Optimisation of Traffic Management in a Three-Dimensional Reward Based Robotic Swarm

Thomas Harrison
Lancaster University
School of Computing and Communications
35584689

***Abstract*—Target congestion and collision avoidance algorithm were implemented into a foraging inspired navigation system simulated in a three dimensional Cartesian coordinate system. The collision avoidance algorithm utilises position states to create a 'traffic light' approach to force objects to wait if they risk a collision. Three different congestion avoidance algorithms were implemented; introducing exit and entrance regions to control objects leaving the common target and returning to the common target respectively; introducing a upper limit in the exit zone which is unique to the objects to reduce the likelihood of head on collision and defining a band in the entrance regions which force object to wait if there is a risk of convergence when entering the common target. These algorithms were shown to reduce congestion and avoid any deadlock scenarios. Additionally the individual performance of the objects in the system were analysed statistically with a varying number of objects (20-50) with two different object radius sizes (0.1, 0.2), and were statistically similar when considering 20-30 objects in the system. This statistical similarity shows efficiency in algorithm with smooth path navigation.**

***Index Terms*—Swarm Robotics, Congestion, Traffic control, Foraging, Multi-Robotics, Artificial Intelligence**

I. Introduction

Swarm robotics is a field of multi-robotics where numerous objects/robots navigate in a decentralised system. These systems are composed of numerous robots/objects attempt to navigate in a confined space. The concept of robotic swarms emerges from natural swarm behaviours, examples include ant colonies and bacterial swarm. Ant colonies are examples of a foraging system and utilise pheromones to mark trails to inform other ants where a food source lies. These behaviours can be adapted and used to create algorithms which simulate this behaviour in multi-robot system.

Foraging is the act of searching for objects such as food, collecting and then returning to a collection point. The act of foraging can be done via robots working individually (solitary foraging) to optimise their own outcome or multiple robots acting collectively (group foraging) where the success is based on the collective group outcome[1][2].

The proposed problem was foraging inspired. This problem assumed the foraging location is known and unique to each object in the system. Other foraging systems operate on a basis where the foraging point is unknown and needs to be located by each object. This can be analogised to an airport being the center target with docking and departing airplanes. Each airplane can be considered an object within a system which knows its own unique target (the location it is travelling to) and must travel there, drop-off and load passengers and return to the original airport (central point). The core principle is to create an environment that best optimises the overall reward whilst similar outputs for each object. Similar comparisons could be made to a drone delivery system. The center target can be considered as a warehouse and each delivery drone an object in this system. For each run the drone in the system has numerous deliveries to make where their address (unique target) is random each time. The drone can only deliver one package per journey and needs to return to the warehouse to restock.

An important factor to consider in multi-robot navigation systems prior to any congestion avoidance methods is collision avoidance. Collision avoidance can be executed using a traffic control method proposed in Marcolino et al which uses a probabilistic congestion control algorithm (pcc) which aimed to reduce congestion by reducing the number of robots that headed to the center target simultaneously[3] . The algorithm used in this simulation was a simplified version using two different position states 'waiting' and 'normal' to control the navigation of objects to force them to stop if their continued path would result in a collision then continue when it safe (no risk of a collision).

After collision avoidance was accounted for there was a need to avoid target congestion, this occurred when a large number of robots attempt to move toward a common target. Marcolino et al proposed numerous different algorithms which successfully aided in reducing congestion [3]. These included defining an entry and exit zone to control to direction of the objects and introducing a danger zone surrounding the target to force objects to wait. These were found to be effective algorithms and were implemented into this project by considering two exit zones and 4 entry zones composed of square based pyramids. Likewise the danger zone was implemented and the values of the inner region of this zone was varied by adopting an ϵ-greedy strategy with weighting based off the total reward for each run. Each run was defined as 5×10^5 iterations when evaluating the lower limit values. Additionally an upper limit was introduced into the exit zones which is unique to each object in the system. This was implemented to reduce the risk of head on collisions between objects; reducing the number of head on collisions will create smoother path navigation.

Traffic congestion is a serious issue that can impact economical growth

II. Related work

There are many studies which involve traffic control in robotic swarms. Many studies utilising animal swarms adopt a foraging strategy strategy which used pheromone-mediated coordination for navigation purposes where ants leave a trail of pheromones to direct other ants to a food source. They managed to successfully reproduce the observed foraging behaviour by implementing a system of partial differential equations of chemotaxis[4][5].

Local and global are two main types of collision avoidance categories [6]. Global based collision avoidance methods (potential field methods for example) have been found to be problematic when the world model changes during processing. This causes a need for repeated adjustments to account for these issues leading to an inherently slow model[7].

van den Berg et al proposed a new concept "Reciprocal Velocity Obstacle" for multi-agent navigation [8], this concept was an extension of the Velocity Obstacle concept in Fiorni et al [9]. The algorithm proposed in Fiorni et al used avoidance manoeuvres to avoid static and moving obstacles in the velocity space. Van den Berg et al expanded on this by accounting for the reactive behaviour of other robots. This was accomplished by assuming that the other robots in the system also make similar collision avoidance reasoning's.

Further research has indicated that the object density (number of objects per total enclosed volume of the system) was a highly influential factor and important to consider for best optimising foraging tasks [10].

Marcolino et al proposed several solutions to avoid target congestion when numerous objects navigate towards a common target. The first of which was a probabilistic congestion control algorithm (PCC) using a probabilistic finite state machine to reduce the number of 'selfish' robots by causing them to wait in a 'danger' zone (a band defined by two limits surrounding the central target). Their second proposed algorithm was Entrance and Exit Regions (EE) which separated the total enclosed area into entrance and exit region to reduce congestion from exiting and entering robots. These two algorithms were combined to for PCC-EE which was shows to improve navigation with statistical significance.

III. Method

A. Setup

Each object is identical and modelled as a ball of radius r_o in a pre-defined volume (V), this volume constitutes a cube of length 10 units composed of six square based pyramids with a common target ($r_t = 1$) at the centre (0, 0, 0) in a Cartesian coordinate system. A cubic system was considered due to the simplicity and symmetry about each plane. The simplicity of the core set up facilitates the ability to alter the size of the entrance and exit zones pyramids as required. In the proceeding analysis the entrance and exit zones were equal in size. There were no obstacles present within the cube; the object could move to any point in the system provided that another object did not occupy that space, this was to keep the environment simple and to test the validity of the collision-avoidance system and congestion issues without any factors that may influence the outcome. The simulation was visualised using the `VPython` library; this was used to make any congestion issues or collisions more prominent.

