in [5] (see Fig. 1). We refer to these worlds as SW1 (Fig. 1(a)) and SW2 (Fig. 1(b)). To showcase the effectiveness of the proposed method, we compare it to different policy parameterizations. These parameterizations are based on common trends that have been already explored for RL-based local planning utilizing range measurements. Fig. 2 depicts all the actor-critic parameterizations that were studied.

The first parameterization, SPARSE (Fig. 2(a)), is a variant of the model proposed in [3], where the observations are constructed by concatenating and normalizing a 10-dimensional range measurements vector (o_{sparse}), the local target position (o_{target}), and the agent's estimated velocity (o_{odom}).

The second and third parameterizations, DENSE and DENSE-STK (see Fig. 2(b)), utilize dense range measurements vectors as part of their observations (o_{dense}). These vectors are 128-dimensional, constructed by uniformly decimating the 683 readings provided by the robot's sensor. As for the SPARSE parameterization, o_{odom} and o_{target} are also part of the agent's observations. The only difference between DENSE and DENSE-STK, is that three temporally stacked laser range measurements are fed to the actor and the critic networks in DENSE-STK, instead of just one as in the case of the DENSE parameterization. Because of the imbalance between the dimensions of o_{dense}, and o_{odom} and o_{target}, each of these components are normalized and fed to the neural networks in a multimodal fashion. Furthermore, o_{dense} is fed to 1D convolutional layers (instead of fully connected layers) to address its high dimensionality, in a similar way to what is proposed in [13]–[15], and [5].

The PCL and PCL-LSTM parameterizations (Fig. 2(c)), are designed as described in Section III-C to handle 2D point clouds as inputs. Finally, the V2R[1] and ASL[2] parameterizations (Fig. 2(d)–(e)) are baseline models which were proposed in [3] and [4], respectively. The V2R model is similar to SPARSE, but uses ReLU activation functions, and has 512 hidden units per fully connected layer. The ASL model takes 36 normalized range measurements as inputs, sampled by applying min-pooling over the sensor's readings [4].

The SPARSE, DENSE, DENSE-STK, PCL and PCL-LSTM models were trained in both of the Stage worlds depicted in Fig. 1 (SW1 and SW2). The baseline models, V2R [3] and ASL [4], and PCL-LSTM were trained in "SW2r." This new environment is almost equal to SW2, but the reward function used was changed to match those used in [3] and [4], namely, the reward defined Eq. (1) was modified so that $r^t_{\text{success}} = 10$, $r^t_{\text{collision}} = -0.4$, and $r^t_{\text{navigation}} = -C(\rho^t_{\text{target}} - \rho^{t-1}_{\text{target}})$, with $C = 2$. The parameters defining this new, simpler reward were primarily taken from [4], however, minor changes (such as the C factor) were introduced to account for differences in the agent's dynamics we use. Training was conducted in an episodic fashion. At each episode, the agent starts in the world's origin, and tries to reach a valid target destination which randomly changes at the beginning of each episode. During training, no noise is added to the agent's observations, and a ground-truth localization system provides the agent with the local coordinates of the target. An episode ends successfully if the agent reaches the target, or prematurely if it collides with the walls of the environment, or a fixed number of agent-environment interaction steps pass. All the training hyper-parameters can be found in Table I. The remaining parameters for DDPG were taken from [19]. The exploration noise is modulated by a factor that decays linearly from 1.0 to 0.05 in $5{\cdot}10^4$ training steps. When training the PCL-LSTM model, we sample traces of length 16 from the replay buffer, and skip updates for their first 4 elements. For all the other models, experiences are sampled uniformly.

C. Validation in Simulations

Training was conducted for 10^5 steps for all models. Every $5{\cdot}10^3$ steps, 50 evaluation episodes were performed (exploration

[1]Named after "virtual to real."

