Discretization and Representation of a Complex Environment for On-policy Reinforcement Learning for Obstacle Avoidance for Simulated Autonomous Mobile Agents

Andreas Dyrøy Jansson [0000-0001-5116-6041]

Department of Computer Science and Computational Engineering,
UiT The Arctic University of Norway, 8514 Narvik, Norway
andreas.d.jansson@uit.no

Abstract. In recent years, the demand for digitalization, automation, and smart systems in the airline industry has accelerated. Furthermore, due to the ongoing global pandemic as of 2022, airlines are faced with the challenge of offering flexibility in both cargo and passenger capacity. Studies show that the use of smart products and autonomous agents are expected to play a key part in the digital transformation of the logistics industry. This paper aims to examine the current state-of-the-art in multi-agent systems and reinforcement learning with special interest in intelligent baggage handling systems. How to simplify, implement and simulate a system of autonomous baggage carts as a software model in order to examine congestion situations will be the main topics of this paper. Furthermore, how the findings from the software model may be applied to real-world scenarios related to Industry 4.0 and baggage handling will also be discussed.

Keywords: Autonomous Agents, Congestion, Edge Devices, Reinforcement Learning, Self-optimization, Simulation.

1 Introduction

Due to the outbreak of Covid-19, aviation demand has fallen considerably. According to [1], leisure trips and tourism were first to recover after previous global crises and are expected to surpass business travel demand as the pandemic subsides. During this crisis, airlines have been taking major financial hits, amassing more than $180 billion worth of debt in 2020 [1]. As a result, the industry as a whole is required to cut costs by restructuring for greater efficiency. This means investing in IT, digitalization, and automation. As of 2019, investing in Business Intelligence (BI) and Smart Systems was a top priority for 95% of airlines and 87% of airports [2].

Furthermore, due to increased globalization, competition and e-commerce, international competitiveness of companies will depend on effective and on-time delivery of products globally. Similarly, global supply chains are dependent on air freight to reach their customers as quickly and efficient as possible. However, as many passenger flights were grounded during the pandemic, cargo capacity was also reduced. [1] estimated

that passenger flights are responsible for half of total air cargo capacity, which means that air freight will be a limited resource going forward. As a consequence, airlines need to be agile and grow cargo capacity while still being flexible and able to quickly adapt to changing demand. As a result, previous efforts in digitization have been accelerated and are expected to play a vital part in this transformation [1, 2].

Sensors enable smart products to detect and report problems, and, in some cases, even make decisions on their own [3]. Smart products appear in several forms and in various domains, and one of the best known is perhaps the automated guided vehicle (AGV). According to [4], using AGVs for resource transportation may help improve effectiveness, safety, and flexibility in the context of Industry 4.0, and by extension, baggage handling and logistics. To achieve this, agents representing such AGVs are required to have a solid strategy for avoiding obstacles in a complex and dynamic environment. However, a limiting factor that must be considered is the computing performance of the AGVs onboard computers. As such, the main topic this paper will examine is how a complex, dynamic environment can be broken down into discrete states to run on low-power hardware while still being useful for obstacle avoidance. This will be done using a simulation of multiple autonomous agents learning an obstacle avoidance strategy through on-policy reinforcement learning. A simple prototype application to simulate and visualize the agents will be developed and tested, and the results discussed. How agents are able the handle dynamic obstacles, in addition to changes in available space will be examined and discussed. Finally, how the findings from this paper may be adapted in order to be useful for real-world challenges regarding baggage handling and logistics will be brought up.

2 Similar Work and Experiments

The use of multi-agent systems (MAS) and reinforcement learning (RL) in the industry have been documented thoroughly. Examples include [5], where they explored the use of smart systems and Internet of Things (IoT) to decentralize control of jobs on the factory shop floor. The main focus of this paper was on the challenge of efficient material flow control through stationary entities. Traditionally, this was performed using a top-down centralized control approach, where agents were structured hierarchically. Using simulation, they demonstrated that self-organized decentralized control was a feasible approach to this challenge. They claimed that a decentralized approach was on-par with or could even outperform traditional implementations of material flow control in speed and robustness. As the manufacturing industry moves towards a more customer-driven market, 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 concept of an auction-based task-allocation system in a manufacturing context was introduced in [6]. Findings showed that this system was robust in the face of disruptions and unpredictable events, due to its dynamic way of allocating tasks through auctions.

The application of multi-agent technology has been thoroughly documented in [7], in which they described a baggage handling system (BHS) as a production system with

a number of random inputs. As the final destination of bags was unknown until its tag was scanned and the bag entered the system, off-line planning and scheduling became impossible. In addition, there was the logistical challenge of storing baggage before loading the plane. This resulted in a system with low flexibility and high level of shared resources. Both full and empty baggage totes shared conveyor belts, which meant that this was a limited, shared resource prone to congestion challenges.

Experiments with reinforcement learning in combination with MAS has been performed on the logistical challenge of air traffic flow management. In [8], it was pointed out that the current air traffic control was centralized, and consequently, slow to respond to changing weather conditions and other unexpected events. They proposed a system of agents who were assigned specific 2D points throughout the airspace, in which each agent was tasked with keeping the separation of incoming traffic to a required distance. Reinforcement learning was used to learn the appropriate control policies. Results showed that using agent-based rewards performed better than using the full system reward, as agents received more direct feedback on their actions, improving learning performance.

