Differential Drive Controlling Robot Comparison

Eric Lindesay

Abstract

In this paper, we developed and compared four differential drive agents on three different maps using a custom environment. We compared a random, reactive, A* and Q-learning agent on the criteria of actions taken, timesteps taken, distance travelled and progress made on the map. It was found that the A* agent completed the map in the shortest distance but required more timesteps compared to the reactive agent. This paper evaluates the four models and provides methods on which to expand.

Contents

3192 words

1. Introduction

This paper presents a comparison of four models solving a racing game on three different maps. The models must go through each checkpoint in order, aiming to reach the end as quickly as possible while avoiding the walls. These models control a Differential Drive Robot (DDR) with multiple sensors to detect walls. DDRs have two wheels, which can be controlled independently, allowing the robot to rotate on the spot. In this work, a random agent, a reactive agent, an A* pathfinder (Foead *et al.*, 2021) and a Q-learning agent (Watkins and Dayan, 1992) are compared.

As a baseline to compare the models, a random agent was included, which takes actions randomly. This was chosen to compare how a model which acts solely on its current state with no knowledge of the past or future performs. Additionally, a reactive agent was chosen to evaluate the performance of a more intelligent model, even without this knowledge. An A* pathfinding agent was chosen to highlight the performance of a model with a perfect understanding of its environment and the optimal path. A Q-learning approach was also included to portray a reinforcement learning technique to see how effective the model is at learning this environment and how that compares to the other models.

It is expected that the A* agent will perform the best due to the environmental knowledge and the random agent to perform the worst. It is difficult to predict whether the reactive agent or the Q-learning agent will perform better in this environment, however, it is predicted that the reactive agent will be more tuned to the environment allowing it to perform better in the chosen maps but potentially worse on new, unseen maps.

1.1. Q-learning

Q-learning is a reinforcement learning technique of machine learning, used on Markovian domains (Sutton and Barto, 2018). It is a model-free approach which means it doesn't attempt to estimate the probability distribution of the state transitions (a result of the action taken) or the reward function associated with the Markov Decision Process (MDP). Model-free methods generally have higher computational efficiency, allowing more information in the state. Q-learning can be viewed as a Dynamic Programming approach (Wang *et al.*, 2019) and it contains a table (called the Q-table) which holds the observed reward for each action it can take at a given state. It uses this table to either take the best action (exploitation) or, at a small random chance, take a random action to explore the environment further. After taking an action, it evaluates the current state and calculates the reward of that movement. It uses this reward to update the Q-table using the Bellman equation (Pesch and Bulirsch, 1994) to learn what actions to take over time. Q-learning is generally used for discrete action and state spaces and it was chosen over alternative methods like Proximal Policy Optimisation (Schulman *et al.*, 2017), which works on a continuous space, due to its computational efficiency which is important on a low specification machine such as the one used to train the models. Additionally, given enough exploration time, Q-learning can identify an optimal action selection policy for any finite MDP which is very important for our goal.

2. Method

2.1. Environment

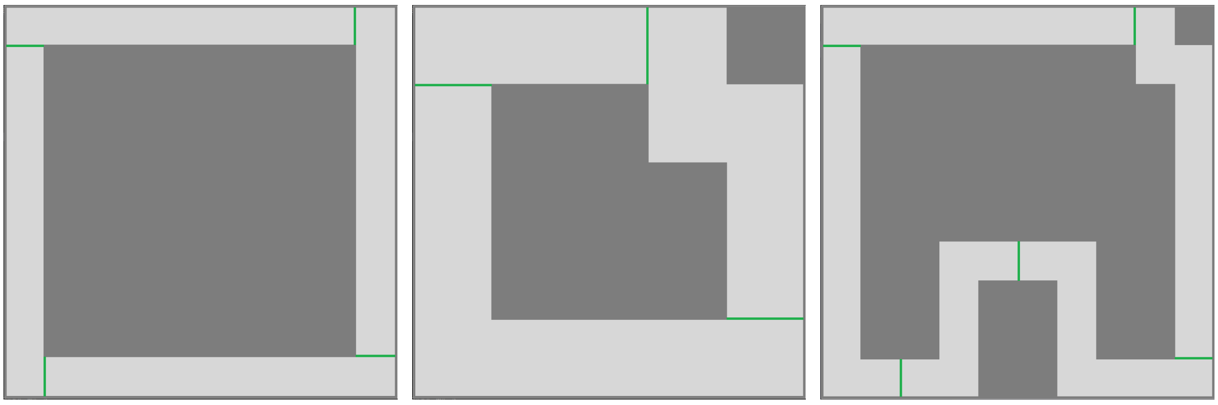
Figure 1: The three maps the agents were compared against, increasing in difficulty from left to right. The walls are coloured grey and the checkpoints are coloured green. The checkpoints are numbered clockwise.

