Comparison of the β-taxis and ω-taxis Swarm Robotics Algorithms

Y6380396

***Abstract*—The aim of this paper is to compare the β-taxis and ω-taxis swarm robotics algorithms. First, swarm robotics and the problem of taxis are introduced, then a short review of previous work in this area is undertaken. Finally, the two algorithms are implemented in simulation and their performance compared.**

I. Introduction

In recent years, many advances have been made in the field of swarm robotics. These advances have come in many different forms, the most important of which are the physical designs of the robots used in swarm robotics and the algorithms which govern the behaviour of these robots.

There are many different tasks that are impossible or difficult for humans to perform, tasks which a robot could potentially perform safely. Humans often wish to work in environments which are unsafe or inhospitable or perform tasks of which our bodies are not capable without assistance. Of course, humans have built robots both large and small to perform many of these tasks for us, however, these robots usually work alone or are controlled directly by a human operator from a safe distance.

What if we required our robots to act of their own accord, in such a way that they can adapt to new challenges and obstacles? A single robot alone cannot accomplish this, it can change its strategy but it cannot adapt its shape and function as readily as, say, 20 smaller robots working together. Indeed, using a collection of robots, known as a swarm, to solve these complex problems is the essence of swarm robotics.

However, using smaller robots presents a new set of issues. In order to successfully miniaturise a robot it must be made simpler architecturally, and, as a result of this, the set of tasks that it can perform is reduced. This is, of course, countered by the number of robots in the swarm, any complexity lost from simplifying the robots is effectively transferred to the swarm as a whole. However, any code which controls an individual robot can only do just that, control an individual robot. It cannot issue orders to the entire swarm as a whole as communication of this sort is extremely resource intensive. Indeed, if global communication throughout the swarm were possible then swarm robotics would be a trivial field.

Knowing the limitations of the robots used in swarm robotics it is possible to design controllers which take these limitations into account. For example, any wireless communication performed between robots is usually only over a very short range and can only convey limited information, the broadcast of robot IDs, for example. Despite this it is possible to design algorithms which exploit high level behaviours that occur as a result of low level behaviours in each robot of the swarm, a process known as *emergence*. One such behaviour, *taxis*, is the purpose of the two algorithms which will be discussed at length in this paper.

Taxis is a simple problem, described as 'swarm motion towards a beacon' [1] and requires a swarm to move towards and optionally encapsulate (surround) a beacon. Though it is referred to as a beacon the target of the taxis can be anything that the robots can detect. It is important that the robots can detect the beacon from a considerable distance as if none of the robots can see the beacon then it is not possible to determine in which direction the swarm should move. However, it is insufficient to simply program each robot to move towards the beacon when it is detected as robots may not always be obscured by obstacles or other robots. Further, it may not be possible to know in which direction the beacon is situated, only that a robot has an unobstructed path to it in some arbitrary direction.

Taxis is not an unsolved problem, there are many algorithms designed to solve the problem, two of which will be discussed at length in this paper. The β-taxis algorithm and the ω-taxis algorithm are both adaptations of swarm clustering algorithms, originally designed to keep a swarm of robots aggregated.

II. Literature Review

Both the β-taxis and ω-taxis algorithms were originally conceived as swarm aggregation algorithms, which were not designed to perform taxis. The two original (non-taxis) algorithms were based on an earlier algorithm known as the α-algorithm. This algorithm was designed for swarm aggregation, which is a feature required of taxis algorithms, but is not sufficient to perform taxis in itself. The α-algorithm is a good base from which to explore more advanced taxis algorithms, however, and is worth describing here.

Aggregation in a swarm is one of the most basic behaviours required in order for the swarm to exist, indeed if the robots cannot remain within a short distance of each other then the swarm breaks up and no useful action can be taken by individual robots. In order to perform aggregation robots must be able to somehow sense other robots, in most cases this is through some form of short range wireless communication which allows robots to broadcast an ID to be received by other robots. If robots are able to effectively *see* their neighbours then each robot can keep track of how many neighbours it has at any point in time. Of course a robots neighbours are only limited to those robots within wireless communication range, but this is to be used to our advantage.