Finally, various approaches for route planning for autonomous guided vehicles has been examined. [9] focused on a closed warehouse application enhanced by industry 4.0 technology. Using this approach, the authors were able to create a dynamic route planning system for real-time applications. The main area of interest in this case was on creating a fast algorithm able to run on hardware with limited computational power. This was achieved by creating an encoded abstraction of the agent's environment, reducing the size of the input vector. Sensor data and previously generated routes were collected and used to build the model to find the optimal route. This model had to be maintained over time, and a long data collection period was required in order to aggregate sufficient training data. A possible remedy for this was the use of simulated datasets, the authors suggested.

Similarly, [10] showed how LiDAR sensor data combined with odometer data could be used to determine a robot's position in an unknown environment. However, the major drawback of this approach was that a manually constructed map created up-front was required.

A similar approach examined a LiDAR-only based navigation algorithm for a weeding robot [11]. Here, they experimented with a general and robust approach for autonomous robot navigation in an unknown environment. Information about the environment was collected and used to construct a model in real-time, which the robot utilized in order to move along detected lines in rows of crops. However, no visual information was available, as this approach relied on LiDAR only. As a consequence, 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.). Despite this, their approach was still promising given the limited data available to the robot, and where no prior information about an environment was available.

The use of visual information combined with LiDAR scans to improve accuracy has been discussed in [12]. Using an extensive set of pre-collected images, a robust and accurate strategy for the robot was created. However, the extensive manual effort

required upfront was a major drawback. Relying on pre-collected image data also meant that changes in the environment could throw the robot off.

In summary, we see many examples of MAS and RL in the industry, both for baggage handling and logistics, and obstacle avoidance and navigation for autonomous agents. However, for the work done with regards to robotics and autonomous agents, the main focus has been on single agents, in environments with few obstacles. [7] did address a system with 300 agents, but they were not dynamic, rather they served the role of mediators controlling resource flow throughout the BHS. Furthermore, the state space in works like [9 - 12] was relative to the static environment, and not to the agent itself, making for a less flexible system. On the other hand, [8] used a system with rewards and state space relative to the agents, which was able to adapt quickly to unexpected events. The agents in [9] used a centralized approach, where the model had to be monitored and re-trained regularly if performance began to decline.

Based on these findings, this paper aims to examine how a highly complex, multi-agent system can be broken down and represented using a simple, tabular approach, with no human interaction. Furthermore, control will be bottom-up and decentralized, and with no information about the environment required up-front.

3 Approach and Implementation

It was decided to examine a model of multiple autonomous, mobile agents in an environment consisting of a large number of obstacles (200+). A simple agent obstacle avoidance simulator based on off-policy RL was developed from scratch [13]. In order to examine the effects of congestion and space allocation in a dynamic environment, this previous implementation was extended, and new benchmarks were defined.

3.1 State Space and Action Definition

The simulation should consist of a simplified model, while still having some relevancy to real-world applications. This was achieved by using a discretized state space, where the current state of an agent was defined by its distance and angle relative to its nearest obstacle, somewhat similar to [8]. The state space was constructed using two different discretization approaches:

Relative Angle and Distance to Nearest Obstacle. In the first approach, the state was based on the distance and relative angle to the nearest obstacle in range. Distance and angle values were discretized to construct the state space, defined as a 2D matrix of width W and height H. W and H thus determined the resolution of the state space in each dimension, which will be discussed later. Sensor values were discretized to a' and d' using the following formulae:

$$a' = \frac{a}{360} \times (H - 1) \tag{1}$$

$$d' = \frac{d}{100} \times (W - 1) \tag{2}$$

This is visualized in the figure below:

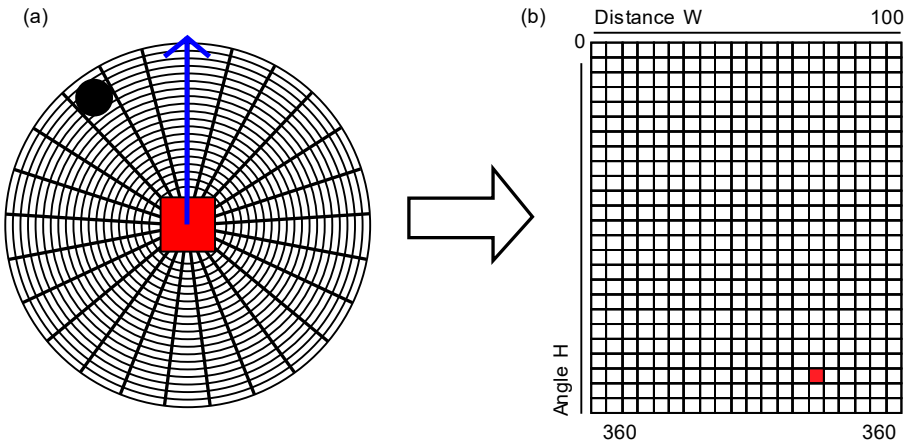

Fig. 1. State space discretization example 1. Current agent orientation and heading is shown by the blue arrow.

