MDP-based Shopping Assistant Robot Simulation using Policy Iteration

Maria Merkulova, Alexandru Dobrescu, Joshua McKay, Thomas Armstrong, Niam Popat
School of Computer Science, University of Birmingham

***Abstract*—In this paper, a Markov Decision Process implementation for autonomous robot navigation in a grocery shop environment is proposed. The robot has the role of a shopping assistant that is given a list of ingredients to collect and return to the customer. The proposed approach is a policy iteration algorithm which creates a near-optimal local path plan of ingredient boxes that need to be visited. This is joint with an addition of people to the environment to make a more realistic representation of a normal shopping trip where uncertainty is present. An advantage of the proposed approach is that policy iteration is guaranteed to find the optimal action for each state. Experiments for path planning with varying variables have been carried out and the results demonstrate the effectiveness of the proposed approach. With further research, the proposed approach can be implemented into the real-world which would be a strong contribution in today's society.**

***Index Terms*—MDP, Policy Iteration.**

I. Introduction

THIS paper aims to implement an MDP (Markov Decision Process) for path optimisation in a mapped environment through the use of a robot and policy iteration. Policy iteration is an algorithm used to find the optimal policy or action for each state in an MDP. The main goal of said robot is to collect all grocery items requested by a customer while avoiding obstacles and over-crowded areas. If an ingredient is not available, the robot must re-route their path to collect a substitute or to collect ingredients for a new recipe. A further goal is to perform one step of the robot in a set amount of time in order to maintain realistic robot motion in a real world environment: this was decided to be **6 seconds**. This project has significance in real-life scenarios. One article [1] states that nine out of ten people with partial or complete blindness "find information on food packaging difficult or impossible to read". As a result, actions such as relying on a carer to get shopping is required. On account of this, a robot that is able to take requests and carry out shopping will be a cheaper and more efficient solution to such a problem. The solution involves a robot simulation using RViz, a ROS (Robot Operating System) visual interface, and policy iteration to calculate a near-optimal path. The rest of this paper is organised as follows: some related works are introduced in part II. Next, the specification of the project is presented in more detail in part III to solidify background knowledge required to understand the proposed solution. In part IV and V, the implementation of the solution and experiments to establish the validity of the project are described, respectively. The final part concludes by discussing the project's outcomes.

II. Related Work

Various articles [2]–[4] were read to see what other people have done in a similar field and to help decide which methods to use.

A. Localisation and Movement

Schlegel [2] created an occupancy grid map and used the Monte Carlo Localisation implementation available on ROS (Robotic Operating System) to localise the robot using 2D laser scans. Prior to reading related work, it was already agreed to use AMCL(Advanced Monte Carlo Localisation), as this had been used by members in a prior assignment. The author also addresses the issues with localisation in a symmetrical environment, such as a supermarket which the robot will navigate, which causes uncertainty about the true location. Schlegel states that this can be avoided by assuming a known start position of the robot. The plan is to have the robot start and end on adjacent cells every process, which allows us to implement Schlegels idea and define the known start position of the robot to improve certainty on the robots location.

B. Efficient Navigation

The navigation methods mentioned in these papers referenced include RRT (Randomly-exploring Random Tree), A* and nearest neighbour. These are used in many fields of computer science due to their completeness and optimal efficiency, however our goal is to use an MDP (Markov Decision Process) to model decision making.

The idea presented in [3] aims to reduce the computation time of state transitions by only considering free states. Cells occupied by obstacles are not free states, therefore the modified algorithm reduces much of computation required when the map is dense with obstacles. The proposed map contains many walls and shelves which the robot cannot walk through or into, and therefore removing the need to consider these cells as a possible state increases the efficiency of the robot's navigation.

C. MDP models

MDP formulations are generally offered in the tuple (S, A, T, R), which consists of a set of states, S, actions, A, transition probabilities, T, and rewards, R. The implementation of these

rewards are particularly important to dictate how the robot should operate in the environment with the desired policy; the aim of MDP is to maximise the reward. This is explored in [4] where they consider four different reward functions for a similar shopping assistance robot. It is found that, when given a limited number of time steps, the robot always looked to prioritise collecting items with the biggest possible reward and often would return without even collecting all of the required items if it was running out of time. One variation they explored, though, was where the biggest reward (10x higher than anything else) was given when the robot collected all of the required items. This seemingly worked much better than other variations where we saw little to no reward for this, and meant that, even in cases where not all items could be collected in the number of available time steps, more items were collected overall.

