International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:15, No:4, 2021

Simulation of Obstacle Avoidance for Multiple Autonomous Vehicles in a Dynamic Environment Using Q-Learning

Andreas D. Jansson

***Abstract*—**The availability of inexpensive, yet competent hardware allows for increased level of automation and self-optimization in the context of Industry 4.0. However, such agents require high quality information about their surroundings along with a robust strategy for collision avoidance, as they may cause expensive damage to equipment or other agents otherwise. Manually defining a strategy to cover all possibilities is both time-consuming and counter-productive given the capabilities of modern hardware. This paper explores the idea of a model-free self-optimizing obstacle avoidance strategy for multiple autonomous agents in a simulated dynamic environment using the Q-learning algorithm.

***Keywords*—**Autonomous vehicles, industry 4.0, multi-agent system, obstacle avoidance, Q-learning, simulation.

I. Introduction

In recent years, the increased availability of high-accuracy sensors, inexpensive computational power and internet-enabled devices democratized smart products and self-optimizing systems in multiple domains. In the context of Industry 4.0, this development opens up the ability to assist humans with dangerous or repetitive work. Smart products span the range from simple, zero-intelligence products to proactive entities. Sensors play an important part in making a product smart, allowing it to detect and report problems, or even make decisions on its own [1]. As the manufacturing industry moves towards a more customer-driven market [2], companies are required to shorten product life cycles and reduce time-to market without negatively impacting quality and costs. Such a shift demands more decentralized, flexible control and increased robustness. The use of smart products in the form of autonomous guided vehicles (AGVs) may help improve effectiveness, safety and flexibility of resource transportation in the context of Industry 4.0 [3]. However, with great power comes great responsibility. Such autonomous agents must therefore have a solid strategy for avoiding collisions in a complex and dynamic environment. In order to manually define such a strategy, every conceivable and even unconceivable situation an AGV may find itself in must be examined and modelled. This is not only time-consuming and resource-intensive, but also counter-productive given the capabilities of modern hardware, both in respect to raw computing power and sensory input and processing. This paper aims to investigate a method for learning a general strategy by means of a model-free approach. Specifically, how to train multiple agents representing AGVs to avoid collisions in an unknown dynamic environment. To do this, a simple multi-agent simulator will be implemented and tested. Furthermore, a set of benchmarks for the system will be defined and run in order to measure learning performance and validate learned obstacle avoidance strategies. Performance will be discussed and further improvements to the approach will be suggested.

A. D. Jansson is with the Department of Computer Science and Computational Engineering, UiT The Arctic University of Norway, 8505 Narvik, Norway (e-mail: andreas.d.jansson@uit.no).

II. Related Work

The use of both model-based and model-free learning combined with sensor data, smart products and autonomous agents have been documented in several instances. Relevant examples include [4] where they used LiDAR sensor data combined with odometer data to determine a robot's position in an unknown environment. Simultaneous Localization and Mapping, along with A-star search, was used for path planning. Simulated trajectories based on the robot's motion model were evaluated based on their ability to avoid obstacles and reach a predefined goal. This demonstrated how a LiDAR sensor could be utilized for obstacle avoidance in an unknown environment. However, this approach required a map to be created up-front, by letting the robot explore its environment exhaustively. During mapping, the robot was controlled manually using a game pad.

A similar approach is the work by Malavaz et al. in which they examined a LiDAR-only based navigation algorithm for agricultural robots [5]. Here, they documented efforts in developing a general and robust approach for autonomous robot navigation in an unknown environment. In this case, the environment consisted of rows of crops with unknown spacing and size. A model of the environment was constructed in real-time using line detection based on 2D point clouds collected by LiDAR. The robot was then tasked to move along detected lines in the crop and remove weeds. Placement of the LiDAR may affect performance if it is placed too low, as tall weeds may blind the sensor. Similarly, if the LiDAR is placed too high, it will not detect crops. The authors suggested using a 3D sensor to overcome these challenges. Furthermore, as this approach relied on LiDAR only, no visual information was available, and thus the robot was unable to identify the type of obstacle. This becomes an issue when weeding crops, as the robot was unable to act according to the type of obstacle, i.e., a harmless obstacle (weeds) and a potentially harmful obstacle (rocks, branches etc.). However, the authors concluded that this approach was promising and served as a robust method when no prior information about an environment was available.