[2]Named after the id given to the model in the source code provided in [4]

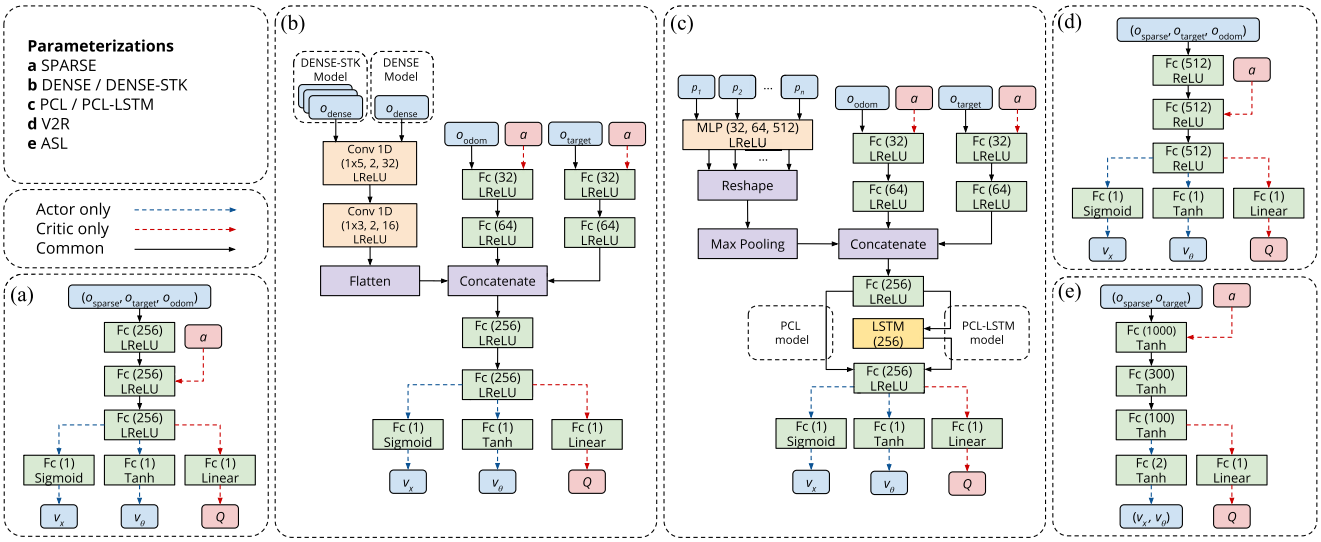

Fig. 2. Depiction of the studied actor-critic architectures. Inputs corresponding to actions are denoted by "a," and the state-action value estimate is denoted by "Q". Layers follow a "Type (Parameters) Activation Function" notation, or just "Type" for non-parametric functions. "Fc" stands for "Fully connected," and its parameter corresponds to the number of hidden units utilized for that layer (this also applies to the LSTM layer's parameter). "Conv 1D" stands for "1D Convolution," and its parameters represent the kernel dimension, stride, and number of kernels utilized. "MLP" stands for multilayer perceptron, and "LReLU" stands for Leaky ReLU.

TABLE I
TRAINING HYPER-PARAMETERS

	Parameter	Value
DDPG	Actor and Critic learning rates	0.0001, 0.001
	Discount factor γ	0.99
	Batch size	256
	Replay buffer size	100,000
	Smoothing factor τ	0.001
Control	Control frequency (Hz)	5
	$v_x^{\max}$ (m/s)	0.5
	$v_\theta^{\max}$ (rad/s)	1.0