Knowing that a robot can detect when it loses a neighbour, it is possible to have a robot act on this stimulus. It is usually the case that when a neighbour is lost the robot is travelling

away from the swarm centre, or *centroid*. If such a robot were to turn 180° it would be travelling back towards the swarm and would regain its lost neighbour. Upon detecting the neighbour again the robot can be set to a random heading in order to allow the swarm to perform a random walk.

However, if all robots were to immediately turn 180° upon losing a single neighbour then every robot in the swarm would constantly be within wireless communication range of all other robots, making the swarm extremely tightly aggregated and potentially trapping robots in the centre. In order to prevent this it is necessary to add a parameter, α, which governs the minimum number of neighbours a robot is willing to have. If the number of neighbours of an individual robot drops below this alpha value then the robot performs a 180° rotation moving it back towards the swarm. This enables the swarm to be more dispersed while keeping it sufficiently aggregated. This is the basic operation of the α-algorithm as described in [1].

There are some issues with this design, however, as robots are only capable of turning 180° in order to return to the swarm it is possible that in very rare fringe cases, a robot could miss the swarm and never return. This issue causes the swarm to occasionally lose a robot which, depending on the physical robot models, could be a costly error.

Additionally, adjusting the α value of the algorithm can drastically change its behaviour. Extremely low α values will cause the swarm to break up as each robot will only try to maintain very few neighbours [1]. It is possible for the swarm to split into smaller swarms which may be a desired behaviour, but it is not guaranteed that these swarms will ever re-aggregate. Adjusting the α value to be too high can cause the opposite problem, the swarm can remain too close together, having the entire swarm remain within 1 wireless sensor range would mean an extremely aggregated swarm which has a small coverage area.

III. Implementation

This section discusses the details of implementing the β-taxis and ω-taxis algorithms in the Pi-Swarm Simulator developed at the University of York. The simulator is written in Python and will be executed on Python 2.7.8 on Xubuntu 14.10. Python has been configured using virtualenv though this is transparent to the simulator. The Pi-Swarm Simulator accepts a class which specifies the behaviour of each individual robot in the swarm. The *drive()* function of this class is then executed for each robot during each simulation timestep. This function must instruct the robot how to behave during each timestep, and update the internal state of the robot in any way necessary.

The simulation environment is initialised to be equivalent to a 500cm by 500cm arena with the beacon 10cm from the bottom of the arena at the centre. The simulator handles the checking of a robots ability to detect the beacon which causes the boolean variable *illuminated* to be set for each individual robot. The robots themselves are based on the e-puck robots developed at the University of York and have two different sensors which are controlled by the simulator. The robots have a wireless sensor which can be used to communicate with other robots in a short range, set to 50cm for the purposes of this paper. Additionally, the robots have six infra-red sensors which are capable of detecting obstacles within 10cm, primarily other robots. These sensors are placed at intervals around the body of the robot and have two variables for detecting obstacles, *contactobs* and *contactdist*. *Contactobs* is set to true when a sensor detects any obstacle in its range and *contactdist* is set to the distance at which the object has been detected. It is important to note that *contactdist* is only reliable when *contactobs* is set to true, otherwise it is set to zero.

The algorithms are implemented using a finite state machine (FSM), this allows each robot to perform different actions based on which state it is currently in. This FSM implementation is easily accomplished using a series of *if statements* and a variable *state* to hold the current state of a robot. This functionality is built into the simulator but must be used by controller implementations in order to control a robot.

The β-taxis algorithm implementation has three states, *"forward"*, *"avoid"* and *"turning"*. All robots begin each timestep by updating their neighbours list. The forward state is the default state in which all robots begin. Any robot in the forward state will first execute its *driveforward()* function causing it to move forwards by some amount.