III. Specification

In this section, we introduce the project problem as well as the robot simulation used in this paper briefly. Then, we present the MDP approach based on policy iteration in detail.

A. Problem statement

The problem of path optimisation in our scenario can be stated as follows. Having inputted a map of the environment, a start state $[s_{start} = (x, y)]$, a list of ingredients and a terminal state $[s_{term} = (x, y)]$, the algorithm is to find a path from start to terminal state with a range of intermediary goal states.

B. Framework of proposed approach

The path is formed using policy iteration. As stated in the slides of presented lectures, a policy gives an action for each state. This algorithm allows us to find the optimal policy which maximises expected utility. This strategy is used to generate a sub-path between neighbouring cells. **Algorithm 1** shows the pseudo-code for the policy iteration algorithm. The potential of approaching state s' from state s using action a is evaluated by the transition probability. The discounting factor γ depends on how much you want the agent to care about future rewards, zero meaning care for first reward only while one means care for all future rewards. Then, the value function, which gives the perceived value of a state, and the policy, which returns an action fora state, is initialised for all states; this may be done arbitrarily or otherwise. As listed in **Algorithm 1**, there are two levels of the algorithm: in the first level, *Policy Evaluation* is used to calculate the value functions using the current policy. In the second level, *Policy Improvement* checks whether utility is maximised with the policy used in that state.

Each box containing requested ingredients has a reward of +100 which switches to 0 once an ingredient is collected, so that the robot does not move back to the state. When the robot moves into a state with an ingredient box, it has reached its sub-goal. Once all ingredients are collected, the terminal state is granted a reward of 100 to lead the robot back to the customer. The combination of all sub-paths of transitions between neighbouring cells results in the global path optimisation.

Algorithm 1 Policy Iteration Algorithm

Require:
Transition probabilities $f_{ss'}(a) = \mathbb{P}(s'|s, a)$
for $s \in \mathcal{S}tates$ **do**
Initialize (potentially arbitrarily):
Value function
Policy
procedure Policy Evaluation
for $s \in \mathcal{S}tates$ **do**
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V(s')]$

procedure Policy Improvement
$policy - stable \leftarrow true$
for $s \in \mathcal{S}tates$ **do**
$action \leftarrow \pi(s)$
$\pi(s) \leftarrow argmax_a \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V(s')]$
if $action \neq \pi(s)$ **then**
$policy - stable \leftarrow false$
if $policy - stable$ **then return** V
return $PolicyEvaluation$

C. Robot motion

The robot can freely traverse forward and backwards however need to rotate while stationary to move into any other direction; the speeds of the robot can be controlled.

Our initial idea for motion was to make the robot 'jump' from the centre of one cell on the map to an adjacent cell in order to avoid hitting walls and other complications that may come with movement. However, to incorporate a realistic movement that can be applied to a robot, the robot moves from cell to cell at a steady velocity. The robot will turn to the direction of the cell it wishes to move to and move forwards.

To allow for realistic motion, we set ourselves a maximum timeframe between each step of the robot - this includes all the necessary calculations explored below that update the transition function and then to complete policy iteration. This was decided to be 6 seconds when having the people move every step of the robot.

D. Simulation

The simulation was created using ROS in python and RViz to map the robot's motion. RViz works by visualising ROS topics published and the map. The map consists of a YAML and World file and would either need to be created by oneself or found online. The states must be hard-coded to define the coordinates of boxes and people. The boxes have grocery items defined by us, the items defined are meant to reflect ones you would usually find in a grocery shop.

