Collision Avoidance for Indoor Service Robots through Multimodal Deep Reinforcement Learning

Francisco Leiva[1], Kenzo Lobos-Tsunekawa[1], and Javier Ruiz-del-Solar[1,2]

[1] Department of Electrical Engineering, Universidad de Chile, Chile
[2] Advanced Mining Technology Center (AMTC), Universidad de Chile, Chile
{francisco.leiva, kenzo.lobos, jruizd}@ing.uchile.cl

Abstract. In this paper, we propose an end-to-end approach to endow indoor service robots with the ability to avoid collisions using Deep Reinforcement Learning (DRL). The proposed method allows a controller to derive continuous velocity commands for an omnidirectional mobile robot using depth images, laser measurements, and odometry based speed estimations. The controller is parameterized by a deep neural network, and trained using DDPG. To improve the limited perceptual range of most indoor robots, a method to exploit range measurements through sensor integration and feature extraction is developed. Additionally, to alleviate the reality gap problem due to training in simulations, a simple processing pipeline for depth images is proposed. As a case study we consider indoor collision avoidance using the Pepper robot. Through simulated testing we show that our approach is able to learn a proficient collision avoidance policy from scratch. Furthermore, we show empirically the generalization capabilities of the trained policy by testing it in challenging real-world environments. Videos showing the behavior of agents trained using the proposed method can be found at https://youtu.be/ypC39m4BlSk.

1 Introduction

Collision avoidance is an indispensable ability in mobile robotics. Avoiding a wide range of both static and dynamic obstacles is a key requirement for safe autonomous navigation, autonomous exploration, and multi-agent coordination. Classical formulations to address the collision avoidance problem often consist of several subsystems interacting in a modular fashion. These subsystems extract and process information from the environment, and then use it to plan and execute collision-free trajectories. The main problem that arises when utilizing these classical, modularized systems is that they generally have many adjustable parameters, whose tuning is time consuming and strongly environment-specific.

In this work, we present an end-to-end approach to endow a mobile robot with the ability to avoid collisions using Deep Reinforcement Learning (DRL). The proposed method allows a controller to derive continuous velocity commands for an omnidirectional mobile robot using depth images, odometry based speed

estimations, and laser measurements as inputs. The latter are incorporated despite the use of depth cameras in order to increase the robot's field of view, thus, improving the observability of the problem. The controller is parameterized by a multimodal neural network and trained using the Deep Deterministic Policy Gradient (DDPG) algorithm [8].

We consider perceptual restrictions commonly found in robots equipped with depth and laser sensors. Consequently, we address the limited observability caused by these restrictions by adding Long Short-Term Memory (LSTM) [4] layers to the controller's network architecture [10]. To further enhance observability, instead of directly using laser measurements as inputs, these readings are aggregated in time as a point-cloud during the policy execution, and handled in a way similar to that proposed in PointNet [14].

As large amounts of data are required to learn a proficient collision avoidance policy, we use the Gazebo simulator [7] for the training process, and then directly transfer the trained controller to a real-world platform by leveraging on ROS [15] interfaces. Because of the aforementioned procedure, we also address the reality gap problem that arises due to the mismatch between simulated and real sensor measurements. In this regard, we use a simple processing pipeline to reduce the differences between simulated and real depth images.

As a case study, we consider indoor collision avoidance using the Pepper robot[12], currently the official Social Standard Platform of the RoboCup@Home competition. The Pepper robot has a 3D sensor on its head, and three fixed laser range finders near its omnidirectional base, which provide sparse range measurements. These sensors provide noisy, short-ranged perception, and are used as inputs to the trained collision avoidance controller.

Through simulated and real-world testing, we show that our approach is capable of learning an effective collision avoidance policy from scratch through DRL. Furthermore, we show that the obtained controller is robust to noisy sensors, and is able to behave adequately when deployed in unseen environments.

The main contributions of this work are the following: (i) An efficient approach to train a robust collision avoidance policy in an end-to-end manner. (ii) A novel method to exploit range measurements through sensor integration and feature extraction (akin to PointNet) for a DRL application. (iii) The utilization of a depth image preprocessing pipeline and its validation as a means to improve the performance of the deployed policy in the robot, by reducing the reality gap.

