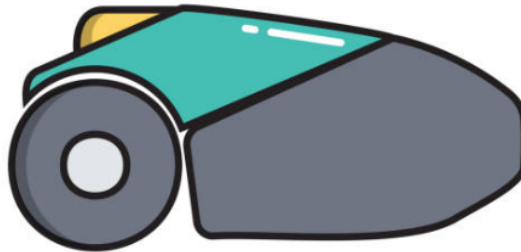




The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA

Robo-Crop



Background

Lawn Mowers are used consistently throughout gardens in the world for landscaping and maintenance purposes.

During a placement year at **BMW**, I had the opportunity to visit their HQ in Munich, in early 2022. At the heart of their innovation centre lay a large square garden around 25m x 25m, which was being mowed by a *single* robot lawn mower.

From observation within the 10 mins in which I was standing outside speaking to colleagues and partners, I had noticed that the robot had moved around very little.

The robot did not follow any consistent movement pattern, rather relied on erratic and random motion, where it sometimes stopped dead in its tracks and rotated continuously for no apparent reason.

While a piece of technology like this eliminates human involvement in tedious tasks like mowing a lawn, we need to explore ways to enhance its intelligence and performance; improving this is based on an *optimization problem* which will be discussed in this study, with the use of imitative experiments.

The experiments will involve different variables, with extensive self-reflection for a *reliable* and *reproducible* analysis of the problem.

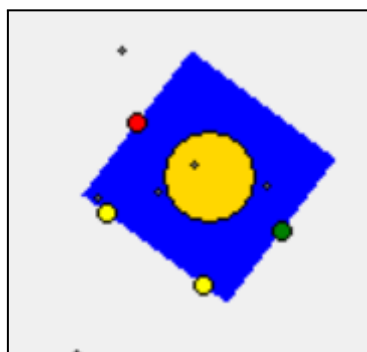
Michael Woolridge (h-index = 99), is a prominent academic and specialist of Intelligent Agents and Systems at University of Oxford. His paper '**Intelligent Agents: The Core Concepts**' [1] (and other papers) will strongly influence themes in our study.

In the end, I would like to discuss *the development* of '**Robo-Crop**'; an autonomous, robotic lawn mowing system based on the findings.

Designing the Intelligent Agent

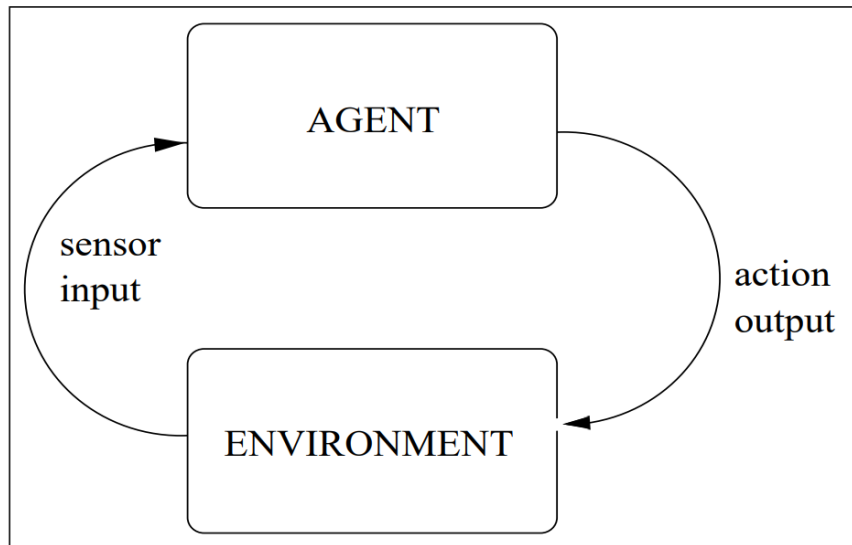
Intelligent Agents

The agent is a robotic lawn mower whose behaviour will be the topic of study across all experiments. While the lawn in Munich had only one mower, **multiple agents will be introduced**. The experiments will discuss the use of multiple robot mowers



- Sensors
- Right Wheel
- Left Wheel

Going by Russel and Norvig's definition (p32), RoboCrop is an intelligent agent because it acts on a stimulus perceived by its **sensors** and acts on its environment through its **actuators (wheels)**. The stimulus being **collision** with other bots, or the specified boundaries.



1.a Woolridge's blueprint for a Simple Reactive Agent [1]

Using Dr Woolridge paper for classification, summarised by Dr Bullinaria [2], RoboCrop is currently a '**simple reflex agent with internal state**'.

The internal state exists by virtue of each bot possessing little bits of information about the coordinates of others bots, and its own behaviour (such as moves remaining till the next turn), while performing simple reflex actions

In other words, it also knows and stores information which is not **directly perceived by its sensors**.

Dr Bullinaria also proposes a '**PAGE**' method to describe the agent:

PERCEPT	Pixels on the canvas which represent 'grass'
ACTION	Removing the dots off the screen, simulating the 'cutting of grass'
GOAL	Remove all dots off canvas; mow the lawn completely
ENVIRONMENT	1200 x 500 tkinter canvas to represent a lawn.

Environment

The aim is to simulate the lawn using a rectangular **tkinter** canvas populated by dots, which represent grass, done by dividing the canvas as a grid, and randomly populating each grid with the dots.