The environment for the agent consists of the walls the agent can collide with and checkpoints the agent must reach in order (numbered clockwise) with the agent starting in the top left corner. A collision means the agent has failed at that attempt. The checkpoints are provided so the agents have goals to reach as they explore the space. It also increases simplicity for the reinforcement learning agent as instead of rewarding based on the total distance moved, the model can be rewarded on the distance to the next checkpoint which means the agent doesn't spin in circles to gain a reward. Three maps were created, shown in Figure 1, on which the agents were tested and compared. The environments were designed to be simple to reduce the complexity of training the agents due to the limited time and hardware resources present.

2.2. Agents

As previously mentioned, the developed models were: a random agent, a reactive agent, an A* agent and a Q Learning agent. The agents are simulated two-wheel differential drive robots, based on the code provided in the labs (Dudek and Jenkin, 2024). Agents consist of sensors which can be used to find the distance to the walls and can be used to find the distance to the next checkpoint.

2.2.1. Random Agent

This agent took a random walk around the map. At each timestep it can either turn left (set the speed of the left wheel to -1 and the right wheel to 1), turn right (set the speed of the left wheel to 1 and the right wheel to -1) or move forwards (set the speed of both wheels to 3).

2.2.2. Reactive Agent

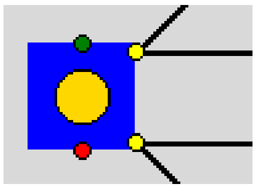
Figure 2: Visualisation of the sensors on the reactive agent. The black lines show projections from the four sensors.

Algorithm 1: Reactive Agent Algorithm

```
input: left, right, left_diag, right_diag    ▷ The distance between each sensor and the nearest wall in the
                                               ▷ direction the sensor is facing
member variable: rotation_frames    ▷ The number of frames the agent is rotating for consecutively
threshold ← 30
rotation_threshold ← 100
if rotation_frames > rotation_threshold then    ▷ If you have rotated for 100 consecutive frames
                                                 ▷ Rotate until you can move forwards
    if left < threshold and right < threshold then
        move forwards
    else
        turn left    ▷ Direction chosen arbitrarily
else if left < threshold or right < threshold then    ▷ If either of the front two sensors are too close
    if left_diag < right_diag then    ▷ If the left diagonal sensor is closer to a wall than the right diagonal sensor
        turn right
    else
        turn left
else if left_diag < threshold then    ▷ If the diagonal left sensor is too close to the wall, turn right
    turn right
else if right_diag < threshold then    ▷ If the diagonal right sensor is too close to the wall, turn left
    turn left
else
    move forwards

if not move forwards then
    rotation_frames ← rotation_frames + 1
else
    rotation_frames ← 0
end
```

This agent consists of four sensors, shown in Figure 2. These are used to sense how close the agent is to the walls. The agent decides its action based on the process detailed in Algorithm 1. The algorithm is very simple: if it gets close to a wall with the front two sensors, it rotates towards the sensor furthest from the wall. If either of the diagonal sensors gets too close to a wall, it rotates away from the wall.