2 Related Work

Obstacle avoidance and similar tasks have been solved by a variety of methods for a long time, with machine learning based methods taking the lead in performance and popularity [5,10,23,17,18]. Methods that rely on labeled data, such as imitation learning, usually require huge amounts of supervised data that is cumbersome to obtain [23,17]. Furthermore, such methods usually limit the system's performance to the ability of the supervisor. On the other hand, DRL-based methods do not require labeled data and are unbounded by the performance

of a supervisor; however, they require large quantities of interactions with the environment to produce good results. Although there are several works on DRL for obstacle avoidance, they either use laser measurements or depth images (thus limiting both performance and applicability) [18,5,21,20], use simplistic environments (halls and fixed environments) [21,20,5], or even fail to converge using just sparse laser measurements [20]. In contrast, we use a multimodal DRL approach [9,13], using both depth images and sparse laser measurements, and consider complex training and validation scenarios.

In some case studies, such as trivial environments, hallways, and some mazes, visual clues and patterns are enough to infer information about regions outside the agent's perceptual range. However, generally, the limitations of such constrained perception must be addressed. A simple approach to overcome the partial observability produced by a limited perceptual range is simply adding more sensors and integrating them using multimodal strategies. However, this method is not always feasible due to physic or monetary contraints. Another common strategy is using recurrent neural networks to alleviate the partial observability [10]. In this work we complement this strategy by integrating sensor readings at different time steps, which allows us to keep a representation of areas outside the perceptual range, while also increasing the density of the measurements.

Finally, we also address the issue of transferring policies trained in simulations to the real world (reality gap) from the perspective of the mismatch between simulated and real observations. Increasingly popular methods to produce real-like observations rely on generative strategies [22], which require labeled data that in the case of depth sensors is difficult or impossible to obtain. Another standard method to produce realistic observations consists of the study of noise and artifact models of 3D sensors [21,11]. In this work, we use a simple pipeline to reduce the mismatch between simulated and real depth images, based on some of the ideas introduced in [6].

3 Preliminaries

We consider the standard RL framework, in which an agent interacts with an environment E in discrete time steps. At each discrete time step t the agent observes o_t, picks an action a_t, and receives a scalar reward r_t. In most real world settings the environment E is partially observed, so the entire history of observation-action tuples may be required to describe the current state s_t, that is, $s_t = (o_1, a_1, ..., o_t, a_t)$.

The environment E is modeled as a Markov Decision Process (MDP) with a state space $\mathcal{S}$, an action space $\mathcal{A}$, transition dynamics $p(s_{t+1}|s_t, a_t)$, and a reward function $r(s_t, a_t)$. The agent's behavior is determined by a policy which may be either stochastic $\pi(a_t|s_t) = \mathbb{P}(a|s)$ or deterministic $a = \pi(s)$.

The return $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ is defined as the total discounted future reward at time step t, being $\gamma \in [0, 1]$ the discount factor, and T a finite time horizon. The goal in RL is to learn a policy that maximizes the expected return starting from an initial state: $\mathbb{E}_{(r_i,s_i)\sim E, a_i\sim\pi}[R_1]$.

4 Collision Avoidance for the Pepper Robot

Collision avoidance can be treated as a sequential decision making problem, where an agent has to navigate through a given environment while avoiding static and dynamic obstacles. In this case we consider indoor environments, and the Pepper robot as the physical agent. To solve this problem using the RL framework we formulate it as an MDP. Namely, the state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $r(s_t, a_t)$ are designed in an ad-hoc fashion.

For the state space we consider processed sensor readings (depth images and laser measurements) and normalized odometry based speed estimations. While depth images are processed primarily to bridge the reality gap due to training in simulations (see Section 4.1), laser measurements are processed to address the limited perception range of the robot (see Section 4.2).

Like several other works on collision avoidance through DRL [5,21,17], velocity commands are used to control the agent. Since Pepper has an omnidirectional mobile base, an action at time t is defined by the tuple $a_t = (v_x^t, v_y^t, v_\theta^t)$, where v_x^t and v_y^t correspond to the robot's instantaneous linear speeds in the x and y-axis, whereas v_θ^t to its angular speed around the z-axis.

