UNIVERSITY OF LIVERPOOL

2017/18

Student Name: Cameron Guest

Student ID: 201120096

Project Title: COMP390
Simulating Robot Swarms: Collaborative Transport

Supervisor: Clare Dixon

DEPARTMENT OF COMPUTER SCIENCE

The University of Liverpool, Liverpool L69 3BX

CONTENTS

ABSTRACT

Swarm robotics is an approach used to coordinate large numbers of robots and manage their collective behaviours. A system using swarm robotics can be defined by the following characteristics: each robot is autonomous, robots only have local sensing and communication abilities, there is no access to a centralised controller or global knowledge and the robots act cooperatively to accomplish a task. These systems have beneficial traits such as flexibility, scalability and robustness. These traits make swarms ideal for working in hazardous environments performing tasks such as disaster rescue or environment monitoring.

The area which this project investigates is known as collaborative transport also sometimes known as foraging. This is a cooperative endeavour where an object must be found within some environment and transported to a goal location. The most common transportation algorithms involve pushing, pulling or caging an object and this paper analyzes three different methods which each use one of these methods. Additionally a simulation of the occlusion based transportation method has been developed, this uses line of sight to determine whether to push an object at a given time. The design and implementation of this simulation is covered in this paper.

An experimental evaluation of the produced simulation has been developed, this is done in two stages. The first compares the simulated occlusion transportation algorithm to its real-world implementation, we compare the time taken for the object to reach the goal point, the efficiency of the path taken and the amount of rotation applied to the object. The second evaluation uses a number of environment variables such as the number of robots in the arena or the shape of the object being pushed and describes how these variables can affect certain aspects of the simulation.

1. INTRODUCTION

1.1 Overview

This project simulates a collaborative transport method using a swarm of robots. This involves searching an arena for a convex-shaped object and then cooperatively pushing this to a goal. From the three different methods outlined in the background section of this paper, the occlusion based transportation method found in [3] was selected to be simulated. This method uses line of sight to push an object across its occluded sides. When a robot is in an area which is occluded it will head towards the goal, pushing the object with it. This solution can be implemented without communication between the robots and in a fully decentralised manner.

1.2 Aims and Objectives

The main objective of this project was to develop an accurate simulation of a collaborative transportation method using a visual interface which allows a user to see the full algorithm in real time.

This simulation should contain a number of environment variables which allow for customisation of a scenario. These variables are to be used in the experimentation section of the paper to see the effect the environment will have on the proposed solution and to setup an arena identical to the one found in [3] for comparison purposes.

Development of a user interface to manipulate these variables was an original goal of the design, this provides a user a way to customise scenarios without having to enter the code.

1.3 Project Challenges

One of the most difficult challenges faced in this project was programming in a new development environment. Extra time was needed during the implementation to compensate for inexperience.

As the project went through a major design change part way through, the deadline of the project became more of an issue.The requirement of more strict time-keeping became necessary to deliver the implementation on time.

1.4 Produced Solution

The produced solution is a visual, physics driven simulation of the occlusion based transportation method. This is setup as a square arena with a goal, an object and a number of robots which change their colour based on the behaviour they are currently executing. Additionally the simulation contains a number of environment variables which allows for customisation of the arena and the elements inside it. Examples of these variables include: max speed, object shape and density.

During the simulation a number of values used in the evaluation are calculated these are the angular error, path efficiency and time elapsed. Once the simulation ends a user can click and these will be displayed.

1.5 Project Effectiveness

Overall I believe the project to be a success, the software produced meets the major requirements I had set out to achieve. The main requirement of the project was to visually demonstrate the occlusion based transportation algorithm which from the implementation can be seen to have been satisfied.

While the results from the experiments performed were not as expected I believe that the evaluation and experimentation provides a reasonable justification to the differences between the implementations.