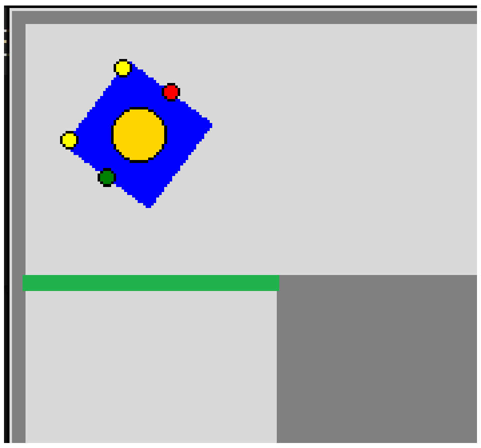

Figure 3: The reactive agent, stuck facing the bottom right corner of map 1.

Initially, during development, there were issues where the bot would get stuck in a corner, as shown in Figure 3, continuously alternating between rotating left and right. To fix this issue, a condition on Line 3 was added. If the agent rotates for 100 continuous frames then we can say (with a high degree of certainty) that it is stuck rotating in a corner. The agent can then continuously rotate until it can move forward allowing it to become unstuck. It was also found that increasing the threshold for when a sensor is too close to the wall encourages the agent to be more central, reducing the chance of it getting stuck or crashing into a wall. However, increasing this sensor sensitivity too high means the agent is constantly within the distance threshold and cannot move.

2.2.3. A* Agent

An agent consisting of a heap-based implementation of A* pathfinding algorithm (Foead *et al.*, 2021) was developed. This agent uses this to calculate the optimal route between its current position and the next checkpoint. This model has full information about the map and wall positions, which other models do not have, so it is expected that this model will perform better than the other agents.

Despite this model having cells which are 5x5 pixels wide, resulting in 40,000 cells, there is not a large time cost as the map is simple and the checkpoints are relatively close. The agent only calculates the distance between the current and next checkpoint. After finding the shortest path, the agent can use its left and right front-facing sensors, and the relative difference in distance to the point, to rotate towards the next point in the path. When reaching the checkpoint, it then calculates the shortest route to the next checkpoint.

2.2.4. Q-Learning Agent

Due to hardware and time limitations and to discretise the search space, the Q-learning agent has simplified movement. Instead of being able to move forward or rotate in either direction by a small amount, it rotates and moves in any of the cardinal directions. This reduces the complexity of the environment.

The Q-learning agent acts upon a state consisting of four features: the current direction the agent is facing, whether it is facing the checkpoint, whether the left diagonal sensor is within a certain threshold and whether the right diagonal sensor is within the same threshold. It then uses this information to take one of four actions: move north, south, east or west. The first two features were chosen so the agent learns it should go towards the checkpoints. The second two features were selected to give the agent awareness of its surroundings and the locations of the walls. Without these final two features, the agent would drive towards the checkpoint and crash into any obstructive walls.

Previously, we tried the model with a different set of features: the distance to the checkpoint, the value of the front left sensor, the value of the front right sensor, the value of the diagonal left sensor and the value of the diagonal right sensor, all rounded to one decimal place, however, this created too large a feature space and the agent would continuously spin in circles. This could be revisited by adding a feature to indicate whether it is facing the checkpoint like in our current implementation, instead of solely the distance to the checkpoint.

The agent was trained on map 1. To increase robustness, it was trained in batches of 5 agents, randomly initialising the agent position and angle each time allowing it to see more of the map. This makes the Q-table more complete, allowing it to choose good actions in different scenarios. Additionally, the hyper-parameters chosen were an epsilon of 0.1 (meaning 10% of the time instead of exploiting the Q-table it explores the environment and takes a random action), a learning rate of 0.05 and a gamma of 0.6 meaning it favours future reward more than immediate.

3. Results

To evaluate our models, they were compared on four criteria: number of actions taken, timesteps taken, distance travelled, and the agent's progress around the map (how far they got as a percentage).

An action is when the agent has to make a choice. For example, the agent may rotate left or, if it is the Q-learning agent, it may rotate north. A timestep occurs every time the agent is drawn to the screen, though the agent might not move each timestep; an A* agent may not move for 5 timesteps as it takes time to calculate the optimal route and the Q-learning agent may take 10 timesteps to complete an action (e.g. rotating north). The number of actions taken and timesteps are very closely linked. It was decided to include them both in order to represent and compare the time the agents require to "think" vs act.