The reward function is designed based on several agent's desirable behaviors when interacting with the environment. To encourage navigation through safe, collision-free regions, the reward defined in Equation 1 is utilized.

$$r_t = \begin{cases} \dfrac{v_x^t}{v_x^{\max}} \cos\left(\dfrac{\pi}{2} \dfrac{K_{v_\theta}^t}{v_\theta^{\max}}\right) & v_x^t \geq 0 \\ \dfrac{v_x^t}{v_x^{\max}} & v_x^t < 0 \end{cases} \tag{1}$$

The variables $v_x^{\max}$ and $v_\theta^{\max}$ correspond to the maximum values for v_x and v_θ, whilst $K_{v_\theta}^t$ is defined in Equation 2.

$$K_{v_\theta}^t = \min\left\{\max\left\{\left|v_\theta^t\right|, \left|v_\theta^t - v_\theta^{t-1}\right|\right\}, v_\theta^{\max}\right\} \tag{2}$$

To prevent the agent from getting dangerously close to obstacles, a punishment signal is provided according to Equation 3, where $l_{\min}^t$ denotes the distance to the nearest obstacle according to the processed laser measurements at time t, and K is a constant set to 1.5. A stronger punishment $r_t = -10$ is given to the agent whenever it fails to avoid an obstacle.

$$r_t = K(l_{\min}^t - 1), \quad l_{\min}^t < 0.6 \tag{3}$$

In this work, the collision avoidance problem is modeled as an episodic task, where an episode ends when the robot collides, or when a fixed number of time steps is reached. To encourage generalization, each episode has randomized initial poses for the robot and most of the obstacles of the environment.

4.1 Depth Images Processing

There is a big difference between simulated depth images and those obtained from the Pepper's 3D sensor. Simulated depth images obtained from Gazebo are ideal, completely noise-free and undistorted. Pepper's 3D sensor readings, on the contrary, usually have many patches of lost, mixed and noisy pixels, and present a clear distortion in the predicted relative distance to some objects. Due to this mismatch, directly transferring the trained policy in simulations to the real world would be unfeasible.

To solve the aforementioned issue, both simulated and real depth images are subjected to a preprocessing stage. Because of the information loss inherent to the 3D sensor embedded in Pepper, it is almost impossible to reconstruct accurate depth maps, specially when the scenes include several shiny or dark objects. Given this limitation, we focus on improving the real sensor readings, but also on corrupting the simulated depth images so they resemble their real-world counterpart.

Mixed and lost pixels are usually found near the border of different objects in a depth map. Therefore a Canny edge detector [2] is applied over the simulated depth images to find regions where pixel corruption would exist. The edge detector is also applied over the depth map's corresponding RGB image, as some edges cannot be directly extracted from the simulated depth map. The extracted edges are aggregated and filtered since some of them arise because of textures, which are usually not associated to places where corruption in real depth images is present. To filter these edges, a Hough transform [3] is applied over the aggregated edge map to detect lines, and the value of pixels parallel to each of the detected lines is compared. If the difference between the average value of the compared pixels is below a threshold, the edge associated to the detected line is likely to be produced because of textures, and consequently is discarded. Finally, the filtered edge map is dilated, and pixel patches are marked over random-sized regions whose position matches the filtered edges. The resulting mask resembles the distribution of corrupted pixels of real depth images, and is applied over the original simulated depth image.

For the real depth images, we only tackle the problem introduced by lost pixels, which we found to be the main cause of image distortion of the real 3D sensor used. A mask is constructed to mark all the lost pixels in the real depth images, and then their value is predicted using Telea's inpainting technique [19]. The same procedure is applied over the corrupted simulated depth maps. Finally, both real and simulated images are decimated (and subjected to an anti-aliasing filter) to reduce their dimensionality to 60×80 pixels. Fig. 1 shows a diagram of the described pipeline.

4.2 Laser Integration

Laser range information is usually used in DRL applications as fixed dimensional representations. Some examples are a fixed-number of sparse laser readings [18], a fixed number of candidates from a dense laser sensor [16], and image-like