According to Woolridge, the environment would be classified as:

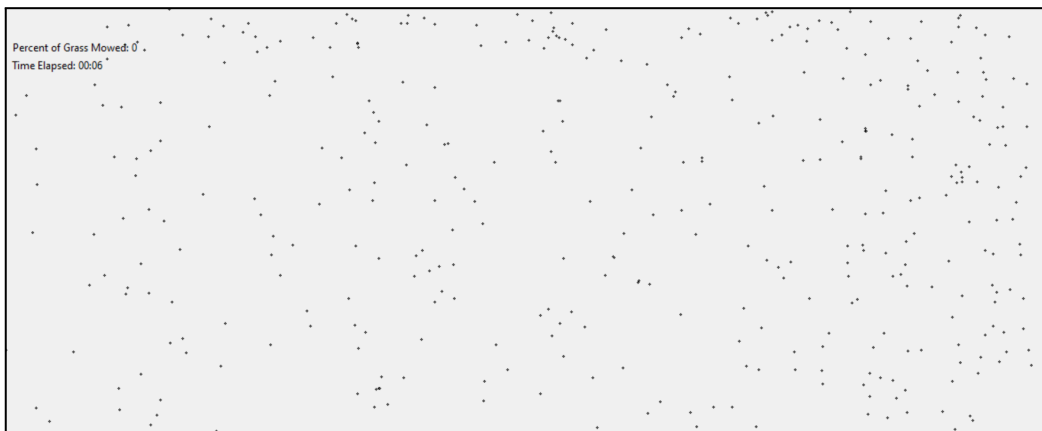
Accessible: all agents (mowers) would have the same, up-to-date information about the environment, and of each other (for example, *distance from each other at every frame*)

Non Deterministic: the position of the grass, its density, and the starting point of the bots is randomised everytime.

Non-episodic: there is no clear end to the bots' movement and they keep running until the program is closed, after which the environment will have to reset.

Dynamic: the agents interact with the grass dynamically

Discrete: only a finite range of 'things' the bots can do, like moving and wiping off dirt. In other words, it has '**Discrete Action Selection**'.



1.b Empty environment representing an empty unmowed lawn

The grass is placed with a degree of **randomness**, to **reduce bias** in the experiments. The grid reshuffles the grass everytime the canvas is redrawn.

```
[ 5  4  7  7  6  7  5  4 10  7]
[ 7  4  5  7  7  4  7  4 10  5]
[ 4  6  6  6  5  5  7  6 10  4]
[ 7  4  5  4  5  5  7  5 10  4]
[ 4  4  7  6  5  5  4  5 10  4]
[ 6  6  4  7  7  4  4  6 10  7]
[ 4  5  4  6  7  7  4  7 10  5]
[ 4  5  5  5  7  7  4  5 10  6]
[ 7  6  4  5  5  5  4  5 10  0]]
```

```
[ 4  6  5  7  4  7  4  7 10  5]
[ 4  6  4  4  7  6  7  7 10  7]
[ 5  5  5  6  6  5  4  5 10  4]
[ 7  7  7  6  7  4  6  5 10  7]
[ 7  4  6  6  4  6  6  5 10  5]
[ 5  5  5  5  6  5  7  6 10  7]
[ 4  6  5  6  7  7  7  7 10  6]
[ 6  5  5  4  6  6  6  5 10  5]
[ 4  7  6  6  5  6  5  5 10  0]]
```

1.c Shuffled Grid

While each dot doesn't represent a single blade of grass, we can think of them as a bundle of grass, with **varying density** as in a real lawn.

A counter on the top right displays the **time elapsed**, and the **percentage of grass cut**; the latter which will be studied as the bot changes behaviour.

Tools and Technologies

The code is built upon code written by Dr Colin Johnson. As mentioned, the agents, and environment is written in Python, namely the **'tkinter'** library.

The experiments are built using **'numpy'**, **'pandas'** and **'scipy'**. More information will be provided in the appendix, along with code authorship.

What are questions to be answered?

What is..

1. The effect of increasing **the number of bots** in an ideal system performance and fault tolerance of the system?

2. The **best movement algorithm** in terms of ideal system performance and fault tolerance?

'Ideal System Performance' and 'Fault Tolerance' are two interesting concepts which will be used to answer both questions.

If a number of bots are working simultaneously to sweep the canvas, '**Ideal System Performance**' is the condition that assumes all bots are working as programmed, with no faults in the system.

Fault Tolerance is a concept which is used in distributed systems where one or more nodes stops functioning. While applicable in various areas, this concept is realised in the context of autonomous robot systems in a paper by Agmon and Peleg [3]. While the pair discuss algorithms that combat the occurrence of faults, this study simply aims to see the effect of **Fail-Stop** faults in RoboCrop, and their effect on the overall system performance.

	number of bots	algorithm
ideal performance	?	?
fault tolerance	?	?

2a. What the study will look like

Collision

Across all experiments, one of the **constants** maintained will be the **collision mechanic**. A highly randomised collision function is used to give a 'bumper car' like effect where all bots involved bounce off each other once they go under a **Danger Threshold**, to maximise distance between them.

```
if self.IsBotColliding(bots):
    #self.turning = 40
    #self.currentlyTurning = True

    if random.choice([True, False]):
        self.x -= random.randint(10, 50)
    else:
        self.x += random.randint(10, 50)

    if random.choice([True, False]):
        self.y -= random.randint(10, 50)
    else:
        self.y += random.randint(10, 50)

    self.turning = random.randint(10, 50)

    self.currentlyTurning = True
```

2b. Randomised Collision

Since all bots on the canvas are governed by the class, this snippet introduces some **chaos** and **variance** in the behaviour of 2 otherwise identical instances of a Bot class, to simulate the real life collision physics.

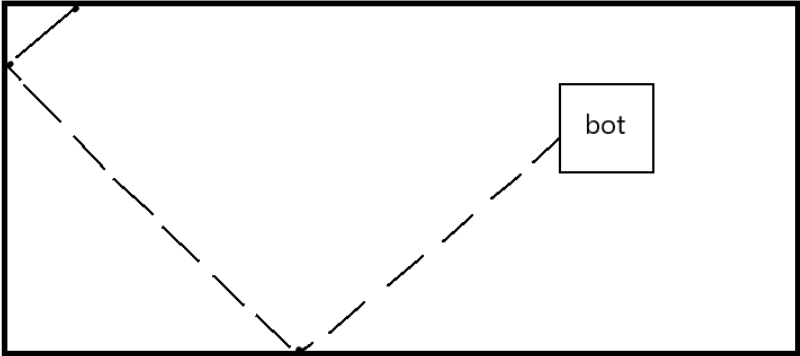
Movement Algorithms

The following algorithms and its mechanics are based on **IEEE's Kuri Robot** [4].