The distance travelled is simply how far the agent moves during its lifespan. This does not account for the fact that the agent may backtrack and repeat itself so the progress metric was included which represents how far the agent travelled around the map. The Q-learning agent does a lot of backtracking which is what caused us to include the progress metric in order to allow fairer comparison with other models.

The Random and the Q-learning models were sampled 500 times for each map to get enough data to provide a mean for the different criteria which can give us confidence that the values are representative of the model's performance. However, the Reactive and A* models were only sampled once since they are deterministic.

Model	Map	Completed	Samples	Actions		Timesteps		Distance		Progress (%)	
				μ	σ	μ	σ	μ	σ	μ	σ
Random	1	✕	500	102.0	64.2	102.0	64.2	102.4	65.5	2.8	1.9
	2	✕		219.7	176.7	219.7	176.7	216.6	175.6	5.3	4.6
	3	✕		103.0	63.2	103.0	63.2	103.2	62.8	2.7	1.8
Reactive	1	✓	1	1233	0	1233	0	3486	0	100	0
	2	✓		1138	0	1138	0	3201	0	100	0
	3	✓		1529	0	1529	0	3987	0	100	0
A*	1	✓	1	2267	0	2317	0	3399	0	100	0
	2	✓		1951	0	2069	0	2832	0	100	0
	3	✓		2673	0	2804	0	3933	0	100	0
Q-learning	1	✕	500	289.8	112.4	1698.4	686.9	868.6	336.5	7.3	7.4
	2	✕		263.1	224.3	1550.6	75.0	788.2	490.7	6.0	4.4
	3	✕		275.8	94.7	1610.4	590.5	826.4	283.5	5.7	5.9

Table 1: Comparison of the models' performance on each map. Comparing: number of actions taken, distance travelled, number of timesteps and percentage progress. "Completed" is true if any of the samples completed the map.

Table 1 shows that both deterministic models complete all three maps whereas neither of the non-deterministic models complete any. Due to the large differences between the two pairs, each model will only be compared against the other in its pair.

From Table 1 it can be seen that the A* model did, unsurprisingly, well. Due to having perfect knowledge of the environment, the agent can calculate the best route, as seen by the fact that it travels the least distance for all maps to complete them. Particularly notable is Map 2: the agent travels 12% less distance than the second-best agent, the Reactive agent. However, the table also shows that the A* agent requires nearly double the number of timesteps compared to the Reactive agent for each map.

The Random agent performs, expectedly, poorly. It has no knowledge of the environment and randomly chooses actions. Despite, these random choices, the agent manages to travel over 100 units for maps 1 and 3 and even more for map 2. It can travel so far without colliding on the first and third maps as the agents are created facing the direction of travel meaning that it can move forwards and rotate small amounts safely. The distance between walls in the second map is much larger, allowing the agent even more space to move safely, reducing the consequence of a poor randomly-chosen action which is likely the cause of the relatively high distance travelled.

The Q-learning agent performs significantly better than the Random agent, travelling much further. It travels over eight times further on maps 1 and 3 and nearly four times longer on map 2. However, the progress doesn't reflect this difference, progressing only 2.6 times further on map 1, 13% further on map 2 and 2.1 times further on map 3. This is due to the agent occasionally doubling back on itself. You can also see that the Q-learning agent requires many more timesteps to progress. This is due to the agent requiring multiple timesteps to rotate to north/east/south/west. This agent also has a much higher standard deviation than the Random agent, showing it is quite inconsistent.