In [6], the use of visual data combined with LiDAR scans was discussed. The authors used a set of extensive pre-collected image data to address the problem of global localization in an unknown indoor environment, dubbed “the kidnapped robot problem”. Their effort resulted in a solid basis for a strategy for the robot, with highly accurate results, the major drawback being the extensive manual effort required upfront. This model also becomes less useful when the environment changes visually.

Combining multiple sensors to increase performance has also been discussed in [7]. In their work, they used an RGB-D camera and a 2D LiDAR in order to learn a classifier in 3D in a semi-supervised context. The classifier was trained online, eliminating the need for data to be collected prior to training. Combined with existing classifiers for the camera and 2D LiDAR, the authors managed to successfully train a human-tracking classifier for a mobile service robot.

III. Approach and Implementation

There already exist several excellent simulation platforms and environments for multi-agent and robotics experimentation and visualization [8], [9]. However, with all their extensive features, they also introduce computational overhead and in general offer higher level representations of robotics and smart products. As more fine-grained control and low-level tinkering ability were desired, it was decided to implement a simple agent simulator from scratch. This simulator was designed and implemented to serve a single objective only, this being the visualization of multiple agents navigating in a dynamic environment. Furthermore, in order to be truly dynamic, a level of interactivity was also required. When running the program, the user should be able to add or remove obstacles in the environment.

Agents learned to avoid the obstacles using Q-learning, which is a model-free reinforcement learning approach [10]. To detect the obstacles, a simple sensor was implemented. This virtual sensor could report distance d and relative angle a of an agent’s closest obstacle. Distance and angle values were discretized to construct the state space, defined as a 2D matrix of width W and height H. Sensor values were discretized to a' and d' using the following formulae:

$$a' = \frac{a+180}{360} \times (H-1) \quad (1)$$

and

$$d' = \frac{d}{120} \times (W-1). \quad (2)$$

For the experiments conducted and results presented in this paper, W = 20 and H = 25.

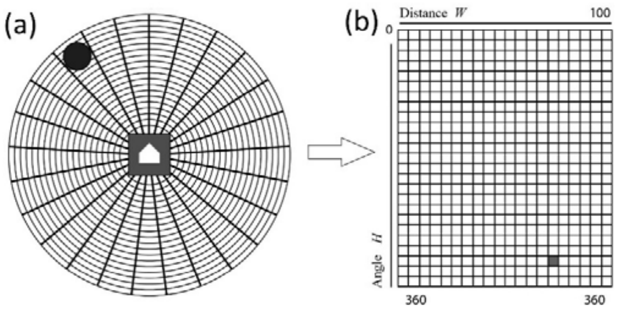

Fig. 1 Virtual sensor range, discretization, and state space mapping

Fig. 1 (a) shows a visual representation of the state space, as seen by the sensor of a square agent. A circular obstacle is present within the sensor’s range, and its position and relative angle defines the current state of the agent. The corresponding state matrix and agent position is shown in Fig. 1 (b). In every state, an agent may take one of three actions: turn left, turn right, or keep going. For example, in the situation seen in Fig. 1, the agent should either A) keep going or B) turn right. Both actions would yield a positive reward in the next time step, with B) being the best action. The reward received was based on the distance from the obstacle in the current state compared to the previous state. However, if the agent turned left, it would receive a smaller reward, since its distance to the obstacle would decrease compared to not deviating from its initial course. In order to drive exploration, agents were programmed to take a random action in 5% of cases, regardless of prior knowledge.

Following the standard Q-learning algorithm definition (6.6, p. 157) in [10], empty q- and r-matrices were initialized and then populated by letting the agent roam its environment in a random fashion.

To demonstrate the performance of the simulator, a set of benchmarks were defined, listed in Table I.