E. Robot's goal

The user trying out the simulation will be prompted to enter their grocery list as well as their acceptable substitutes. The

program will carry out policy iteration to indicate to the robot where to go. However, if an ingredient is unavailable and the substitute is not in the same box, policy iteration must be recalculated to take into account altered rewards. At the same time, the robot must update its knowledge of out of stock items to avoid heading there next time. Furthermore, if no substitute is available, the robot must make contact with the customer remotely. The customer is given the option to either continue without the required ingredient, request new ingredients or end the shopping task. In these latter two scenarios, unwanted items in the robots inventory must be replaced in their original boxes. The same process can be applied if the requested quantity of an item is unavailable.

F. Heat-map

A heat-map should also be created to reflect crowd clusters in a supermarket. These people will influence the probability of moving to a state as the robot would want to avoid these states to move around more efficiently and avoid getting knocked over (if said robot was implemented in the real world). The quantity of people in states must change throughout the day to be more realistic of a real-life scenario. As a result, transition probabilities and policy iteration must be updated.

IV. Implementation

A. Map Creation

In order to create a map, we first looked on the internet for a satisfactory map of a supermarket, however after not finding anything suitable, it was decided to create our own. This was done in such a way to imitate a real supermarket, with aisles that the robot could move up and down and with categorised boxes of groceries arranged along them.

The initial idea was to create this map using grid lines to split it into cells, which would represent the different states, however it was discovered that any black lines within the map would be marked as an obstacle,. Meaning that the robot could not move between cells.

To combat this, it was instead decided to use these black boundary lines to only represent the aisles and boxes, rather than individual cells, so the robot moved around these fixed obstacles - not through them - and so that it could move freely between the unbound cells. It was realised that the boxes should still be accessible to the robot, hence, their boundaries were edited to have one side open as an 'entrance' for the robot to access them.

Further to this, each box across the map should contain a different item variety, ranging from 'Fish' to 'Dairy', so in order to check whether the robot had visited the correct boxes, they needed to be labeled accordingly. In order to do this without creating additional obstructions in the map, labels were added in adjacent obstacle blocks.

B. State identification

The next challenge after creating the map was identifying all of the possible states of the robot.

The cell occupancy was hard-coded with 100 defining an obstacle and 0 defining an available cell. Since our map is sized 9 x 11, we had 99 potential states however, cells that were defined as obstacles were not defined in our list of states so that they would not be considered when doing policy iteration.

The potential actions the robot could take were defined as UP, DOWN, LEFT, RIGHT. Later on, TERMINAL was added to define states that once visited, would end the process. The only terminating state is when the robot returns to the customer.

Since we are implementing an MDP which is fully observable, all the potential actions from each state were defined.

C. Robot Movement

Since the robot needs to move from one state to another, an implementation must be found to shift in one motion.

The approach to this begins with the robot being located at the centre of cell 9, for example. Cells are 3.5 pixels in width and height, with the cells forming a 9x11 grid across the map. The robot can move forwards by 3.5 pixels in one movement which will relocate the robot at the centre of the cell in front of the robot, and can rotate any direction in increments of 90 degrees. The robot is aware of the direction in which it is facing, and is also aware of the direction of the cell that it wishes to move to. The robot rotates so that it is facing the direction of the cell which it wishes to move to and then move forwards to the centre. For example, if the robot is facing UP and wishes to move to the cell on its right, the robot will rotate 90 degrees clockwise and move forwards by 3.5 pixels to the centre of the next cell. This is the best approach for the robot simulation, as it represents realistic movement which would be used by a real robot to travel from one cell to another.

D. Employing policy iteration

Initially, value iteration was attempted with one goal/terminal state as it is a simpler algorithm to implement and the transition probabilities were chosen by us. Once this was achieved, policy iteration was implemented. The value function for each state was initially set to zero and the policy to a random action out of the possible actions that can be taken from that state. The value of each state is recalculated using the current policy and the transition probability. Once the heat-map was implemented, a formula was created that lowers the probability of moving to a state, the more people are in that state.

$$uncertainty = 0.1 + (num\_people\_in\_state/num\_people * 0.85) \quad (1)$$

$$prob(trans) = 1 - uncertainty$$

This formulation for the uncertainty was selected to maintain that the more people that are in a state, the lower the probability of the robot to enter the state. Whilst also maintaining that a transition from the current state to a state with the maximum number of people in will always have a transition probability 0.95

Another aspect of the algorithm is the rewards, rewards of -1 were given to all states to promote optimising reward and completing the task with minimal actions. To define a goal ingredient state, a reward of 100 was given to promote movement towards that state; this switches to a negative reward once the box had been visited. Throughout this, the goal state of heading back to the customer will have a negative reward until all other states have been visited. Once all states are negative, the goal state reward switches to 100 which should lead the robot there after another round of policy iteration.