Another requirement of the software was to produce a interface to allow input into the program, due to a design change midway through the project a large portion of code had to be re-written putting a greater strain on the time limit of the project. Due to this the interface was not completed in time as I considered a lower priority than other sections of the program.

2. BACKGROUND

Swarm robotics is an approach used to coordinate large numbers of simple robots. The approach is inspired from the behaviour of social insects such as bees and ants. Through the study of the eusocial organisation of these biological systems we can begin to understand and develop swarm robot algorithms which demonstrate robustness, flexibility and scalability [1]. Robustness is a systems ability to function after faults or a change in conditions. Flexibility is the capability to adapt to changes in the environment. Scalability is the ability to expand the system without affecting the performance levels too greatly. These characteristics are desirable as they allow a system to operate under many different varying conditions.

In [2] the works using the swarm robot methodology are differentiated from those using a multi-robot approach by using the following characteristics:

- Each robot in the system should be autonomous.
- Robots are placed within the environment and can act to change it.
- Robots only have local sensing and communication capabilities.
- There is no access to a centralised controller or any global knowledge.
- There is cooperation between robots to accomplish a task.

These provide a framework for developing a swarm robot system that is robust, flexible and scalable as each agent is designed to have no, or very little dependency on other agents in the system.

The benefits of this approach are discussed in [7] and include:

- Exploitation of the sensing capabilities of large groups.
- They support superior situational awareness.
- Higher level of robustness than systems using individual agents.
- Distributed workloads tend to achieve more significant results.
- They carry out a large range of tasks with simpler and cheaper robots.

These give swarm robotics a wide variety of applications as described in [11]. An important example of these systems in used would be in disaster rescue missions; agents can be sent to areas humans can't reach and due to the hazardous environment a high level of robustness is needed.
However there are drawbacks to this approach which prevents usage in certain situations. Referring to [7], having a human operator controlling the swarm is difficult due to the decentralised nature of the system. Global system behaviour can also be hard to predict as this emerges from interactions between the agents at a local level, this can be an issue in safety-critical systems where a definite description of system behaviour is needed.

The problem within swarm-robotics which this paper addresses is known as box-pushing, this is a problem within the field of multi-robot transportation.Multi-robot transportation, sometimes referred to as foraging [1], is where "a group of robots has to cooperate in order to transport an object. In general, the object is heavy and cannot be moved by a single robot, making cooperation necessary" [2]. This area is a long-standing and difficult goal of multi-robot systems [5]. The problem can be split into two phases, an exploratory phase and a transportation phase. During the exploratory phase each agent in the system must explore the environment to find the object or prey item that is to be retrieved. In the Transportation a agent must position itself and the object and then transport this to some goal location.

Common approaches to this type of problem involve pushing, pulling and caging [3]. Pulling an object consists of a number of robots connecting themselves to an object and then travelling to their goal location. The pushing method is a simple way of transporting a large object without much mechanical complexity. Caging is a special form of pushing where a group of robots surround the object in such a way so that the object is caged, if this position is maintained the object will follow the robots as they move. This section will review the benefits and disadvantages of algorithms using these different approaches.

2.2 Group Transport Along a Robot Chain

Another related problem to this area is known as foraging. "The aim of robots in a foraging task is to find the preys and bring them to the nest" [1]. An example of this using the pulling approach is the chain based path formation and group retrieval algorithm found in [4]. This algorithm explores the arena, forms a chain to the object and then pulls it along this chain back to the nest. The software uses two modules a exploration module and a transportation module, the behaviours are found in [6] and are as follows:

- **Search**: the robot performs a random walk it travels forward and turns on the spot when an object is detected.
- **Explore:** an explorer moves along the chain towards the tail. If a robot leaves a chain, it will move back to the nest and follow a different chain.
- **Chain:** a robot joins the chain and activates its LED's to let other robots know it's part of the chain.
- Assemble: The robot approaches a nearby prey object and attaches to it.
- **Transport-Target:** The robot aligns itself with the nearest chain member and begins pulling.
- **Transport-Blind:** sense the torque on the grippers and calculate the direction to push.