4. Discussion

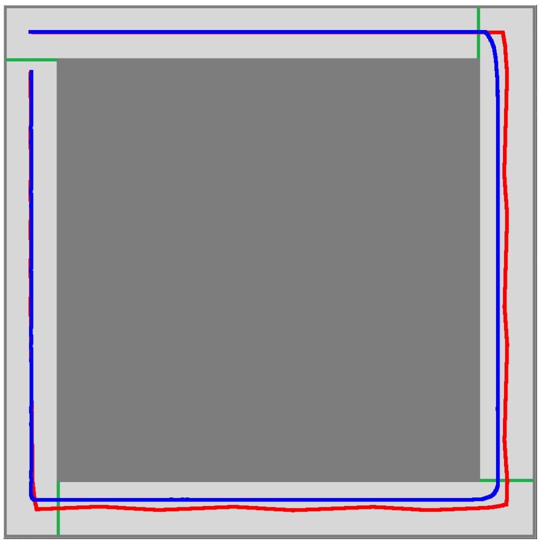
Figure 4: Comparison of the A* agent (blue) and the Reactive agent (red).

The A* agent performs better than the Reactive agent, taking a shorter path as seen in Figure 4. However, since it frequently makes small adjustments in rotation to get to the next location in the path, it requires more timesteps. Two methods for making the agent more timestep efficient are pointing the agent towards a position further along the optimal path (e.g. 5 positions into the future) or splitting the map into larger cells, meaning each point in the path is further apart. The main downside of the former is if you look too far into the future, you may crash into a wall (e.g. when going around a corner). The main downside of the latter is it creates a trade-off between timesteps and distance: a larger cell size means the path found will not be the shortest, but it will reduce the number of timesteps. Alternatively, you could use a variant of A* which is faster, such as iterative deepening A* (Yap, 2002). This may reduce the number of timesteps required for the agent to think, reducing the total time it requires to finish the map.

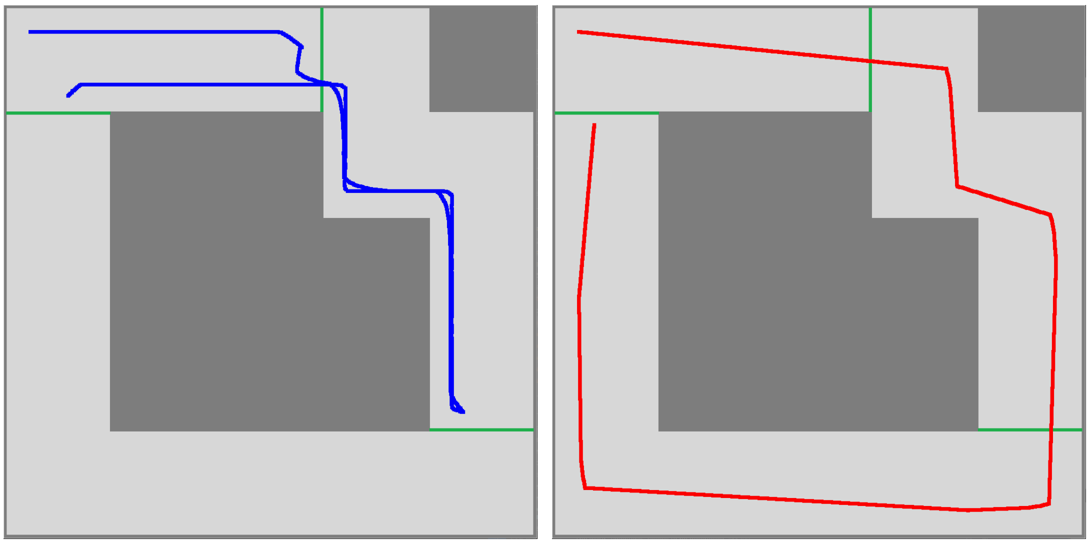
Figure 5: Path of the A* agent (left) and Reactive agent (right) on map 2. Note that although the A* agent's path doesn't touch the checkpoint, due to the width of the agent the checkpoint is collected

The main benefit of the A* agent compared to the other agents is that it has knowledge of the map which allows it to double back on itself, as seen in Figure 5, which results in a 12% lower distance travelled compared to the Reactive agent on map 2. This is unlikely to occur in the Reactive agent since the Reactive agent doesn’t take the location of the checkpoint into account. Figure 5 also shows the Reactive agent's reliance on using the walls to turn. If the walls are far apart, the agent follows an increasingly sub-optimal path.