Parameters used	Value	Symbol
Length Side	10	l_s
Object radius	0.1, 0.2	r_o
Target radius	1.0	r_t
Unique target radius	0.1, 0.2	-
Mass object	1	m_0
Position vector	-	$\vec{p_i}$
Number objects	20, 30, 40,50	N
Total volume system	10^3	V
Direction vector	-	$\vec{\rho_i}$
Object density	-	ρ_{object}
Volume density	-	ρ_{volume}

Table I: Main parameters used within the experiments with their corresponding value and symbol. Areas marked with '-' represent values which changed periodically or depended on the combination of other parameters.

There are several environment parameters explored as show in table I.

- **World border** - The length of the border itself was left dimensionless so that it can be compared and scaled to the object radius. The initial test was implemented on a scaling factor of 100 where the length of each border was 10 and $r_o = 0.1$.
- **Object density** - The object density can be defined as the number of objects in the swarm per volume of the confined space($\frac{N}{V}$). This will give insight as to whether object density has the potential to converge to an optimal value. When applied to the foraging scenario it may reach a point at which adding more objects into the system will reduce the overall reward (total number of object which reach their destination and returned to the starting point over a set period of time). This result would help optimise any industry applications by increasing the output with a reduced number of objects (drones/machinery) leading to economical benefits.
- **Foraging target separation** - There was no constraint within the position of the foraging target, the target could lie within any point of the confined space. This was to explore as many path navigation scenarios of the objects as possible to identify and resolve any issues that arose. If only certain paths and set positions were considered then the model could not be justified to work in a general scenario.
- **Unique foraging return** - The total number of successful foraging tasks (reached the foraging target and returned) for each object in the system. In each run there is no limit to the number of times an object can perform a foraging task during each run of 1×10^6 iterations. For defining the convergence band (explained in section VII) the number of iterations was reduced to 5×10^5.

The foraging return will give a strong insight to the performance of the model overall; evaluating an overall score of the system alongside the individual scores of the objects operating within this system will show whether the proposed algorithms perform effectively as a whole. This means no objects will be inefficient and 'stuck' in deadlock.

IV. Defining entry and exit regions

Each object will navigated towards the main target simultaneously; without any constraints on the objects path they could approach from any direction which led to head on collisions where they could not move. This caused them to become 'stuck' in a deadlock scenario.

One method which was proposed in Marcolino et al to alleviate target congestion used entrance and exit zones which proved to be effective at improving navigation with statistic significance [3].

To test if an object was located in the top or bottom pyramid based its relative x_t, y_t and z_t position were used. Taking the current y-coordinate of the object the corresponding x-coordinate of the intercept between the exit zone and entrance zone could be determined. The same procedure was repeated in the xz-plane, xy-plane, zy-plane to determine if the object was in the left/right region or front/back region. The full conditions are as shown in equations1-3 and visualised in figure 2:

Determine if the object is in top or bottom pyramid:

$$\left.\begin{matrix} \left(|x_t| < |y| \times \tan(\frac{\pi}{4})\right) \\ \left(|z_t| < |y| \times \tan(\frac{\pi}{4})\right) \end{matrix}\right\} [\text{-l}_s, l_s] = \{(x,y,z) \in \mathbb{R} | -l_s < (x,y,z) < l_s\} \tag{1}$$

Determine if in left or right pyramid:

$$\left.\begin{matrix} \left(|z_t| < |x| \times \tan(\frac{\pi}{4})\right) \\ \left(|x_t| > |y| \times \tan(\frac{\pi}{4})\right) \end{matrix}\right\} [\text{-l}_s, l_s] = \{(x,y,z) \in \mathbb{R} | -l_s < (x,y,z) < l_s\} \tag{2}$$

Determine if in back or front pyramid:

$$\left.\begin{matrix} \left(|z_t| > |y| \times \tan(\frac{\pi}{4})\right) \\ \left(|x_t| < |z| \times \tan(\frac{\pi}{4})\right) \end{matrix}\right\} [\text{-l}_s, l_s] = \{(x,y,z) \in \mathbb{R} | -l_s < (x,y,z) < l_s\} \tag{3}$$

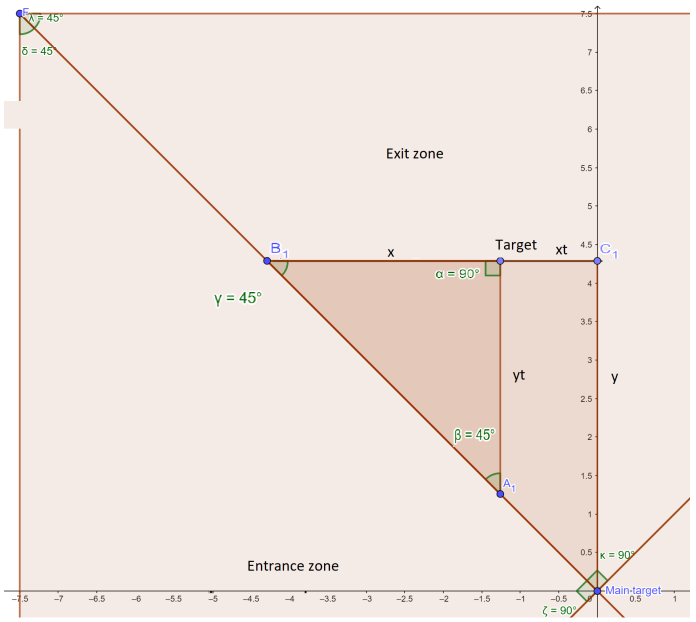


Figure 1: 2D visualisation of an object in the xy-plane. The object position is shown by (xt, y). This setup was used with the above conditions to determine whether the object is in the exit zone or entrance zone.

V. Defining upper limit in exit zone

The exit/entrance zone were a strong addition for the objects to create smoother path navigation. An upper limit was defined within the exit zones, this was implemented to further reduce congestion and used for collision avoidance. This limit was based on the unique targets for each object (explained in section VI).

Two variations of an upper limit were explored; the first variation defined an upper limit which was common to all targets, the second variation based the upper limit on the y-coordinate of the unique target.

First variation of the upper limit constraint:

All objects initially moved vertically (y direction) away from the center target. An upper limit was defined which was common to all targets in the system. After each object reached the upper limit they could move with probability $\rho_{limit} = 0.8$ towards the region which held the individual target otherwise they continued to move vertically. If the individual target lay within the exit zone the object would navigate towards this target and the upper limit had no impact on this objects navigation. This implementation was found to have some issues, numerous objects collided head on since there were no additional constraints on the y direction vector. Once an object reached this limit it then navigated towards its unique target which are positioned randomly; implementing a common upper limit proved to have no additional benefit due to the foraging styled system. Whilst the problem with these head on collisions could be solved it did not create a smooth path navigation since the colliding objects had to divert their path away from each other which also leads to in inefficient system. The ideal scenario was to design a system to reduce the likelihood of direct collisions whilst not increasing the time it takes for the object to reach its unique target. This led to the design of the second upper limit.