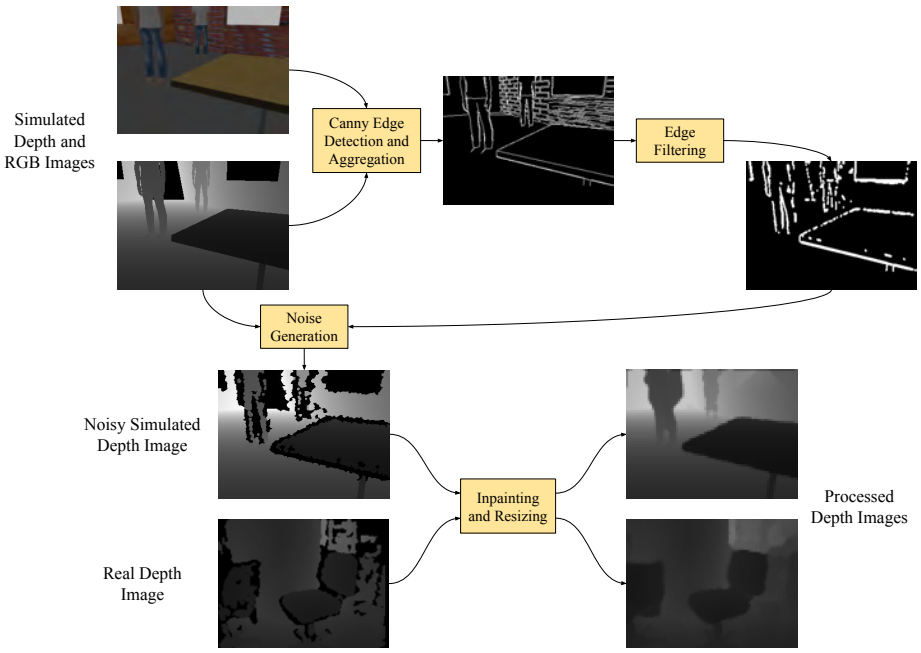


Fig. 1: Diagram of the preprocessing pipeline for simulated and real depth images.

representations [13] (e.g 3D LiDAR). For sparse lasers and fixed candidates, their distribution and resolution become critical in the observability of the MDP, but a high dimensionality may produce slow learning since in most cases these elements are fed to fully connected layers, which do not necessarily exploit the nature of the representation. While image-like representations do exploit the nature of the readings through convolutional layers, all these representations fail to identify openings (areas without obstacles in the sensor range) since in those cases the readings must be saturated to their maximum value to maintain a consistent representation. Furthermore, in all cases, the representations only consider obstacles in a direct line of sight, failing to model complex environments.

In this work we construct and unordered, size-independent representation of the environment, which we will refer to as "local map". This local map is made by simply integrating laser scans across time in a fixed-frame, and storing them as a point-cloud. In in this case, the fixed-frame utilized to perform such integration is set by the robot's odometry. As odometry also integrates the error of the encoders, navigating through large environments using this methodology produces clearly distorted local maps. To address this issue, the laser scan integration is limited to a fixed distance D from the robot's local frame, as points within this range are less likely to be distorted by the odometry error. In the case of the Pepper robot, D is set to 6.5 m.

As environments are subject to change due to dynamic obstacles, we also include a point-removal step to be able to model this phenomenon. To do this, points from the local map that should be perceived in the current laser scan but are not, are discarded. This process includes a tolerance in the criterion, which is set to the maximum angular error of the laser. This step also helps reducing the distortion of the map, since non-coherent points are deleted. Fig. 2 shows the difference between raw laser range readings and the proposed representation of the environment.

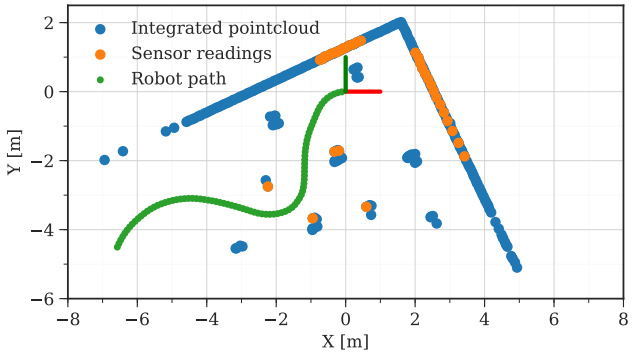


Fig. 2: Difference between raw laser measurements and the proposed odometry based local map. The green dots represent the trajectory followed by the robot, whilst orange points correspond to its raw laser range measurements. The blue points represent the obtained local map, constructed by integrating and filtering laser readings.

The advantages of this representation are clear. It encapsulates information to distinguish between opening areas from objects at the saturation distance. It also allows to model the environment with increasingly precision as the robot navigates through it, while modeling out-of sight objects. Finally, it allows the use of cheaper sensors, as it can artificially augment the sensor's resolution.

4.3 Algorithm and Networks' Architecture

To train the collision avoidance policy we use DDPG [8], a continuous control DRL algorithm. Since DDPG is an actor-critic method, two independent neural networks are designed to parameterize the policy (actor) and the state-action value function (the critic). In this work, both of these networks share almost the same structure, differing only in their inputs and outputs.