E. User input

The customer must be able to communicate to the robot which ingredients that they want the robot to gather for them. We implemented this by asking a question at the beginning of the process, in which the user is prompted to enter a list of ingredients that they want. On top of this, once the user has typed the ingredients they want, the robot further prompts them to enter any ingredient replacements they would be happy with for each ingredient if it is out of stock. The user is able to request multiple ingredient replacements for a single item, ordered by priority, in case the replacements themselves are also out of stock, or simply not ask for a replacement ingredient at all. If the robot discovers an ingredient is out of stock and furthermore the replacement ingredients are out of stock (or the customer did not wish for a replacement ingredient), the robot will ask the customer remotely if they would like a different replacement item, to ignore the item entirely or to cancel the order and place collected ingredients in the robots inventory back to their original boxes.

F. Generating locations of people around the map

A representation of supermarket activity was created using a heat-map. This simulated movement and how many people to spawn in order to make a good representation was generated randomly. In the initialisation, the people are spawned randomly in valid states around the map. After each step the robot takes, the people are randomly allocated a valid transition from their location to a neighbouring state. If there are more then 3 people on the desired state, the individual that wants to perform that transition can either stay in the same place or move to a different state; this way we simulate the movement of real people. From the robots perspective, the transition uncertainty increases based on how many people there are on each state, capping at 95% chance of not being able to move.

Each time the robot needs to make a choice of movement, the transition cost map updates and the policy iteration is rerun.

One issue that was difficult to address with the given time constraints was a scenario where the robot would move back and forth between the same states due to crowds blocking its path. Optimally the robot should pass through the crowd and ignore the low probability of entering that state after it had moved back to the previous state once. This is an example of an area of the project that could be optimised with more time.

V. Experimental Results

In this section, the performance of the proposed approach was tested using the robot simulation introduced in **Section III** using the same 9 m x 11 m map. The speeds of the robot were set as 0.5 m/s forwards and $\pm$ 0.2 radians per second to rotate, in all the simulations. Furthermore, there was set to be a maximum number of 3 people per cell to encourage the heatmap to be more spread for more effective testing in that area.

A. Policy Iteration Vs. TSP

The performance of this policy iteration algorithm was firstly compared to a TSP(Traveling Salesperson Problem) algorithm. These are known to find the best possible path between nodes in a graph in order to minimise cost. However, the TSP algorithm does not account for dynamic changes in the heuristics, such as people moving, when traversing through the nodes in a graph. So this comparison could only be completed after removing the heat-map from the proposed solution which uses policy iteration. This meant that the inconsistencies in crowds were avoided and ensured that the most accurate comparison of results could be attained over 20 separate trials. In the initial 10 trials, these ingredients were requested such that the robot had to visit all of the boxes spread across the shop. The ingredients requested were kept constant, once again, to avoid extraneous variables, however, the amount of boxes visited was later changed for 10 further trials to test the difference that this parameter makes. Figure 1 shows the path using the proposed implementation and Figure 2 shows the path created by a TSP solver while the ingredients are kept the same and no people are present.