Kuri follows a simple pattern where:

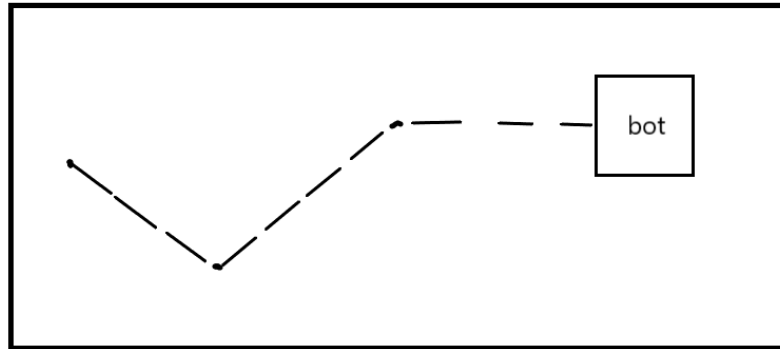
1. **Movement:** It moves in a straight line
2. **Trigger:** An event triggers it to change direction
3. **Method:** used to select a new direction

RoboCrop's algorithms follow roughly the same blueprint as Kuri's movement algorithm of 'Movement', 'Trigger', and 'Method'.

Algorithm Name	Description
Linear Wander	<p>This is the most basic algorithm of the set.</p> <p>Movement: <i>linear across the whole environment.</i></p> <p>Trigger: Collision with wall or other bots</p> <p>Method: Randomised</p>  <p>The diagram shows a rectangular environment. A dashed line starts from the top-left corner, moves diagonally down to the bottom-center, and then diagonally up to the right, ending at a small square labeled 'bot'. This illustrates a randomised path between collisions with walls or other bots.</p>
Random Wander	<p>Similar to Linear Wander, but with added randomisation:</p> <p>Movement: <i>linear across the whole environment.</i></p> <p>Trigger: <i>Collision with wall or other bots, + random</i></p>

direction change

Method: *Randomised*



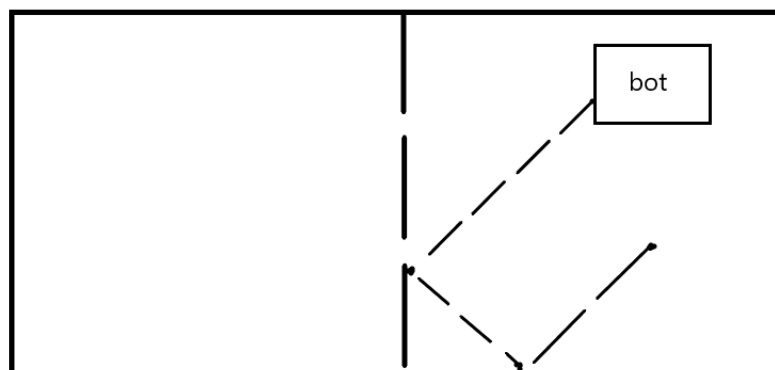
Bisector Search

The canvas is split into 2 halves, and the bots only search within the half they are randomly placed in.

Movement: *linear across the allocated half.*

Trigger: *Collision with wall or other bots, + approaching the soft border (invisible bisector line)*

Method: *Randomised*



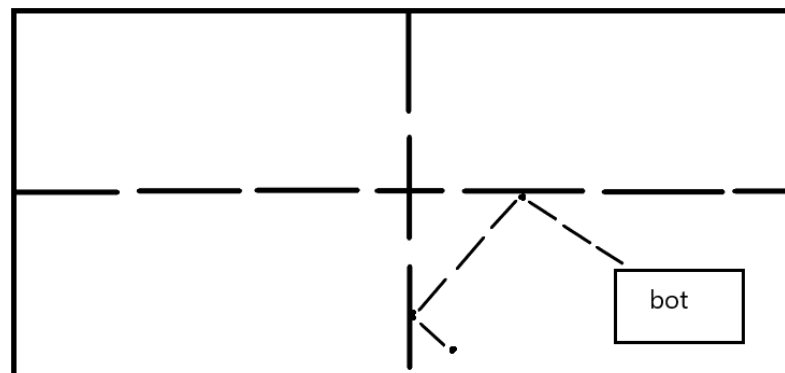
Quadrant Search

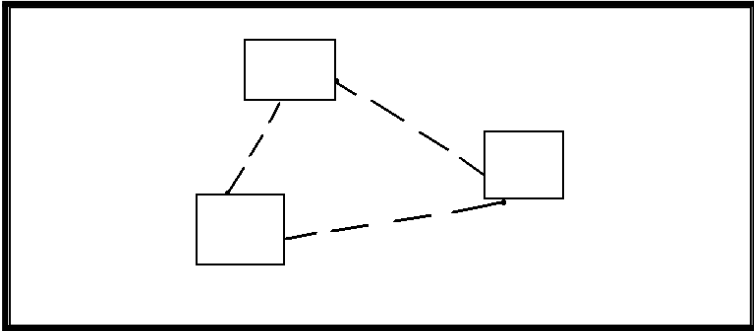
Similar to above, but the canvas is split into 4 quadrants, and the bots only search within the half they are randomly placed in.

Movement: *linear across the allocated quadrant.*

Trigger: *Collision with wall or other bots, + approaching the invisible border (quadrant line)*

Method: *Randomised*



Repel	<p>The bots 'repel' each other like similar poles of magnets so that they maintain a minimum distance from each other</p> <p>Movement: <i>linear across the canvas</i> Trigger: <i>Collision with wall, + too much proximity to other bots</i> Method: <i>Randomised</i></p> 
-------	---

Experiments, Results & Analysis

To ensure each run has enough time to sufficiently cut the grass, a constant limit of **500 moves** has been set so that the program ends when that limit has been hit. For one experiment, the timer feature on the canvas has been utilised and is used as a limit.