Second variation of upper limit constraint:

The concept of implementing an upper limit was kept but instead of having a general upper limit which was common for all objects each object had its own unique upper limit which was the y coordinate of the unique target. This implementation reduced the likelihood of head on collisions between the objects since this method constricted the y-value of the direction vector (ρ_y) in such a way that ρ_y will either be 0 or directed towards the target. Any deadlock scenarios that remained were accounted for through use of the cross product from the vector that connected the centers of each object.

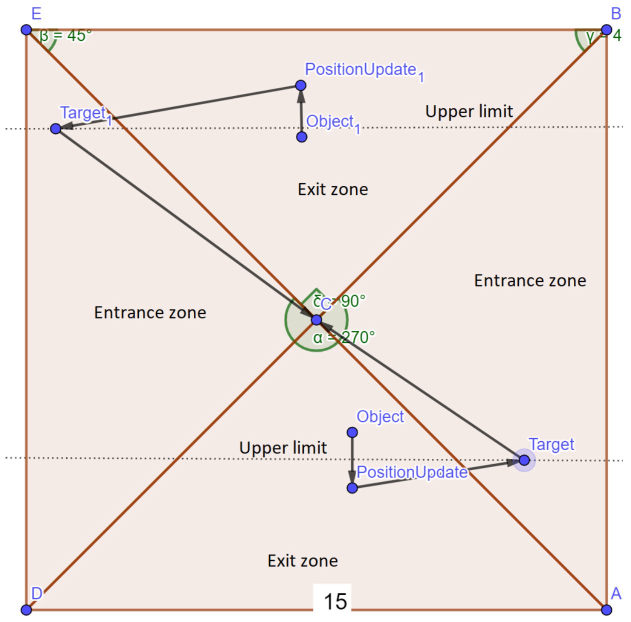


Figure 2: 2D visualisation of two objects passing through the upper limit and return to the center target in the xy-plane.

If the vector which connects the center of object 1 and object 2 is given by

$$\vec{p} = (p_x, p_y, p_z) \tag{4}$$

Where $\vec{p}$ is a vector composed of any real numbers. Then a given vector which is perpendicular can be given by

$$\vec{q} = (p_z, 0, -p_x) \tag{5}$$

Since it is known that the dot product of two orthogonal vectors is zero then test to see if

$$\vec{p} \cdot \vec{q} = 0 \tag{6}$$

$$\vec{p} \cdot \vec{q} = (p_x, p_y, p_z) \cdot (p_z, 0, -p_x) \tag{7}$$

$$= p_x p_z + 0 - p_z p_x = 0 \tag{8}$$

Therefore the new direction vectors of the object 1 ($\vec{\rho_i}$) and object 2 ($\vec{\rho_j}$) are $\vec{\rho_i} = \vec{p}$ and $\vec{\rho_j} = \vec{-p}$ respectively. This allowed them to move opposite to each other for their next position update then test to see if they are no longer at risk of colliding; if they are at a 'safe' distance they will revert back to there original state and continue along their original path (navigating towards the center or individual target).

VI. Introduced Position States

In real world applications such as a number of drones moving around a set space or a network of roads, collisions would result in damage to the machinery and deadlock scenarios. Position states were incorporated as a form a traffic control to approach the issue of collision as adapted from Marcolino et al [3]. This assumed that each object could communicate with each other and gave updates of their position on a local scale. Each object examined its next position update at each iteration; the objects examined other nearby objects and tested if the next position update caused a collision. If a collision was imminent the at risk objects communicated with each other and test their position states; if one object was in the 'waiting' state then the other continued moving. If neither object was currently 'waiting' then the original object remained in the 'waiting' state until it was safe to move. Any head on collisions caused the objects to be stuck in a deadlock scenario which was accounted for through the use of the orthogonal vectors which connected the centers of each object as described in section V. This implementation is an example of local path planning.

- "**Normal**" - If object is in the exit regions then it will move vertically away from the centre target until it reaches the y limit of its designated target in which it will direct itself towards this target.
- "**Return**" - Once the object has reaches its target then the return state will cause the object to move directly towards the centre target.
- "**Waiting**" - If the new position vector of the object will result in collision with another separate object then this object will remain stationary.

As well as position states each object was assigned a 'target condition', these correspond to a 'hit' and 'not hit' which represented an object which had reached its' individually assigned target or not reached respectively.

The position vectors of all the objects were initialised in the 'exit' regions (top and bottom pyramids) with position state 'normal', current score of 0 and a target condition of 'not hit'.

An object which had a 'target condition' of 'hit' and then returned to the main centre target gained an individual score of +1 and a combined score of +1. The combined score is the combined score of all objects in that run. The introduction of individual target is a form a foraging [11]. Foraging was effective at measuring the individual performance of each object in the system. If the overall score of the system was measured there would be no method to consider the performance variation of individual objects. One approach could increase the performance for some objects but decrease the performance of others. The introduction of the foraging technique helped identify a technique that increased the performance across all objects in the system.

At each iteration the direction vector was normalised and the position vector of the object was updated given

$$\vec{p_i} = \vec{p_i} + \frac{\vec{\rho_i}}{m_o}\delta x \tag{9}$$

$$\delta x = \begin{cases} 0 & state = \text{‘}waiting' \\ 0.1 & otherwise \end{cases}$$

where $\vec{p}$ is the position vector of the object, $\vec{\rho}$ is the direction vector, m is the mass. δx was introduced to cause the position vector to be updated in small increments. The direction vectors were normalised at each iteration to represent objects that moved at constant speed.

VII. Reducing convergence issue

A common issue found with the above model occurred when a large number of objects attempted to enter the centre target simultaneously. The objects collided and were unable to move to a different point which caused congestion. This is a common issue with multi-robot systems. To avoid this conflict an upper limit and lower limit were introduced in all entrance regions. The concept for this algorithm was adapted from the EE algorithm proposed in Marcolino et al [3]. Whilst an object was in this region; if the number of objects below the entry region that were directed towards centre target abided $nV_o > \alpha\text{‘}V_T$ where V_T represents the total volume of the entrance pyramid, nV_o is the volume of objects in the region and α is a predetermined ratio then the object was in the ‘waiting’ state. If the constraint was in accordance to $nV_o < \alpha V_T$ then every object that lay within this band would continue to move forward. This adaptation required that each object in the band knew the values of V_T at each iteration. This was through communication with the central processing unit (center target), which could monitor the number of objects below the band but had not yet reached the center target, and the objects within the band.

