Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 1107 – 1120

doi:10.3906/elk-1907-112

Crash course learning: an automated approach to simulation-driven LiDAR-based training of neural networks for obstacle avoidance in mobile robotics

Stanko KRUŽIĆ[1,*], **Josip MUSIĆ**[1], **Mirjana BONKOVIĆ**[1], **František DUCHOŇ**[2]
[1]Faculty of Electrical Engineering, Mechanical Engineering, and Naval Architecture, University of Split, Split, Croatia
[2]Institute of Control and Industrial Informatics, Slovak University of Technology, Bratislava, Slovakia

Received: 15.07.2019 • **Accepted/Published Online:** 16.01.2020 • **Final Version:** 28.03.2020

Abstract: This paper proposes and implements a self-supervised simulation-driven approach to data collection used for training of perception-based shallow neural networks for mobile robot obstacle avoidance. In the approach, a 2D LiDAR sensor was used as an information source for training neural networks. The paper analyzes neural network performance in terms of numbers of layers and neurons, as well as the amount of data needed for reliable robot operation. Once the best architecture is identified, it is trained using only data obtained in simulation and then implemented and tested on a real robot (Turtlebot 2) in several simulations and real-world scenarios. Based on obtained results it is shown that this fast and simple approach is very powerful with good results in a variety of challenging environments, with both static and dynamic obstacles.

Key words: Autonomous mobile robots, obstacle avoidance, neural networks, simulation-based learning

1. Introduction

Truly autonomous robots need to be able to perform a number of goal-oriented tasks in dynamic environments, using available on-board sensors and computational resources (which are usually somewhat limited). One such task useful in a number of applications, such as surveillance and search-and-rescue, is obstacle avoidance [1]. Due to the uniqueness of different environments, it is usually not convenient or even possible to apply tailor-made obstacle avoidance algorithms for every conceivable situation and environment. Thus, there is a need for a more general approach to the problem. In recent years, reemergence of neural network (NN)-based algorithms has led to some interesting applications in the field of mobile robotics [1–5]. For example, in [2], NNs were used for feature extraction from stereo images as a part of a complex system for long-range vision used in the navigation of autonomous off-road vehicles (robots), while in [3], NNs were used to infer control action to keep the robot on a forest trail. In [4], the authors demonstrated the effectiveness of their NN-based modelless obstacle avoidance algorithm in an indoor environment. This reemergence is mainly due to their generalization ability, which then leads to good results when new and previously unseen samples occur. This, however, requires the existence of large datasets for the training of NNs, whose collection is a time-consuming and labor-intensive task, and one that usually requires human supervision [3, 5, 6]. Thus, it becomes a bottleneck for the whole algorithm development process. A number of different approaches to tackling the issue have been proposed in the literature, but they all, in general, take a significant amount of time and resources and pose a certain risk of damage to the robot and its surroundings [5, 6]. In some instances, they rely on additional

*Correspondence: skruzic@fesb.hr

algorithms, such as various forms of deep reinforcement learning [7, 8]. An additional issue that arises while learning obstacle avoidance for mobile robots is that human operators tend not to collide with objects in the environment while collecting the data, thus limiting the existence of negative examples and consequently the algorithm's performance. The majority of available literature on the subject uses a camera-based approach (although LiDAR-based ones also exist [9], but in significantly lower number and not for ground-based mobile robot obstacle avoidance), in which some form of deep learning is used to interpret the scene captured by a robot-mounted camera and infer control actions [1–5, 10]. While images are inherently information-rich, they do not possess depth information and consume significant computational resources. Of course, depth cameras (like Kinect and RealSense) can be used to avoid the issue [4, 11], but this is then a completely different approach and one better tackled by 2D/3D LiDAR [12]. Another related issue is that training the networks on synthetic images generated in simulation does not necessarily guarantee good real-world performance (due to simulation limitations).[1] Such NNs need to be retrained with additional real-world examples [5], thus prolonging and complicating the deployment process. This is referred to as the reality gap, which may be defined as a difficulty in transferring simulated experience into the real world [12]. There have been various approaches to overcoming the reality gap. In [12], for vision-based grasping, training data were collected in simulation and presented a very challenging problem of transferring obtained data to the real world. In [8], deep reinforcement learning was used with synthetic images to train obstacle avoidance for mobile robots and the authors claimed to have achieved high performance using the proposed approach. In this research, we try to bridge this gap by using as simple data as possible and a simple metric (distance). Thus, the difference between simulation and reality should be reduced. Some approaches have recently been proposed using simulation data for training [12, 13], but there, a human supervisor may still be needed and more complex networks are used. In [13], an approach to training vision-based navigation (flight) in simulation using only synthetic images (i.e. without a single real-world image) was presented. It is reported that by highly randomizing rendering settings for the synthetic images, a policy that generalizes properly in the real world can be trained. In [14], some issues of deep reinforcement learning-based approaches are resolved by using simulation environments for collecting training data for visual navigation. The proposed method generalizes appropriately and can be transferred to a real robot with some small amount of fine-tuning. Closest to our work, the work that motivated us was the research by Gandhi et al. [5], in which an unmanned aerial vehicle was used to collect data for training a deep neural network by randomly crashing it (10,000 times) into objects within its environment. Whereas their work was based on camera data, ours is based on simpler LiDAR data and a simulation environment for data collection (making it safer for both the robot and its surroundings). The rest of the paper is structured as follows. In Section 2, the problem is introduced and formulated. In Section 3 training data generation is explained, while Section 4 describes the NN training procedure. In Section 5, the experimental setup is introduced, and the results are presented and discussed. Finally, in Section 6, conclusions are drawn based on the obtained results.

2. Problem formulation and contribution

The goal of the research was to develop a procedure to train and test an efficient NN architecture for mobile robot obstacle avoidance while keeping human intervention as minimal as possible (i.e. self-supervised nature). The approach should provide numerous positive and negative (i.e. crash) examples to a learning algorithm but

[1] Bousmalis K, Levine S. Closing the Simulation-to-Reality Gap for Deep Robotic Learning. Google AI Blog [online]. Website https://ai.googleblog.com/2017/10/closing-simulation-to-reality-gap-for.html [accessed 22 08 2018].