Across ALL experiments, the '*% of grass cut*' is the **dependent variable**, while the '*number of bots*' is the **independent variable**.

When answering the 2 questions, two different functions containing nested loops were used to organise and tabulate the results in a way where it could be visually represented. Their pseudocodes are slightly different, as explained below.

```
function runSetOfExperiments(numberOfRuns, numberOfBots, code):  
    listOfMeans = []  
    for i in range(1, numberOfBots+1):  
        grassPercentCollectedList = []  
        for j in range(numberOfRuns):  
            grassPercentCollectedList.append(runOneExperiment(i, code))  
        averageGrassPercent = sum(grassPercentCollectedList) / len(grassPercentCollectedList)  
        listOfMeans.append(averageGrassPercent)  
    return listOfMeans
```

3a. Pseudocode for Q1

For Q1, the **range of the number of bots forms the outer loop** since we are measuring the percent of grass cut as a function of the number of bots. Since we are concerned only with the number of bots, only a single line graph is produced.

For Q2 however, we wish to map the performance of *each* algorithm along with an increase in the number of bots, so while the *independent and dependent variable stay the same*, multiple lines representing a different algorithm are charted. This is done by having the **range of different algorithms as the outer loop**.

```

function SetOfExperimentsWithNumberOnly(numberOfRuns,numberOfBots):
    listOfMeansOfAllAlgorithms = []
    movementAlgorithms = ['lw', 'rw', 'b', 'q', 'r']
    for i in range(1, numberOfBots + 1):
        listOfMeansOfAllRuns = []

        for code in movementAlgorithms:
            grassPercentCollectedList = []

            for _ in range(numberOfRuns):
                grassPercentCollectedList.append(runOneExperiment(i, code))

            averageGrassPercent = sum(grassPercentCollectedList) / len(grassPercentCollectedList)

            listOfMeansOfAllRuns.append(averageGrassPercent)

        averageOfAllRuns = sum(listOfMeansOfAllRuns) / len(listOfMeansOfAllRuns)

        listOfMeansOfAllAlgorithms.append(averageOfAllRuns)

    ShowResult(listOfMeansOfAllAlgorithms)

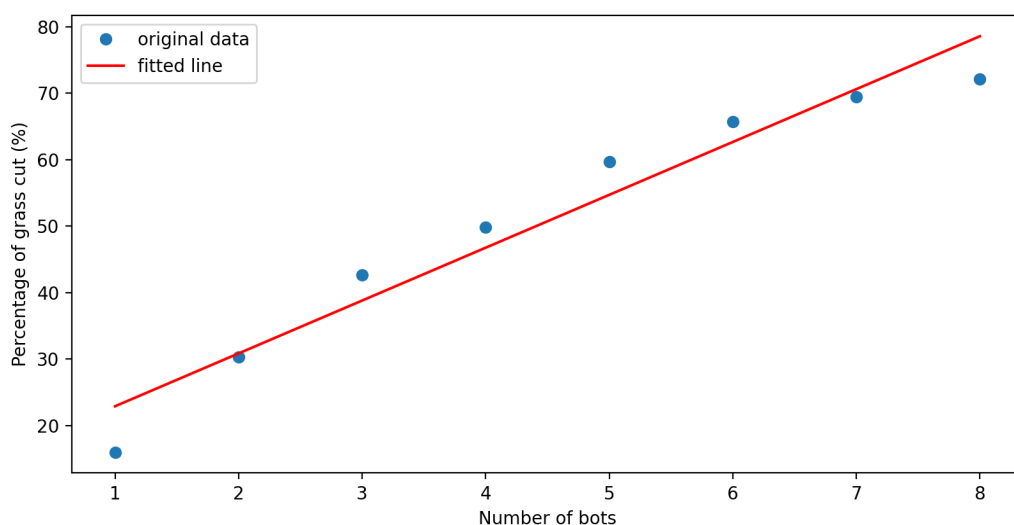
```

3a. Pseudocode for Q2

Ideal System Performance

In a system like RoboCrop which comprises many intricately designed machines, there are high chances that faults will occur.

This is **NOT** under consideration for the first set of experiments, and it will be assumed that each bot works exactly like how it is programmed to. There are no stoppages, faults or general breakages in their movement patterns.



3.c Number of runs = 10, limit = 500 moves

From the above, the clear pattern is that there is an **increase in the percentage of grass cut with an increase in the number of bots**.

To statistically verify this, Python's numpy library has been used to perform a *Linear Regression* test.

```
Slope: 7.949690476190475  
Intercept: 14.973642857142863  
R-value: 0.9730830518741751  
P-value: 4.7775859800235785e-05  
Standard Error: 0.768617516590847
```

The *Slope* is **7.95** (2dp) which means that there is an **average increase of 7.95%** of grass cut **with every extra bot** introduced to the system.