As we combine different sensor measurements to conform the state space, a multimodal strategy is followed to perform sensor fusion. In this regard, every sensor is processed independently by different feature extractors (which are part of the same neural network), and the resulting embeddings are concatenated. However, given that the laser measurements representation (the local map) has

a non-fixed dimensionality, neither fully-connected nor convolutional layers can directly take it as input. Instead, we follow the approach presented in [14], using shared weights and symmetric activation functions to go from a variable input size to a fixed dimensionality embedding.

The approach followed consists of two steps. First, every point from the map is processed by a series of fully connected layers independently (the same fully connected layers are applied to every point). Then, the outputs from this operation are aggregated using max pooling to obtain a global feature vector, which encapsulates the input's information in a fixed-dimensional representation. Even though the number of points conforming the local map may vary at every time step during the policy execution, we select a fixed number of points from this representation during training to allow batch processing. If the number of points from the local map exceeds the fixed selected number of points, uniform sampling is performed. On the contrary, if the points from the local map are not enough, we randomly repeat points until the requirement is satisfied. At test time, the complete local map is utilized.

The architectures of the designed actor and critic are summarized in Fig. 3, where hyper parameter and design choices were primarily taken from [10]. The depth image feature extractor is conformed by three convolutional layers: Conv1 with 16 8×8 filters (stride 4), Conv2 with 32 4×4 filters (stride 2), and Conv3 with 32 2×2 filters (stride 2). The feature extractor for the local map consists of three fully connected layers of 64, 64 and 1024 hidden units respectively. Finally, the feature extractor for the odometry speed estimations (and actions, for the critic) consists of two fully connected layers of 32 and 64 hidden units each.

The embeddings obtained from these modules are concatenated and used as input for a fully connected layer (FcFus), which has 200 hidden units. The output from this layer goes to an LSTM layer, and then to another fully connected layer (FcPre), both having 200 cells/hidden units. The final layer (FcOut) outputs the action for the actor and the state-action value for the critic, respectively.

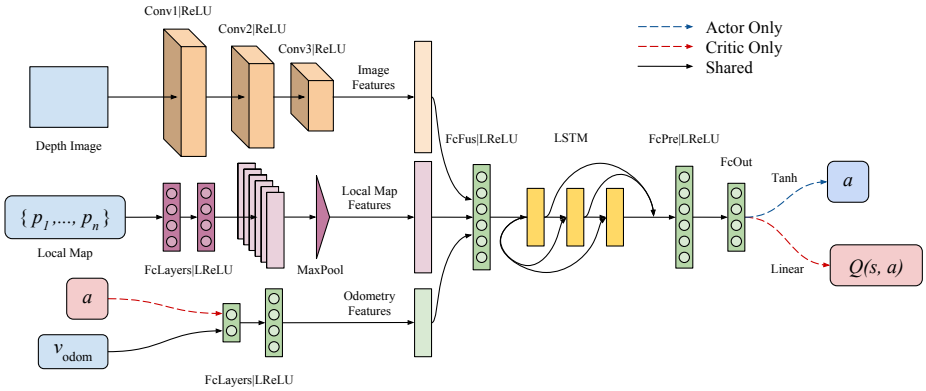


Fig. 3: Representation of the proposed multimodal network architecture.

We use LSTM layers to address the partial observability due to the robot's sensors limitations, that is, to rely on several observations to approximate the actual environment state at each time step. These layers process the concatenated embeddings obtained from the inputs, thus, being able to integrate sensory information through time. To effectively train the neural network with these layers, uniform sampling from the replay buffer cannot be performed. Instead, the method proposed in [10] is utilized to sample sequential experience traces.

5 Evaluation Results

5.1 Experimental Setup

The training environments are constructed using the Gazebo simulator [7], whilst TensorFlow[1] is used to implement DDPG and the associated neural network models. To bridge the simulations with the training algorithm, we rely on ROS [15] interfaces.

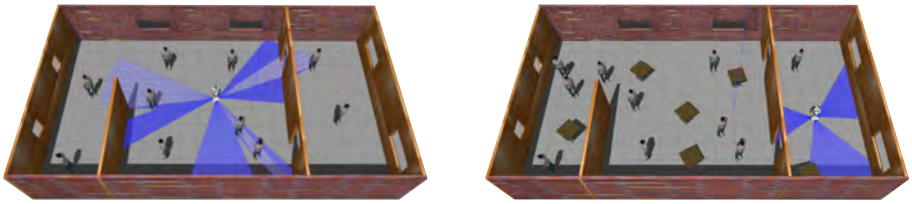
Episodes are limited to 500 time steps, but may end prematurely whenever a collision between the agent and any obstacle is detected. At every time step, a velocity command is executed for 400 ms. The maximum instantaneous speeds for the Pepper robot are set to 0.5 and 0.3 m/s for the x and y-axis, and 0.5 rad/s for the angular speed around the z-axis.