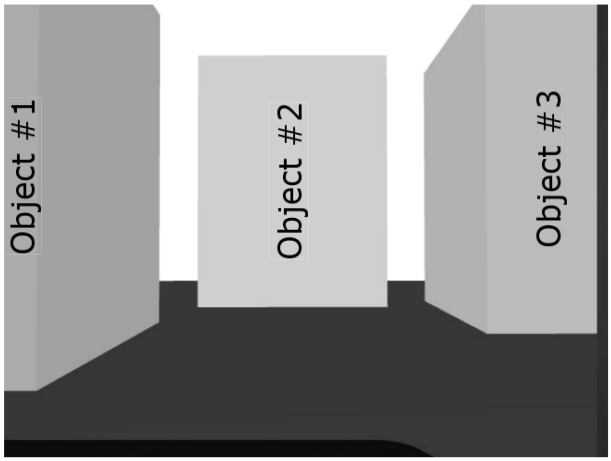
should do that in a manner that is safe both for the robot and its environment (i.e. in simulations). It should also enable the collection of large amounts of data with very little human effort and cost. To that end, it was decided to use 2D LiDAR instead of a camera because, although cameras can provide much more data than LiDAR, LiDAR provides sufficient data for obstacle avoidance. However, it should be kept in mind that LiDAR sensing can fail in some instances (e.g., in the presence of glass surfaces or if the distance is below sensing threshold). Furthermore, the LiDAR output is less complex than video streams, which should translate into simpler NN architecture and shorter training time. This less complex nature of input data has an attractive side-effect in that real-world measurements are more similar to simulation-based ones than in the case of a video stream. Finally, mass-produced and low-cost LiDARs are on the horizon, making them a logical choice for mobile robots' main sensors in the future.

There are two fundamental reasons why simulation was chosen in this research, the first one being that crashing a real robot into obstacles may result in damage to both the robot and the environment, and other being that usage of simulation makes possible the collection of huge amounts of data with different crashing scenarios with very little effort and negligible cost, while achieving a high degree of similarity between data obtained in simulation compared to real-world data by using a LiDAR sensor. An additional reason for using a simulation environment is that the whole approach (including NNs) can be fine-tuned in simulation and deployed only when an acceptable level of performance is achieved (as was done in this research).

Differences between the simulated camera and LiDAR compared to the real-world camera and LiDAR data are illustrated in Figure 1. From the figure, it can be seen that differences exist between simulation and real-world cases both for the LiDAR and camera. However, due to different lighting conditions (which are difficult to reproduce in simulation) and the existence of textures and shadows, the image-based comparison produces larger discrepancies than the LiDAR-based one. Please note that same distance from the robot to obstacles was used in the real-world scenario and in the simulation. It should be noted that recently an approach that uses simple graphics (i.e. not photorealistic) but with a huge number of variations in light and texture (for better generalization) has been investigated and good results were reported [12]. Thus, the contributions of the paper can be summarized as follows:

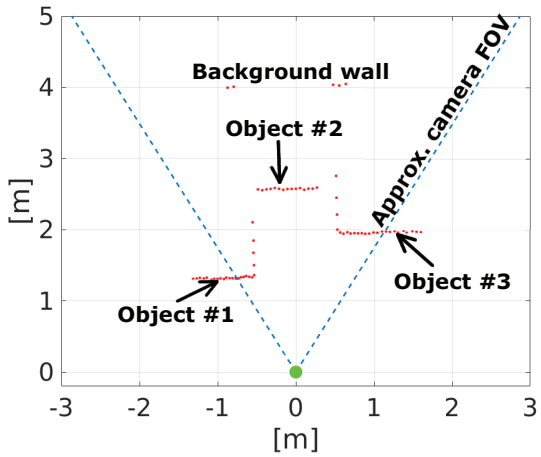
- Development and testing of a simulation-based method that does not require any human intervention during data collection and NN training (thus saving time).
- Simulation-based collection of both positive and negative (i.e. crash) data samples for NN training, which also opens up the possibility of using reinforcement learning-based approaches. Since everything is done in simulation, crashes do not pose a danger to the robot or its surroundings.
- Use of 2D LiDAR sensor/data for obstacle NN avoidance training (and its real-world implementation), while in the literature video-based NNs are usually employed. This, in turn, enabled us to use and train much simpler NNs than in the case of video/picture (e.g., convolutional-type networks) with good performance results (as is demonstrated in this paper).
- Demonstration of seamless transfer (without any retraining) of trained NN to real-world tasks using the Trutlebot 2 mobile robot platform.

Please note that, although simulation-based NN training approaches do exist in the literature (as already presented in Section 1), they usually employ video-based sensors, use complex convolutional networks for obstacle avoidance, do not collect both positive and negative examples, and do not do the NN training process