Robots in the forward state then determine if they need to change state. The specification of the β-taxis algorithm states that robots must perform coherence in two cases, if their loss of a neighbour causes that neighbour to have fewer than β neighbours or they lose a neighbour that is illuminated by the beacon. This is implemented through the use of a check for each of these conditions which causes a change of the robots state to turning with a bearing of 180°. If this check does not cause a state change then a further check is performed, to see if any of the front infra-red sensors are active. If any sensors are active then the robot enters the avoid state.

The last two states are more simple than the forward state. Robots in turning will simply check each time-step if they have turned the amount set in the heading that was set upon entering turning. Once this heading has been reached the robot will transition back into the forward state. Robots in the avoid state will turn away from any of the front sensors that are active. This is accomplished by checking for *contactobs* on the front four sensors. As the left sensors are checked first there is a slight bias here to cause the robots to turn left if all front sensors are active. A possible improvement here would be to test for the presence of an obstacle on all front sensors first and turn in a random direction if this is the case. However, this change is unnecessary as the bias does not affect the operation of the algorithm. A robot in state avoid which cannot detect any front obstacles will transition back into state forward.

A robot in any state will finally update its previous neighbours list *prevneighbours*, which is used to determine lost neighbours. This functionality is repeated for each time-step for all robots in the simulation. A point of note regarding the functionality of the β-taxis algorithm is the order in which the checks are performed for the transitions from forward to avoid and turning. This order has not been modified from the basic code provided as the provided code performed the β-algorithm correctly.

Taxis is an emergent property of this implementation due to the two different conditions checked for the transition

from forward to turning. Robots will turn back immediately if they lose a neighbour which is illuminated, this causes robots which are generally travelling away from the beacon to turn towards the beacon more readily. If a robot loses a neighbour which is not illuminated then this action will not be taken immediately, only if a lost neighbour has fewer than β neighbours remaining. This causes robots which are illuminated, which will often be travelling towards the beacon, to turn back towards the swarm less often than those which are not illuminated, which causes a net movement of the swarm towards the beacon.

The implementation of the ω-taxis algorithm is largely similar to that of the β-taxis algorithm. The ω-taxis algorithm utilises the same three states as β-taxis but has state transitions on different conditions. Upon robot instantiation, the algorithm initialises the *timer* variable for each robot to zero. Each robot also keeps a variable textitsrange which is initialised to half of the full infra-red sensor range value.

During each time-step a robot will first increase the value of its *timer* by one. Robots will then perform an action specific to their state. Robots in the forward state will execute the *driveforward()* function to move forward by some amount and then check for state transition conditions.

The first of these checks determines if the front sensors can detect an obstacle and if so moves into the avoid state. However, unlike the method used in the β-taxis algorithm it is not possible to simply check for *contactobs* on all front sensors. Because the specification of the ω-taxis algorithm requires that illuminated robots have a larger avoidance radius than those that are not illuminated, it is necessary to perform separate checks here based on current illumination status. Robots that are illuminated will detect obstacles at the full infra-red sensor range, whereas robots that are not illuminated will detect obstacles at range *srange*, which is always initialised to half of the full infra-red sensor range. This functionality has been abstracted into the function *is_front_obstacle_detected()* for ease of use.

If the previous check does not pass, a robot will then check if it's *timer* has exceeded *omega*. If this is the case then the robot will enter state turning with a heading towards a position computed from the mean position of all other robots. This position, the centroid of the swarm, is computed using information unavailable to the robots locally, including the exact positions of all other robots. This is necessary because the simulation robots do not have a long range infra-red sensor which would be used to locate the centroid of the swarm in reality. Though this strategy violates the swarm algorithm paradigm forbidding the use of global information, it is necessary due to the lack of this type of sensor.

Robots in the avoid state behave almost identically to those in the β-taxis algorithm, they turn left or right to avoid other robots detected by the front sensors. It is important to note that this detection accounts for the *illuminated* status of the robot to enable the avoid state to account for the change in range of infra-red detection. If a robot in the avoid state does not detect an obstacle on the front sensors then it transitions to state forward and resets its *omega* value to zero.