Defining a potential ‘waiting’ region had its drawbacks, if the band was too small then a large number of objects could move towards the centre and collide which caused them to become ‘stuck’ in the waiting state. If the band was too large then the model became inefficient as numerous objects would be in the ‘waiting’ state simultaneously. An ϵ-greedy strategy was implemented to select the optimal size. A range of values for the lower limit were defined from $\{0.1k | k \in \{\frac{l_s}{8}, ..., \frac{l_s}{4}\}\}$.

Each of these values had assigned weights which were initialised to $1/n_l$ where n_l is the number of tested values.

For each run the ϵ-greedy strategy selected the best lower limit so far with probability ρ_l or any value at random with probability $1 - \rho_l$. Any tie was broken through random selection. After each run the weights were updated. Two different versions of weight updates were explored, the first involved taking the mean score ($\bar{c}$) to date using each run and if the current score (c) for that run satisfied $(\bar{c} - \frac{(1-0.95)}{2}\bar{c}) < c < (\bar{c} + \frac{(1-0.95)}{2}\bar{c})$ then the weights (Q) were updated with

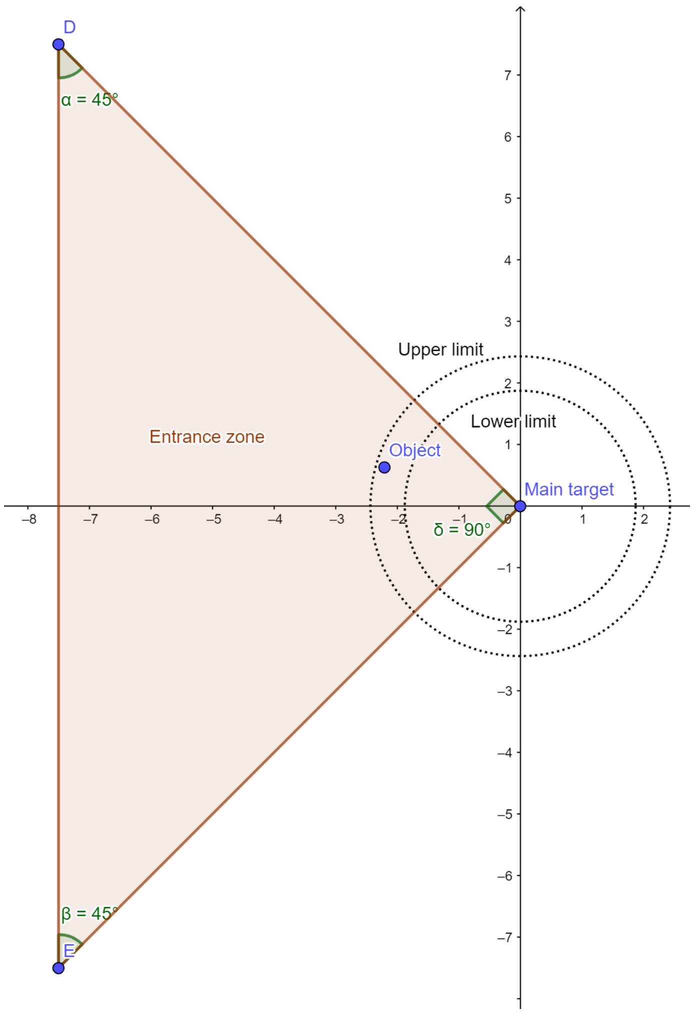


Figure 3: 2-D demonstration of the convergence band surrounding the entrance zones. The outer ring shows the upper limit and the inner rings demonstrates the lower limit that was varied across a range. These bands form spherical sectors across the entrance zones.

a reward R=0 since there was no significant change in the current score in comparison to the mean score. If c satisfied the condition $(\bar{c} + \frac{(1-0.95)}{2}\bar{c}) < c$ then this corresponded to a reward of $R = 1$ likewise if c satisfied the condition $c < (\bar{c} + (1 - \frac{(1-0.95)}{2}\bar{c})$ then this corresponded to a reward of R=-1 since it was less beneficial to use the corresponding lower limit value. After each weight was updated, the weights were normalised to 1.

$$R = \begin{cases} 0 & (\bar{c} - \frac{(1-0.95)}{2}\bar{c}) < c < (\bar{c} + \frac{(1-0.95)}{2}\bar{c}) \\ 1 & (\bar{c} + \frac{(1-0.95)}{2}\bar{c}) < c \\ -1 & c < (\bar{c} + (\frac{(1-0.95)}{2}\bar{c}) \end{cases}$$

The issue of definite rewards (-1, 1, 0) is that a reward of 1 will be the same for a lower limit value that produces a c score significantly higher than $\bar{c}$ and a lower limit value that produces a c score that is only marginally higher. To account for this issue a second model was devised which calculated the reward based on the percentage increase/decrease between c and $\bar{c}$.

$$R = \frac{c - \bar{c}}{\bar{c}} \tag{10}$$

$$Q(a) = Q(a) + R \tag{11}$$

$$Q = \begin{cases} \frac{Q}{|Q|} & min(Q) > 0 \\ \frac{Q - min(Q)}{|(Q - min(Q))|} & min(Q) < 0 \end{cases}$$

These form of weight updates required a baseline mean score ($\bar{c_0}$) prior to any weight updates. To determine $\bar{c_0}$ the first run consisted of selecting each value for the lower limit in sequence and tested this lower limit vale over n_i iterations and recorded the overall score for each of the objects. $\bar{c_0}$ was taken to be the mean of these values. After each run the mean score was updated and this updated mean score became $\bar{c}$ in the next run. A demonstration of the proposed algorithm is shown in algorithm 1.

Algorithm 1 Calculate weighting of lower limit values

```
1: procedure WEIGHT UPDATE
2:     "ASSIGN EMPTY LIST to list"
3:     Q ← [$1/n_l, ..., 1/n_l$] "Initialise weights to $\frac{1}{n}$
4:     $\bar{c} \leftarrow 0$
5:     for run in reruns do $c = 0$
6:         if rerun == 1 then
7:             for "Action in lower limit do
8:                 for $i$ in objects do
9:                     if Object has completed path then
10:                        $c \leftarrow c + 1$.
11:                        $c_i \leftarrow c_i + 1$.
12:                list ← $c$.
13:                $\bar{c} \leftarrow$ list/mean(list)
14:        if rerun ≠ 1 then
15:            Chose action 'a' derived from Q($\epsilon$-greedy)
16:            for $i$ in objects do
17:                if Object has completed path then
18:                    $c \leftarrow c + 1$.
19:                    $c_i \leftarrow c_i + 1$.
20:            list ← $c$.
21:            $\bar{c} \leftarrow$ list/mean(list)
22:            Store $c_i$
23:        $Q(a) \leftarrow Q(a) + \frac{c - \bar{c}}{\bar{c}}$
```