TABLE II
AVERAGE EVALUATION PERFORMANCE OF THE TRAINED AGENTS IN GW1, ACROSS FIVE TRIALS

Agent	SR	TR	CR	Avg. Steps
$\text{SPARSE}_{\text{SW1}}$	0.74±0.14	0.02±0.01	0.25±0.14	87.7± 13.0
$\text{DENSE}_{\text{SW1}}$	0.64±0.11	0.02±0.02	0.34±0.11	123.8± 48.2
$\text{DENSE-STK}_{\text{SW1}}$	0.64±0.08	0.08±0.06	0.28±0.12	136.1± 39.6
PCL_{SW1}	0.85±0.02	0.06±0.05	0.09±0.05	95.8± 20.2
$\text{PCL-LSTM}_{\text{SW1}}$	0.88±0.03	0.08±0.02	0.04±0.02	100.0± 5.7
$\text{SPARSE}_{\text{SW2}}$	0.74±0.07	0.05±0.03	0.22±0.08	119.6± 15.9
$\text{DENSE}_{\text{SW2}}$	0.61±0.18	0.12±0.20	0.27±0.12	187.1±104.6
$\text{DENSE-STK}_{\text{SW2}}$	0.46±0.12	0.24±0.09	0.31±0.15	275.0± 51.1
PCL_{SW2}	0.86±0.04	0.06±0.03	0.08±0.06	95.7± 14.8
$\text{PCL-LSTM}_{\text{SW2}}$	0.88±0.02	0.09±0.02	0.02±0.02	103.4± 11.8
V2R_{SW2r} [3]	0.59±0.11	0.24±0.08	0.17±0.09	186.5± 41.2
ASL_{SW2r} [4]	0.36±0.12	0.12±0.12	0.52±0.23	151.8± 50.0
$\text{PCL-LSTM}_{\text{SW2r}}$	0.88±0.05	0.05±0.03	0.07±0.05	73.7± 11.4

noise is turned off). For each trial, and omitting the evaluation process, training took about 3 to 4 hours for the SPARSE, DENSE and baseline models, and 5 to 6 hours for the DENSE-STK, PCL and PCL-LSTM models. Fig. 3 shows the average and standard deviation for the success rate, the average number of steps, and the average un-discounted reward obtained per episode, across five independent trials per model. Fig. 3(a) and 3(b) show the performance evaluation of the SPARSE, DENSE, DENSE-STK, PCL and PCL-LSTM models in SW1 and SW2, respectively. Fig. 3(c) shows the performance evaluation of the baselines V2R [3] and ASL [4], and our proposed model, PCL-LSTM, in SW2r.

From the obtained results, it is observed that almost all models consistently learn proficient local-planing policies in their respective worlds. The performance differences between models trained in SW1 are quite subtle compared to those observed for models trained in SW2, where PCL and PCL-LSTM display superiority in terms of success rate. The peak in the average number of steps per episode displayed by some models at early stages of training can be explained because, in those cases, the agents quickly adopt a collision avoidance behavior, being able to survive during whole episodes, but unable to reach the navigation targets.

To evaluate the generalization capabilities of the trained models when deployed in unseen environments, as well as their response to slightly different dynamics, we constructed two worlds in the Gazebo simulator, GW1 and GW2 (Fig. 4).

All the trained agents were first evaluated in GW1 (Fig. 4(a)), where they must reach valid target destinations starting from the world's origin, for 500 episodes. At each episode the target is randomly changed, and the agent receives it in local coordinates using a ground-truth localization system. Episodes are considered successful if the agent reaches the target, and unsuccessful if it collides, or more than 500 agent-environment interactions pass. The results for this experiment, averaged across all the trained instances, are shown in Table II, where the subscript on each model's name refers to the world it was trained in, SR

