Zain Rashid – 20172929 – PSYZR2

Designing Intelligent Agents Report

Introduction

Automated household vacuum cleaners have seen a large rise in popularity in recent years due to advancements in technology and intelligent agents. However, there is still a large debate on what methods are best to maximise area coverage and efficiency, while reducing collisions. This is partly due to several factors needing to be considered such as position localisation and the environment mapping. So, to start to breakdown this optimisation problem these bots must take into account a multitude of different considerations and thus must incorporate intelligent systems. This means sensors, mapping algorithms and collision detection methods are essential to producing a cleaning bot which performs well. Where it is suggested that the approaches utilised should incorporate some form of learning. This is preferred due to the bots needing to adapt to a variety of constantly changing variables and thus some form of learning is important to maximise efficiency and predict possible issues like collisions.

Project Inspirations and Aim

In this report I aim to compare and evaluate multiple movement algorithms to see which perform best against performance metrics such as dirt collected, percent map coverage [1] and amount of collisions. This was inspired by a paper which aimed to assess the effectiveness of different coverage algorithms [2] where I also utilised the spiralling algorithm referenced. This was a good algorithm to implement as it was a great comparison method to my random wandering approach.
Additionally, a grid made up of square cells was important and was formed through "approximate cellular decomposition"[3]. The grid wasn't limited to purely square cells and could be made up of triangle cells or even hexagonal cells. Where it was reported that with triangular cells, paths are shorter due to being more flexible [4]. However, it also had been stated that the differences were insignificant and could be replicated through increased resolution of the grid [5]. This grid allowed for algorithms which can plan their next move, such as only moving to unvisited cells or cells with greater amounts of dirt.
Furthermore, I intend to experiment with different collision algorithms to evaluate the best method to minimise collisions. I utilised a Machine Learning statistical model, Logistic Regression, as it was a useful model for "dependent binary variables", where in this case "our binary output variable would take on the value 0 if no collision had occurred and 1 if a collision had occurred" [6]. These questions will be tested alongside a varying number of bots to see how performance is affected.

Questions

- **How do different movement algorithms compare in terms of maximising dirt collection, map coverage, and reducing collisions?**
- **How do different collision algorithms compare in terms of predicting collisions?**
- **How does the performance of the system change as I vary the number of agents in it?**

These questions will be explored through breaking down the environment and intelligent agents being used in the simulation. Running multiple experiments in the constructed simulation to assess bot performance with the varying factors. The results will then be visualised and analysed where I will conclude the findings and aim to answer the referenced questions.

Environment

The environment is an enclosed room split up into a 20x20 grid where bots can collide with the edges of the room. The grid is populated with dirt within cells and when a bot has passed over a cell it's marked as 'visited', this grid is utilised by two of the movement algorithms for the agent. A charger and bin are utilised to allow the bot to clear it's collected dirt once full and charge when it's battery is low.

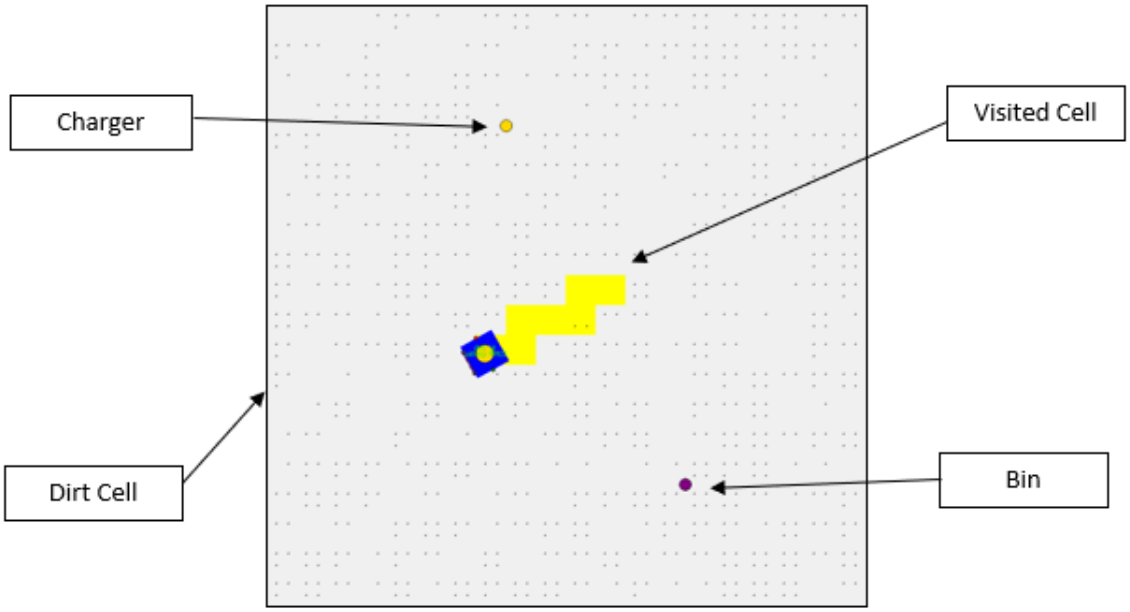

Figure 1: Environment view

Intelligent Agent

The bot has sensors to help detect obstacles and objects. The sensors are used to move towards the charger and bin. The sensor values are also used by the collision detection algorithms to predict and classify whether a bot should change its course because it's about to collide with another object or wall.