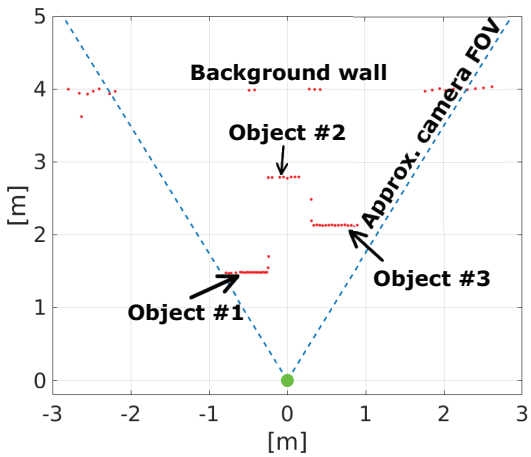


(a) Camera image from simulation environment

(b) Camera image from real robot



(c) LiDAR data from simulation environment



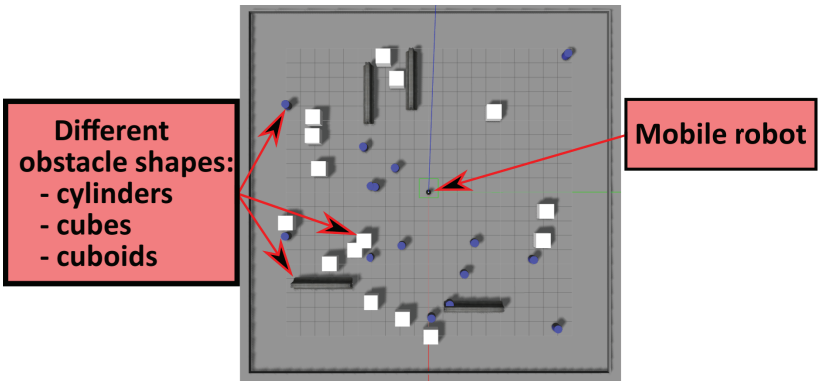
(d) LiDAR data from real robot

Figure 1. Differences between camera images and laser scans in simulation and on the real robot of the approximately same scene.

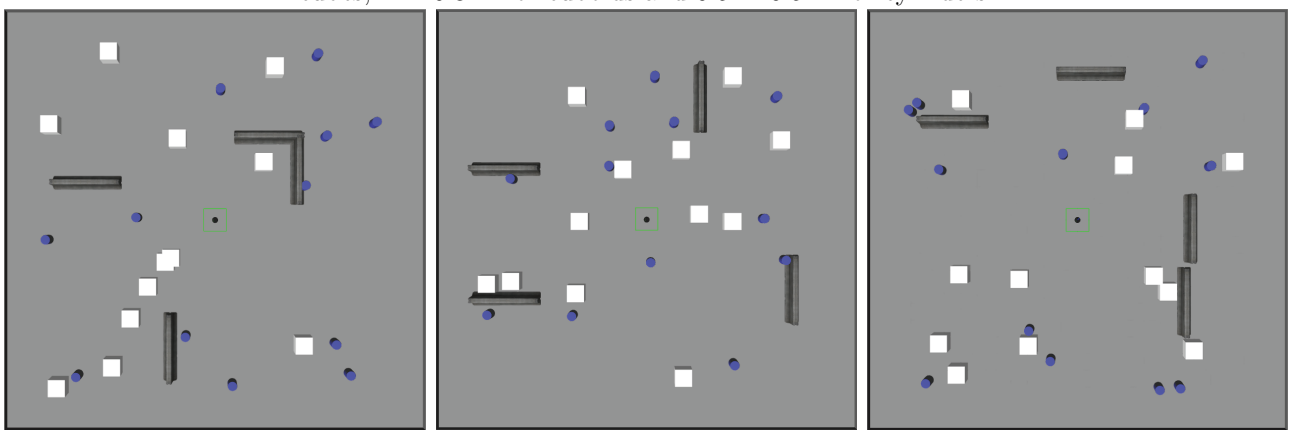
automatically. The focus of this paper is thus to develop an approach that enables the above-mentioned contributions and in the process also answers the following questions: Can simulation-driven LiDAR data be used for training NNs and seamless transfer to real-world applications? How does such an NN perform in the real world and how does it compare to some standard obstacle avoidance algorithms? What is the appropriate NN architecture in such a case and how large should the training dataset be?

3. Training data generation

Training data were generated and collected in the Gazebo simulation environment [14] as follows. First, a simulation was automatically initialized, and various obstacles were randomly scattered within it, as shown in Figure 2. The dimensions of the simulated environment were always kept the same (12.5 m $\times$ 12.5 m) but can be freely chosen/changed by the user. Then a robot was spawned in its center with random orientation and was given a constant linear velocity of 0.4 m/s. While the robot was in motion, LiDAR data were collected. The recording was carried out until the robot crashed into the obstacle or the perimeter wall.



(a) Gazebo world and various obstacle shapes. Please note that obstacles had footprints of 0.5 m 0.5 m for cubes, 4 m 0.8 m for cuboids and 0.5 m 0.5 m for cylinders.



(b) Examples of different randomly generated Gazebo Worlds

Figure 2. Gazebo simulation environment.

Each of the laser range finder scans is timestamped and the timestamp of the moment when the robot hits the obstacle is also recorded. The preceding procedure is repeated a large number of times (specified by the user) to obtain enough data for the NN to generalize properly. In our experiments, we obtained a total of 3754 crash events (in 19 simulation worlds with different obstacle setups), which took about 2 workdays of real-time self-supervised execution (with no human intervention) on a computer with Intel Core i3 6100 processor and 4 GB of RAM. This, in turn, resulted in 396,377 training samples (both positive and negative ones).

Before the NN training, obtained data were preprocessed. First, failed LiDAR measurements were replaced with maximal possible range values. Failed measurements occurred when either the obstacle was closer than the declared minimum sensor range (0.15 m) or farther away than the declared maximum sensor range (6 m). Obtained scan data were then divided into three overlapping groups so each of the three networks is trained on a different dataset. The network for forward motion used scans of 45° to both the left and right of the forward direction (90° in total), while the network for left and right motion used scans of 90° to the left and to the right of the forward direction. This is graphically depicted in Figure 3 from the robot's viewpoint.

The last two seconds of the robot's motion before collision were treated as negative examples (this duration was always guaranteed since there were no obstacles in a 1-m diameter around the robot's initial position), but only for the network(s) representing a side of the robot on which the bumper was activated during the collision,

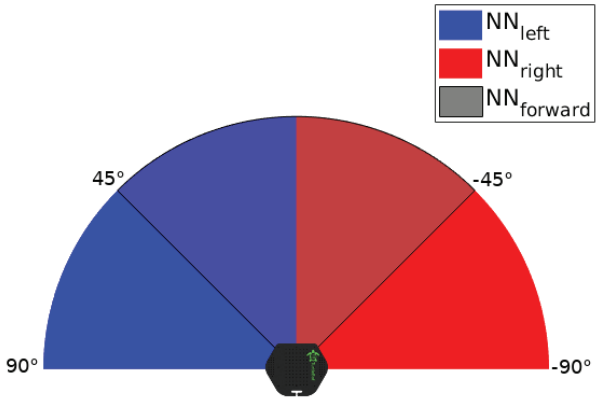


Figure 3. LiDAR scans divided into three overlapping groups (left, forward, right).

while the other sides' network labels remained positive. If a particular measurement lasted longer than 4 s, the first 2 s were treated as positive examples while the remaining (middle) part of the data samples (excluding the last 2 s) was discarded and not used for the training. Otherwise, all the remaining data samples were treated as positive examples.