Figure 4 shows the weight updates after each run of 5×10^5 iterations. These results demonstrated a proof of concept that over a large number of iterations the weights will converge to 1 for the optimal value of the upper limit. Results for 50 objects showed much larger deviation than the previous results. This demonstrated that the upper limit had a much larger influence which was expected as more objects were introduced into the system. The algorithm produced more varied results compared to previous tests but still converged to an optimal value. Due to large computation times this could only be explored over 20 runs of 5×10^5 iterations for three different values for the lower limit. In figure 4 each line represents the weighting at the end of each run, as the number of runs increased the weighting tended towards one for a specific value whilst the weighting of the remaining values tended towards zero. These results were also only explored with $r_o = 0.1$, this algorithm was not performed on the second radius size. The convergence was noticeable for 50 objects but the variation may suggest that further additions need to be made to account for more objects in the system. These results for 50 objects suggest that the ratio values chosen $\alpha = 0.1$ was not optimised; objects 20-40 showed clear transitional convergence across the repeated runs which suggests that a better optimised α has the potential to show these results for 50 objects.

One approach that was considered but not implemented was to base the constraint $n_l < \alpha$ where n_l is the number of objects below the lower limit and α is the chosen limit that can be below the lower limit. This was initially rejected since the volume that α is not consistent across the varied ranges of object radius.

VIII. RESULTS

Each of the objects have a unique 'current score' attached to them; a 'current score' of 1 represents an object which has reached its own designated target and returned to the centre point once. This was included to understand the impact of the waiting state and see the influence of the proposed congestion/collision avoidance algorithms on the individual objects.

If the simulation were to run as expected the objects would remain in the 'waiting' state for a small number of iterations whilst the opposing object moved away; the time spent in the 'waiting' state would become negligible after a larger number of iterations. The expectation is that the mean score of the individual objects will be statistically similar across 'n' repeated runs however the standard error may vary as the path they take to reach the individual target alters. This was visualised in figure 5. Each sample taken for each of the mean score values were expected to follow a Gaussian distribution.

The path each object took to reach the individual target and return to the center was random for each set in a confined space, provided the influence of the 'waiting' was minimal ($\frac{n_w}{n_t} \approx 0$) the sample could be approximated as i.i.d (independently and identically distributed) where n_w is the number of iterations spent in the waiting state and n_t is the total number of iterations. This means that the law of large numbers (LLN) could be considered which states that for a sample $\bar{c_1}, \bar{c_2}, ..., \bar{c_n}$ that is i.i.d where $\bar{c_1}$ has a finite expectation $[\mathbb{E}[\bar{c_1}]] < \infty$ then the sample average is

$$\bar{c} = \frac{\bar{c_1} + \bar{c_2}, ..., \bar{c_n}}{n} \rightarrow \mathbb{E}[\bar{c_1}] \tag{12}$$

The sample average is expected to converge to its expected value in probability [12]. This means that as n becomes large the sample mean $\bar{c}$ will approach the expected value of the random variable $\mathbb{E}[\bar{c_1}]$. Considering an arbitrarily small value $\phi > 0$.

$$\mathbb{P}([\bar{c} - \mathbb{E}[\bar{c_1}]] > \phi) \rightarrow \ 0 \text{ as } n \rightarrow \infty \tag{13}$$

Figure 5 shows the mean value of each set of objects (20, 30, 40 and 50) over four different seeds for each simulation. These demonstrate the expected results where the mean scores of the objects are approximately similar (further analysis is shown in figure 6) and whilst the standard error bars differ in magnitude across the set of objects they are not excessive; an excessive error bar would be considered one which tended to zero which would indicate a deadlock scenario where one or more objects were unable to move.

These results were gained through a lower limit value of 1.2 and the parameters as shown in table I. Whilst these results demonstrate that no deadlock scenario they do not indicate whether a relationship exists between them.

To further explore if there is a relationship between the mean scores of the objects, building a one-way ANOVA model generated the associated P-value between the mean score of each object under the hypothesis on the 0.05 statistical significance level.

Hypothesis $\mathbf{H_0}$ (Null hypothesis)**:** No difference amongst the mean score of each object.

Hypothesis $\mathbf{H_1}$ (Alternate hypothesis)**:** The mean score of at least one object differs significantly amongst the mean of the other objects.

The null hypothesis H_0 was evaluated at the 0.05 significance level which meant that H_0 was accepted if $p > 0.05$ and H_0 was rejected in favour of the alternate hypothesis H_1 is $p < 0.05$.

One way ANOVA is more suited to determine if there is a difference amongst the mean scores of the objects since it