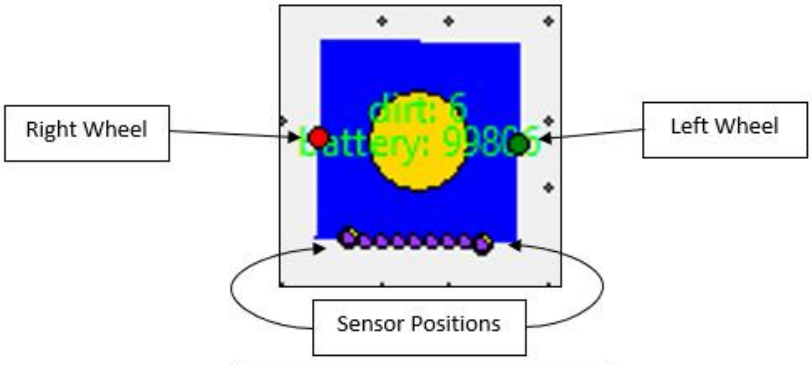

Figure 2: Intelligent Agent model

Movement Algorithms

Random
Acts as the control in comparing the different algorithms. It moves in a straight line for a randomly generated distance, then turns at a random angle, continuing to move forward. This pattern is repeated.

Spiral
Moves the bot in an outward circular motion, increasing in size. When a bot collides with another object or wall it will move in a straight line to a new location. From here it will again start to spiral.

Cell Search
Utilises the grid to plan the bot's next move. It searches the grid for the nearest cell to the bot, then it utilises the sensors on the bot to move towards the designated cell. The cell is marked as visited once passed and a new target cell will be found till the whole grid is visited.

Cell Search Dirt
Based off Cell Search, however, it selects the target cell differently. Instead of prioritising the nearest cell it checks the grid for positions with the most amount of dirt. Prioritising cells with four pieces of dirt, then three, two, one and finally the empty cells. It then finds the nearest cell with the max amount of dirt and moves towards that.

Attract
A teamwork-based algorithm becoming more efficient and preferable with multiple bots. Functions by having nearby bots follow the same path and angle , so they move in a group. This is done by searching for bots nearby within a threshold then finding the average angle of the nearby bots. The calculated average angle will then become the bot's new orientation.

Repel
Another teamwork-based algorithm which works in the opposite manner to the Attract algorithm. It has bots stay a fixed distance away from each other, forcing them to different parts of the grid. This works by finding the closest bot to the bot's location, then setting the current bot's angle to one which would make it avoid the closest bot.

Collision Algorithms

When a bot encounters a wall or another object it should react to it and avoid a collision, as otherwise a realistic simulation isn't being created. Three algorithms will be compared which all vary in how they detect whether a collision is about to occur. Additionally, when a collision is imminent, an object avoidance algorithm is run. It works by having the bot turn till it is at an angle to move away and avoid a collision. This algorithm is combined with every movement algorithm.

Distance-based
Used as a control as it doesn't utilise the bot's sensors to judge whether a collision will occur. It instead calculates the distance from the centre of the current bot to every other bot on the map. If a distance calculated is below a set threshold and the other bot is in front of the current bot, then a collision warning will be given.

Danger Threshold

Utilises the bot's sensors and a training set to generate a mean threshold value for when collisions are most likely to occur. First, the bot is trained to form a training set made up of sensor values and whether or not a collision occurred. The sensor values where a collision occurred are then extracted from the set and the max value of each sensor grouping is determined. Finally, the mean of these max sensor values is calculated. Now when a detecting a collision, the max sensor value is calculated then compared to this threshold, if the sensor value is above the threshold, a collision is said to be imminent.

Logistic Regression

The final approach utilises Logistic Regression, a Machine Learning method. This is the go-to for binary classification problems, in this case being whether a collision is about to occur or not based on sensor readings. To train this model I use the same training set created when generating the Danger Threshold using the max sensor values. The training set is divided into a train/test split where the test split allows me to evaluate the model's classification performance and the train split trains the model. Instead of using only the danger values as with the threshold method, I will use both safe and danger values. This is because to train a Logistic Regression model you must pass pre-classified data so the model can then classify new data it's given based on the dataset originally passed. Once the model is trained it is saved and now when detecting a collision, the sensor value is passed to the model and either "safe" or "danger" will be outputted to state whether a collision is about to occur or not.

Challenges

With the Logistic Regression method instead of using the nine sensor values to train the model, I used only the maximum value of the nine as I did with the Danger Threshold. I unfortunately had to do this since when training the model with the sensor array, it would only detect collisions when all nine sensor values had a value above its generated threshold. So for instance, [0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.0] would be incorrectly flagged as "safe" while [0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9] would be correctly matched as "danger" even though both have values which indicate an imminent collision.

Another challenge faced would be how the sensor detected objects, as it was detecting objects in front and behind the current bot. This would lead to scenarios where a bot would detect a collision although it's path ahead was clear, as it had detected an obstacle behind it. To fix this problem, I first calculated the distance from the current bot to the suspected object, I then calculated the distance from the front of the current bot to the suspected object. If the distance from the front to the other bot was shorter than the originally calculated distance that means the other bot was in fact in front of the bot and not behind. Solving the issue.