After the initial labeling, all the labels are further examined and those that are positive are converted to negative if there were at least five laser scan points closer than the predefined threshold (1 m in our case). This is done in order to prevent motion too near to the obstacles that does not result in a crash. An example of such a situation would be a robot moving parallel to the wall. The number of laser scan points (five in our case) and the threshold were chosen experimentally [15] and can be changed depending on user requirements, environment type, and quality/resolution of the used sensor(s).

4. Neural network training

The NN-based controller was designed so that there were three independent NNs: for left, forward, and right directions. Several network architectures were considered and trained using a different number of hidden layers (1 to 4) and different numbers of neurons per layer (20, 40, 60, 80), the results of which are depicted in Figure 4. This amounted to a total of 16 different network architectures per direction (left, right, and forward, giving a total of 48 architectures whose performances were analyzed). Each of the networks was trained as a multilayer perceptron (MLP) using the MATLAB Deep Learning Toolbox, with scaled gradient descent as the optimizer and mean squared error (MSE) as the loss function (and performance metric). The performance of all trained networks was analyzed on the validation set and the network with 1 hidden layer and 60 neurons per layer were chosen among them as the best compromise between MSE value and network size (and, consequently, computational complexity and training time). Its architecture with details on the size of data in a particular stage is depicted in Figure 5. Shallow NNs were sufficient for this task due to the low complexity of input (LiDAR) data. Networks with a higher number of layers and neurons may perform somewhat better, but not sufficiently better to justify spending more time for training the networks (i.e. significantly higher computational complexity and training time for negligibly better performance).

Besides the previously described NN training procedure, the selected NN was also trained but with samples from 75%, 50%, and 25% of randomly selected data samples (297,283, 198,189, and 99,095 training samples,

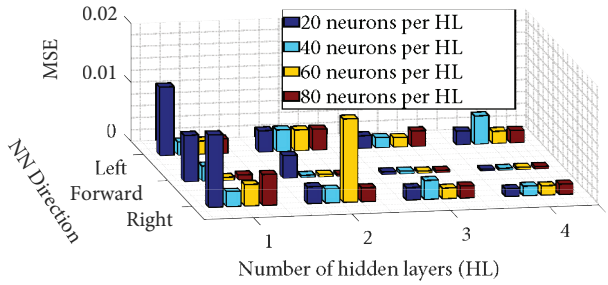


Figure 4. Performance analysis for different trained NN architectures based on MSE metric obtained on validation set.

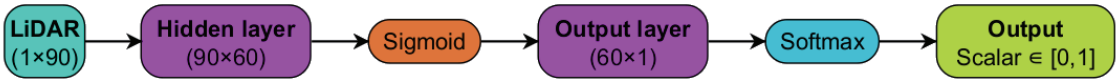


Figure 5. Selected NN architecture used in the experiments.

respectively) in order to approximately estimate the number of collisions needed for proper generalization of NNs. Those NNs were tested in two different real-life scenarios: a U-shaped obstacle and driving down the narrow corridor augmented with additional obstacles.

Each of the NN outputs was a number between 0 and 1, one meaning *'go in this direction'*, which may also be thought of as the probability that the space in the appropriate direction is obstacle-free. Network outputs are postprocessed in order to obtain linear (v) and angular (ω) velocities for robot motion. If the forward direction probability was above a predefined threshold (0.5 in our case, experimentally determined), then the direction was computed as a proportion to a difference between probabilities of left and right directions (NN_{left} and NN_{right}, respectively) being obstacle-free. If the forward direction probability was below the threshold, the robot would slow down (to a small positive speed, and not stop, in order to avoid local minima) and rotate either to the left or right depending on which side was obstacle-free. Once the forward-facing NN probability increased above the threshold, it would speed up and continue as before. Thus, it can be concluded that linear velocity had two predefined levels, while angular velocity could attain any value between –2 rad/s and 2 rad/s, and the value is computed using Eq. 1.

$$\omega = \begin{cases} 2 \cdot NN_{left} & NN_{left} \geq NN_{right} \\ -2 \cdot NN_{right} & NN_{left} < NN_{right} \end{cases} \tag{1}$$
$$NN_{left}, NN_{right} \in [0, 1]$$

5. Experimental setup and results

Several experiments were performed in order to assess the performance of the proposed approach, both in simulation and in the real world, to show that NNs trained with simulation data can perform well without any retraining. The first experiment was conducted in simulation in four random environments that were generated the same way as were those used for collecting training data and are shown in Figure 2. There were five test runs in each of the environments (20 test runs in total). Each run began with the robot in the center with random orientation and lasted until the robot crashed into the obstacle or a maximum time limit of 10 min

was reached. In the experiment, a crash did not happen at all, and all tests terminated after 10 min without a crash (200 min of driving without a crash). These results demonstrated appropriate behavior in simulation, which motivated us to test the approach in the real world. Examples of trajectories obtained during testing in the simulation are shown in Figure 6. It should be noted, however, that if a more dense obstacle configuration were used, crashes might have occurred, but we believe that the used obstacle configuration (remember the size of the whole perimeter is 12.5 $\times$ 12.5 m) is a good representation of a general office-type environment.