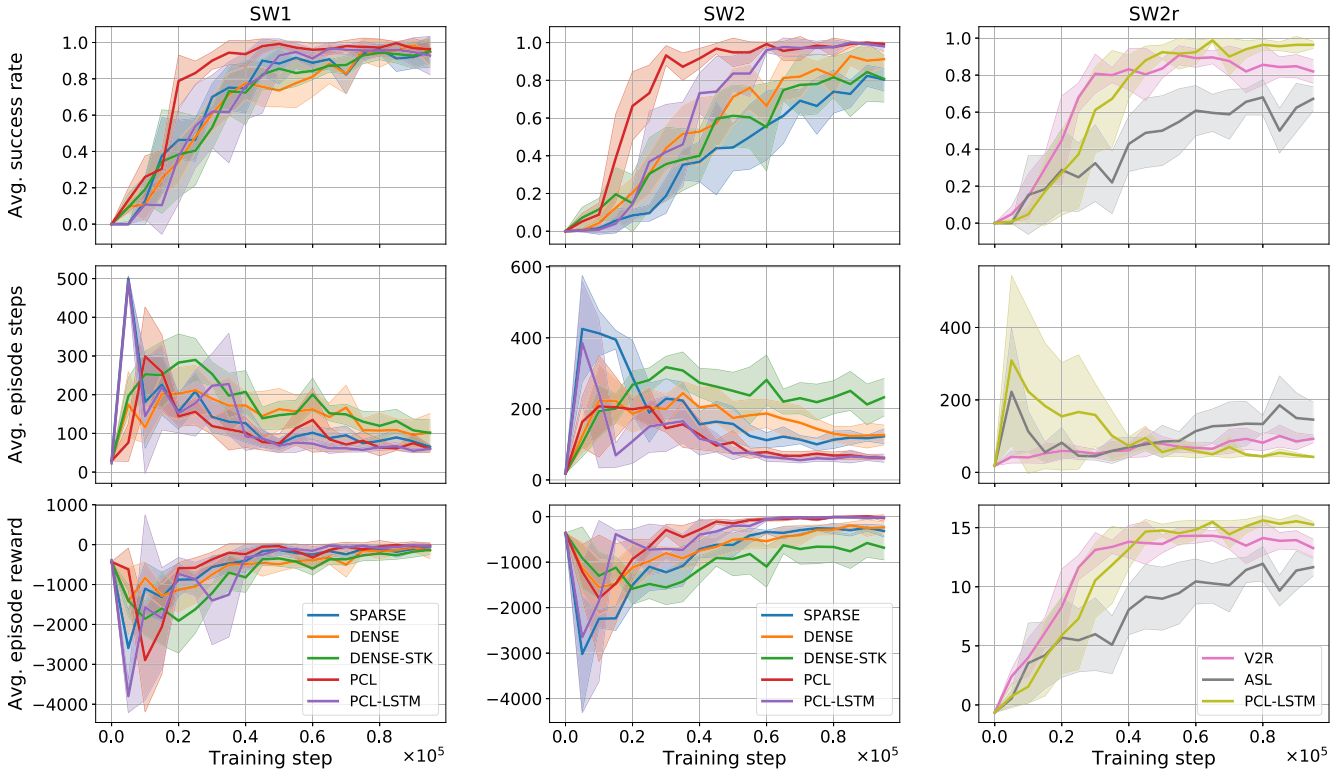

Fig. 3. Evolution of success rate, average number of steps per episode, and obtained reward during the training process of the studied policy parameterizations. (a) and (b) show the evolution curves for the models trained in SW1 and SW2, respectively. (c) shows the evolution curves for PCL-LSTM and the baseline models trained in SW2r.

(a) Gazebo world 1 (GW1) (b) Gazebo world 2 (GW2)

Fig. 4. Simulated worlds used to validate the performance of the trained models. (a) GW1 is 10 m long by 10 m wide, whilst (b) GW2 is 20 m long by 20 m wide.

stands for "Success Rate," TR for "Time-out rate," and CR for "Crash Rate."

The results show that PCL and PCL-LSTM are the best performing models across those trained in SW1 and SW2. Furthermore, the SR displayed by these models on GW1 is almost the same regardless of the world they were trained in. It is observed that including LSTM layers to the architecture that we propose is beneficial, as PCL-LSTM displays an overall better performance than PCL, showing a lower CR. It is also observed that PCL-LSTM outperforms the studied baselines by a large margin. Moreover, PCL-LSTM models display almost the same performance in terms of SR, regardless of the world and reward function that was utilized to train them. The results also show that the DENSE and DENSE-STK models show a lower performance compared to the SPARSE model. This can be due to the SPARSE model having lower dimensional observations, thus, making the training process easier. The foregoing demonstrates the importance of designing adequate observations and policy parameterizations for RL-based local planning, as they noticeable influence the agent's performance.