TABLE I
Benchmark definitions

ID	Number of agents	Type of obstacle
1	5	100 static obstacles
2	5	100 initial static obstacles, with 100 static obstacles added after 30 seconds
3	10	100 initial static obstacles, with 100 static obstacles added after 30 seconds
4	10	100 dynamic obstacles disappearing and appearing randomly every 2 seconds
5	20	100 initial static obstacles, with 100 static obstacles added after 30 sec.
6	20	100 dynamic obstacles disappearing and appearing randomly every 2 seconds
7	20	200 static obstacles

Benchmarks were conducted in two configurations, where in the first, agents did not share knowledge and maintained their own set of q- and r-matrices. For the next set of benchmarks, agents were able to communicate and share experience in the form of q- and r-matrices. The performance was measured in the form of collisions over time and the total number of collisions. To account for the random nature of the simulation, each benchmark was performed 10 times and the values averaged. This also meant that running the simulator for longer than 60 seconds for each benchmark became impractical.

All benchmarks were performed and timed on a system with the following specifications:

- CPU: AMD Ryzen7 5800X@4.4GHz
- Memory: 32 Gb
- Windows 10 20H2 64-bit build 19042.

IV. Results

For visualization purposes, agents representing AGVs were rendered as solid-colored squares, with a straight line representing the current heading and sensor range. Each agent also left a trail to easier track their movements in the environment. Obstacles were randomly generated and visualized as black dots. Furthermore, the last action taken along with the reward received was displayed in a label attached to each agent (Fig. 3). The program consisted of an interactive window where it was possible to add or remove obstacles by left- and right-clicking in the simulator area as shown in Fig. 2. The simulation was set to run automatically, with agents exploring the environment as soon as the program was launched.

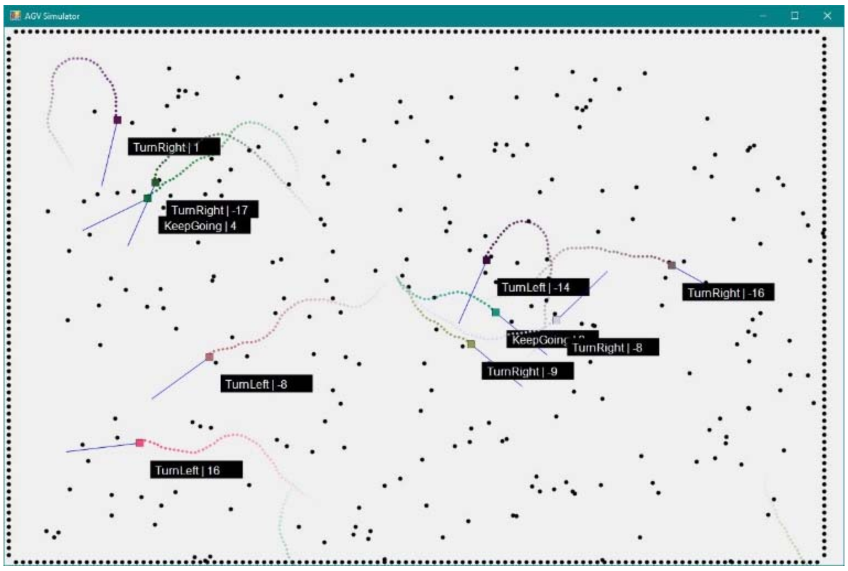

Fig. 2 Screenshot of simulator window

In the first seconds of the simulation, agents often left the designated area. Over time, agents started to learn to avoid the obstacles, as well as staying within the defined world boundary, as may be seen in Fig. 3.

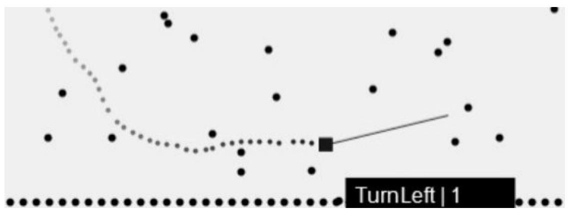

Fig. 3 An agent with status label avoiding obstacles

Figs. 4-10 show the number of collisions over time (a) along with the total number of collisions (b) in individual knowledge mode (black) and shared knowledge mode (gray).