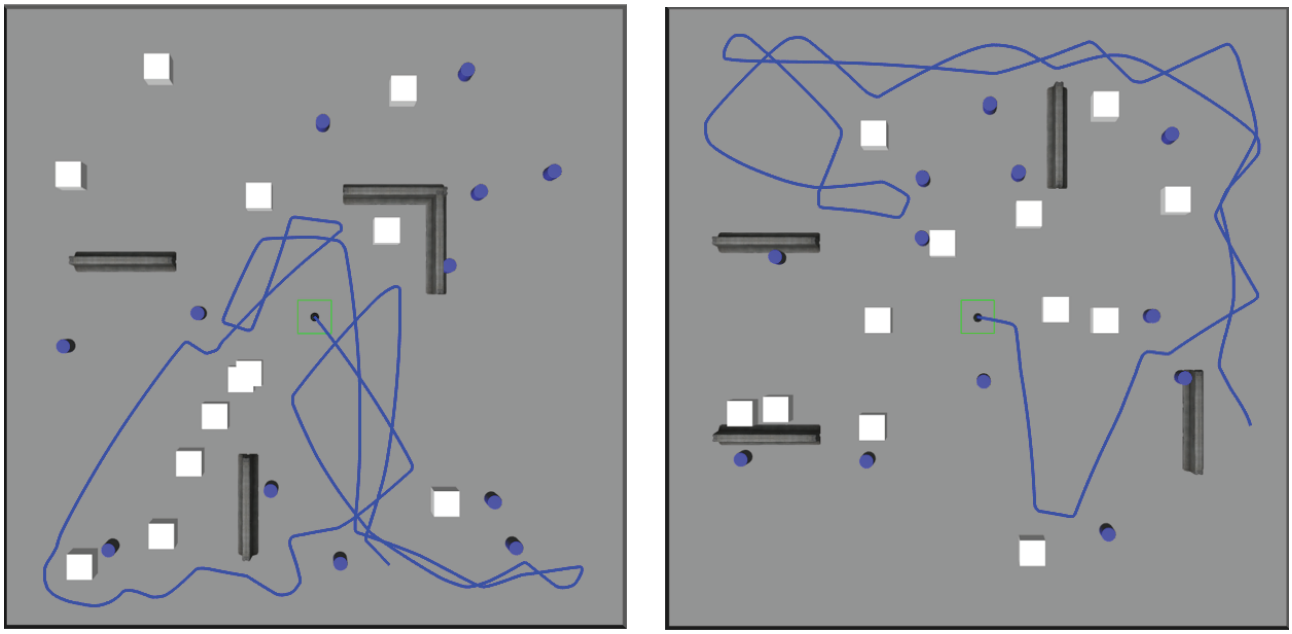


Figure 6. Example trajectories obtained while testing the proposed approach in simulation.

The second simulation experiment was conducted in a small, 5 $\times$ 5 m environment with a single moving obstacle, as depicted in Figure 7. The robot was left for 10 min to roam around the environment. The experiment was run four times, each time with a different (but constant) obstacle velocity (0.1 m/s, 0.2 m/s, 0.4 m/s, and 0.8 m/s; please note that in Figure 7, arrows depict directions of obstacle movement). It was observed that the robot had no problems avoiding the moving obstacle if the obstacle velocity was small (0.1 m/s and 0.2 m/s, roughly less than robot velocity). However, with higher obstacle velocities, crashes occurred (6 crashes with 0.4 m/s obstacle velocity, and with 0.8 m/s obstacle velocity, the robot could not get past the moving obstacle in any instance). While interpreting these results please keep in mind that the NNs controlling the robot were trained without any moving obstacles in the scene, and improved performance could possibly be achieved if moving obstacles are appropriately included in the training set.

Other experiments were conducted in the real world on a real robot using the (heavily modified) Turtlebot 2 mobile robot (shown in Figure 8), with a LiDAR sensor mounted on-board. The 2D LiDAR used was Rplidar-A1M8, which has a resolution of 1 degree (i.e. gives 360 points per measurement cycle) and was run with 7 Hz rotation frequency. It should be noted that LiDAR of the same characteristics was also used in the simulation environment. The computational time needed to run the proposed approach was also examined (average values for 500 LiDAR scan cycles are reported; of note is that median values were smaller than mean values in all cases). Each neural network (right, left, forward) took 0.773 ms to produce an output while preprocessing

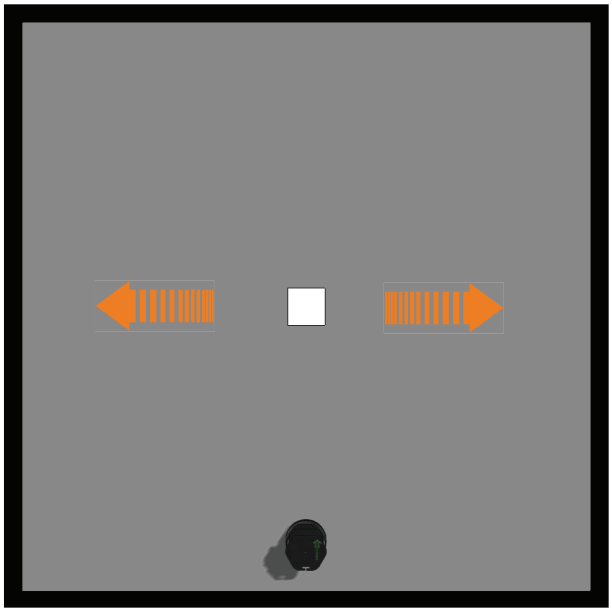


Figure 7. Simulation environment for testing robot behavior in the presence of moving obstacle. Arrows indicate motion directions of the moving obstacle.

of raw LiDAR data (mainly separation in appropriate chunks and formatting) took another 0.144 ms and postprocessing (generation of velocity commands to the mobile robot) took an additional 0.237 ms. Thus, in total on average, it took the algorithm 1.554 ms to produce the velocity command to the robot based on its input. This computational speed was more than enough in our case since LiDAR maximum rotation frequency was 7 Hz (i.e. it took about 142 ms to make the single rotation and provide new raw data), indicating that it can accommodate much faster 2D LiDARs.