In (1), the measured angle value a is discretized into intervals of 15 degrees. Similarly, in (2), the measured distance d is divided by the total sensor range of 100 pixels and discretized into intervals of 5 pixels. Fig. 1 (a) shows a visual representation of the state space, as seen by the sensor of a square agent. The values a' and d' thus become the indices of the state space matrix in Fig. 1 (b). In the above example, an obstacle represented as a black circle 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 the current implementation, H was set to 25 and W was set to 20. This produced a 2D state space matrix of size $25 \times 20 = 500$ possible states.

Cardinal Partitions and Number of Obstacles in each Partition. In the second approach, the state of the agent was defined based on relative angle and the number of obstacles in each interval. Instead of discretizing the angles into 25 15-degree intervals, four 45-degree partitions based on the cardinal directions were manually defined: Front, Right, Rear, and Left:

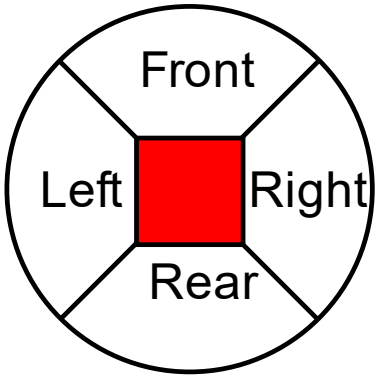

Fig. 2. State space partitions based on cardinal directions.

A visual representation of the state space discretization and partitioning is shown in Fig. 2. Next, the actual state was determined based on the presence of one or more obstacles in each quadrant. The ideal state would thus be no obstacles in the Front partition, where the agent is heading. Similar to the first approach, states were mapped to indices in a 1D state matrix by the following method:

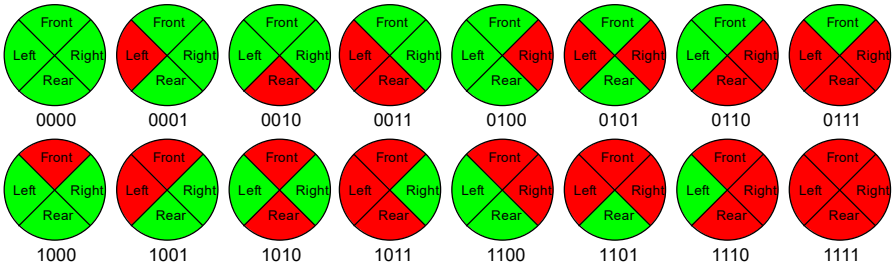

Fig. 3. Partition state mapping visualization.

Fig. 3 shows how indices were generated based on the presence of obstacles in each of the quadrants. This is simply a true or false value, represented as 0 and 1, going clockwise from the Front partition. Since there were four partitions of two "sub-states" each, the total size of this state matrix became $2^4 = 16$. Indices of the state matrix were then generated by converting from the binary to decimal value.

Action Definition. In every state of both approaches, an agent could take one of four actions *A'*: turn left, turn right, wait, or keep going. The actions for turning left and right were performed in the simulator by rotating the agent's direction vector $\vec{V}$ by a fixed amount of 10 degrees in either direction. The wait action was implemented by not updating the agent's position for 10 simulator ticks. As the simulator ran at 125 ticks per second, the agent would wait for 1250 milliseconds.

3.2 Reinforcement Learning Implementation

The reinforcement learning algorithm used was SARSA, which is an on-policy reinforcement learning algorithm [14]. Advantages of SARSA may include faster convergence compared to off-policy, since the agent relies more on previous knowledge when selecting its next action, using a greedy policy. However, balancing exploration versus exploitation becomes even more important using this approach. To address this, a random exploration factor r was added.

Next State Estimation. In order to determine the next state following the selected action, "look-ahead" functions for each of the two approaches were implemented. For the first approach, to calculate an agent's next position, its current trajectory according to the selected action *A* was used, as shown in Fig. 4. The result from this computation was in turn used to estimate the next distance and relative angle d_t+1 and a_t+1, and thus $S_t+1 \mid A$:

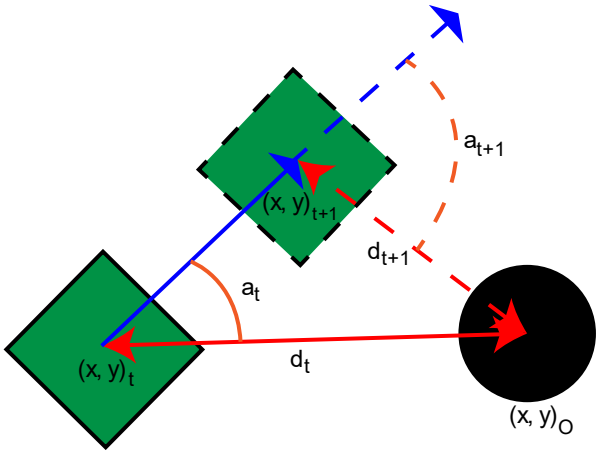

Fig. 4. Look-ahead computation.