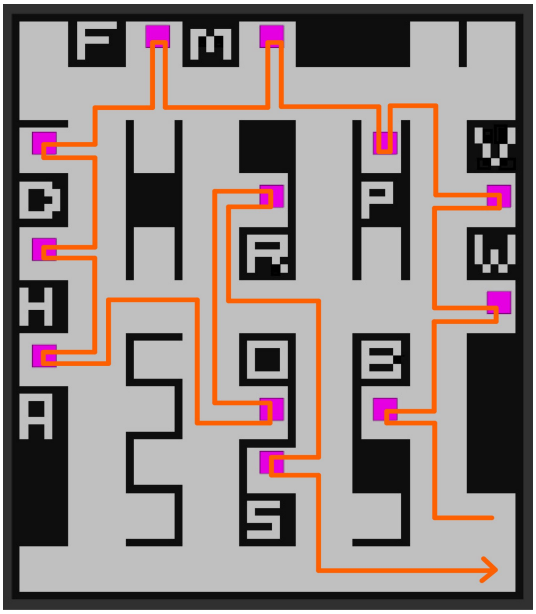

Fig. 1: Proposed approach route to visit all boxes

As can be seen in figure 2, the path generated by TSP is slightly different to the policy iteration algorithm in terms of the order that boxes are visited, however the number of

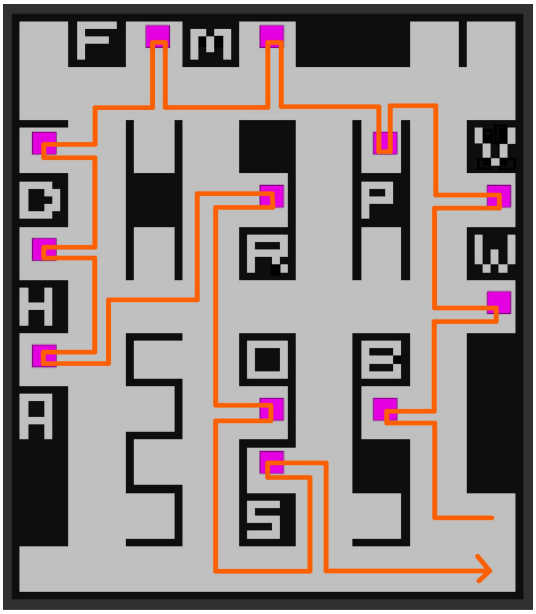

Fig. 2: TSP solver route to visit all boxes

TABLE I: Comparison of results from second trial using each approach

	Test	1	2	3	4	5	6	7	8	9	10
Steps	PI	49	47	43	43	47	47	49	53	51	51
	TSP	43	43	43	43	43	43	43	43	43	43

steps taken in each case is calculated to be the same: 65. As such, the policy iteration algorithm in this can be considered optimal. In the next 10 trials, the performance was once again compared using a fixed number of ingredients to collect: 4, though in this case 30 people had populated the map. This is a difficult variable to test since the location of the people around the map is randomised per run to achieve uncertainty, however, by doing 10 trials, an average of the performance can be calculated. Compared to TSP's 43 steps for the same route with no people, the policy iteration algorithm recorded results of between 43 and 53 steps, as seen in Table I. On average it took 5 additional steps to return to the customer and reach the end goal.

Once again it is shown here that even when computing a very different path, an optimal solution, which matches TSP, is reached by the policy iteration algorithm [see Test 4] to visit all required boxes and return to the customer. The variance in results across these tests also demonstrates the effects of crowds in the heat-map which direct the robot to seek alternative routes to avoid busy areas. This is explored further below.

B. Performance in differing number of people

For the purposes of this test, we have set there to always be a group of three people in state s26 to test if the robot will re-route around that that state. Also during the test, the robots policy will be continuously updated at each step in order to keep track and avoid groups of other moving people who are not shown on the diagrams. When the robot is not moving within a time step and stays in the same state, the diagrams show a small deviation in the displayed route.

The path for these tests are always the same to allow for accurate comparison of journey time: from the customer, to the spice box, to the fish box and then to return back to the customer.

This test is carried out to ensure a robust performance in multiple realistic scenarios comparing differing numbers of customers in the shop and to verify that we have achieved one of our project goals - explained below. The test ensures that the robot is not only avoiding the constant barriers, but also so that the robot is altering it's route to avoid potential shoppers in the way.

The metric of these tests are as follows: As a metric for efficiency in computation, a goal of 6 seconds for each movement was set by the team. The movement consists of regenerating the heat-map, using the new positions of the people to then recalculate the transition function that robot must follow. After this policy iteration can then executed once more and a new policy for the robot to follow is produced.