In the first experiment, a U-shaped obstacle was used to test how well the NN recovers from such complex obstacles and challenging situations, while in the second one, a narrow (2.16 m wide) corridor was augmented with additional obstacles, making it more demanding for obstacle avoidance algorithm. Please keep in mind that in training, a distance of 0.8 m to 1 m on each side was used as a distance for obstacle avoidance, and the robot itself has a diameter of 0.35 m. In both real-world test cases, there were 5 measurement repetitions per case (5 repetitions $\times$ 2 test cases $\times$ 4 NN cases depending on the used number of samples = 40 test cases in total). Examples of obtained results for both cases are presented in Figure 9 and Figure 10. In the subfigures pertaining to raw LiDAR scans, the same color scheme as in Figure 3 is used in order to illustrate which data points were fed to which NN. Also, note that data points that were not used in any of the NNs (i.e. in the back of the robot) are not depicted in the figure.

It should be noted that for the U-shape scenario (Figure 9), the robot crashed at least once for all cases (25% of data 3 out of 5 times, 50% of data 3 out of 5 times, and 75% 1 out of 5 times), except for case with 100% data. This led us to conclude that maximum collected crash data are needed (or even more) for the reliable performance of obstacle avoidance, while any reduction in training data resulted in reduced reliability (especially in 25% and 50% test cases).

In the corridor test case (Figure 10), we placed obstacles in such a manner that we could, with a high degree of certainty, assume that the obstacle avoidance would fail (since free gaps at the last obstacle – rightmost

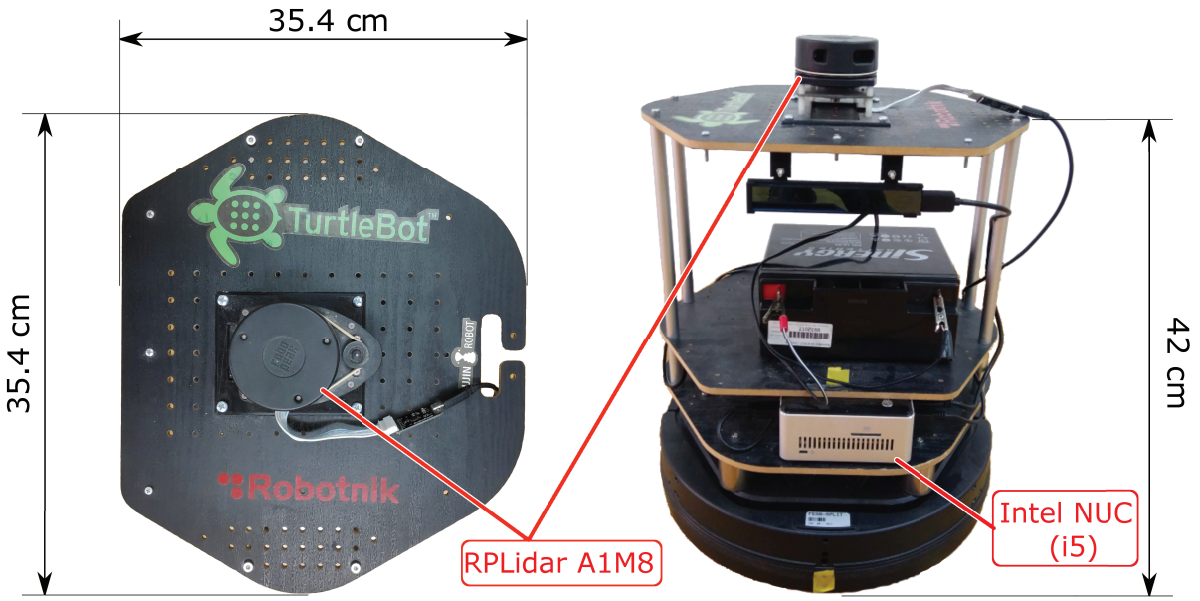
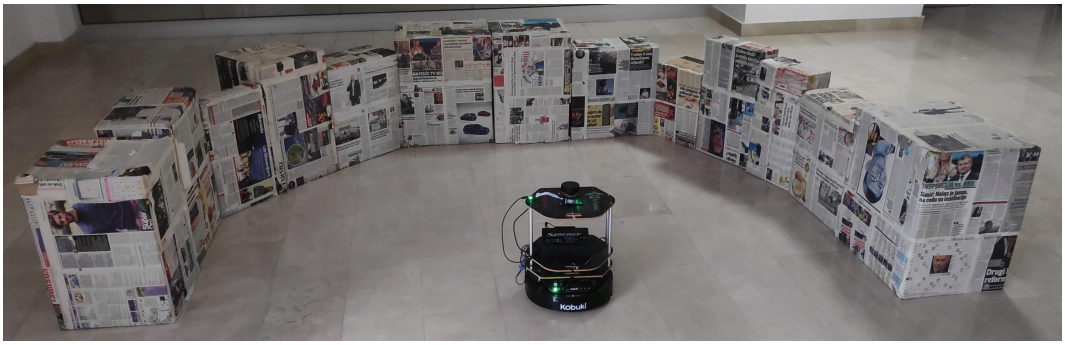


Figure 8. Turtlebot 2 mobile robot used in real-world experiments.

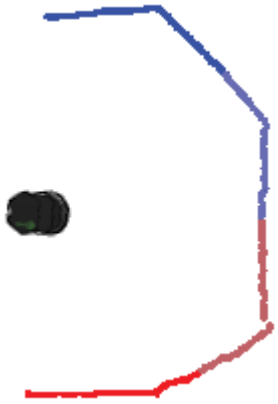
one in Figure **??** – are well below clearances used in the training). In all 20 test cases, the robot did not pass the corridor as was intended (signaling that there is still room for improvement). Two interesting observations were made during the experiments: 1) in 25% and 50% cases the robot performed a U-turn and thus did not crash into the obstacle; however this behavior was not the desired one, but strictly speaking it did avoid obstacles; and 2) for the 100% case the robot moved fastest but crashed into the final obstacle in all 5 test cases. However, we observed that in all cases a crash occurred because the obstacle was too close to the robot's right side (while the robot started moving left and forward), so when it started to turn right it simply did not register the obstacle due to LiDAR minimum range issues and the way we processed such data. Thus, it can be thought of as a failing of the sensor rather than the method (which could be reduced/eliminated with the inclusion of additional sensors like ultrasound rangers).