Experiments

To fully evaluate my algorithms I had each of my runs last for 1000 moves, this allowed enough movement to show patterns and correlation between the different algorithms. For each run the positions of the bots on the grid was randomised to help prevent position bias in certain algorithms.

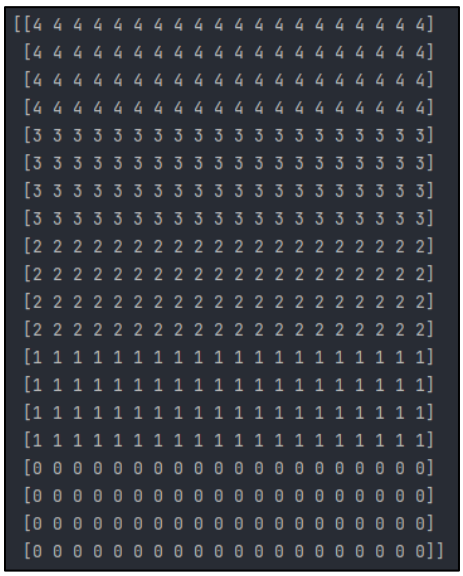

Figure 3a: Unshuffled Grid – 800 dirt

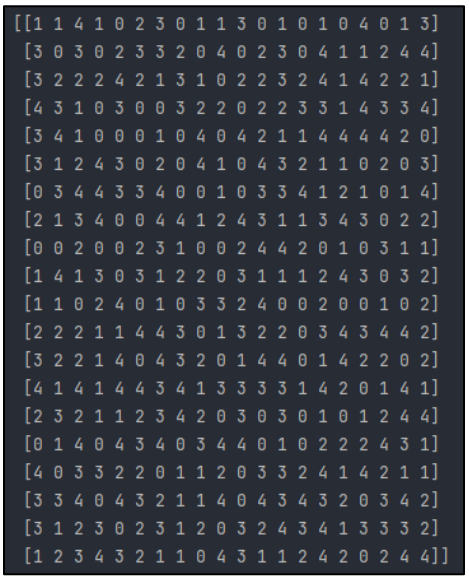

Figure 3b: Randomly Shuffled Grid - 800 dirt

Next, dirt was smartly allocated within the grid where there was always 800 dirt (3a), which was randomly distributed to every cell (3b). A cell could have zero to four pieces of dirt. This allowed for variance when doing multiple experiment runs. Within each cell I set dirt configurations for when there is one (4a), two (4b), three (4c), or four (4d) dirt in a cell. Again this allowed for less randomness in the experiments allowing me to more closely control what is being adjusted per experiment.

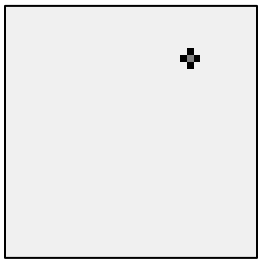

Figure 4a: cell with 1 dirt

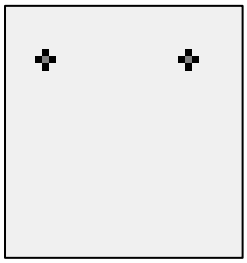

Figure 4b: cell with 2 dirt

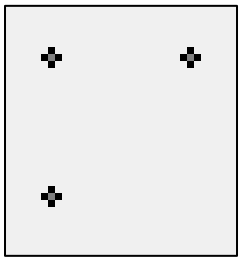

Figure 4c: cell with 3 dirt

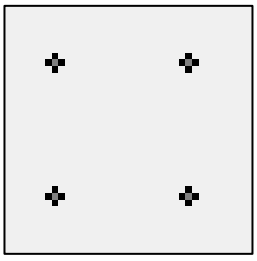

Figure 4d: cell with 4 dirt

Each algorithm ran with a varying amount of bots (1-10) where each bot amount ran 10 times to reduce outliers and calculate means. After the 10 runs per bot amount, key performance metrics such as the average collisions, map coverage and dirt collected were evaluated. This process was repeated for each movement algorithm and then line graphs were plotted with the received data, comparing the number of bots to the averaged metrics for each movement algorithm.

I repeated this process for each collision algorithm too, running a different collision algorithm with every movement algorithm. This allowed me to see how different movement algorithms were impacted when used in combination with varying collision algorithms. The Danger Threshold and Logistic Regression methods were trained with a dataset of 20000 values to reduce the possibility of data inaccuracy and impact of outliers from the training runs.

Once all collision algorithms had run, I produced a line graph of the average collisions over all the movement algorithms vs the number of bots. This showed a clearer correlation between the performance of the collision algorithms. Three line graphs were also generated to compare the bots effect on the system, using the average over all algorithms (movement + collision) for each metric vs the number of bots. A box plot was also plotted to decipher the variance in data between the different collision algorithms. Finally, a Two Sample T-Test was performed to conclude whether there was a significance difference between the results given for the collision algorithms.