V. Discussion of Results

Letting autonomous agents roam around in a random fashion is only practical in a simulated environment, where no physical damage can be done to obstacles in the case of collisions. This random exploration enabled learning without any upfront knowledge of the environment, nor the position of obstacles. This is the major benefit of the proposed approach. When examining the results in Figs. 4-9, it is clear that the number of agents has an impact on learning performance. As seen in Fig. 4, when running the system with five agents, sharing knowledge is only slightly beneficial in the long run, and hurts short-term performance. When doubling the number of obstacles in the environment, sharing knowledge appears to become even less beneficial with few agents. It is clear from Fig. 5 that the number of collisions start to increase at this point. Also seen in Fig. 5, the number of collisions was consistently lower for the shared knowledge run up until the point where more obstacles were added. This may indicate that the agents' strategy emerged too quickly to be able to adapt to the increased number of obstacles. A similar trend is seen in Fig. 6, where single- and multi-agent performance are similar up until more obstacles are added, even though the number of agents was double compared to benchmark 2. Performance deteriorates further in Fig. 7, where obstacles disappear and reappear randomly, although it appears to stabilize in the long term compared to Figs. 5 and 6.

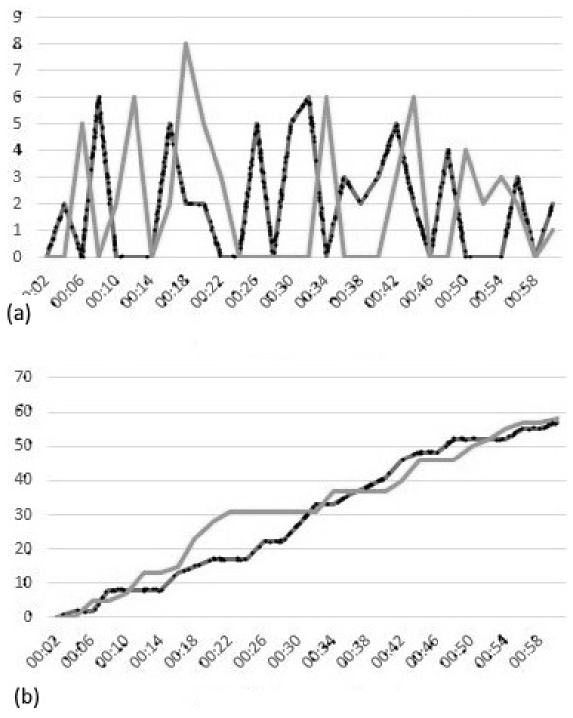

Fig. 4 Benchmark 1 collisions per time step and total collisions

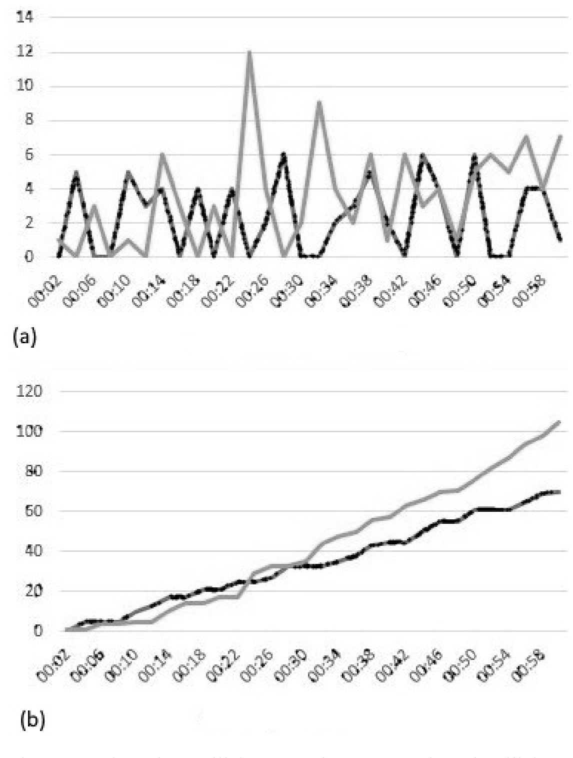

Fig. 5 Benchmark 2 collisions per time step and total collisions

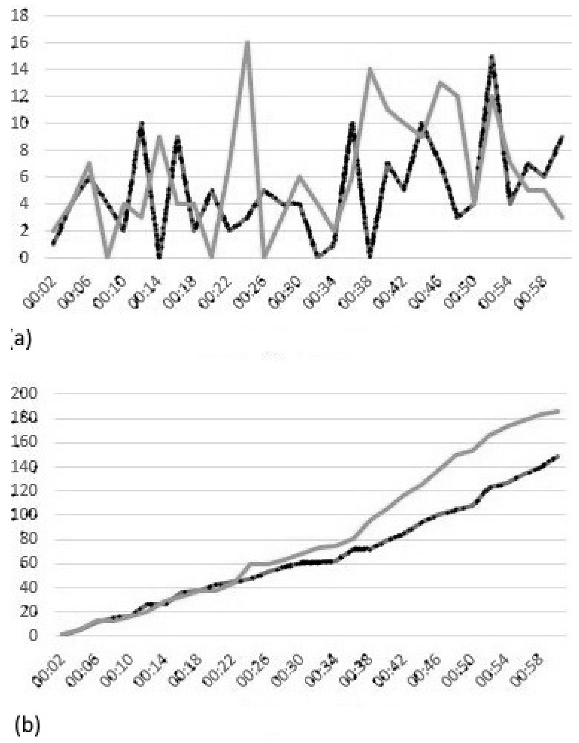

Fig. 6 Benchmark 3 collisions per time step and total collisions

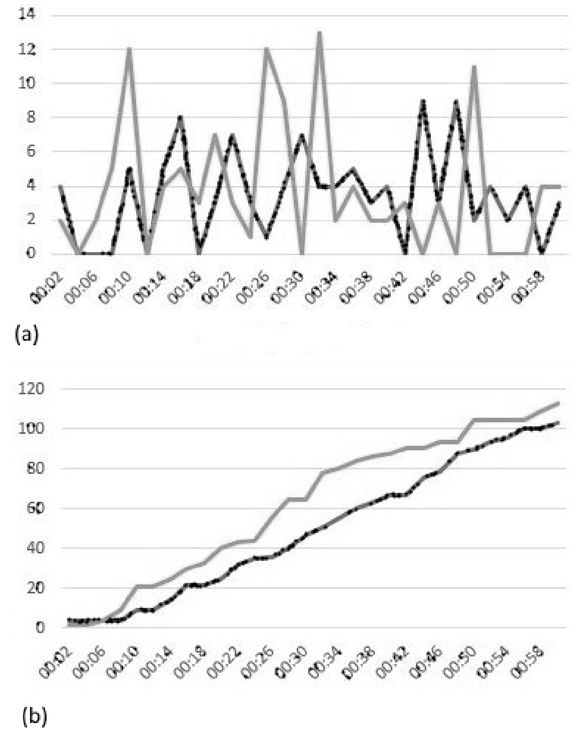

Fig. 7 Benchmark 4 collisions per time step and total collisions

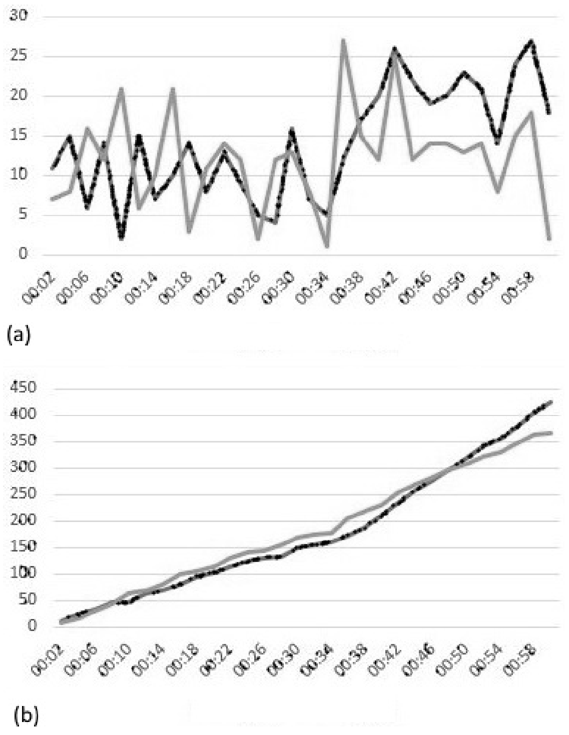

Fig. 8 Benchmark 5 collisions per time step and total collisions

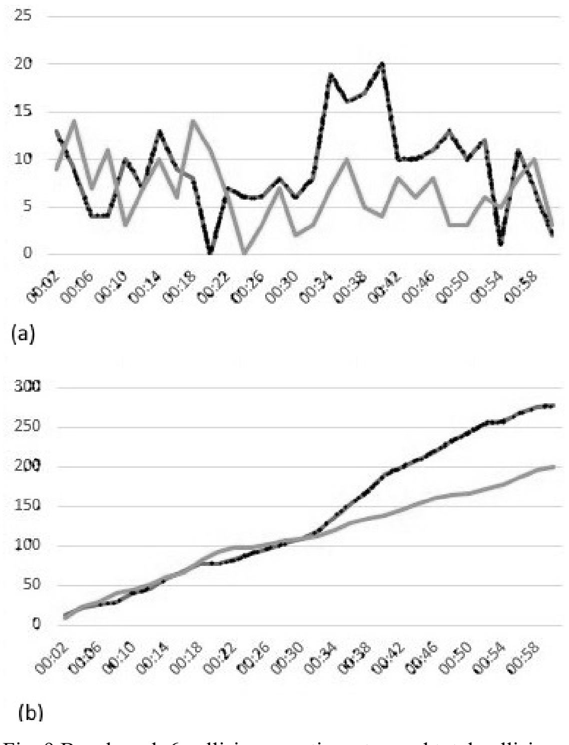

Fig. 9 Benchmark 6 collisions per time step and total collisions

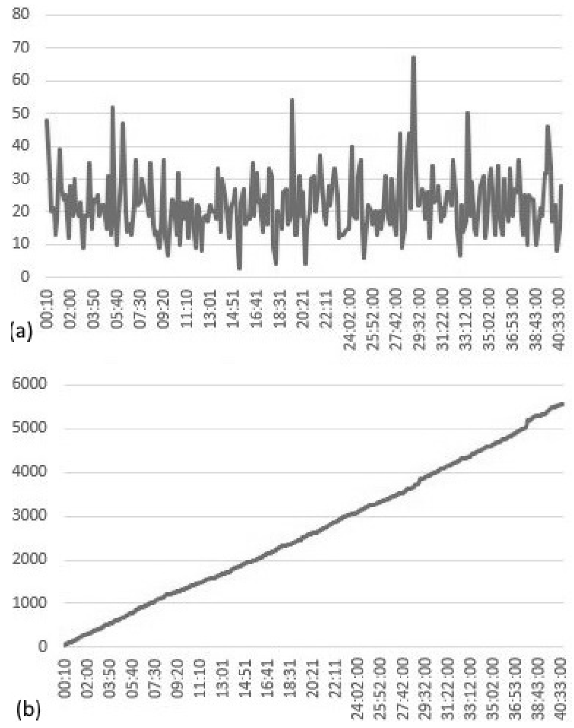

Fig. 10 Benchmark 7, after running the simulator for 40 minutes

When increasing the number of agents to 20, performance of the shared knowledge approach (gray) starts to surpass that of a single agent (black). As seen in Fig. 8, multi-agent performance is only slightly worse than that of a single agent and starts to surpass it after an average of 50 seconds. The long-term total number of collisions drops below that of a single agent. Furthermore, when comparing Fig. 5 to Fig. 8, it shows that quadrupling the number of agents from five to 20 did not quadruple the total number of collisions in the shared knowledge case (gray). The results from Fig. 9 show the most promise, as both the number of collisions per time step (a) and the number of total collisions (b) are lower in the multi-agent approach. Performance also seems to be more stable in the long term. As this was a highly dynamic environment, with obstacles changing positions every 2 seconds, this approach to learn a collision avoidance strategy with multiple agents shows some promise. The fact that an agent is indeed able to learn a strategy to avoid obstacles is perhaps best observed when looking at Fig. 3, where the path taken to avoid the obstacles can be seen clearly.

Running the simulator for longer did not produce better results, as shown in Fig. 10. Performance was stable and the total number of collisions (b) increased in a linear fashion.