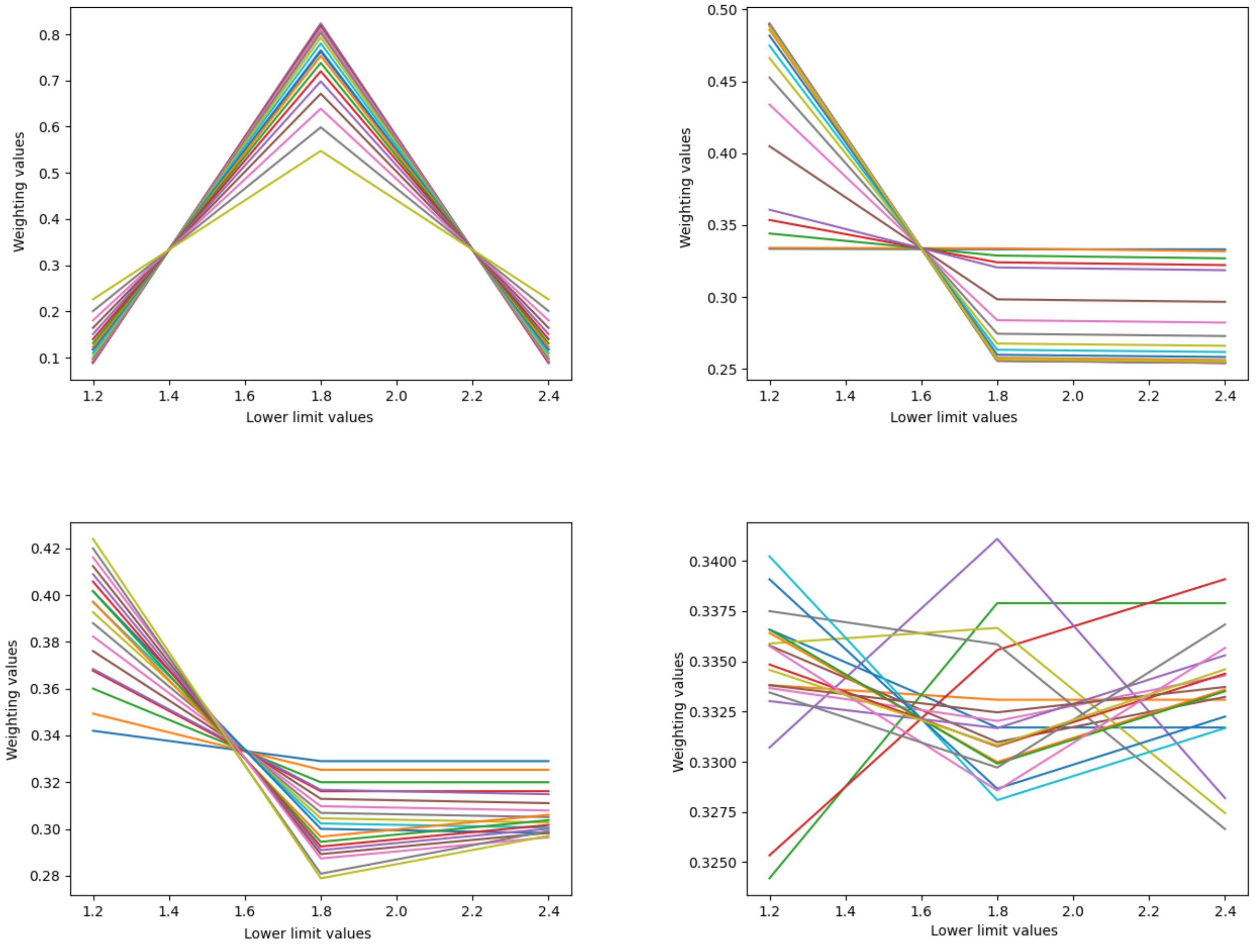


Figure 4: Weighting shown over 20 runs of 5×10^5 iterations for lower limit values of 1.2, 1.8 and 2.4 for 20, 30, 40 and 50 objects respectively.

utilises the F-test for statistical significance. This test allows for the comparison of multiple mean values simultaneously. The F-test calculates the variance of each of the objects mean score from the overall group variance. This F-value was analysed to obtain a probability (P-values) to determine whether the difference between the groups were statistically significant[13].

The data used to perform this analysis was raw data; no pre-processing techniques were performed prior implementing one-way ANOVA. This was to ensure a more precise analysis, there was no reason to assume any extreme mean scores were definite outliers and could have been informative to the simulation (potential signs of large waiting times for example). If significant outliers were present then this could have been a sign of the impact. To visualise the residuals of all the mean scores, the residuals were plotted alongside the fitted values (mean scores) which are shown in figure 6. A full set of values over repeated runs are shown in table II.

O_r	seed	Number of agents			
		20	30	40	50
0.1	10	0.109	0.463	0.00126 *	0.0826
	20	0.381	0.0521	0.00176 *	0.278
	30	0.465	0.587	0.0106 *	0.000751 *
	40	0.111	0.17	0.764	0.00212 *
	ρ_{volume}	8.4e-5	1.3e-4	1.7e-4	2.1e-4
0.2	10	0.2319	1.77e-05 *	0.00122 *	
	20	0.1615	4.38e-05 *	0.000113 *	
	30	0.02174 *	0.758	0.279	
	40	0.3291	7.18e-06 *	0.00897 *	
	ρ_{volume}	6.7e-4	1.0e-3	1.3e-3	1.7e-3
	ρ_{object}	0.02	0.03	0.04	0.05

Table II: P-values generated through one way ANOVA alongside their associated seeds in python for objects 20, 30, 40 and 50 objects respectively. P-values shown with * indicate statistical significance.

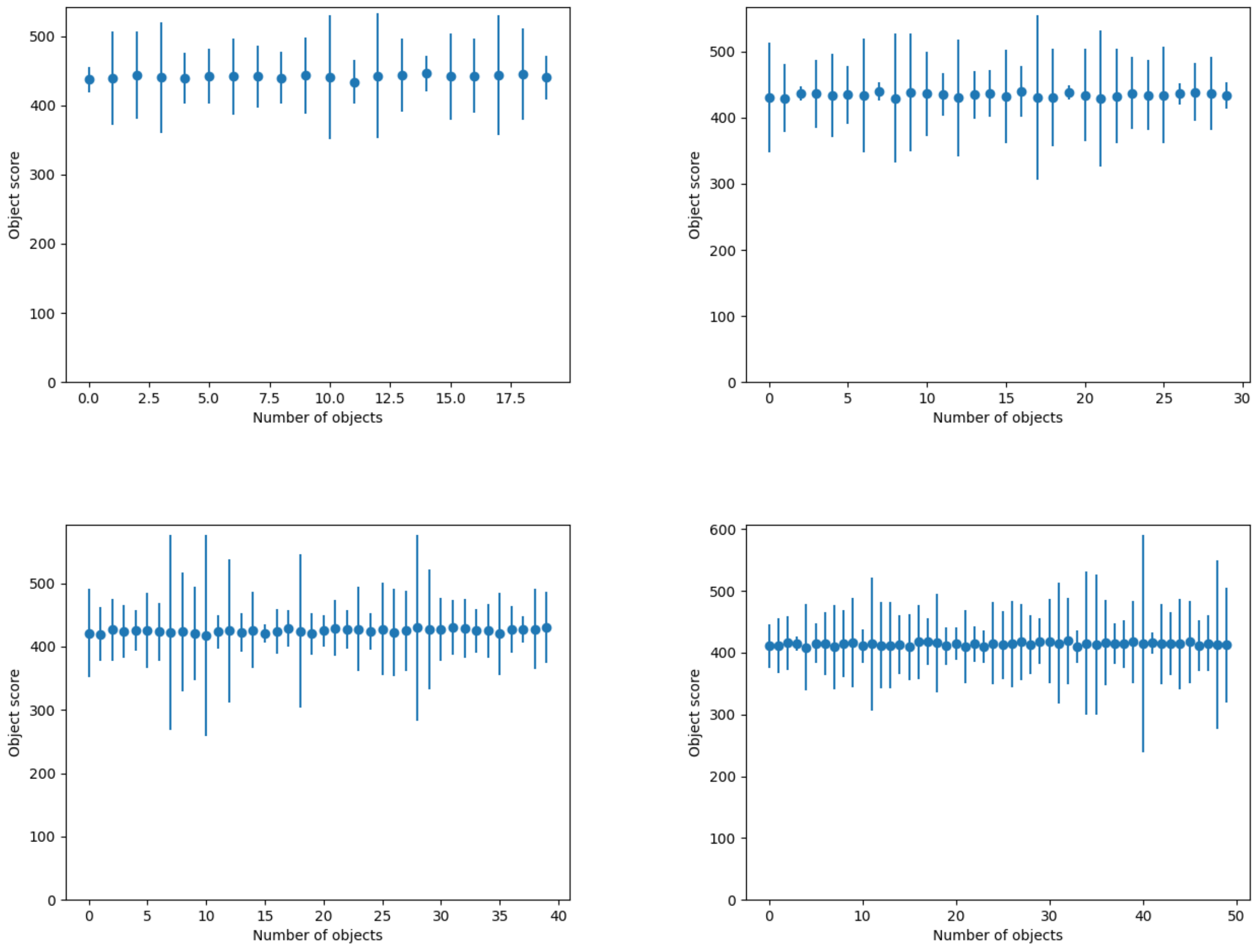


Figure 5: Mean score of individual objects across 10 runs of 1×10^6 iterations alongside the standard error tested for 20, 30, 40 and 50 objects with a pre-defined lower limit of 1.2. These demonstrate no deadlock scenarios since no scores tend towards zero.