Results + Analysis

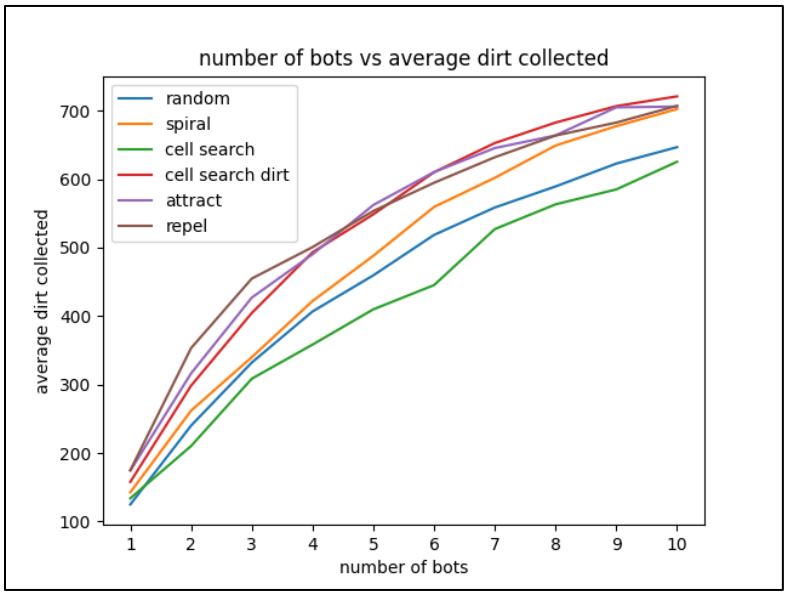

Figure 5a: Dirt Collection Results – Mean over all three collision algorithms

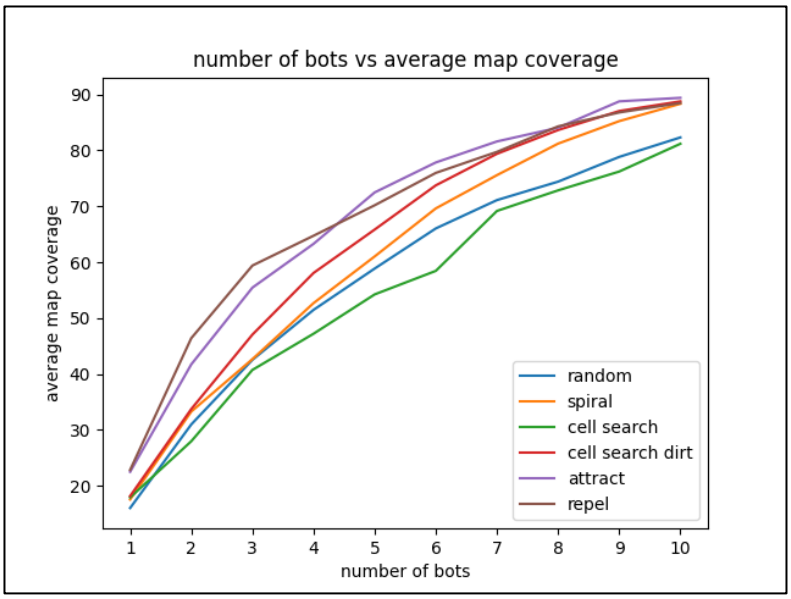

Figure 5b: Map Coverage Results – Mean over all three collision algorithms

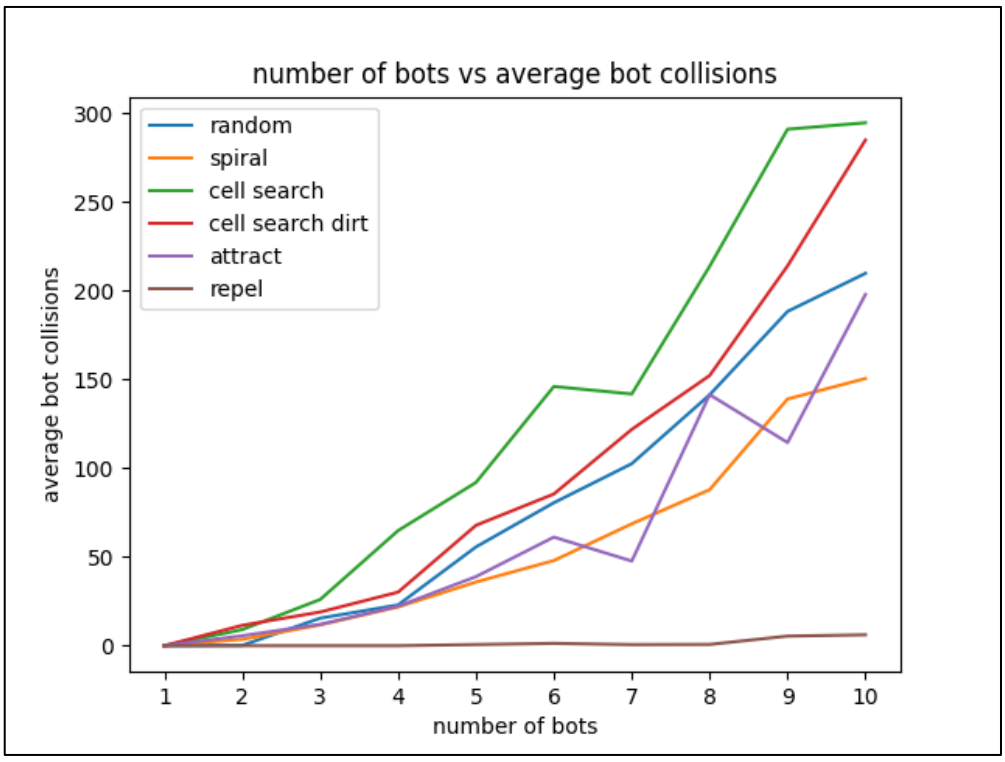

Figure 5c: Bot Collision Results – Mean over all three collision algorithms

Zain Rashid – 20172929 – PSYZR2

How do different movement algorithms compare in terms of maximising dirt collection, map coverage, and reducing collisions?