To measure that the robot is taking differing routes due to avoiding moving customers, we recorded the number of steps taken by the robot on each test. With this information we then will check that the robots movement count is always differing from the optimal route count of 40 steps set by the travelling salesman formulation. The metric for success in this test is defined as follows: the average number of movements cannot be less than 110% of the optimal path. With this in place, the test ensures that the robot must be either taking evasive action or waiting for group of people to pass. The optimal path is more unlikely with people to avoid on the map.

To measure the fact that the robot does enter state 26, the paths of each robot journey is recorded and checked to ensure that s26 is not present within it.

Below is an example of the data recorded from the test. Each list of states in the journey is checked for **s26**. If the state is present then the test is failed. To save space on this report, the full list of recorded routes is not shown however all tests passed this condition.

tests[with 40 people][test 8] : 335.71465301513675 with route['s15', 's14', 's7', 's6', 's5', 's5', 's12', 's20', 's20', 's19', 's20', 's12', 's5', 's5', 's4', 's3', 's2', 's1', 's1', 's9', 's16', 's22', 's30', 's35', 's44', 's50', 's57', 's57', 's64', 's65', 's73', 's73', 's65', 's66', 's66', 's66', 's59', 's51', 's46', 's37', 's38', 's38', 's39', 's39', 's39', 's38', 's39', 's40', 's41', 's34', 's28', 's28', 's21', 's14', 's15'] of length 55

Clearly shown in table 2 with supporting data from table 3, the average time per movement for each range of people is greatly impacted by outliers. In table 2 it can be seen that one task took over 3500 seconds to complete, which was not anticipated in any manner. The reasoning behind this occurrence is as follows- the computer taking the test is very old, and had been running for many hours straight before this.

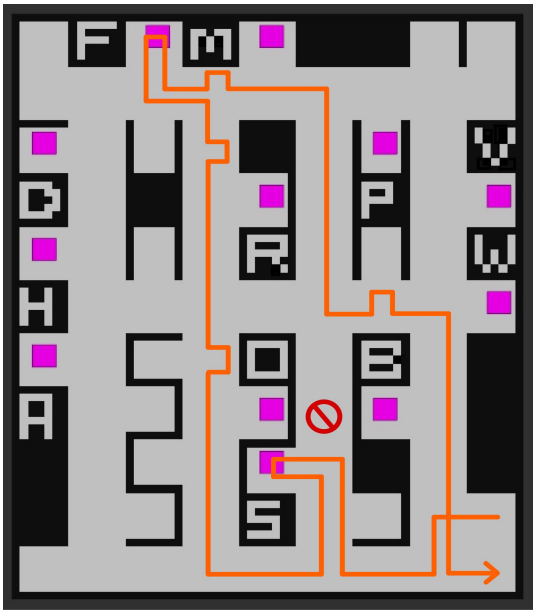

Fig. 3: The route taken by the robot moving from the customer, to the spice box, to the fish box and then back to the customer when there are 20 people in the shop. This avoids the group of people in s26, denoted by the STOP symbol.

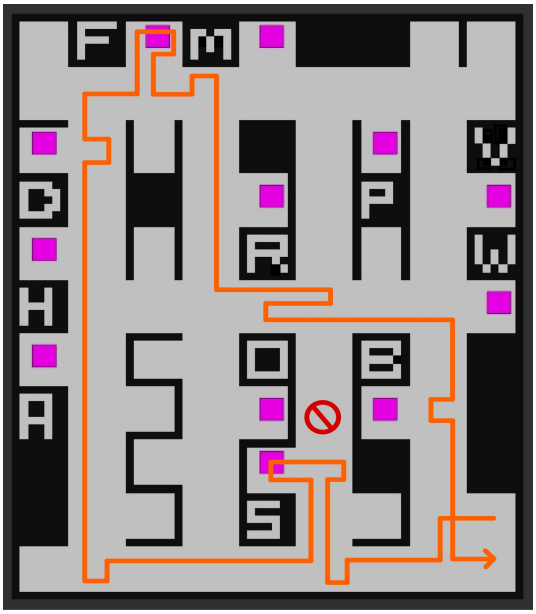

Fig. 4: Similarly, the route taken by the robot for the same set of goal ingredients but with 40 people in the shop.