We utilize a batch size of 64 for training, whereas learning rates, weight initializations and exploration strategy remain the same as in the original DDPG paper [8]. All processing is conducted in a laptop equipped with an Intel i7-7700HQ processor, and a Nvidia GeForce GTX 1060 GPU. With the available hardware, 7,500 training steps take approximately one hour.

5.2 Simulation Results

To validate our method in simulations, we constructed a virtual environment consisting of a room of 18 m long by 10 m wide. This room has internal and external walls, windows, doors, and static obstacles placed over free regions (see Fig. 4). Using this environment, three experiments were conducted.

To validate the effectiveness of laser integration, we compare the performance of agents trained using the local map representation (Experiment 1) against agents trained using normalized sparse range measurements (Experiment 2). In these experiments 3D perception is not available, as we limit the state space to processed laser readings and odometry based speed estimations. To account for this, the environment's obstacles are constrained to objects that can be completely perceived using laser readings (see Fig. 4a). To test the complete proposed system, we train a policy which uses processed depth images, integrated laser measurements (the local map representation), and odometry based speed estimations (Experiment 3). In this case the environment's obstacles include tables, as 2D range measurements are not enough to model their collision geometry (see Fig. 4b).



(a) **(b)**

Fig. 4: Virtual environments constructed for training and validation in simulations. (a) Environment used for Experiments 1 and 2. Contains 10 simple obstacles (persons). (b) Environment used for Experiment 3. Contains 10 simple obstacles (persons) and 6 complex obstacles (tables).

To evaluate the performance of the trained agents, two metrics are utilized: the total reward obtained in an episode $R = \sum_{t=1}^{T} r(s_t, a_t)$, and the average instantaneous forward speeds of an episode (penalizing high angular speeds) $V_{\sum} = \frac{1}{T} \sum_{t=1}^{T} v_x^t \cos(v_\theta^t)$.

Fig. 5 shows the obtained results for the conducted experiments. The training process was limited to $7.2{\times}10^4$ time steps (approximately 10 hours per trial) for Experiments 1 and 2 (Fig. 5a and 5b). For Experiment 3 (Fig. 5c and 5d), training was conducted for $11{\times}10^4$ time steps (that is, for approximately 15 hours per trial). Every $8{\times}10^3$ and $10{\times}10^3$ time steps, respectively, 50 validation episodes are performed (the policy is executed directly, i.e., the exploration is turned off). The graphs show the average and standard deviations from 10 independent trials.

The obtained results show that policies trained using the proposed method (using both high and low dimensional state spaces) are able to efficiently avoid obstacles in the constructed virtual environment. From Fig. 5a and 5b it is clear that agents trained using raw sparse laser measurements fail to get a good performance, as the environment's observability is heavily limited by the robot's perception range. In contrast, including the local map in the state space allows the trained agents to learn proficient obstacle avoidance policies, thus, validating the benefits of such representation.

Fig. 5c and 5d show that agents trained with the proposed method consistently get good performance, regardless of the high variance across training and validation episodes due to robot and obstacles initial pose randomization. This also shows the ability of the proposed method of combining multisensory inputs, as the trained policies successfully learn to avoid tables in spite of contradictory information between lasers and depth images (the constructed local map detects tables as point obstacles or collision free regions, and their 3D geometry is only captured by the depth images).

There is a slight reduction in the overall performance and its variance (both in terms of the reward and average speeds) compared to the results obtained using only the local map (Fig. 5a and 5b); however, this is due to the difference