Afterwards, the proposed approach was compared to a more standard obstacle avoidance algorithm shipped with the ROS (Robot Operating System) [16] navigation stack – the dynamic window approach (DWA) [17]. The measurement setup was fairly simple, with the robot always starting from the same point in space and having to go to the point behind a simple rectangle-shaped obstacle that was not in the original map. This is depicted in Figure 11. The experiment was repeated 3 times for each algorithm, 6 times in total (however, only four of those are shown in the figure for clarity reasons). It should be noted that the DWA always avoided the obstacle to the left, while out approach chose the right side two times and the left side once, demonstrating the stochastic nature of the approach. The experiment demonstrated that the proposed approach can be integrated into navigation-based algorithms and can perform well within them. In Figure 9, please note parts of the trajectories for NN-based obstacle avoidance outlined with different colors, in which the NN-based obstacle avoidance was in full control of the robot (otherwise, the navigation part was in control).

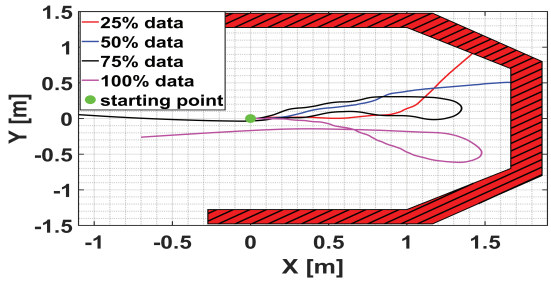
Finally, the robot was placed inside a more complex, self-contained obstacle course similar to the simulated environment, which is shown in Figure 11, and in which it freely ran for 20 min. The course also had dynamic



(a) Real-world experimental setup



(b) Raw LiDAR scan

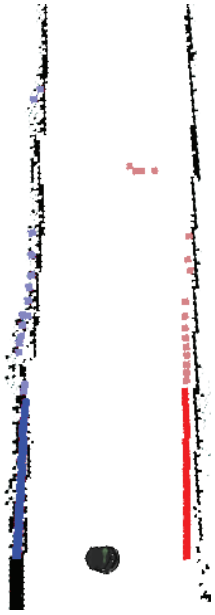


(c) Experimental results

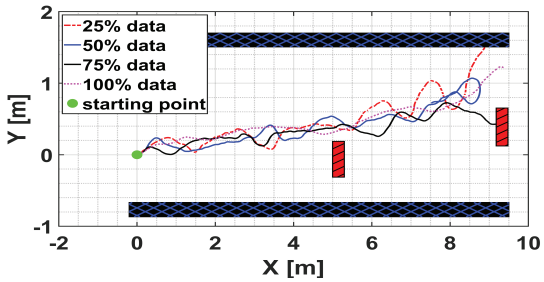
Figure 9. Performance of the approach in the presence of U-shaped obstacle.

obstacles in the form of moving mobile robots (Turtlebot 3 Burger and Waffle) with predefined trajectories. The number of collisions, the time between collisions, and the distance between collisions were recorded. Obtained results were as follows: 8 collisions, 127 s ($\pm$102.88 s) for mean time between collisions (and standard deviation), and 21.59 m ($\pm$17.49 m) for mean distance between collisions (and standard deviation). This is degraded performance compared with the simulation (in which there were no moving obstacles and which had a smaller number of obstacles per meter squared - 0.194 vs. 0.223), but still, one that shows that the approach is valid and has potential for practical applications. Obtained results are also in line with results from [5] for time and somewhat lower for distance. It should be kept in mind that our test scenario was cluttered with a number of obstacles, a case that would not be common in everyday applications and one not used in [5]. In standard office setups and uncluttered corridors, the NN performance was improved. Also, out of 8 crashes, 4 were with moving obstacles (in cases when it was moving directly towards the robot with higher speed, again in line with results obtained from simulation with a single moving obstacle) and 1 with static obstacles with a slim profile (less than 5 scan points needed for detection of obstacles in this approach). With a slightly different setup (e.g., lower number of scan points) and additional and better trained neural networks (e.g., for going backwards, or trained on a larger and more diverse dataset), we believe some of the crashes could have been avoided, and these are some of our future research directions. Another possible improvement is a unification of NNs into a single NN for possible smoother trajectory. When considering possible improvements, three additional possibilities were

(a) Real-world experimental setup



(b) Raw LiDAR scan



(c) Results

Figure 10. Performance of the approach in a narrow corridor.

noted. First, since a number of crashes occurred with moving obstacles, it could be beneficial to add moving obstacles into a simulation environment for training purposes. Secondly, the middle part of the obtained data that were discarded in training could potentially be used as additional training examples for improved network performance. Finally, as in [13], variations in shape and configuration of the test environment could lead to improved performance.

6. Conclusions

This research aimed to develop a self-contained approach for safe and accurate generation of NN-based training data and implementation of such data for the example of mobile robot obstacle avoidance. While it was previously shown that NNs can be used for obstacle avoidance in mobile robotics, experiments in this paper demonstrate that it can be done based on the simulated data in a self-supervised manner with no or very little human intervention. This, in turn, makes collecting vast amounts of training data easier, a task which is very difficult and time-consuming using a real robot. The paper also demonstrated that by using LiDAR-based data instead of more commonly used live camera images/feed, a high degree of similarity between simulation and real-world scenarios can be achieved, resulting in a lower complexity level of the NN that is trained on simulation-