TABLE II: Results after averaging findings

	Average		
People	Time per route (s)	Movements	Time per movement (s)
10	278.8266399	43.66666667	6.377375556
20	500.9394737	47.13333333	10.62813594
30	350.1494646	48	7.294780513
40	340.1475175	54.5	6.241238854
50	394.2792463	60.7	6.495539478
60	474.6381562	63	7.547249833

It froze midway through execution of the program and had to be shut down. Upon restart and the opening of the record file the value in question was stored, we believe that the freezing process had damaged the computer's ability to measure time accurately. The recording without the anomaly is graphed in figure 7.

In terms of our second metric, it is clear to see that for each amount of customers in the shop tested, the average number of moves are all above the goal of 44 steps. As the number of people in the shop increases, so does the number of average steps. This is crucial evidence to prove that the robot must be altering it's route in order to avoid the numerous groups of people around the map. It is not only that the route is differing, but that with more people, more evasive action must be taken. This is demonstrated well in Figure 4, for example. Figure 3 more closely follows the optimal path shown in Figure 6, however, as well as avoiding the constant group of people in s26 like figure 3, figure 4 also appears to avoid another randomly moving group of people that look to have blocked the second aisle. For reference, dynamically moving people are not displayed on the diagrams.

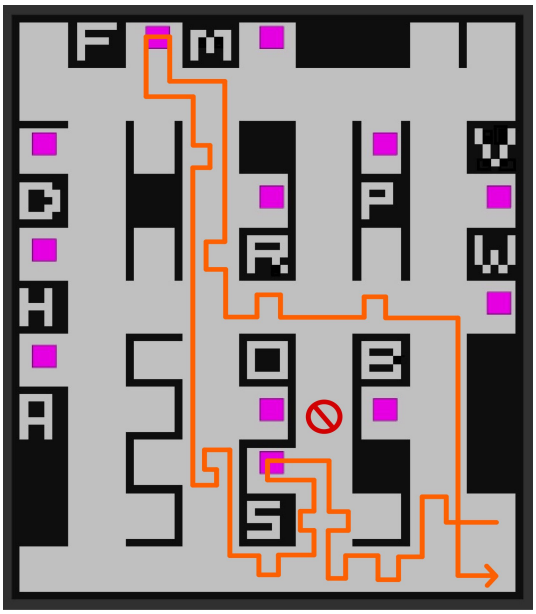

Fig. 5: Finally, the route taken by the robot when 60 people are in the shop.

It is interesting to note that the average time taken for movement in the trial with 30 people is higher that the results for both the 20 and the 40 people scenarios, countering what was initially hypothesised: that the more people in the shop, the longer it would take to calculate the best move. In this case, it would require additional testing to discover the reasoning behind this result as none of the readings were clear outliers, and the current hypothesis only comes down to the randomly chosen locations of people at the time. If these people are

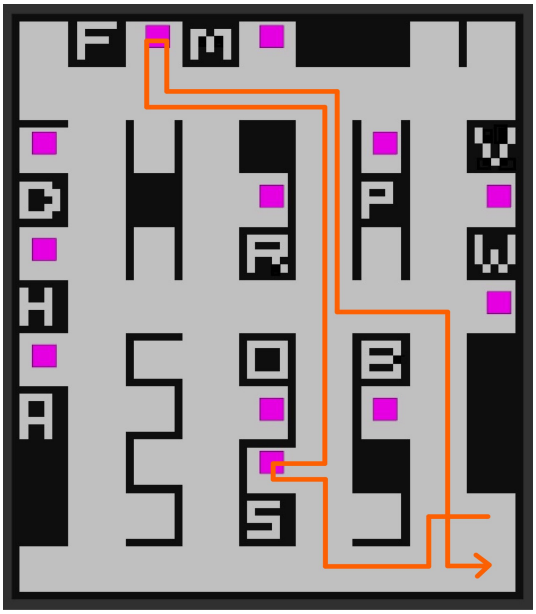

Fig. 6: For comparison, the route taken by the robot when the shop is empty of other people and no re-routing is necessary