The final state, turning, again behaves almost identically to the same state in the β-taxis algorithm with one key difference. A robots that transitions from state turning to state forward will reset its *omega* value to zero. The manipulations of the *omega* value in this algorithm cause robots that have not interacted with another robot in a time period specified by *omega* to attempt to turn towards the centroid of the swarm. This is the basic operation of the vanilla ω algorithm without taxis. Taxis is an emergent property of this implementation due to the varying infra-red sensor range based upon a robots *illuminated* state. Robots that are illuminated will avoid robots at twice the distance of non-illuminated robots, when this occurs, the non-illuminated robot is not affected. These types of collision are most likely to occur when the non-illuminated robot is travelling toward the beacon and the illuminated robot travelling away from it. The illuminated robot will turn to avoid its non-illuminated counterpart causing it to face the beacon, this causes the swarm to have a net movement towards the beacon, as both robots in this interaction are now travelling towards it.

IV. Analysis

The aim of this section is to analyse and compare how the β-taxis and ω-taxis algorithms perform in simulation through the use of various statistical tests. As an initial investigation it was necessary to determine the optimum parameters for both algorithms. In order to determine this each algorithm was executed in simulation using 20 robots and with all other parameters set to defaults. The *beta* and *omega* parameters were varied across a range in order to determine their optimum values and for each value of these parameters four individual simulations were carried out using a different random seed. The results for these experiments can be found in the *"analysis_scripts/figures/"* directory.

For the β-taxis algorithm it is clear that taxis is an emergent property of the algorithm for values of *beta* below 10. A value of *beta=2* provides the best solution, which reaches the beacon faster than any other value, this value was chosen as the optimum *beta* parameter. For the ω-taxis algorithm the results are less pronounced. For values of *omega* between 30 and 40 taxis is performed with other values seeming to initially perform taxis and then travel away from the beacon. *Omega = 35* was chosen as the optimum value as it reaches the beacon faster than other values.

Having determined the optimum parameters for both algorithms it is necessary to formulate a hypothesis which this paper intends to verify or refute. The chosen hypothesis is that *"The ω-taxis algorithm performs more efficient swarm taxis than the β-taxis algorithm"*. In order to measure how *efficient* a particular algorithm is at performing taxis the following metrics will be employed: time to reach beacon, mean distance from swarm centroid and number of 'lost' robots.

These metrics were considered due to their ability to distinguish an algorithm that moves the swarm to the beacon in the fastest time possible while keeping the swarm aggregated. The time factor of this measurement is accounted for by measuring the time taken for robots to reach the beacon. By measuring the mean distance from the centroid of all robots in the swarm it is possible to determine how well aggregated the swarm is, a high value for this metric indicates a swarm that is sparse, in which robots cover more area as they perform taxis,

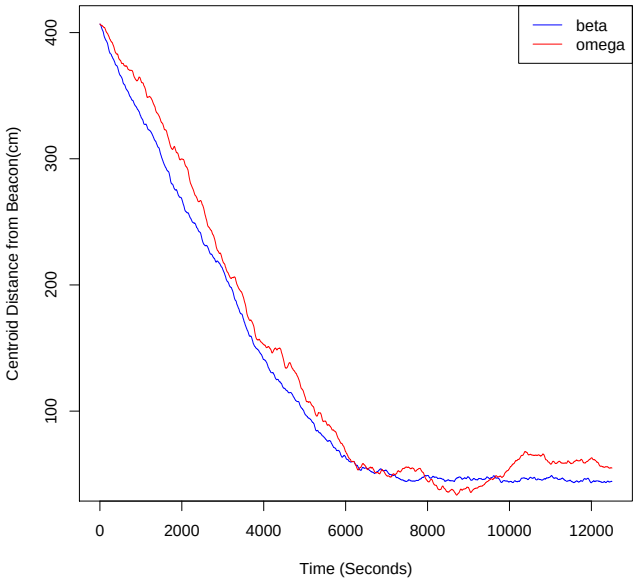

Fig. 1. Distance from Beacon against Time for *beta=2* and *omega=35*

while a low value indicates a swarm is compact, covering less ground but remaining more aggregated. Finally, measurement of the number of lost robots, where a lost robot is any robot without another within wireless sensor range, allows for further determination of the aggregation of the swarm.