While there are some differences in performance when comparing those model instances trained in SW1 and SW2 (e.g. PCL_{SW1} vs PCL_{SW2}), the DENSE-STK models trained in SW1 vastly outperform the DENSE-STK models trained in SW2. We hypothesize that stacking observations makes this model prone to over-fitting: the same distribution of stacked observations experienced in a complex world is unlikely to be experienced again in an unseen environment.

We conducted a second validation experiment, this time on GW2 (Fig. 4(b)), where long-term planning is required to reach the targets. For this experiment, the AMCL localization system,[3] and the A* global planner[4] are used. Only the best performing models from the previous experiment, the PCL-LSTM models trained in SW2, were subjected to this evaluation. In this experiment, the agents are guided towards the navigation goal by a sequence of way points obtained by sampling the global plan generated by A*. The same performance metrics (SR, TR, and CR) are measured for 500 episodes, and averaged across five

[3][Online]. Available: http://wiki.ros.org/amcl
[4][Online]. Available: http://wiki.ros.org/action/fullsearch/global_planner

TABLE III
COMPARISON BETWEEN PCL-LSTM INTEGRATED WITH LOCALIZATION AND GLOBAL PLANNING, AND `Move_Base` IN GW2

Agent	SR	TR	CR	Avg. Steps
PCL-LSTM$_{\text{GW2st}}$	0.90±0.09	0.07±0.10	0.03±0.04	295.1±60.8
`move_base`$_{\text{GW2st}}$	0.85±0.05	0.13±0.07	0.02±0.02	287.2±56.3
PCL-LSTM$_{\text{GW2rnd}}$	0.67±0.09	0.17±0.11	0.16±0.04	313.0±42.6
`move_base`$_{\text{GW2rnd}}$	0.42±0.10	0.21±0.12	0.37±0.15	192.3±65.2

TABLE IV
EVALUATION RESULTS FOR DIFFERENT PERTURBATIONS APPLIED TO THE OBSERVATIONS OF THE PCL-LSTM AGENTS IN GW1

Variant			SR	TR	CR	Avg. Steps
FoV	$\lvert o_{\text{pcl}}\rvert_{\max}$	AGGR				
180°	128	✗	0.87±0.02	0.09±0.03	0.03±0.02	105.5±13.4
120°	128	✗	0.83±0.04	0.11±0.03	0.06±0.04	116.1±14.4
90°	128	✗	0.76±0.10	0.15±0.09	0.09±0.05	135.8±39.4
240°	64	✗	0.88±0.02	0.09±0.03	0.03±0.03	104.0±13.6
240°	32	✗	0.87±0.02	0.10±0.03	0.03±0.03	106.1±12.7
240°	16	✗	0.85±0.04	0.10±0.03	0.05±0.05	107.6±14.9
180°	128	✓	0.87±0.02	0.07±0.04	0.06±0.05	96.7±16.0
120°	128	✓	0.86±0.04	0.07±0.02	0.07±0.05	102.0±14.2
90°	128	✓	0.83±0.08	0.10±0.03	0.08±0.06	119.4±24.3

trials. The PCL-LSTM agents are compared to `move_base`.[5] Two variants of GW2 are considered: a static variant, GW2st, where the only obstacles for the agent are the environment's walls, and a randomized variant, GW2rnd, where different objects are randomly placed in the world, changing their position at the beginning of each episode (Fig. 4(b)). The results obtained for evaluation are displayed in Table III.

The results for this experiment showcase the reactive nature of the proposed approach, as it vastly outperforms `move_base` when deployed in GW2rnd, where not mapped obstacles are present. For the experiments conducted in GW2st, our approach also outperforms `move_base`, as the latter often gets stuck in narrow passages.