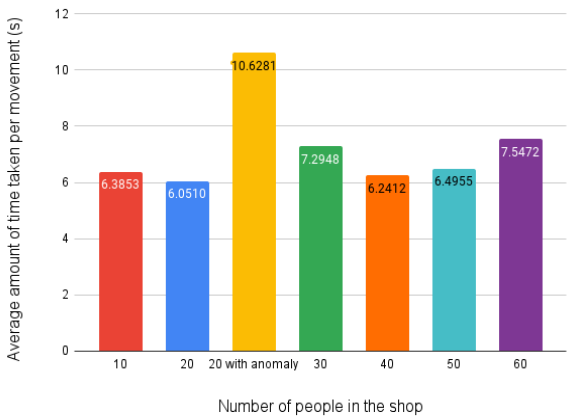

Fig. 7: The effect of number of people on time taken per movement

TABLE III: Sample of results from 20 people in the shop

Test	Time taken to complete route (s)	Steps taken to complete route
1	294.3002007	53
2	351.2489741	50
3	366.0463357	42
4	332.0820036	42
5	199.7979548	48
6	218.4418218	51
7	295.9041977	49
8	291.7160597	51
9	299.1305408	49
10	263.2649257	44
11	270.4192219	44
12	253.8053584	40
13	268.3214533	47
14	295.2352353	51
15	3514.377823	46

evenly spread out, with the same concentration of people in each state, then this could increase the time taken for policy iteration to be carried out by a small amount. To further investigate this occurrence, we suggest a more in-depth testing strategy in future, one of which could be running the same tests on multiple different computers, and another could be to increase the number of tests per category.

Even if these anomalies are not taken into account, seen graphed in Figure 7, despite being close, the movement times are all outside of the self-set goal for this project. Therefore, the robot failed these tests. However, as this shop is understood to be small in a real world scenario, it would be unlikely for it to contain any more than 40 customers at once. A slight trend observed from graphing the data shows that an increased number of people in the shop causes the robot to take longer per move. With this considered, despite failing the efficiency test, it may be considered that this solution is acceptable regarding a real life scenario applicable to our robot.

VI. Discussion

Tasks were assigned through the *plan-of-action* channel in the group's discord server. This worked well since there were daily updates and contact between members however in hindsight, a more agile framework such as SCRUM would have been favourable. Paired programming was put in place when members were having difficulty completing an assigned task. This meant deadlines were met while members having equal contribution to the project.

As mentioned, there were unexpected results regarding the impact of amount of people on the transition functions. When 30 people occupied the shop the robot moved slower than compared to 40 people over an average of 15 tests. In the future it would be beneficial to increase the number of tests in our experimentation to see if this was an outlier due to chance or a problem which needs improvement.

Being granted access to a real supermarket while operating a robot would be difficult and creating an appropriate experimental environment can be quite challenging since there are many extraneous variables that cannot be controlled in a real-life scenario. Nevertheless, recreating this project with a real robot would be a favourable future capability. It would allow the incorporation of other facilities, such as communication through speech or reading aloud ingredient information for customers with the incapability to read.

VII. Conclusion

This paper proposed a Markov Decision Process approach to a shopping assistant robot simulation. In the proposed approach, the policy iteration algorithm and a crowd implementation was combined to create a realistic simulation of a grocery shop. The experimental results have demonstrated the effectiveness of the proposed approach. Project management

was discussed showing the collaboration of the team members throughout the project. A real world implementation of such a project would benefit to reinforce the quality of the proposed approach, but the current program is a promising basis for advancements in a future research project.

REFERENCES

[1] Tanner C. 'It's impossible for blind people to go supermarket shopping – that needs to change'. 2021.

[2] Schlegel K, Neubert P, Protzel P. Building a Navigation System for a Shopping Assistant Robot from Off-the-Shelf Components. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 12228 LNAI. Springer Science and Business Media Deutschland GmbH; 2020. p. 103-15.

[3] Achour N, Braikia K. An MDP-based approach oriented optimal policy for path planning. 2010:205-10.

[4] Gillani R, Nasir A. Incorporating artificial intelligence in shopping assistance robot using Markov Decision Process. In: 2016 International Conference on Intelligent Systems Engineering (ICISE); 2016. p. 94-9.