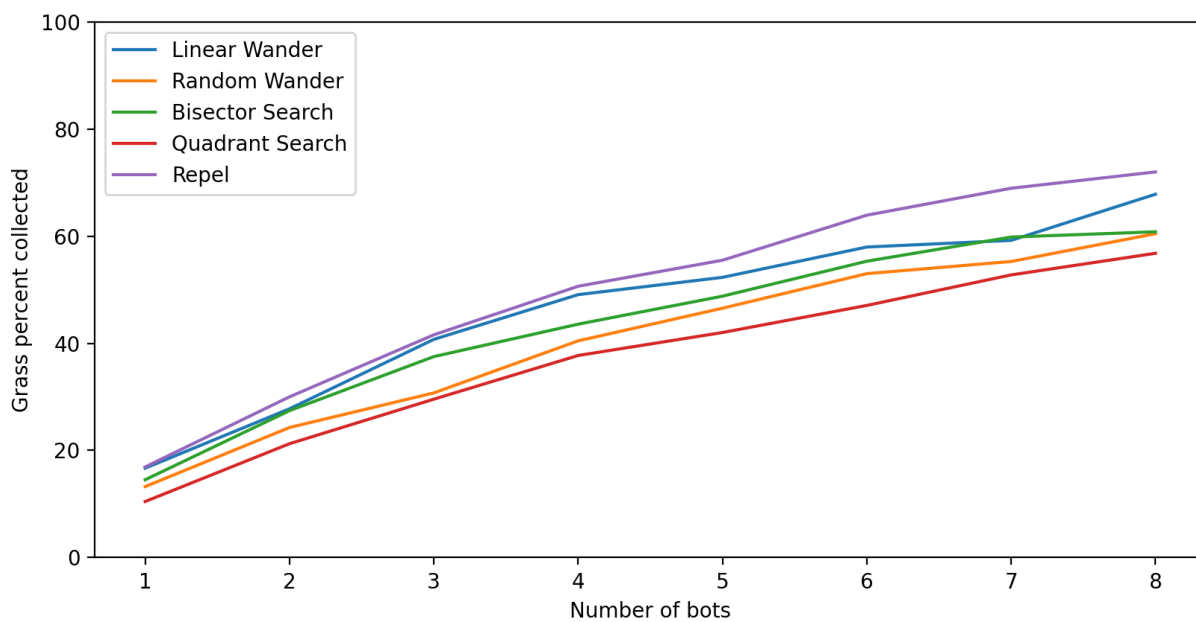
The *R-value* is **almost 1.0** which implies a **very strong positive correlation** between the 'number of bots' and the 'percentage of grass cut'.

The *P-value* is extremely small (**0.000048**) and is significantly less than **0.05**, which suggests that the linear relationship between 'number of bots' and '% of grass cut' is *statistically significant*, and *unlikely to be due to chance*.

The Standard Error is also quite low and this suggests that the line is generally a good fit for the data points, so it is reliable.

At Least in terms of ideal system performance, it can be concluded that:

“Increasing the number of bots, improves and increases the % of grass cut in the environment”



3.d Number of runs = **10**, limit = **500 moves**

The above graph analyses the performances of different movement algorithms

The first trend which can be observed is that there is an increase in the % of grass cut with an increase in the number of bots, *across all algorithms, individually*.

While this has already been suggested to be the case with the previous experiment, there was always the possibility that any algorithm didn't follow the pattern, and that this was covered up by the average performance of all algorithms, for that number of bots. This has obviously been disproven now.

Logically, more bots cover more area in the same number of steps/same amount of time, coming closer to the ideal goal of 100% of all grass cut.

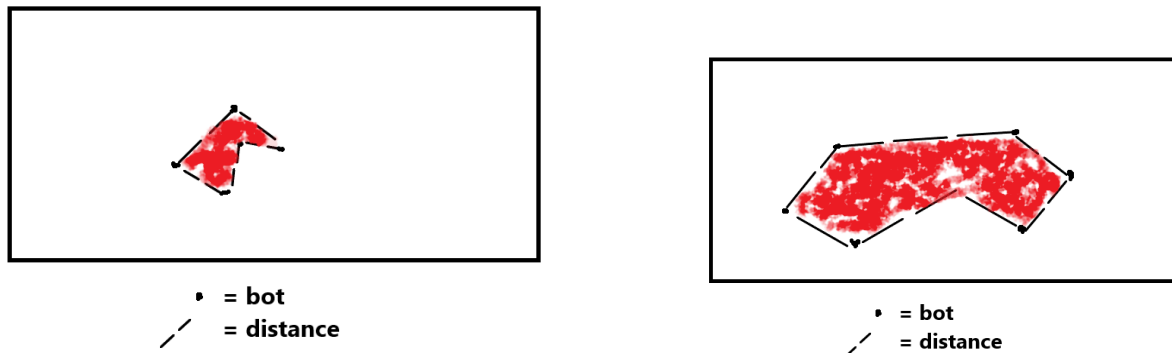
While most coverage stays under 80% for all runs, it can be confidently be said that the bots would eventually clean the whole canvas given more time /or moves.

The algorithms also maintain their relative positions across all numbers of bots, apart from when the number of bots is 7, when Bisector Search very narrowly beats Linear Wander to come at position 2.

The algorithms show an equal, linear increase in the % of grass cut with the number of bots. While the performance of the algorithms are similar at a low number of bots, the gulf between them increases with the number of bots.

Repel demonstrates best performance compared to other algorithms. It is probably explained by the fact that the bots aim to keep a minimum distance between each other, so the whole network of bots covers a larger area than other ones where the bots are more social and allowed to be within close proximity of each other.

The following diagram explains this better



3.e 'A bigger constellation'

Repel is the best performing algorithm which makes use of the internal state of the individual lawn mower, and uses it to its advantage.

The next best performing algorithm after Repel is **Linear Wander**. While this is a simple reactive algorithm for the most part, a bot's path stays uninterrupted unless another bot is in the way, allowing a relatively undisturbed traversal letting it mow down whatever is in the way.

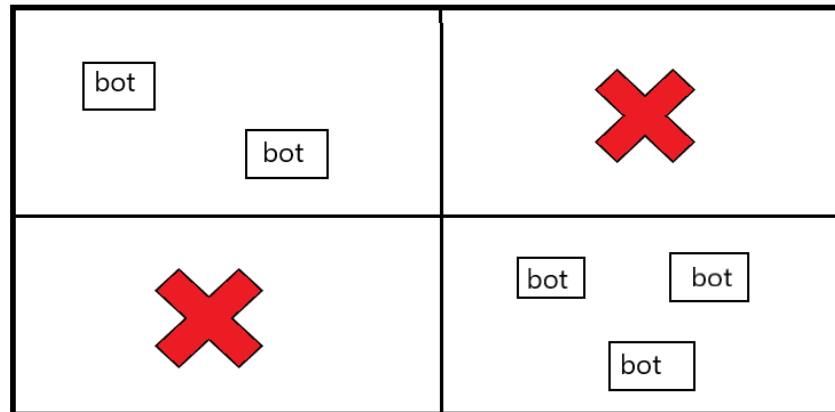
Bisector search and **Random Wander** perform average. They are also the 2 algorithms which perform most similarly.