The 'Cell Search Dirt' algorithm performed best at collecting dirt (5a). I believe the reason why it performed so well is because it is a planning algorithm, so it utilises the dirt grid to move specifically towards cells with greater amounts of dirt and avoid cells already visited. This means this algorithm will have less cell overlap than other none-planning algorithms and thus greater map coverage (5b). In addition, it will avoid cells with no or small amounts of dirt, leading to an increase in overall dirt collection. On the other hand, this algorithm suffers from a higher amount of detected collisions with other bots as can be seen from 5c; performing second worst over all bot amounts. This behaviour can be chalked down to bots being attracted to the same max dirt cell on the map if they are nearby each other. This can cause collision issues as the bots are now more likely to be moving towards the same location and thus will interact more often.

The 'Attract' algorithm performed best at map coverage, I deduce that this is because of its greater coverage where nearby bots move in parallel; thus meaning, they are less likely to cross paths. Due to this parallel movement it also means more dirt is collected, however, since the bots stay a certain distance apart, due to collision detection, dirt may be missed in the gaps between the moving group of bots. This could possibly explain the smaller amount of dirt being collected compared to the 'Cell Search Dirt' algorithm. The algorithm sees an average amount of collisions compared to the other methods, which I feel is because since the bots move in groups and at the same angle so there is less chance of collisions when moving across the map. However, at edges and corners problems could arise due to the grouped bots needing to reposition together to avoid a collision, thus the chance of collisions increasing.

The 'Repel' algorithm performs very closely to the 'Attract' algorithm with excellent coverage and dirt collection which I believe is to do so with the bots communicating and sticking to separate sections of the map. This forced bot separation is apparent at lower bot amounts where it performs best out of the rest of the algorithms in both dirt collection and map coverage. However, as the number of bots increase the algorithm drops in performance slightly. This is most likely due to the bots being pushed to the edges of the map as the number of bots increases as they try to avoid each other. To improve upon this issue I could implement variable thresholds for when the number of bots increase. Finally, as this algorithm keeps bots at a set distance away from each other it causes the least amount of collisions, having a near zero average.

The 'Spiral' algorithm performed slightly worse than the top three teamwork algorithms, however, still gave strong results for all metrics; it is also the best performing non-teamwork algorithm(5a/5b). I believe it performs better than the 'Random' algorithm at all metrics since firstly, the spiralling method can cover a large distance around it's starting point where it's less likely to pass cells it has already visited, due to its outward moving pattern. Furthermore, when a collision is detected, the algorithm will move the bot a set distance away from a wall or object to then resume spiralling, as to give the bot enough space to spiral. This also means it is more likely to start spiralling again in a new unvisited area, leading to better dirt collection and coverage. This movement away from areas where a collision has occurred in turn also reduces the total number of collisions, where the spiralling algorithm has the second lowest rate of collisions (5c).

The control 'Random' algorithm with random turning and movement performed poorly compared to the majority of algorithms. This was to be expected as the algorithm doesn't employ any strategies to traverse the map to improve the performance of a judged metric. It encounters a lot of collisions only being behind the two planning algorithms. I believe this to be so as this algorithm makes more frequent turns compared to other algorithms. This means the chances of a bot intersecting with the path of another bot is greater than if it was moving only in a straight line.

The worst performing algorithm in all metrics was surprisingly the standard 'Cell Search' algorithm. This greatly juxtaposes the performance of the related pathing algorithm, Cell Search Dirt. Although I suspected this algorithm performing well, I am able to understand what causes it to faulter. First off, it suffers from the same problems as the 'Cell Search Dirt' where it encounters a lot of bot collisions. As stated, this was because if multiple bots are in close proximity they will be drawn to similar or the same cells. This can cause a knock-on effect where they end up following the same path as one another, thus, getting stuck while trying to avoid a collision and move to an unvisited cell. This algorithm performs much worse than the 'Cell Search Dirt' too because it's movement is more restricted as it moves only to nearby cells instead of prioritising max dirt cells. This means it isn't as efficient as maximising dirt collection, as it mainly covers the area around where the bot starts instead of exploring further out as with the other algorithms. Furthermore, since this algorithm uses the sensors to adjust movement, the bot moves at a slower rate as it is making small turns and adjustments to align its path towards the designated cell. This again will decrease map coverage although it's trying to explore new cells.

Overall, I believe the best algorithm is between the teamwork-based algorithms: 'Cell Search Dirt', 'Attract' and 'Repel'. In a real-world environment the 'Cell Search Dirt' algorithm would be harder to implement as it requires a predefined dirt map of the area to be cleaned to work. However, the 'Attract' and 'Repel' algorithms aren't bounded by such restrictions, this makes them more reliable and efficient in my opinion. They also perform better at smaller bot amounts which is more beneficial and practical.