The virtual sensor implemented in the simulator was only able to sense its nearest obstacle. This means that an agent was only able to make decisions that were rewarding in the short run. An agent could for example navigate head-first into a cluster of obstacles in order to avoid a single obstacle, and then be unable to successfully avoid all obstacles in the cluster. Extending the virtual sensor to classify a cluster of obstacles as a general region to avoid is expected to help increase performance in this regard.

The way the state space was discretized is also expected to have negatively impacted performance. In order to remedy this, some alternative approach to continuous state space reinforcement learning should be investigated. One approach that may produce better results is the use of deep reinforcement learning, for instance deep-Q-learning. An alternative approach could be to increase the resolution of the virtual sensor and thus the size of the Q- and r-matrix. However, this may impact performance in other ways, especially if using this approach on a physical robot with limited computational power. The approach presented in this paper requires only simple computations for mapping and reward calculation. Finally, although implementing a simulator from scratch gave full control over all aspects of the program, visualization is minimal compared to existing simulators. The basic Q-learning approach presented in this paper can easily be ported to various programming languages and simulation environment if desired.

VI. Conclusion

Results indicate that using a model-free reinforcement learning approach in order to learn an obstacle avoidance strategy in an unknown and dynamic environment has some potential. Its major benefit compared to similar work is that no prior information about the environment is required, and agents are able to learn in a virtual environment without human intervention. Increasing the number of agents and having them share knowledge was beneficial compared to fewer agents, especially in a highly dynamic environment. Running agents in this random fashion in a virtual environment is also safer compared to a physical environment. Although visualization is minimal compared to comparable simulators, it is possible to notice some learning progress by observing the movement of the simulated agents. With some more attention to sensor implementation and state space definition, the proposed approach may have some potential to be useful for avoiding dynamic obstacles when applied to autonomous resource transportation vehicles in enclosed spaces within the context of Industry 4.0.

Appendix

A complete copy of the simulator source code may be found at bitbucket.org/ADJansson/selfdrivingcarsimulator

References

[1] G. G. Meyer, K. Främling, and J. Holmström, “Intelligent products: a survey”, *Computers in Industry*, vol. 60, issue 3, pp. 137-148, Apr. 2009.

[2] T. Skjoett-Larsen, “European logistics beyond 2000”, *International Journal of Physical Distribution & Logistics Management*, vol. 30, issue 5, pp. 377-387, June 2000.

[3] R. Sella, A. Rassõlkinb, R. Wanga, and T. Otto, “Integration of autonomous vehicles and industry 4.0”, in *Proceedings of the Estonian Academy of Sciences*, Tallin, Estonia, 2019, pp. 389–394.

[4] Y. Cheng, and G. Y. Wang, “Mobile robot navigation based on lidar”, *2018 Chinese Control and Decision Conference (CCDC)*, Shenyang, China, 2018, pp. 1243-1246.

[5] F. B. P. Malavazi, R. Guyonneau, J.-B Fasquel, S. Lagrange, and F. Mercier, “LiDAR-only based navigation algorithm for an autonomous agricultural robot”, *Computers and Electronics in Agriculture*, vol. 154, pp. 71-79, Nov. 2018.

[6] B Z. Su, X. Zhou, T. Cheng, H. Zhang, B. Xu, and W. Chen, "Global localization of a mobile robot using lidar and visual features," *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Macau, Macao, 2017, pp. 2377-2383.

[7] Z. Yan, L. Sun, T. Duckctr, and N. Bellotto, "Multisensor online transfer learning for 3D LiDAR-based human detection with a mobile robot," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 7635-7640.

[8] M. Freese, S. Singh., F. Ozaki, and N. Matsuhira, “Virtual robot experimentation platform V-REP: a versatile 3D robot simulator” in *Simulation, Modeling, and Programming for Autonomous Robots*, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, Ed. Berlin Heidelberg: Springer-Verlag, 2010, pp. 51-62.

[9] S. Nakaoka, "Choreonoid: Extensible virtual robot environment built on an integrated GUI framework," *2012 IEEE/SICE International Symposium on System Integration (SII)*, Fukuoka, Japan, 2012, pp. 79-85.

[10] R. S. Sutton, and A. G. Barto, *Reinforcement learning: an introduction*, 2nd edition. Cambridge, MA: The MIT Press, 2015, ch. 6.5.