For simplicity, the P-values in table II will be referred to using the notation $N(r_0 = A)$ where N is the number of objects used in each test and A object radius.

Table II demonstrate that for object numbers 20, 30 and 50 the P-value is in accordance to $P > 0.05$ meaning that H_0 is accepted and the mean scores of each object do not differ significantly in comparison. The P-values for 40 objects does not support H_0, the associated P-values of 0.00126 and 0.00176 indicate significant statistical difference between the mean score of the objects. The statistical difference between the mean scores found with 40 objects could be due to anomalies; the associated residual graph for $Pr(> F) = 0.00126$ is shown in figure 6. This graph identifies three outliers with a large residual (points 11, 20, 36) which would have a large influence on the final result. A larger sample size would need to be tested to explore whether these points are true outliers.

The simulations run for $20(r_o = 0.1)$ and $30(r_o = 0.1)$ demonstrated clear statistical similarity; $Pr(> F) > 0.05$ was consistent across four different seeds which supported H_0. Many results for 40 agents and 50 agents did not support H_0 but rejected it in favour of H_1. This was intuitive since the larger the number of objects that navigate a confined spaced the more likely they were to collide and therefore change their position state to ‘waiting’.

Following these results, the radius of each object was doubled to consider whether it was the increase in objects in the system or the volume of objects in the system that had this influence.

By doubling the radius of each object (initial radius was 0.1 and the new radius if 0.2) then these contributed to each object occupying eight times the original volume $v \propto r^3$ since each object was considered as a sphere of radius r.

A second set of P-values were recorded for the new object size. Considering the simulation results for $20(r_0 = 0.2)$; the volume that these objects occupy combined are four times that of $40(r_0 = 0.1)$. Despite $20(r_0 = 0.2)$ occupying more volume than $40(r_0 = 0.1)$ the results show that 3/4 samples supported H_0

Figure 7 demonstrates the mean score across ten runs of 1×10^6 iterations for each number of objects and their corresponding radius. For objects $N(r_o = 0.1)$ the mean score is consistent across 20-50 objects with a gradual drop throughout. Objects $N(r_0 = 0.2)$ suffer a substantial drop in mean score between 30-40 objects. This drop is approximately

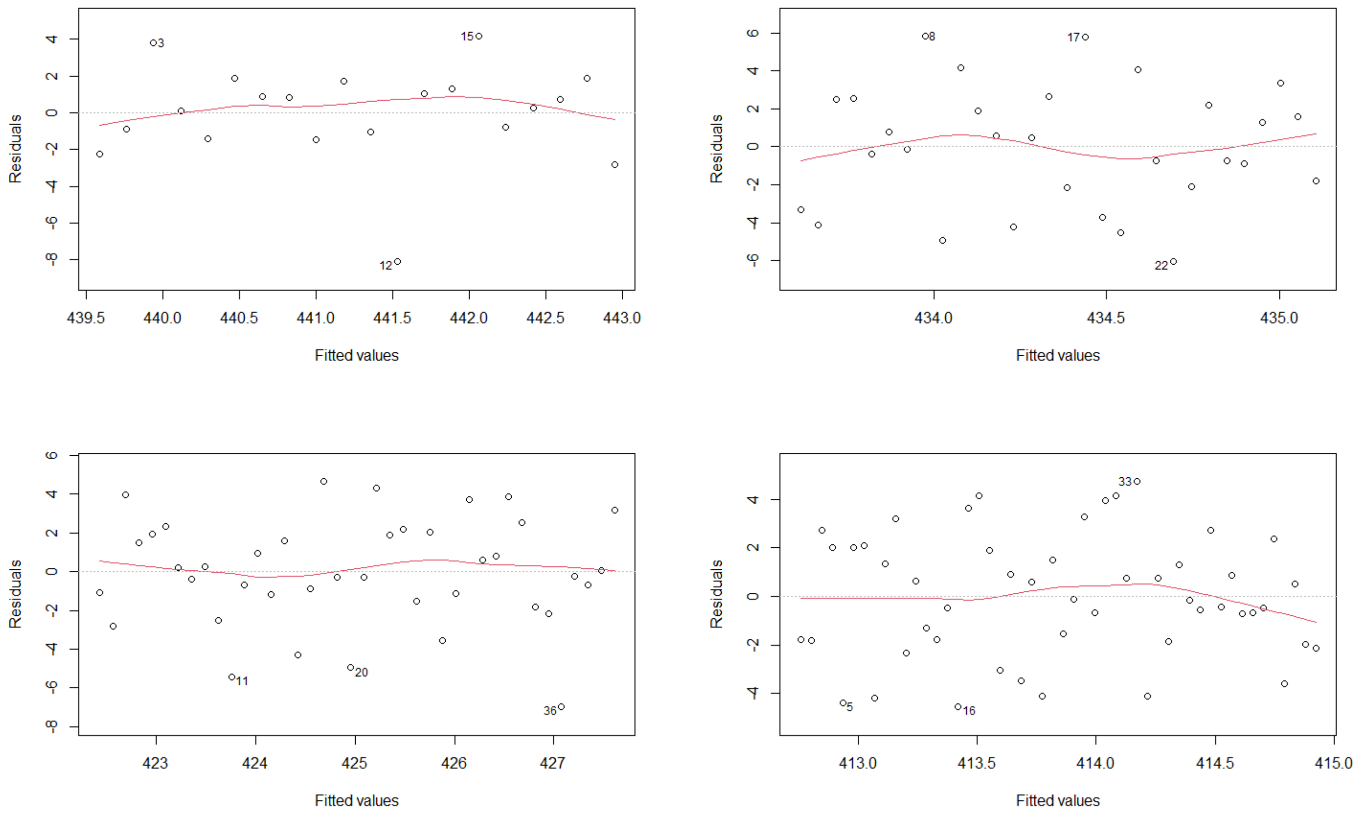


Figure 6: Mean score of individual objects across 10 runs of 1×10^6 iterations alongside the standard error tested for 20, 30, 40 and 50 objects with a pre-defined lower limit of 1.2 using seed 10. The highlighted points identify outliers within the residuals.

nine times the drop seen with objects of $N(r_o = 0.1)$. There is no observable maxima for $N(r_o = 0.1)$ further values of data would need to be taken to identify this point however there is a noticeable maxima for $30(r_0 = 0.2)$.