The data used for the comparisons of the algorithms will be generated over 12 individual simulations for each algorithm with a different random seed for each. The means of the data across all simulations will be compiled and used per algorithm and metric.

Figure 1 plots how long each algorithm takes to reach the beacon. There is very little difference between the two algorithms with regard to this metric, they both reach the beacon at approximately the same time. The β-taxis algorithm does remain closer to the beacon after taxis has been performed but the ω-taxis algorithm obtains the closest distance to the beacon. This metric does not prove that there is a significant difference between the algorithms, however it has not been measured in a particularly reliable way. It would have possibly been more correct to use a statistical test to determine significance, but as the data is time based there are few tests which are appropriate. It is simple enough to inspect the graph and see that these algorithms differ very little when measured in this way.

In order to determine which algorithm performs best with regards to swarm aggregation the measurements of the mean distance from the swarm centroid must be compared. Unlike the previous metric, this data is not time based, as the aggregation of the swarm should remain constant throughout the experiment. In order to measure this data the Vargha-Delaney A measure will be used. This measure determines if one data set is significantly larger than the other. The result of this test when used to compare the mean distance from the swarm centroid of both data sets is 0.8958964, though there is a difference in swarm aggregation between the two algorithms it is not possible to say with 95% confidence that is the case.

Upon inspection of the data gathered regarding the number of lost robots, it transpired that neither algorithm ever lost any robots. This metric is therefore unable to distinguish between either of the algorithms and does not indicate that either performs more efficient taxis than the other. It is worth noting, however, that the design of the ω-taxis algorithm prevents it from ever losing robots, as all robots can determine the swarm centroid no matter their location relative to it. This would suggest that the β-taxis algorithm is the superior choice here, but no valid data can be collected to prove this for these implementations.

Using the three metrics chosen it is not possible to determine that "The ω-taxis algorithm performs more efficient swarm taxis than the β-taxis algorithm", rather, that the null hypothesis must be assumed, "the ω-taxis algorithm and the β-taxis algorithm perform taxis at the same efficiency." However, it must be noted that very few, only twelve, simulations were performed for each algorithm due to time constraints, and the validity of these results should be questioned as a result.

V. The Reality Gap

The e-puck robots on which the simulator is based could be used to implement either of the algorithms described, but how well would either of them work? The β-taxis algorithm was implemented using only standard functions of the simulator, both the wireless sensor and infra-red sensors are used, both of which feature in the e-puck robots. It is important to note that the β-taxis algorithm requires the transmission of neighbour lists between robots, which is implemented in the simulator in a way not compatible with the physical robots. It would be necessary for each robot to listen for broadcasts of neighbour lists instead of querying each neighbour for the lists.

The ω-taxis algorithm is of further complication to port to real robots. Due to the way simulated robots detect the centroid of the swarm a long range infra-red sensor is required which the e-puck robots do not have. The addition of this sensor would be required in order to implement the ω-taxis algorithm successfully.

VI. Conclusion

In this paper, two swarm robotics algorithms have been implemented and tested against each other in order to determine which one is superior. Testing swarm robotics algorithms requires a vast amount of time and processing power due to the many simulation runs required in order to generate significant data. It would be wise for future research to focus on the improvement of simulation solutions available in the field of swarm robotics, as the current offerings can be difficult to use and consume large amounts of resources.

References

[1] J. D. Bjerknes, A. F. Winfield, and C. Melhuish, "An analysis of emergent taxis in a wireless connected swarm of mobile robots," in *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE.* IEEE, 2007, pp. 45–52.