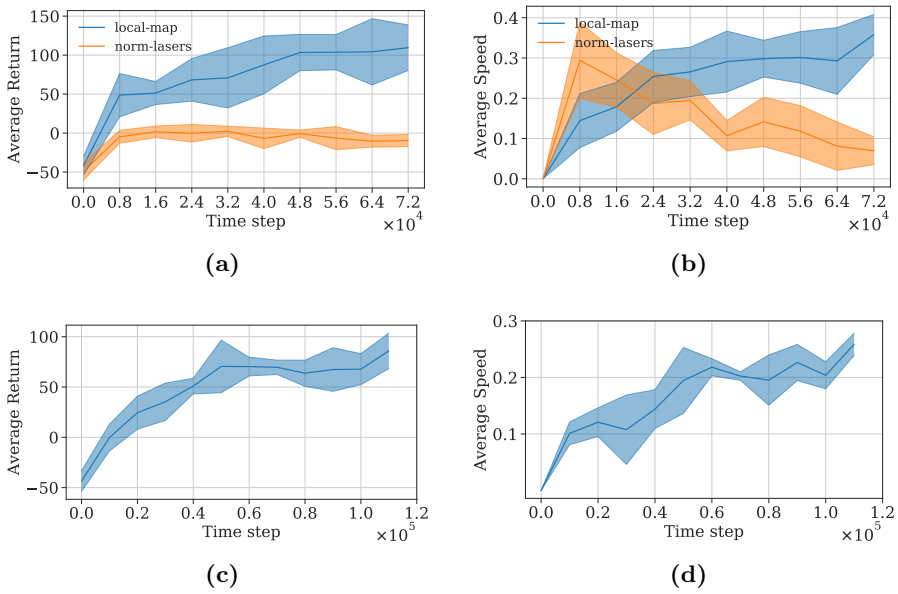


Fig. 5: Performance evaluations for the experiments carried out. (a) and (b) show the comparison between agents trained using the proposed local-map representation, and agents trained using normalized sparse range measurements (Experiments 1 and 2). (c) and (d) show the performance of agents trained using the complete proposed method (Experiment 3).

between the number of obstacles utilized for Experiment 1 and Experiment 3 (refer to Fig. 4). As we randomize the initial pose of the robot and the obstacles for each episode, a small number of obstacles allows the existence of easy episodes (which results in occasionally higher rewards). On the other hand, the difficulty is higher, but remains similar across episodes, when the environment is highly cluttered (which results in lower rewards).

5.3 Validation on the Physical Robot

The collision avoidance policy trained using the full proposed method in simulations (Experiment 3) was directly transferred to a real Pepper robot by using external processing and ROS interfaces. Depth images and laser measurements were manipulated according to the methodology described in Sections 4.1 and 4.2, respectively. To evaluate the performance of the policy, we considered three different indoor environments: A study room (static obstacles), a hall (static and dynamic obstacles) and a cluttered laboratory (dynamic obstacles, maze-like environment).

The agent was able to navigate successfully through collision free regions in all these environments, regardless of the variety of unseen obstacles it encoun-

tered with respect to simulations. Furthermore, it behaved adequately even when undergoing maze-like environments. Examples of the trained policy deployed in real world environments can be found at `https://youtu.be/ypC39m4BlSk`.

(a) **(b)** **(c)**

Fig. 6: Real work environments utilized to validate the trained collision avoidance policy. (a) Study room, (b) hall, and (c) laboratory.

The generalization capabilities of the trained policy may be explained considering that (i) simulated and real local maps are similar, and (ii) the avoidance of 3D obstacles in the real world only requires coarse information regarding their shapes, which can be obtained using the processed depth images. These conditions allowed the policy to perform well in the testing environments even though it was trained with limited information in simulations.

On the other hand, it was observed that the policy's performance was severely undermined when the robot was deployed in environments where outdoor lighting was present, as Pepper's depth sensor produced highly corrupted images under such conditions. The same performance detriment was observed when unprocessed depth images were used as observations. This behavior can be explained considering the domain mismatch between simulated depth images used for training the policy, and the heavily corrupted depth maps that were being fed to the policy in the aforementioned scenarios.

6 Conclusions and Future Work

In this work we presented a successful approach to solve the obstacle avoidance problem using the Pepper robot, with experimental validations in real-world environments. We also presented a novel approach to use point-clouds, and more generally, unordered and non-fixed dimensional inputs for RL applications. Finally, a simple preprocessing stage to bridge the reality gap between simulated and real depth images was presented and validated.

Although in this work we only tackled the obstacle avoidance problem, similar tasks, such as visual navigation, could be solved relying on the proposed modelling. Furthermore, during our experiments, the real world performance of the policy was strongly dependent of the input preprocessing applied during training and deployment. Consequently, realistic input simulation and domain adaptation arise as interesting areas of study for future work.

Finally, in this implementation we integrate laser scans using encoder-based odometry to construct the unordered representation of laser measurements. This greatly limits the quality of the integration. We propose the use of proper point cloud registration (for instance, using Iterative Closest Point) when LiDAR information allows the use of such methods, and also the use of point-clouds produced by methods such as visual SLAM as inputs of our policy. Furthermore, these methods could also be applied to register point-clouds from the depth sensor, as the local map feature extraction extends to 3D points.