The agent is shown in Fig. 4 as a green square, with an obstacle shown as a black circle. The current distance d_t and relative angle a_t were provided by the sensor. $|\vec{V}|$ was also known, as this was the constant velocity of all agents. The next estimated position given action *A* was then found by:

$$(x,y)_{t+1} = (x,y)_t + \vec{V}|A\,,\ \vec{D}_{t+1} = [x_O - x_{t+1}, y_O - y_{t+1}] \tag{3}$$

Based on (3), d_{t+1} and a_{t+1} could be calculated:

$$d_{t+1} = \sqrt{x_D^2 + y_D^2}\,, a_{t+1} = \cos^{-1}\frac{\vec{V}\cdot\overrightarrow{Dt+1}}{|\vec{V}|\times|\overrightarrow{D_{t+1}}|} \tag{4}$$

For the partition approach, the next state following a selected action was estimated by simply "rotating" the observed state seen in Fig. 3 according to the current action. This is shown in the below figure:

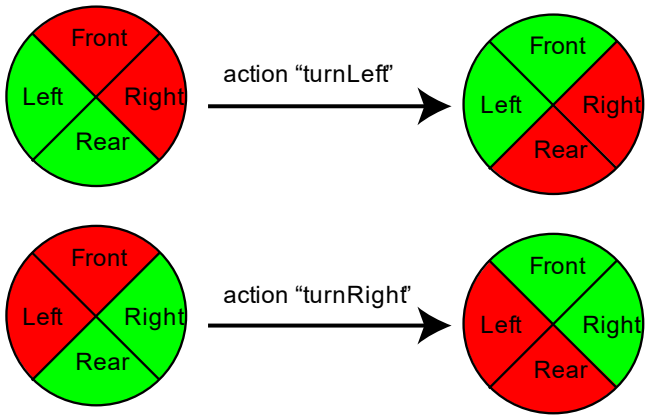

Fig. 5. Next state estimation for cardinal partition approach.

Next, the function to determine what action to take in any given state was defined as follows:

Algorithm 1

Pick a random decimal number $n \in [0, 1)$
if $n < r$ **then**
    return random action $A \in A'$
end if
Max value $q \leftarrow -\infty$
$A \leftarrow keepGoing$
for each action $a : A'$ **do**
    if $Q(S_t, a) > q$ **then**
        $A \leftarrow a, q \leftarrow Q(S_t, a)$
    end if
end for
return A

Next, the estimated reward was calculated. For the relative angle and distance approach, the expected reward was based on the predicted distance to the nearest obstacle:

Algorithm 2

if $d_{t+1} < 10$ **then**
    return -10
else if $d_{t+1} < 20$ **then**
    return 10

```
5: else
6:    return d_{t+1}
```

This meant that if the agent was less than 10 pixels away from an obstacle, it counted as a collision, and the agent received a penalty in the form of a negative reward. Next, if the agent was closer than a "safe distance" of 20 pixels, it received a low, positive reward. Only if this safety distance was exceeded would the agent receive a larger reward. For the partition approach, the reward was based on the number of obstacles in the Front partition after "rotation":

Algorithm 3

```
1: if number of obstacles in Front > 0 then
2:    return −(number of obstacles in Front)
3: else
4:    return 5
```

Finally, combining Algorithm 1, 2, and 3 with (1, 2, 3, 4) resulted in the following final algorithm used for agent learning:

Algorithm 4

```
1:  Initialize Q(s, a) with default values 0
2:  Select initial action A
3:  while true do
4:      Get sensor readings d_t, a_t for current agent position t
5:      Determine state S_t using (1) and (2)
6:      Apply rotation associated with A to V⃗
7:      Calculate d_{t+1} and a_{t+1} for next state S_{t+1} using (3) and (4)
8:      Get action A_{t+1} ∈ A′ with the highest Q-value for S_{t+1} from Q(s, a) using Algorithm 1
9:      Calculate reward R based on d_{t+1} using Algorithm (2, 3)
10:   Q(s_t, A) ← Q(s_t, A) + α × (R + γ × Q(S_{t+1}, A_{t+1}) − Q(s_t, A))
11:     A ← A_{t+1}
12: if A is not WAIT then
13:     Translate agent position according to V⃗(S_t ← S_{t+1})
```

The values of control parameters *H*, *W*, α, and γ used in the implementation are shown in Table 1.

Table 1. Control variable values

Variable	Value
r	0.038
α	1.0
γ	0.8
W	25
H	20

3.3 Obstacle Generation, User Interface, and Interactivity

A level of interactivity was desired in order to make the simulator more dynamic. In the simulator window, cursor coordinates were captured when clicking, creating a new obstacle in the specified location. By default, a preset number of obstacles were generated and distributed in the environment for the agents to avoid. No explicit hard world border was defined. In addition to the randomly distributed obstacles, lines of obstacle objects were added, surrounding the environment. This was done in order to make agents learn to stay within the screen using the same algorithm as for obstacle avoidance. How this addition affected the agents' behavior will be discussed later. When agents did wander off screen, they were repositioned in their original starting location and continued exploring. In the simulator, agents were also considered obstacles. The sensors used could detect both static obstacles, and other agents present in the scene.

4 Results

A set of experiments were performed in order to examine the agents' ability to learn to avoid obstacles and deal with congestion. Firstly, it was of interest to test a single agent in a static environment, to see if the basic concept of the implementation worked as intended. A single agent was added to an environment of size 1184x761 pixels, with 250 obstacles randomly distributed. Both approaches to the state space discretization and mapping produced similar visual results:

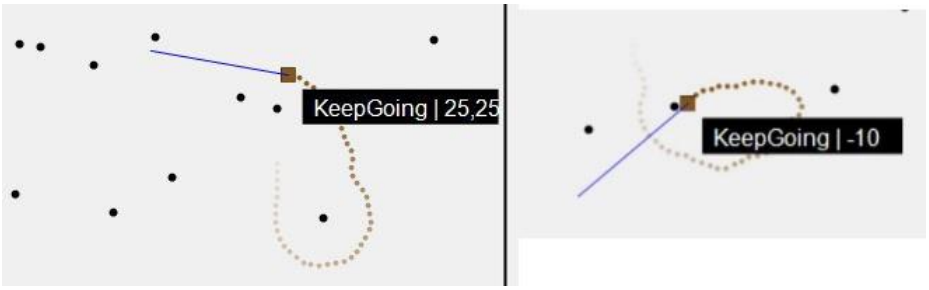

Fig. 6. Single agent with action and reward label.