For Bisector Search, the strength of the algorithm lies in the fact that the canvas is split up in 2 halves, and that a bot is made to stay in its allocated half, **ensuring a higher chance of a thorough sweep of the whole canvas.**

On the flip side, this can also prove to be its weakness, where if all bots are randomly generated on one half, then the other side remains untouched. The equal chances of this happening is likely the explanation of its average performance.

Random Wander likely performs decently due to the fact that a single bot can travel anywhere across the canvas, but a large number of collisions due to its erratic bee-like movement can result in a lower overall coverage compared to its cousin, **Linear Wander**.

Quadrant Search performs the poorest. From 5 bots onwards, it consistently mows on average 20% less grass than the best performing algorithm, Repel. This is explained by the phenomenon that although splitting the canvas into 4 quadrants should result in better overall coverage, a quadrant may not have any bots generated in it, leaving it untouched.



3.f 'Why QS may be the weakest'

While the program being fair and random in generating bots, this is unavoidable. In real life however, this can be mitigated by the human operator placing bots in each quadrant.

While the above are likely explanations for the behaviour observed, it is imperative to gauge their likelihood using a statistical test like ANOVA.

After performing ANOVA (from Python's SciPy) on these results, these were the values obtained:

```
F-statistic: 0.674795458116367  
p-value: 0.6139544598235438
```

3.g ANOVA

The **F-statistic** which is **relatively low (0.67 dp)** shows that there isn't much variability between the means of all algorithms. This can be best explained due to the fact that all algorithms behave generally the same in terms of ideal system performance.

However, the '**p-value**' is significantly higher than the commonly used threshold of 0.05, which means that we don't have the evidence to reject the null hypothesis that the behaviour of the different algorithms is due to anything other than chance.

This is probably explained by the fact that while there exist subtle differences between the algorithms, their movement mechanics are still similar on a fundamental level. They all have **mostly linear movement** with **some degree of randomness** along with **identical collision physics**.

In other words, while there is some difference observed in the behaviour of the different algorithms in terms of ideal system performance, **it isn't conclusive enough to draw a distinct relationship between the algorithms**.

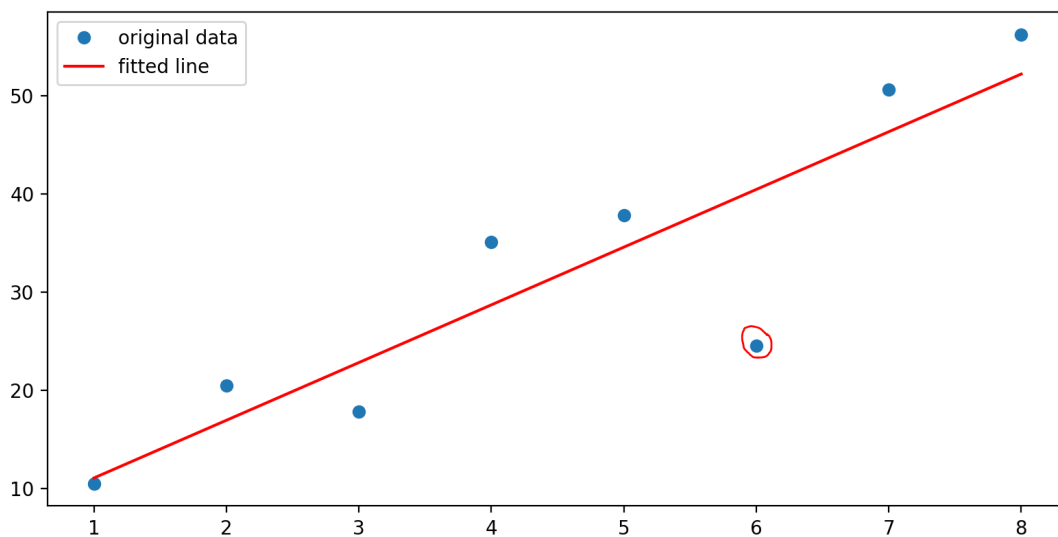
Fault Tolerance

To keep the degree of faultiness in the bots fair, randomness was used once again. A simple snippet of code was introduced in the constructor of 'Bot.py':

```
if random.randint(1,4) == 3:
    self.isFaulty = True
else:
    self.isFaulty = False
```

3.h Fault rate = $\frac{1}{4}$

The chance of a faulty bot is 25%, and if a bot is chosen to be faulty, it remains stationary throughout the run.



3i. Number of Runs = **10**, limit = **500**

For just the number of bots alone, the trend stays the same as before. There is *still an increase in the average percent of grass cut across all algorithms, with an increase in the number of bots*. There is seemingly an outlier at 6 bots, but this doesn't radically change the line of best fit.

```
Slope: 5.872785714285715
Intercept: 5.227714285714285
R-value: 0.8907486660555569
P-value: 0.0029987354441780058
Standard Error: 1.2233299947590175
```

3j. Linear Regression

The Linear Regression test seems to corroborate this further. There is a slight decrease in the slope compared to an ideal system, but it is still positive.