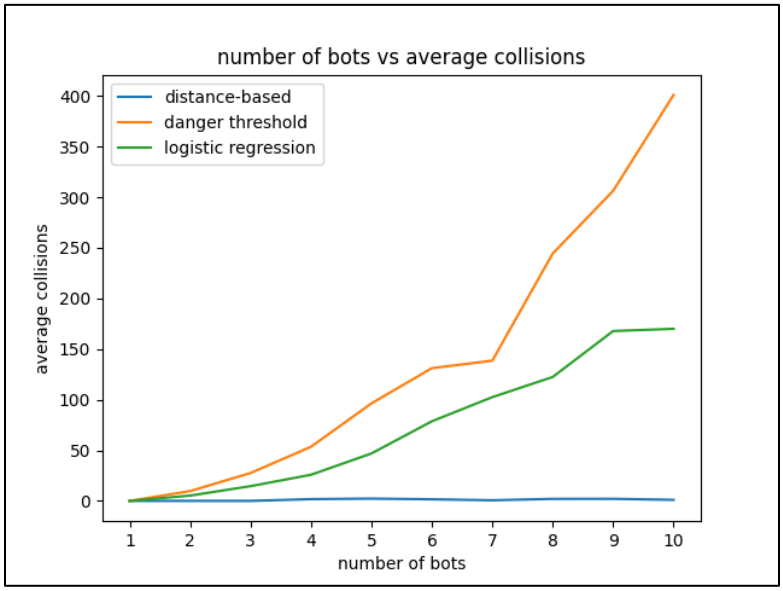

Figure 6a: Bot Collision Results – Mean over all movement algorithms

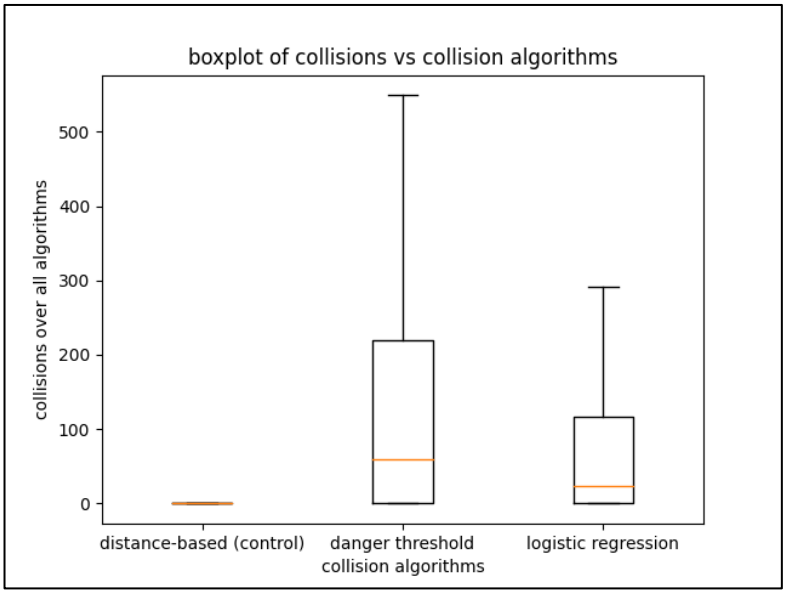

Figure 6b: Bot Collision Results – Boxplot

```
Accuracy score:  0.9412
Safe F1 Score:  0.940975707689219
Danger F1 Score:  0.9414225941422594
Safe Precision score:  0.9445787988714228
Danger Precision score:  0.9378721714966256
Safe Recall score:  0.9374
Danger Recall score:  0.945
Confusion Matrix:
 [[4725  275]
 [ 313 4687]]
```

Figure 6c: Logistic Regression – Model Evaluation

Two Sample T-Test (Danger Threshold vs Logistic Regression)

- alpha threshold: 0.01
- p-Value: 3.547185756039875e-14
- t-statistic: 7.669911262206594

Figure 6d: Two Sample T-Test – Mean Significance Test

How do different collision algorithms compare in terms of predicting collisions?

Firstly, the Distance-based algorithm, as stated previously, was to be used as the control since it didn't utilise the camera sensors on the robot meaning it was more likely to accurately detect whether a collision was about to occur. Due to this it can be seen (6a, 6b) this method very rarely encountered a collision.

It can also be deduced that the Danger Threshold algorithm performs the worst. In 6a it has a much steeper gradient than the Logistic Regression plot inferring that it's performance almost gets exponentially worse as the number of bots is increased. The boxplot also reveals that Danger Threshold has a worse median, range and interquartile range to Logistic Regression. This means there is a larger spread and dispersion of data leading to less consistent results for this method. I believe the reasoning behind this poor performance is because of the way the sensors work on the bot. Since they can only detect obstacles in its peripheral vision, this means if two bots are moving parallel at slightly converging angles, a collision can occur without it being detected. So since this method doesn't have variable threshold ranges due to using only a training set to generate a singular value, the method can't adapt to these scenarios in real-time. This is why I also believe the Logistic Regression approach is superior as it is able to dynamically alter its threshold based on the data parsed through training. Which I believe is why Logistic Regression had a statistically significant T-Test (6d) which proved its effectiveness.