Finally, the response of PCL-LSTM to perturbations on its observations was evaluated in GW1. We studied changes in performance when (i) decreasing the agent's FoV, (ii) restricting the maximum number of points conforming the observed point cloud ($\lvert o_{\text{pcl}}\rvert_{\max}$), and (iii) limiting the agent's FoV, but performing an aggregation (AGGR) through time of the resulting point clouds using odometry information, and filtering points outside a radius defined by the agent's sensor max. range (5.6 m). This sensory integration was achieved as in the construction of the "local maps" described in [17]. The results for this experiment are shown in Table IV.

It is observed that the performance of PCL-LSTM decreases when restricting the agent's FoV. As training was conducted using a 240° FoV simulated LiDAR, this result is expected. This limitation, however, can be partially addressed by performing sensory integration, as indicated by the performance improvement observed when aggregating observations. In addition, the results show that the model only experiences a slight performance detriment when the sensor's resolution is decreased. The results support our hypotheses regarding the system's flexibility from a practical view point. As the trained agents allows variable sized inputs, the use of cheap, low resolution sensors at deployment time is possible. It also permits using different techniques, such as point cloud registration and sensor fusion, to artificially augment the agent's perception.

[5][Online]. Available: http://wiki.ros.org/move_base

D. Validation in the Real-World

To validate the applicability of our approach in the real-world, we evaluated the performance of PCL-LSTM when being integrated in a full navigation stack, and deployed in two environments: a small room resembling an indoor space, and a long corridor. We relied on AMCL for localization, and A* for global planning. In this experiment, the agent is guided towards the navigation goal by way points sampled from the plan generated by A*.

We augmented the robot's perception by using an RGB-D camera to project range readings (from depth maps) into a two-dimensional space, as in [15]. These readings are then coupled with those provided by the laser range finder mounted on the robot. This allows the robot to detect objects inside the camera's FoV which may not be fully observed by 2D perception, such as chairs and tables.

Furthermore, the navigation system was deployed on-board: the processing required to localize, generate global plans, sample way points, pre-process sensor readings, and perform policy inferences was performed by a Raspberry Pi 3 Model B, mounted on the Pioneer 3DX. The PCL-LSTM model was able to run in real-time, with an average maximum inference time of about 75 ms (measured by feeding point clouds conformed by 128 points to the controller).

During the real-world evaluation, different valid target locations were queried to the robot sequentially in the two considered environments. Examples of the trajectories that were performed by the agent are shown in Fig. 5(a)–(b) (indoor-like room) and Fig. 5(c) (large corridor). During the corridor experiments, sometimes humans acted as dynamic obstacles. The execution of some of the trajectories displayed in Fig. 5 can be seen in https://youtu.be/AzvRJyN6rwQ.

To get a quantitative measure of the agent's real-world performance, we conducted three experiments to evaluate its behavior on challenging tasks: obstacle avoidance through augmented perception (chair avoidance), fine-grained control (passing through doors), and short-time planning (passing blocked paths). We performed 30 trials per task, and recorded the metrics used for simulated validation (SR, TR, and CR). As we aimed to validate the local planner performance, we did not rely on A* (as in the previous experiment). The obtained results are summarized in Table V.

These results show that the trained agent is able to perform well in real-world environments, under the presence of noisy sensor measurements, localization errors, dynamic obstacles, and actuation delays. Furthermore, the system is able to (i)

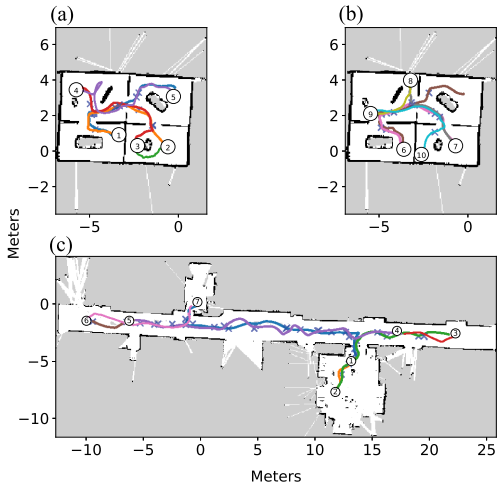

Fig. 5. Examples of the performed trajectories by the PCL-LSTM model in (a)–(b) an indoor-like environment, and (c) a large real-world corridor.