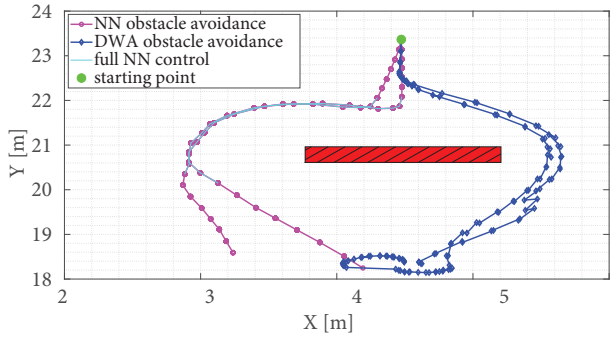
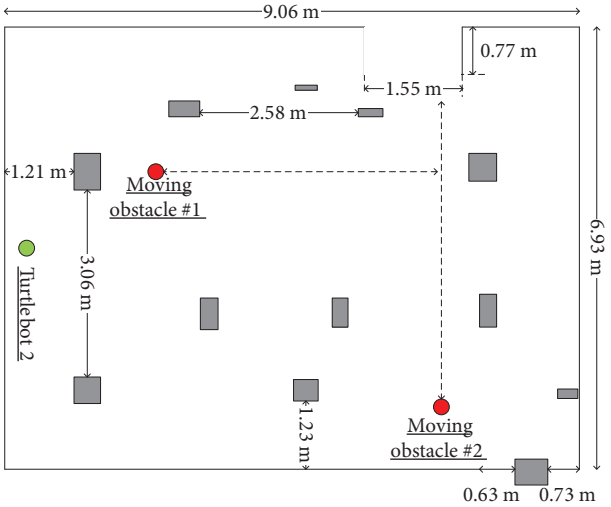


Figure 11. Comparison of obstacle avoidance in navigation task.



(a) Ground plan of the complex obstacle course where the final experiment was conducted, with marked starting positions of test vehicle and moving obstacles. Dotted lines show predefined trajectories of moving obstacles (Turtlebot 3 mobile robots).

(b) A snapshot of real-world testing environment

Figure 12. Real-world testing environment with static and dynamic obstacles.

based data and which performs satisfactorily without the need for retraining within real-world environments. Good results were achieved in experiments where the proposed approach was tested in simulation and on the real robot in several complex obstacle avoidance tasks in demanding environments, demonstrating the practical applicability of the proposed approach. However, some of the disadvantages of the approach that were identified during testing are as follows: in narrow spaces right- and left-based NNs fight for dominance, resulting in somewhat jittery movement, lower-level performance in the presence of moving obstacles, and the need for the robot to constantly move forward in order to avoid getting trapped in local minima. Thus, our future research will be devoted to reducing some of the mentioned drawbacks, which we believe can be achieved with the development of a single (possibly time-dependent) NN for obstacle avoidance or using a fuzzy-based mediation-type mechanism for fusing together outputs from left- and right-based NNs, implementing an in-place rotation mechanism and/or reverse driving, and implementing moving obstacles in the simulation environment

so that the NN training incorporates such examples and consequently improves robot avoidance capabilities in the presence of moving obstacles.

References

[1] Sullivan K, Lawson W. Reactive ground vehicle control via deep networks. In: Robotics: Science and Systems: Workshop-New Frontiers for Deep Learning in Robotics; Boston, MA, USA; 2017. pp. 1–7.

[2] Hadsell R, Sermanet P, Ben J, Erkan A, Scoffier M et al. Learning long-range vision for autonomous off-road driving. Journal of Field Robotics 2009; 26 (2): 120–144. doi: 10.1002/rob.20276

[3] Giusti A, Guzzi J, Cireşan DC, He FL, Rodríguez JP et al. A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters 2015; 1 (2): 661–667

[4] 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; Daejeon, Korea; 2016. pp. 2759–2764.

[5] Gandhi D, Pinto L, Gupta A. Learning to fly by crashing. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems; Vancouver, Canada; 2017. pp. 3948–3955.

[6] Pinto L, Gupta A. Supersizing self-supervision: learning to grasp from 50k tries and 700 robot hours. In: 2016 IEEE International Conference on Robotics and Automation; Stockholm, Sweden; 2016. pp. 3406–3413.

[7] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J et al. Human-level control through deep reinforcement learning. Nature 2015; 518 (7540): 529-533. doi: 10.1038/nature14236

[8] Xie L, Wang S, Markham A, Trigoni N. Towards monocular vision based obstacle avoidance through deep reinforcement learning. arXiv preprint. arXiv: 1706.09829, 2017.

[9] Maturana D, Scherer S. 3D convolutional neural networks for landing zone detection from LiDAR. In: 2015 IEEE International Conference on Robotics and Automation; Seattle, WA, USA; 2015. pp. 3471–3478.

[10] Liu C, Zheng B, Wang C, Zhao Y, Fu S et al. CNN-based vision model for obstacle avoidance of mobile robot. In: 2017 3rd International Conference on Mechanical, Electronic and Information Technology Engineering; Chengdu, China; 2017. pp. 1-4.

[11] Correa DSO, Sciotti DF, Prado MG, Sales DO, Wolf DF et al. Mobile robots navigation in indoor environments using kinect sensor. In: 2012 Second Brazilian Conference on Critical Embedded Systems; Campinas, Brazil; 2012. pp. 36–41. doi: 10.1109/CBSEC.2012.18

[12] Bousmalis K, Irpan A, Wohlhart P, Bai Y, Kelcey M et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. arXiv preprint. arXiv: 1709.07857.

[13] Zhu Y, Mottaghi R, Kolve E, Lim JJ, Gupta A et al. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: IEEE International Conference on Robotics and Automation; Singapore; 2017. pp. 3357-3364. doi: 10.1109/ICRA.2017.7989381

[14] Koenig NP, 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; Sendai, Japan; 2004. pp. 2149–2154. doi: 10.1109/IROS.2004.1389727

[15] Kružić S, Musić J, Stančić I, Papić V. Influence of data collection parameters on performance of neural network-19based obstacle avoidance. In: 3rd International Conference on Smart and Sustainable Technologies; Split, Croatia; 2018. pp. 1-6.

[16] Quigley M, Conley K, Gerkey B, Faust J, Foote T et al. ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software; Kobe, Japan; 2009. pp. 1-6.

[17] Fox D, Burgard W, Thrun S. The dynamic window approach to collision avoidance. IEEE Robotics Automation Magazine 1997; 4 (1): 23–33. doi: 10.1109/100.580977