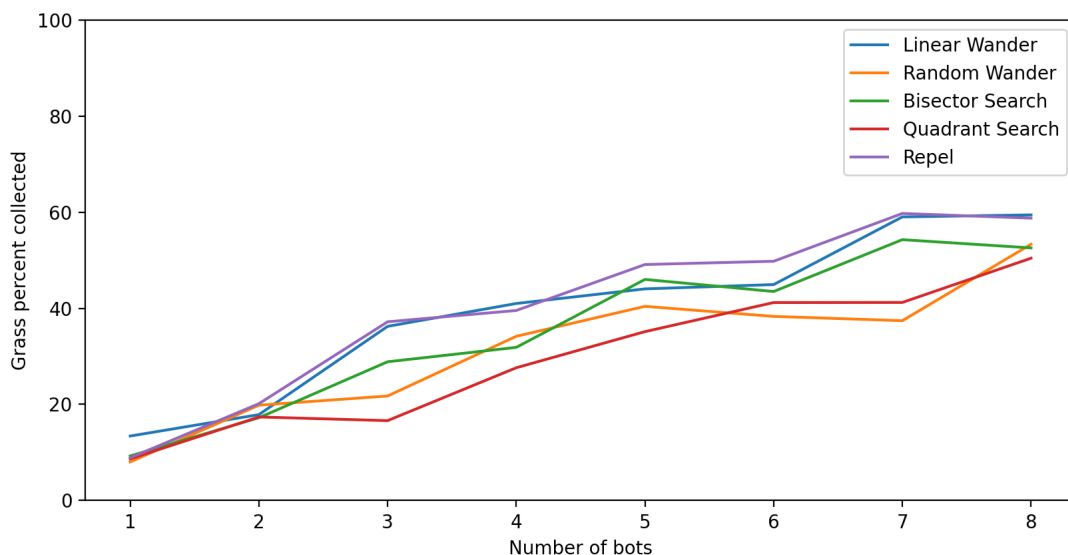
The R-value whilst being slightly lower is still close to 1.0, indicating a **very strong positive correlation** between the number of bots, and % of grass cut.

The P- value and SE are also slightly higher than the previous ones, but are still quite low suggesting strongly that our results are **statistically significant**, and that the model is **still generally a good fit**, respectively.

Overall, **increasing the ‘number of bots’ has a strong positive effect on the % of grass cut**, even in a **faulty system**.

So this is a strategy which has **high fault tolerance** and can help us achieve our main goal of mowing the lawn clean.

The following graph shows the performance of different bots in a faulty system. Stability is a concept which will be talked about in this context, and it will refer to whether an algorithm maintains a relatively linear trend in its graph.



3k. Number of runs = 10, Limit = 500 moves

On the whole for each algorithm, there may be an increase in the percent of grass cut with an increase in the number of bots, but the percent of grass cut is less than for the same algorithm and number of bots, in an ideal system.

This is justifiable as each broken bot is effectively as useful as having one less bot.

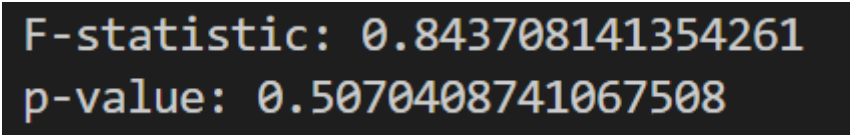
Moreover, the shapes and relative positions of each algorithm are not consistently maintained, revealing lower stability in certain ones.

Repel remains the best performer; it is also somewhat stable and has less steep troughs in its graph, being overtaken by Linear Wander only twice. This is probably due to the fact that it aims to cover a larger chunk of the map, and since each bot can move somewhat freely throughout, they can overcome the limitations of a stationary bot.

Bisector and **Random Wander** seem to have the most erratic performance, and change their relative positions frequently. Unfortunately, there isn't enough evidence to explain a possible reason behind this phenomenon, and more studies may be required to do so.

Quadrant Search continues to perform worst (except when it overtakes Random Wander at 6 bots). This is likely explained by the fact that in addition to partition, if a sector only has faulty bot randomly spawned in it, its grass remains uncut.

Just like the previous set of experiments, ANOVA was used to gauge the statistical significance of the graph.



F-statistic: 0.843708141354261
p-value: 0.5070408741067508

3I. ANOVA

Compared to an ideal performance, The F-statistic is higher here, implying a **higher variability in the means** of the different algorithms, owing to differing levels of stability in each algorithm in a faulty system.

The p value, while still being fairly high, is closer to the statistical significance threshold of **0.05**. While the results may still be insignificant, there is an emerging trend here; *a greater number of bots, runs and/or fault rate may reveal a wider, more statistically significant difference in behaviour of different algorithms.*

Overall, there is a slight trend in the difference of performance of the bots in a faulty system, but the **evidence isn't strong or statistically significant enough to draw any solid conclusions**. Further research will be required for it.

The answers to both questions are summarised in the below table. ✓ means that a **statistically significant correlation has been found**, whereas ✗ means that **nothing conclusive was found**.

<i>Conditions</i>	Number of bots and the performance of the system	Different movement algorithms and the performance of the system
<i>Ideal System</i>	✓	✗
<i>Faulty System</i>	✓	✗

Evaluation of Study and Experiments

Successes

The study was kept as fair as possible. The following were kept constant and random where appropriate to ensure the results would be reliable, and reproducible if repeated under similar conditions.

Random	Constant
Arrangement of Grass across the canvas	Grass Generator Function (which ensured that while the grass was randomly generated, the function which generated it remained the same across all runs.)
Coordinates where the bots were generated	Size and velocity of the bots
Faulty bots	Function which generated faulty bots

	Size of canvas
	Length and number of runs
Collision Response	Collision Function

Due to the consistency and transparency used in all methods above, not to mention the limitations mentioned below, the study should be **reliable, repeatable and reproducible**.

If the same methods were to be employed by another researcher, similar results should ideally be obtained.
This in itself should be classed as a success.

Moreover, of the 2 questions asked, both were answered concisely, and while one was found to have statistical significance, **research can always be furthered**.
More extensive research could therefore help shed further light on both, and potentially find some substantial explanations for the second one, so the above study provides a good foundation for that.

Limitations, Challenges & Improvements

It is hoped that this appraisal will help further the research on RoboCrop by highlighting the gaps to be filled for future studies.
Here, limitations of the study and experiments are detailed along with possible improvements if time or other factors permitted.