Acknowledgements

This work was partially funded by FONDECYT Project 1161500 and CONICYT-PFCHA/Magíster Nacional/2018-22182130.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org
2. Canny, J.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-8**(6), 679–698 (Nov 1986). https://doi.org/10.1109/TPAMI.1986.4767851
3. Duda, R.O., Hart, P.E.: Use of the hough transformation to detect lines and curves in pictures. Commun. ACM **15**(1), 11–15 (Jan 1972). https://doi.org/10.1145/361237.361242
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997). https://doi.org/10.1162/neco.1997.9.8.1735
5. Kahn, G., Villaflor, A., Ding, B., Abbeel, P., Levine, S.: Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 1–8 (May 2018). https://doi.org/10.1109/ICRA.2018.8460655
6. Kim, S., Kim, M., Ho, Y.: Depth image filter for mixed and noisy pixel removal in rgb-d camera systems. IEEE Transactions on Consumer Electronics **59**(3), 681–689 (August 2013). https://doi.org/10.1109/TCE.2013.6626256
7. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). vol. 3, pp. 2149–2154 vol.3 (Sep 2004). https://doi.org/10.1109/IROS.2004.1389727
8. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: ICLR (2016)
9. Liu, G.H., Siravuru, A., Prabhakar, S., Veloso, M., Kantor, G.: Learning end-to-end multimodal sensor policies for autonomous navigation. In: Proceedings of the 1st Annual Conference on Robot Learning. Proceedings of Machine Learning Research, vol. 78, pp. 249–261. PMLR (13–15 Nov 2017)

10. Lobos-Tsunekawa, K., Leiva, F., Ruiz-del-Solar, J.: Visual navigation for biped humanoid robots using deep reinforcement learning. IEEE Robotics and Automation Letters **3**(4), 3247–3254 (Oct 2018). https://doi.org/10.1109/LRA.2018.2851148
11. Nguyen, C.V., Izadi, S., Lovell, D.: Modeling kinect sensor noise for improved 3d reconstruction and tracking. In: 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission. pp. 524–530 (Oct 2012). https://doi.org/10.1109/3DIMPVT.2012.84
12. Pandey, A.K., Gelin, R.: A mass-produced sociable humanoid robot: Pepper: The first machine of its kind. IEEE Robotics Automation Magazine **25**(3), 40–48 (Sep 2018). https://doi.org/10.1109/MRA.2018.2833157
13. Patel, N., Choromanska, A., Krishnamurthy, P., Khorrami, F.: A deep learning gated architecture for UGV navigation robust to sensor failures. Robotics and Autonomous Systems **116**, 80–97 (2019). https://doi.org/10.1016/j.robot.2019.03.001
14. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. CoRR **abs/1612.00593** (2016), `http://arxiv.org/abs/1612.00593`
15. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: Ros: an open-source robot operating system. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics. Kobe, Japan (May 2009)
16. Sampedro, C., Bavle, H., Rodriguez-Ramos, A., de la Puente, P., Campoy, P.: Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1024–1031 (Oct 2018). https://doi.org/10.1109/IROS.2018.8593706
17. Tai, L., Li, S., Liu, M.: A deep-network solution towards model-less obstacle avoidance. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2759–2764 (Oct 2016). https://doi.org/10.1109/IROS.2016.7759428
18. Tai, L., Paolo, G., Liu, M.: Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 31–36 (Sep 2017). https://doi.org/10.1109/IROS.2017.8202134
19. Telea, A.: An image inpainting technique based on the fast marching method. Journal of Graphics Tools **9**(1), 23–34 (2004)
20. Xie, L., Wang, S., Rosa, S., Markham, A., Trigoni, N.: Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 6276–6283 (May 2018). https://doi.org/10.1109/ICRA.2018.8461203
21. Xie, L., Wang, S., Markham, A., Trigoni, N.: Towards monocular vision based obstacle avoidance through deep reinforcement learning. CoRR **abs/1706.09829** (2017), `http://arxiv.org/abs/1706.09829`
22. Yang, L., Liang, X., Xing, E.P.: Unsupervised real-to-virtual domain unification for end-to-end highway driving. CoRR **abs/1801.03458** (2018), `http://arxiv.org/abs/1801.03458`
23. Yang, S., Konam, S., Ma, C., Rosenthal, S., Veloso, M.M., Scherer, S.: Obstacle avoidance through deep networks based intermediate perception. CoRR **abs/1704.08759** (2017), `http://arxiv.org/abs/1704.08759`