In Fig. 6 we see a single agent with trail and status label avoiding obstacles. On the left, the agent was able to stay more than 20 pixels away from the obstacle when selecting action “KeepGoing”, and received a reward based on distance. In the right-hand part of the figure, the agent came too close to an obstacle, and thus received a reward of -10, as described in Algorithm 2. It is also possible to see from the trail that the agent was able to navigate and avoid the obstacles. A selection of screenshots of single agent navigation are presented below:

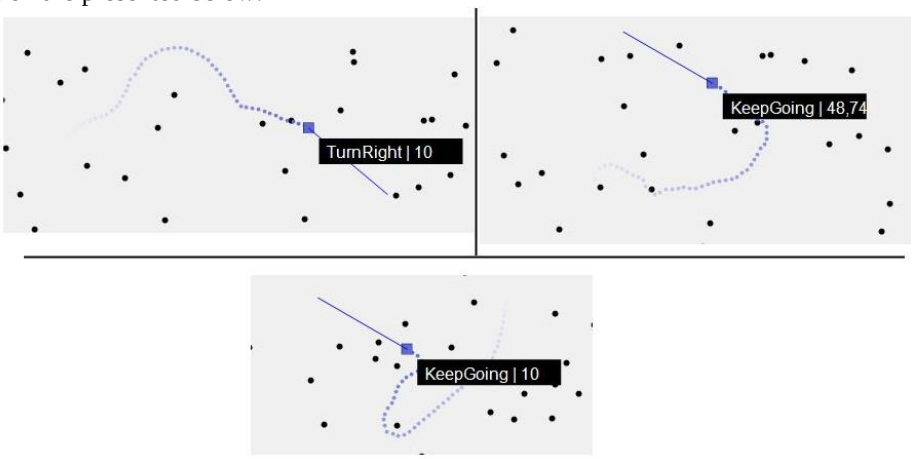

Fig. 6. Blue agent navigating.

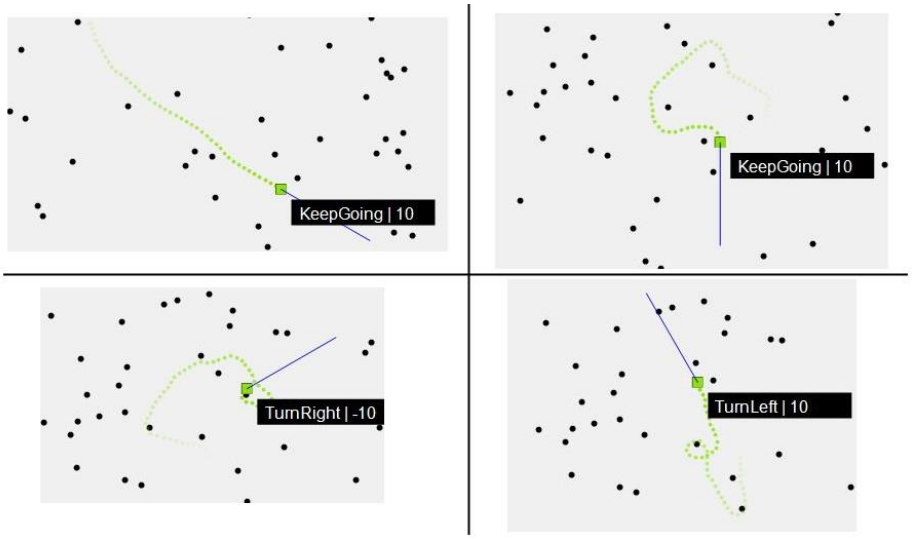

Fig. 7. Lime agent navigating.

As previously mentioned, no explicit world border was defined in the reward function. To discourage agents from leaving the screen, a border made up of static obstacle

objects was added, in addition to the randomly distributed obstacles, as can be seen in Fig. 9. Agents would sometimes leave the scene even with this border in place, but were more likely to swerve back into view, also seen in Fig. 9:

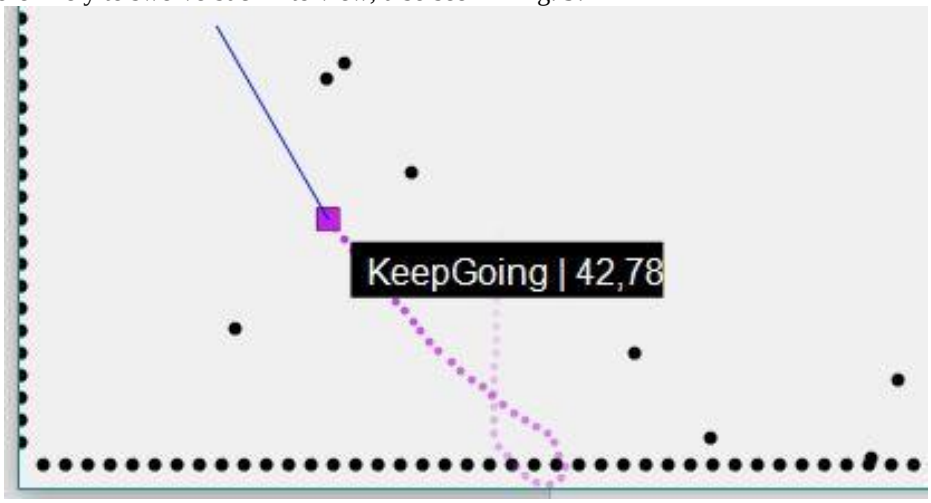

Fig. 8. Agent responding to border of obstacles.

This border also served as a mechanism to introduce space restraints and congestion to the system. To get a better understanding how this border affected the agents, the number of times agents left the scene over time, with and without the border was recorded. 25 agents and 250 obstacles were added to the environment. 10 instances of the simulator were run simultaneously for 10 minutes, and their numbers averaged to account for the randomness inherit in the reinforcement learning approach.

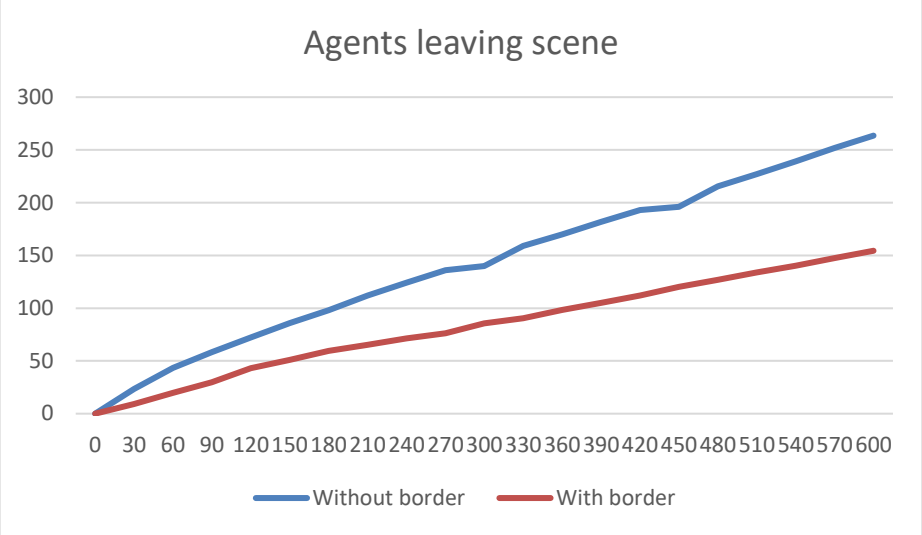

Fig. 9. Agents leaving scene with and without obstacle border.

Since this benchmark was based on time, the Wait action was disabled. Otherwise, the agents would potentially get good results by simply standing still for the duration of the simulator.

Fig. 10 shows that agents had left the scene more than 250 times after 10 minutes (600 seconds) when no border was defined. When introducing the obstacle border, agents were more inclined to stay within the screen, and had wandered off 154,4 times on average.

Another interesting aspect regarding AGVs in enclosed spaces is to examine how the size of the environment affects congestion. In the next experiments, the size of the simulator window was reduced, and the number of collisions for each configuration was measured. As in the previous experiment, the number of agents used was 25 and the number of obstacles 250. Each configuration was run 10 times for 300 seconds, and the resulting values averaged. The size of the window in each experiment is listed below. Due to software constraints, the actual environment in the simulator is slightly smaller than the total window size. This discrepancy was accounted for in the calculations performed.

Table 2. Simulator environment sizes used in experiments.

Window size in pixels	Environment size in pixels	% of default area
1200x800 (default)	1184x761	100
900x600	884x761	55
600x400	584x361	23.4

When the size of the area was compared to the total number of collisions after 300 seconds, the following graph was produced:

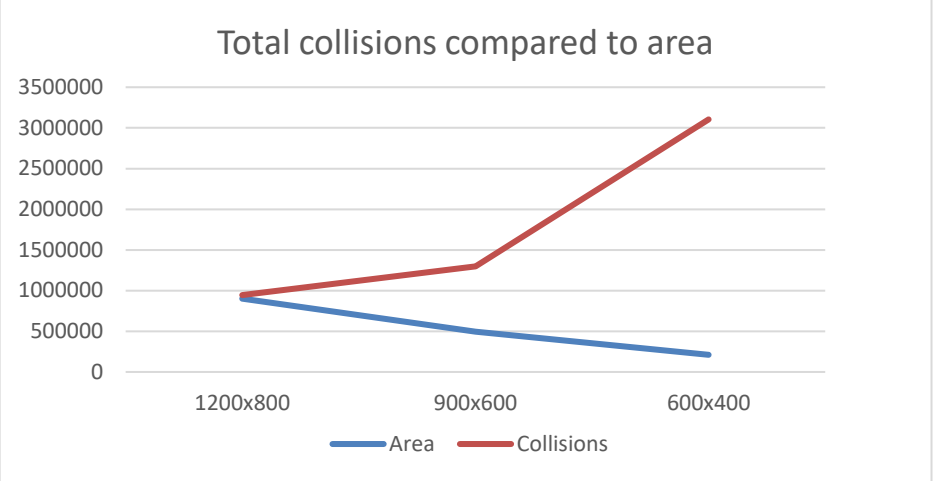

Fig. 10. Total collisions compared to simulator area.