TABLE V
RESULTS OBTAINED IN ISOLATED REAL-WORLD EXPERIMENTS

Task	SR	TR	CR
Avoiding chairs	0.93	0.00	0.07
Passing through doors	0.93	0.00	0.07
Passing blocked paths	0.87	0.03	0.10

artificially attain 3D perception by using a simple sensor pre-processing procedure, and (ii) run in a processing constrained platform in real-time.

V. CONCLUSION

In this work, a robust approach for training RL-based local planners was introduced. Our approach allows the use of non-fixed size point clouds as the agent's observations, instead of the more commonly used fixed size range measurement vectors. This endows the trained local planner with flexibility with regards to its observations: the sensor resolution can be changed, and sensor pre-processing can be used to artificially augment the agent's observability on the fly. Although we only explored the idea of using 2D point clouds, richer information can be harnessed by using sets of higher dimensional points. We left this as future work.

REFERENCES

[1] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236
[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, Jan. 2016.
[3] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 31–36.
[4] M. Pfeiffer *et al.*, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4423–4430, Oct. 2018.
[5] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2018, pp. 6276–6283.
[6] K. Lobos-Tsunekawa, F. Leiva, and J. Ruiz-del-Solar, "Visual navigation for biped humanoid robots using deep reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3247–3254, Oct. 2018.
[7] A. Faust *et al.*, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2018, pp. 5113–5120.
[8] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2017, pp. 1527–1533.
[9] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," in *Proc. 1st Annu. Conf. Robot Learn.*, Proceedings of Machine Learning Research, vol. 78, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., Nov. 2017, pp. 185–194. [Online]. Available: http://proceedings.mlr.press/v78/gao17a.html
[10] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2017, pp. 3357–3364.
[11] P. Mirowski *et al.*, "Learning to navigate in complex environments," in *Proc. Int. Conf. Learn. Representations*, 2017.
[12] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente, and P. Campoy, "Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2018, pp. 1024–1031.
[13] J. Jin, N. M. Nguyen, N. Sakib, D. Graves, H. Yao, and M. Jagersand, "Mapless navigation among dynamics with social-safety-awareness: A reinforcement learning approach from 2D laser scans," 2019, *arXiv:1911.03074*.
[14] A. Wahid, A. Toshev, M. Fiser, and T. E. Lee, "Long range neural navigation policies for the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2019, pp. 82–89.
[15] J. Choi, K. Park, M. Kim, and S. Seok, "Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view," in *Proc. Int. Conf. Robot. Autom.*, May 2019, pp. 5993–6000.
[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[17] F. Leiva, K. Lobos-Tsunekawa, and J. Ruiz-del Solar, "Collision avoidance for indoor service robots through multimodal deep reinforcement learning," in *Proc. Robot World Cup XXIII*, S. Chalup, T. Niemueller, J. Suthakorn, and M.-A. Williams, Eds. Cham, Switzerland: Springer, 2019, pp. 140–153.
[18] M. Hüttenrauch *et al.*, "Deep reinforcement learning for swarm systems," *J. Mach. Learn. Res.*, vol. 20, no. 54, pp. 1–31, 2019.
[19] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2016.
[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vision. Pattern Recognit.*, Jul. 2017, pp. 77–85.
[21] N. Heess *et al.*, "Emergence of locomotion behaviours in rich environments," 2017, *arXiv:1707.02286*.
[22] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017.
[23] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2019.
[24] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intell.*, vol. 2, no. 2, pp. 189–208, Dec. 2008. [Online]. Available: https://doi.org/10.1007/s11721-008-0014-4
[25] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. IEEE Intl. Conf. Robot. Autom. Workshop on Open Source Robot.*, Japan, May 2009.
[26] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.) (IEEE Cat. No.04CH37566)*, Sep. 2004, vol. 3, pp. 2149–2154.