However, when evaluating the Logistic Regression's classification of safe and unsafe scenarios, the re still were collisions. This can be seen from 6a where there are still collisions occurring for each bot amount, this is especially evident when looking at 6c. Where I can deduce that although the model has strong Accuracy, Precision, F1 and Recall scores it is still prone to misclassifications for when a collision may occur, where there were about 600 misclassifications on the test data, seen in the Confusion Matrix. I suspect this is in part due to the lack of cameras around the whole bot as discussed, but also, I believe it is a limitation of the Logistic Regression model itself. As this model can't be tuned very precisely, only being able to alter the amount of training data passed and the random state of the model. Furthermore, the data being classified is very hard to distinguish between due to the difference between a collision occurring or not, can be decided by a sensor value change of as little as 0.01.

In summary, based on these results I can confidently say that Logistic Regression is the prefe rence of collision algorithm as it greatly reduces the amount of collisions in comparison to Danger Threshold due to its adaptiveness. Where it was proved through the Two Sample T-Test (6d) performed on the Logistic Regression and Danger Threshold data. The test returned a p-value which was far below the alpha threshold needed to reject the NULL hypothesis of the test (that the two algorithms don't have conclusive and distinct results).

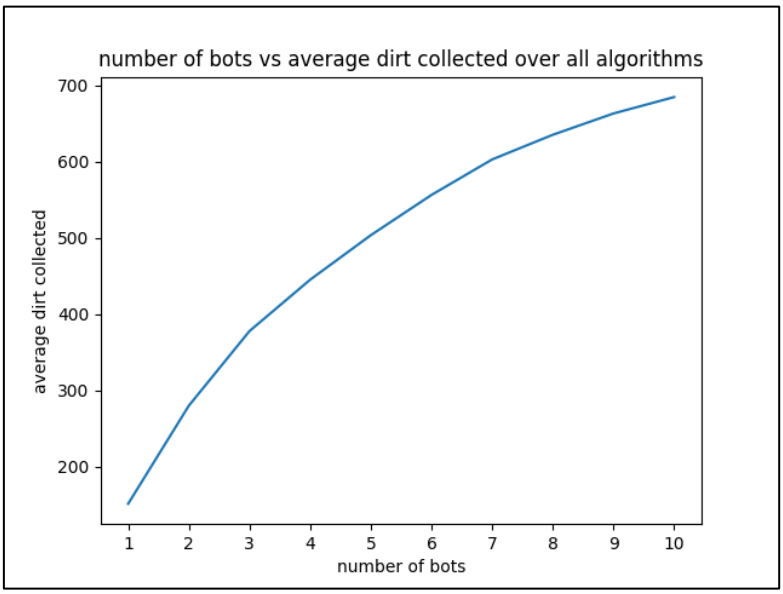

Figure 7a: Bots vs Mean Dirt Collected Over All Algorithms (Mapping + Collision)

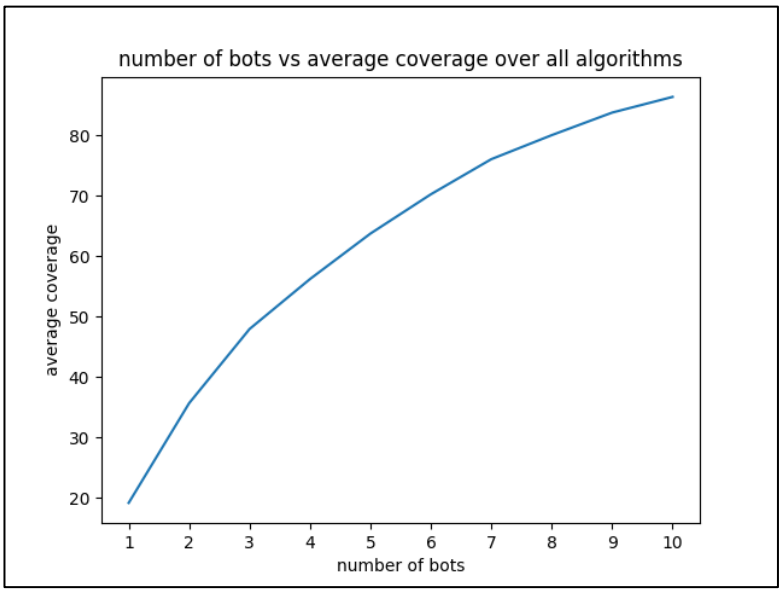

Figure 7b: Bots vs Mean Coverage Over All Algorithms (Mapping + Collision)