Finally, the performance of both approaches was measured and compared based on the number of collisions per distance in pixels travelled by each agent. For this experiment, the number of agents was 20, the number of obstacles was 200, and the Wait action was enabled. The screen size was left at the default, and the simulator was run for 5 minutes in both configurations. For the relative angle/distance approach, the average number of collisions per distance in pixels was 0.337. For the cardinal partition approach, the resulting average was 0.072. Both approaches had a similar memory usage of less than 30 MB of RAM, as seen in Fig. 12:

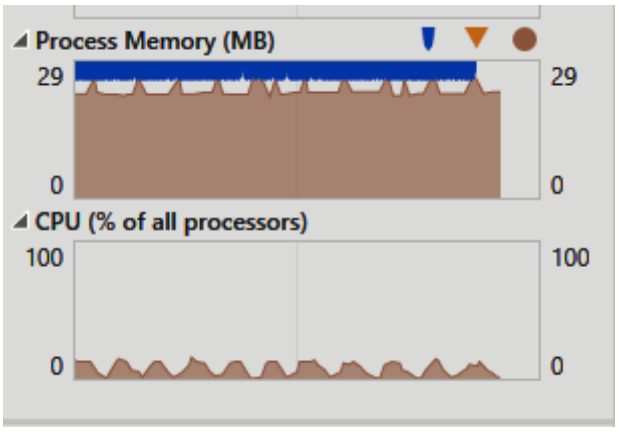

Fig. 11. Memory and CPU usage of the implementation.

5 Discussion of Results

As seen in Fig. 6-9, agents were able to avoid obstacles, and received a negative reward when they failed to do so. However, screenshots can only show so much when dealing with moving objects. Using trails to visualize the path of the agent were imagined to be helpful in order to show the movements of agents over time. This became clearer when introducing the world border consisting of standard obstacle objects. As seen in Fig. 9, the agent wandered off slightly, then was able to reorient itself and navigate back into view. This screenshot was picked specifically to demonstrate this behavior, as agents would sometimes disappear completely off screen. In those cases, the agent was automatically repositioned to its random starting position. No penalty was given for leaving the scene other than obstacle proximity. The effect of this is even clearer in Fig. 10, which compares the number of times agents left the scene with and without the border in place. From this, it is possible to see that the agents were incentivized to stay within the screen solely based on the learned obstacle avoidance strategy.

Next, the effects of limiting the space of the agents were examined. In these experiments, a collision was counted every time an agent received a negative reward, as defined in Algorithm 2. When looking at Fig. 11, we see the relationship between the number of collision and total area. As one might expect, there was an inverse correlation. What is interesting is that when reducing the area to 55% of the default, collisions only went up by 37%. In other words, half the area did not equal double the collisions. However, this did not hold when reducing the environment further to 23%. In this case, collisions rose by 228%, as seen in Fig. 11. The limitation of agent actions is also expected to have had an impact, since agents could only turn left or right or keep going straight. With less space to “idle” in, constantly moving while still maintaining a safe distance becomes much more difficult due to congestion. This suggests that this approach only performs acceptably down to a certain environment size, while maintaining the number of obstacles and agents. Space itself becomes a limited resource. As agents were required to maintain a minimum safe distance to each other and other obstacles in the environment, the challenge of allocating this between agents presented itself. Obstacles were not able to move, and thus agents were required to negotiate. However, no explicit negotiating mechanisms were implemented, meaning that agents just had to skirt around each other as best they could. In larger environments this becomes less of a problem, mostly due to limited agent vision. An agent does not consider an obstacle it cannot detect.

Furthermore, when comparing the performance of the two approaches, it appears that the cardinal partition approach outperforms the relative angle/distance approach when we only consider collisions per distance travelled. This is interesting since that approach only relied on 16 states, while the angle/distance approach used 500 states. The next-state calculation was also simpler and was achieved in practice by simply swapping the indices of the four partitions according to the selected action. Being able to achieve better of performance with a reduced number of states could be beneficial when running this system on low-power hardware for real-life applications and suggests that a partition-based approach shows some promise for further investigations.

5.1 Further Work and Recommendations

As mentioned above, the introduction of a negotiating mechanism to help with space allocation in small environments is expected to benefit the system and help deal with congestion. In practice, all agents have to share a common good in the form of space, or more specifically, the maximum distance from obstacles. Negotiations could be in the form of agent-to-agent, or a mediator agent could be introduced, like in [7]. There, bag totes had to share a common good in the form of conveyor belts. Such a mediator agent would need to have an overview of the environment and every agent position in order to designate spaces the agents could move to. This is imagined to speed up decisions when faced with congested situations. Furthermore, using multiple, or even other types of sensors, to define the state space should be looked into. Using LiDAR only has been proven to be a drawback previously [11] and introducing visual information did improve performance in certain situations, as shown in [12].