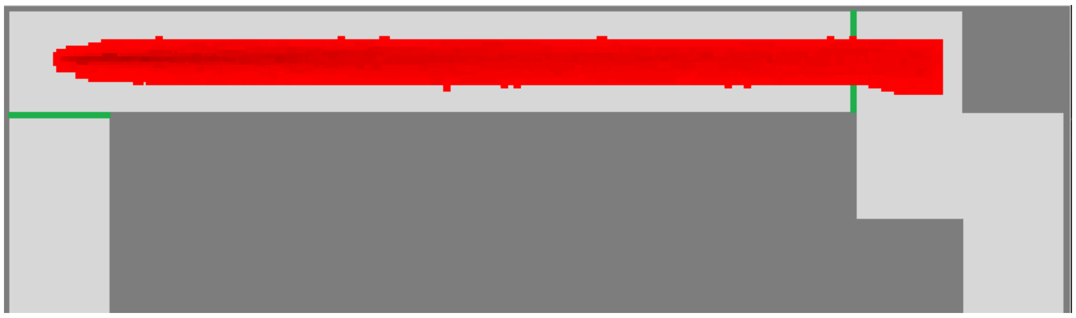
Figure 6: Heatmap of the Q-learning agent positions over 500 samples on the top portion of map 3.

The Q-learning agent performs poorly on the three maps, not achieving 10% progress on any of them. The model learnt to move right, as seen in Figure 6, but struggled to identify that it should move down after getting to the first checkpoint. This can likely be improved by allowing the model more time to train and training it on a more complex map which has a lot of bends, allowing it to learn how to overcome them. Additionally, the model will likely become much more robust by providing the sensor values as a feature for the model and allowing more precise rotation instead of limiting it to 90°. However, Q-learning typically performs better in discrete action spaces, so another method, such as Proximal Policy Optimisation (Schulman *et al.*, 2017) may be better as it can work on a continuous space, allowing for a continuous sensor input.

5. Conclusion

In conclusion, it was shown that the A* pathfinder performs better than the reactive agent when comparing distance but requires more timesteps to complete the map. These agents are then followed by the Q-learning and random agents, neither of which performed very well with neither of them having a mean progress of 10% on 500 samples. Future works could consider alternative agents such as using a variation of A* pathfinding algorithm like iterative deepening A* (Yap, 2002), improving on our current Q-learning agent by training it for longer on a wider variety of maps or creating a different reinforcement learning agent for this environment, such as a Proximal Policy Optimisation agent which may allow for continuous rotation rather than limiting it to 90 degrees rotation, as well as testing and comparing the agents on a wider variety of environments.

References

Dudek, G. and Jenkin, M. (2024) *Computational Principles of Mobile Robotics.* 3rd ed. Cambridge: Cambridge University Press

Foead, D. *et al.* (2021) "A Systematic Literature Review of A* Pathfinding." Procedia Computer Science, 179, pp. 507–514. Available at: https://doi.org/10.1145/3132747.3132780

Pesch, H. and Bulirsch, R. (1994) "The Maximum Principle, Bellman's Equation and Caratheodory's work." Journal of Optimization Theory and Applications, 80(2), pp. 203–229. Available at: https://doi.org/10.1007/BF02192933

Schulman, J. *et al.* (2017) "Proximal Policy Optimisation Algorithms". Available at: https://doi.org/10.48550/arXiv.1707.06347

Sutton, R. S. and Barto, A. (2018) *Reinforcement learning: An Introduction.* 2nd ed. The MIT Press

Wang, W. *et al.* (2019) “Data-driven adaptive dynamic programming for partially observable nonzero-sum games via Q-learning method.” International Journal of Systems Science, 50(7), pp. 1338–1352. Available at: https://doi.org/10.1080/00207721.2019.1599463.

Watkins, C. J. C. H. and Dayan, P. (1992) “Q-learning.” Machine Learning, 8, pp. 279–292. Available at: https://doi.org/10.1007/BF00992698

Yap, P. (2002) “Grid-Based Path-Finding.” Advances in Artificial Intelligence. Available at: https://doi.org/10.1007/3-540-47922-8_4