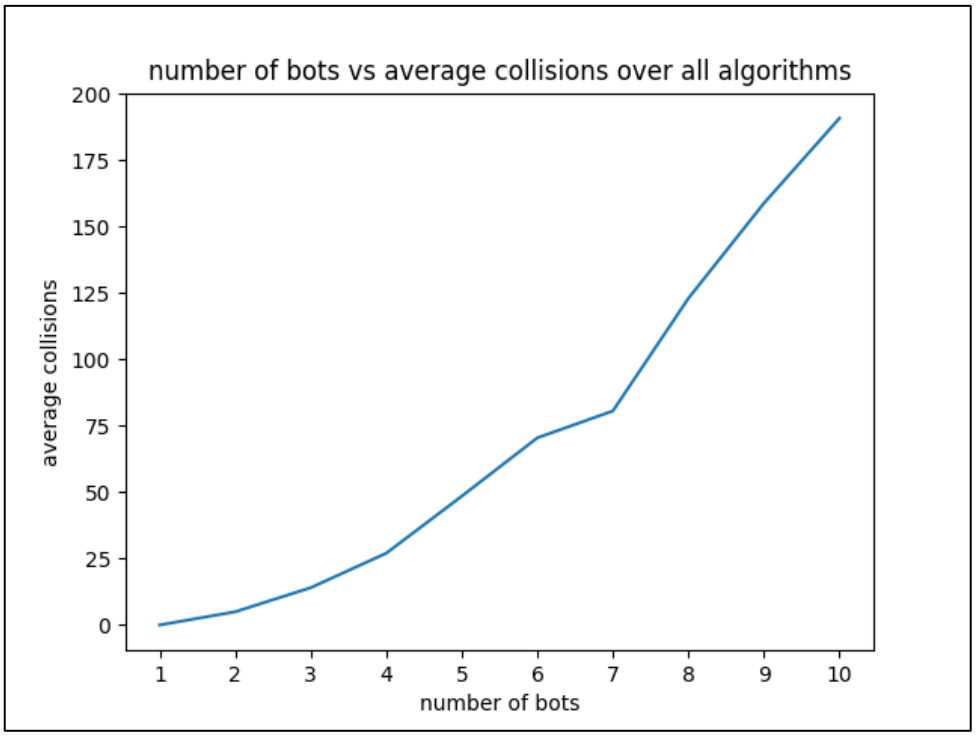

Figure 7c: Bots vs Mean Collisions Over All Algorithms (Mapping + Collision)

How does the performance of the system change as I vary the number of agents in it?

From 7a and 7b it can be seen that there is a strong correlation between the amount of dirt collected / total map coverage and the number of bots. Both graphs resemble a logarithmic graph where for a lower number of bots there is a steeper gradient, but as the bots increases further the gradient becomes flat. This shows that increasing the number of bots does improve dirt collection and total map coverage; but this improvement becomes marginal after a certain point. The reasoning behind this is that at lower bot numbers there is less chance of bot's visiting already explored cells, but as bots increase, cell overlap increases, where it becomes harder to reach new cells. Furthermore, when there are more bots on the grid, there is a higher chance of bot interactions, thus the bot's coverage may decrease as it spends more time moving to avoid collisions. This can be seen by 7c where the line resembles a linear relationship. So as the number of bots increases, the number of collisions also proportionally increases, while dirt collection and map coverage will plateau. Meaning more collisions will occur but not with an equal gain in dirt collection and map coverage, leading to a reduction in efficiency.

Conclusion

To conclude, I believe I've created a solid project which answered multiple interlinked questions with good depth. I tried to create robust experiments which would aim to prevent outliers and inaccurate results. This was done through multiple runs, and the strict controlling of random elements within the experiments. I displayed my data in multiple formats to provide a stronger analysis and evaluation, setting out to clearly answer the set of questions I proposed in thorough detail; clearly stating limitations and possible improvements to my algorithms to help better future results.

Through this analysis I came to the first conclusion that the most efficient movement method to utilise would be the 'Attract' or 'Repel' algorithm, due to them scaling well with varying amount of bots, maximising both dirt collection and map coverage. They also encountered smaller amounts of collisions compared to other top performing algorithms. Furthermore, these two algorithms were less complex to implement and thus seen to be more practical in a real-world setting. While I found the best collision algorithm to be the Logistic Regression method where it drastically reduced the amount of collisions in comparison with the Danger Threshold method, where the results were found to be statistically significant. Finally, there was a clear positive correlation between varying the number of bots and the performance of the dirt collection, map coverage and bot collision metrics. Where increasing the number of bots past a certain point led to diminishing returns in dirt collection and map coverage, but a linear increase in collisions. Thus making it not optimal to add too many bots to an environment, as efficiency will decrease.

To improve upon the work provided in this project, I could develop upon the collision detection utilised. As I believe it is possible to make it more efficient at moving away from other objects, in a manner which is predictable instead of a crude randomised turning method. I feel this would reduce bot downtime in the experiments where they are trying to get out of a collision zone, and instead allow them to focus on the main goal, which is collecting dirt. Thus, it's likely to lead to more reliable results and conclusions. I would also like to have compared more learning-based methods such as Decision-Tree learning or a genetic algorithm. I believe this would add greater depth to my analysis as I can make more complex comparisons between the algorithms, for example, evaluating the tuning of hyper-parameters. In addition, I could attempt to combine different movement algorithms to see how performance metrics are affected.

References

1. Kuisong Zheng, Guangda Chen, Guowei Cui, Yingfeng Chen, Feng Wu and Xiaoping Chen. “Performance Metrics for Coverage of Cleaning Robots with MoCap System” (2017).

2. Robin Gunning. “A performance comparison of coverage algorithms for simple robotic vacuum cleaners” (2018)

3. Tommy Joe Lund. “Autonomous Robot Coverage Paths” (2017)

4. Joon Seop Oh, Yoon Ho Choi, Jin Bae Park, and Yuan F Zheng. “Complete coverage navigation of cleaning robots using triangular-cell-based map” (2004)

5. Enric Galceran and Marc Carreras. “A survey on coverage path planning for robotics. Robotics and Autonomous Systems” (2013)

6. Kollision and Vorhersage. “Collision Detection for a Mobile Robot using Logistic Regression.” (2019).