Further expanding or enhancing the state space, as demonstrated in the cardinal partition approach should also be looked into. Even though that approach showed promise with only 16 states, introducing additional granularity to the partitioning should also be investigated. One approach that comes to mind is adding additional substates to each partition, like “no obstacles”, “a few obstacles” and “many obstacles”. This would result in a state space of 81, which is still considerably less than 500. More partitions could also be added to achieve a finer level of control for specific situations. Memory wise, using 500 or 16 states had minimal impact during benchmarks. Adding additional states would also mean that manually defining rules of action for each state would be increasingly difficult, strengthening the argument that the agents should create their own models without human intervention. In summary, using a small, discretized state space as proposed did show some promise, but continuous state spaces and actions should also be investigated.

6 Conclusion

Two methods to simplify and model the state space for autonomous mobile agents was presented, and a simulator was implemented and demonstrated in software. The presented approaches demonstrated how a complex environment with multiple obstacles could be simplified using a discretized state space based on a single LiDAR-like sensor. Agents were implemented with an internal representation of their environment created in real-time, and an on-policy reinforcement learning algorithm was used in order to learn to avoid randomly distributed obstacles. Discretizing the state space allowed for a simple, tabular approach to reinforcement learning, which is less resource intensive than high-dimensional or even continuous state spaces. Even with this simplified approach, it was possible to observe learning. The environment was made dynamic by introducing multiple agents, each moving around and trying to avoid each other as well as a high number of static obstacles. No information about the environment was required up-front, and the agents were purely self-optimizing and relied on decentralized control.

Results showed some promise using a single LiDAR-like sensor to construct simple state spaces, which can be useful when dealing with cheap, low-power equipment in the real world. The computational overhead for a single agent is low compared to more complex system of high continuous state- and action spaces. A complex environment with multiple agents and 200+ obstacles was broken down into a small state spaces, requiring less than 30 MB to run including the graphical user interface, while still being useful for obstacle avoidance. One might even draw parallels to "social distancing", which has become much more relevant in the time of writing. However, there is still room for improvement, especially when considering the use of multiple sensors to increase performance. A system of negotiation among agents could also be useful in situations when space is limited and should be investigated further. Creating high-level actions like "turn left" and "turn right", or "keep going", made debugging, demonstration and general understanding of the system easier. Using a similar approach when building and configuring RL systems for real-world AGVs may make simple proof-of-concepts easier to deploy rapidly. Further tweaking to low-level actuator control can thus be made while still providing a consistent interface to end users and developers.

In summary, using additional sensors in addition to a system of negotiation is imagined to increase performance of the system. Finally, additional functionality could be implemented in the simulator itself in order to model and test even more real-world use cases related to baggage handling and logistics.

References

[1] Back to the future? Airline sector poised for change post-COVID-19, https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/back-to-the-future-airline-sector-poised-for-change-post-covid-19, last accessed 2021/06/08.

[2] Baggage IT Insights 2020, https://www.sita.aero/resources/surveys-reports/baggage-it-insights-2020, last accessed 2021/03/04.

[3] Meyer, G., Främling, K., Holmström, J.: Intelligent products: a survey. Computers in Industry 60(3), 137-148 (2009).

[4] Sella, R., Rassõlkinb, A., Wanga, R., Otto, T.: Integration of autonomous vehicles and Industry 4.0. In: Proceedings of the Estonian Academy of Sciences, pp. 389-394. Estonian Academy Publishers, Tallin, Estonia (2019).

[5] Thürer, M., Fernandes, N. O., Stevenson, M., Qu, T., Huang, G. Q.: Self-organizing material flow control using smart products: an assessment by simulation. Journal of Industrial and Production Engineering 38 (2), 148-156 (2021).

[6] Bussmann, S., Schild, K.: Self-organizing manufacturing control: an industrial application of agent technology. In: Proceedings Fourth International Conference on MultiAgent Systems, pp. 87-94. IEEE (2000).

[7] Hallenborg, K., Demazeau, Y.: Dynamical control in large-scale material handling systems through agent technology. In: 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 637-645. IEEE (2006).

[8] Tumer, K., Agogino, A.: Distributed agent-based air traffic flow management. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, pp. 1-8. Association for Computing Machinery, New York, NY.
[9] Duc, D. N., Huu, T. T., Nananukul, N.: A dynamic route-planning system based on Industry 4.0 Technology. Algorithms 13(12), 308, (2020).
[10] Cheng, Y., Wang, G. Y.: Mobile robot navigation based on lidar. In: 2018 Chinese Control And Decision Conference, pp. 1243-1246. IEEE (2018).
[11] Malavazi, F. B. P., Guyonneau, R., Fasquel, J.-B., Lagrange, S., Mercier, F.: LiDAR-only based navigation algorithm for an autonomous agricultural robot. Computers and Electronics in Agriculture 154, 71-79 (2018).
[12] Su, Z., Zhou, X., Cheng, T., Zhang, H., Xu, B., Chen, W.: Global localization of a mobile robot using lidar and visual features. In: 2017 IEEE International Conference on Robotics and Biomimetics, pp. 2377-2383. IEEE (2018).
[13] Jansson, A. D.: Simulation of obstacle avoidance for multiple autonomous vehicles in a dynamic environment using Q-learning. International Journal of Computer and Information Engineering 15, 267-272 (2021).
[14] Sutton, R. S., Barto, A. G: Reinforcement learning: an introduction. 2nd edn. The MIT Press, Cambridge, Massachusetts, MA (2015).