Type	Limitation	Possible Improvement
Tools and Technology	'Python tkinter' cannot simulate the wind, weather conditions, unevenness of the ground and other factors.	Add elements of 'wind' by causing bots to slow down if moving in one direction.
	The shape of the canvas remains constant throughout	Obstacles could be added on the canvas to improve and measure

	<p>all experiments, and it cannot be assumed that the performance of all agents will stay consistent on a different shape of lawn.</p>	<p>the robustness of the experiments.</p> <p>The experiments could also be run on different shapes of garden.</p>
Experiments	<p>The lack of conclusive evidence for Q2 may be down to several factors in the method of experimentation:</p> <p>The number of runs is only 10.</p> <p>Each run is only tested with 500 moves</p> <p>The percent of grass cut is the only independent variable being measured</p>	<p>Increase the number of runs. Although this is time and power intensive, this needs to be considered.</p> <p>Test with varying number of runs, and other features such as time.</p> <p>Consider other potential independent variables like map coverage.</p>
Code	<p>Collision mechanics may look crude and unrepresentative of a real collision when the frame rate is slow</p>	<p>Add a feature where the bots gently halt when about to collide, followed by a turn in the opposite direction.</p> <p>Inspiration taken from the novel Starship Food Delivery Robot operating in Milton Keynes [5], which halts in front of obstacles.</p>
Code	<p>Collision of cross-sector and cross-quadrant bots in Bisector and Quadrant Search algorithms.</p> <p>If bots on both sides of the border are too close, the</p>	<p>Have an additional detector function which only detects collisions within the allocated sector or quadrant of the bot.</p>

	<p>collision mechanic kicks in causing either robot to get pushed into the other half or quadrant, where it isn't supposed to be.</p> <p>This could potentially explain the poor performance of the 2 algorithms, especially the latter, since a half or quadrant with a single bot now has nothing inside it, after a wrongly triggered collision</p>	
Code	<p>The Repel algorithm has a bug in it, where if two or more bots approach the repel threshold, sometimes the reaction fails to kick in and they simply pass over each other.</p>	<p>This needs to be investigated and fixed as this may be unfairly causing the algorithm to beat every other one in terms of performance.</p>

All in all, the biggest limitation is probably the intelligence of **RoboCrop**.

Woolridge classifies the following 3 tenets of an intelligent agent:

- Reactivity
- Proactiveness
- Social Ability

RoboCrop currently focuses only on **Reactivity**.

While the bots appear to be working together, they are essentially individual entities which happen to be sharing the same environment. As the graphs about algorithms suggest, the movement algorithms are quite simple and have similar underlying reactive mechanics.

Introducing elements of Artificial Intelligence such as **Pathfinding, Machine Learning, along with distributed coordination** would help introduce '**Proactiveness**' to RoboCrop. These techniques augment the capabilities of simply cutting grass and attempt to make the goal achievable in a more optimal time and manner.

For instance, a possible improvement to the movement algorithm would be to mention the use of **Computer Vision** and **Neural Networks** to efficiently map other robots, and avoid travelling the same grids of grass again. This is inspired by the Ecovacs robot cleaner system which employs these techniques, with the help of **TensorFlow** [6]. Naturally this would require the necessary actuators such as **LiDAR**, **cameras** etc.

One of the biggest possible improvements to RoboCrop is the introduction of a **connected network of intelligent bots**, called **Distributed Control**. Each bot would be programmed with an individual algorithm, but also the awareness of the whereabouts of other bots, as proposed in Yu Zhou's paper [7].

This would encourage them to be less social; avoid each other's path under normal circumstances, but show coordination during times of faulty bots like in the experiments. Hence, Woolridge's final principle of **Social Ability** will come into play, where a single bot would interact efficiently when required with other bots to ensure the long term goal is achieved. For instance, to prevent a **Byzantine fault**, the remaining bots can be correctly alerted so that they can pick up the slack for their lost comrade.

In the end, having considered all of the above, it is safe a long term goal would be to turn RoboCrop from a simple reflex agent with internal state, to a more complex, efficient **Utility based agent**; an agent which has **state**, **reaction** but also an idea of the goal along with process where it constantly learns techniques to achieve that goal more optimally.

In Conclusion

RoboCrop is a promising idea in theory, and it has the potential to revolutionise the landscaping industry by increasing efficiency and reducing costs and time of lawn mowing.

However, the experiments conducted don't provide conclusive evidence of what is the optimum algorithm, so we can't gauge whether it will be more efficient compared to a human operator.

In other words, further extensive research is required for improving the intelligence of RoboCrop so that it can supplant the most intelligent agent in existence, Human.

References:

"We don't count towards the word limit"

[1] Woolridge, Michael

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7434cc7952a7d9b7fb74856d8ab5d60917aa1e0a>

[2] Bullinaria, John

<https://www.cs.bham.ac.uk/~jxb/IAI/w4.pdf>

[3] Agmon, Peleg

<https://u.cs.biu.ac.il/~agmon/AgmonSICOMP06.pdf>

[4] IEEE,

<https://spectrum.ieee.org/wandering-robots-in-the-wild>

[5] Starship,

<https://www.starship.xyz/robot-food-delivery-in-milton-keynes/>

[6] Ecovacs,

<https://blog.tensorflow.org/2020/01/ecovacs-robotics-ai-robotic-vacuum.html>

[7] <https://www.intechopen.com/chapters/6589>

Appendix:

"I don't either"

All the source code is contained within the '**RoboCrop**' folder.
'**cd**' into it after you unzip the file.

The code is based on Colin's lab tasks, but has been significantly changed and built upon.

Each function has a comment under its signature which indicates whether the code is entirely mine (Yash's), Colin's, or a mix of both. If the entire class was made by either party, then that is stated at the top before the imports.

Since there are 2 questions, there are 2 experiment files dedicated to each, called 'ExperimentsQ1.py' and 'ExperimentsQ2.py'.