IX. Conclusion

Several different algorithms were presented in order to account for collision avoidance and target congestion on a foraging inspired system.

The first algorithm proposed was a collision avoidance system utilising a 'traffic light' approach through the introduction of position states. This method was a simple but effective approach at avoiding collision. Further expansion of this method would be the inclusion of a 'repulsive force' which occurs when the objects are close and a collision is at risk.

The second implementation was including entrance and exit zones. The defined area was composed of two exit zones and four entrance zones of equal sizes. This approach was effective at reducing congestion; no results were able to be obtained prior to implementing this algorithm due to full convergence of the objects (a complete deadlock scenario) however, implementing this algorithm the convergence issues greatly reduced to ≈ 5 objects reaching a deadlock scenario over the set runs testing with $N = 20$.

The concept behind introducing the convergence band within the entrance zones was proven to work conceptually. This introduction removed any remaining congestion issues however it has not been tested to best optimise its performance. Time constraints and computations times only allowed for a small number of iterations and values to be examined. Further work on this implementation would include testing a wider range of values for the lower limit values alongside α to best optimise the algorithm.

The combination of these algorithms proved to be an effective multi-agents system. No convergence issues occurred over repeated runs using four different seeds testing various values for N and r_o.

The results showed that the scores of the objects individually were more largely influenced by the number of objects within the system as opposed to the size of the objects themselves. These results have shown a cutoff point for $30(r_0 = 0.2)$ in terms of overall score. After increasing the number of objects further resulted in a decrease in performance. This has shown that the system can be tested and optimised for peak performance with equally performing objects.

Further improvements could involve predictive behaviour for the waiting state, in this current example it is assumed that each of the robots can communicate immediately to give updates on their positions, whilst the communication for collision avoidance only needs to be on a relatively short scale (several times the radius of the object); the implementation of the upper limit for the congestion issue requires a much larger communication radius. Whilst this is feasible in simulations in real world applications there is likely to be a time delay between one robot sending its position update and the others to receive and process it. Similarly if there is a malfunction with one robot and it is unable to send a transmission to the others the robots would be unable to know its location which could lead to a collision. A more robust model could involve simulating sensors over a predefined distance to use the current path of the robot to simulate a predicted trajectory and wait if there is a likely collision.

References

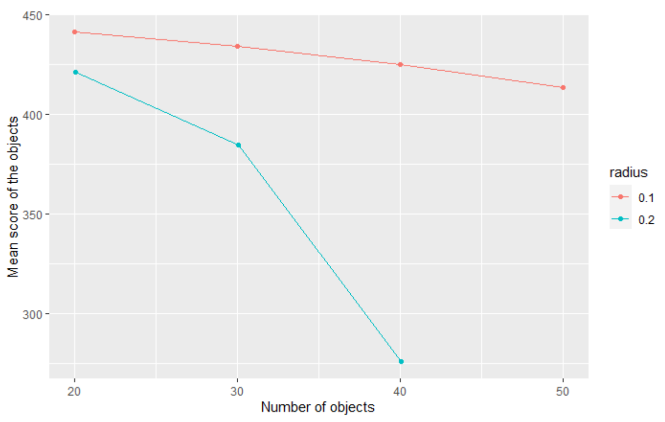
[1] R. Brooks, P. Maes, M. Mataric, and G. More, "Lunar base construction robots," in *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, 1990, pp. 389–392 vol.1.

[2] P. A. Abrams, "Foraging time optimization and interactions in food webs," *The American Naturalist*, vol. 124, no. 1, pp. 80–96, 1984.

[3] L. S. Marcolino, Y. T. dos Passos, Á. A. F. de Souza, A. dos Santos Rodrigues, and L. Chaimowicz, "Avoiding target congestion on the navigation of robotic swarms," *Autonomous Robots*, vol. 41, no. 6, pp. 1297–1320, 2017.

[4] M. Nakamura, "Regulation mechanism of task-allocation and formation mechanism of ants' distribution pattern in collective behavior of ant colony models," in *Soft Computing as Transdisciplinary Science and Technology*. Springer, 2005, pp. 937–948.

[5] A. Drogoul, "From tom thumb to the dockers: Some experiments with foraging robots," 04 1999.



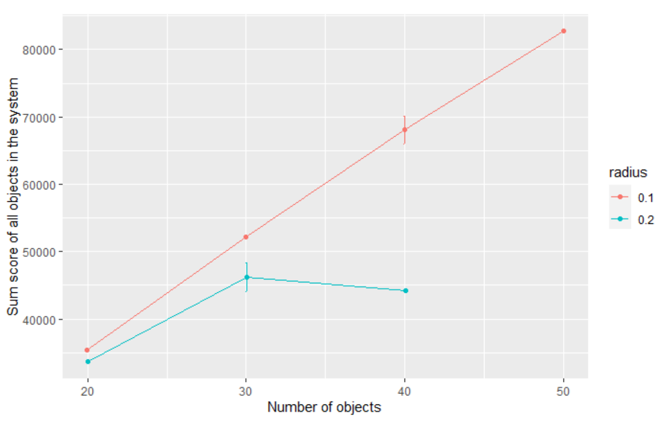


Figure 7: Mean score of the objects alongside the sum of all objects within the system.

[6] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[7] J. T. Schwartz, M. Sharir, and J. E. Hopcroft, *Planning, geometry, and complexity of robot motion*. Intellect Books, 1987, vol. 4.

[8] J. Van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1928–1935.

[9] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[10] N. R. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal, “Two foraging algorithms for robot swarms using only local communication,” in *2010 IEEE International Conference on Robotics and Biomimetics*. IEEE, 2010, pp. 123–130.

[11] O. Zedadra, H. Seridi, N. Jouandeau, and G. Fortino, “A cooperative switching algorithm for multi-agent foraging,” *Engineering Applications of Artificial Intelligence*, vol. 50, pp. 302–319, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197616000294

[12] I. D. Dinov, N. Christou, and R. Gould, “Law of large numbers: The theory, applications and technology-based education,” *Journal of Statistics Education*, vol. 17, no. 1, 2009.

[13] T. K. Kim, “Understanding one-way anova using conceptual figures,” *Korean journal of anesthesiology*, vol. 70, no. 1, p. 22, 2017.