The experimental results in [6] show that this method is both robust and reliable however it requires more robots than other examples of multi-robot cooperative systems. Another problem with transportation utilizing pulling mentioned in [3] is the complexity of the physical mechanisms, this could limit the method being used in certain areas.

2.3 Transportation Via Caging

The method outlined in [9] demonstrates a swarm robot transportation algorithm which cages the relevant object. The method makes a number of assumptions about the environment it is operating in:

- Every robot has the same body shape.
- All robots know how many robots are in the system.
- Robots can detect the distance to other robots and the object at all times.
- There are enough robots to cage the object.

This method is realised in with two behaviours: Object Approach and an Inescapable Formation Search. Starting from a random position on a side of the arena the Object Approach behaviour activates, each robot must approach the object independently. A potential field based model is used where a potential force radiates from the object guiding the robots towards it, additional attractive or repellent forces are applied based on the proximity to other robots allowing for collision avoidance.

Once the object has been reached it may not be entirely caged, this is where the Inescapable Formation Search behaviour is used. The robots follow the perimeter of the object until a condition is satisfied that satisfies the requirement of object closure. Object closure is defined as there being no feasible path for the object to any point outside the space enclosed by the robots. When this step is completed a simple system to maintain the swarm's formation can be used to push the object to the goal.

The advantages of this method discussed in [9] include good scalability and the ability to work with objects where the object geometry cannot be precisely determined. However this method struggles when using robots with a range of diverse shapes and with how object closure is hard to precisely define.

2.4 Occlusion Based Cooperative Transportation

The occlusion based transportation method found in [3] is the algorithm used in the implementation of the simulation. Unlike the previous two algorithms this utilises pushing as the method of transportation. The idea behind this strategy is that by having the goal location emit light an object can be continually pushed along its occluded side to reach this goal. The assumptions made about the environment in [3] are:

- The object and goal are recognisable by the robots.
- The object is large enough to occlude the robots view to the goal.
- The robots can perceive the goal from any location except if occluded by the object.
- There are no obstacles in the environment excluding the boundary.

Starting from a random position within a initialisation region an agent must firstly explore its environment using a relevant search algorithm, in this case there is a bounded environment

so a random walk is used. Once the object is detected the agent will move towards it, when in range it will attempt to see if the goal can be seen from its position. If the goal is seen the agent will move around the object and check line of sight to the goal again. If the goal is not seen the robot will move towards where it knows the goal to be, pushing the object along the way.
An extension to this strategy exists which allows the system to operate in more complex environments where the goal may not be perceived from any area due to obstacles, or due to the sensor range not being able to support the distance between the entities. In this scenario a mobile robot can be used as the goal location, navigating around a complex environment to a final destination.

The simplicity of this strategy makes it suitable for use on robots with limited capabilities, an example being on robots at very small scales operating in a vascular network to deliver drugs. However problems exist with this method due to the difficulty of perceiving the goal imposing certain limitations on usage. An example given in [3] discusses sensors which are positioned higher than the object in a 2-D environment imposing limitations on the height of the object.

2.5 Development Environments

Initial development of the project was done using an IDE known as Netlogo. It is described as “a programmable modeling environment for simulating natural and social phenomena.” [8]. It provides modeling tools to simulate a large number of agents in an environment and allows the study of how local behaviours affect the global behavior of a system. Netlogo also contains a built-in interface designer allowing for quick adjustment of the environment variables, an agent monitoring system for inspection of individual agents and a speed slider allowing visualisation of the simulation at different speeds. This IDE however contains no libraries for true physical collisions between agents and the object which was to be pushed which made it unsuitable for this project as modelling the physics behind a collision can become complex and an acceptable representation of this could not be developed in the time-frame of the project.

The final implementation of the project is done using an IDE called processing. It is described as “a flexible software sketchbook and a language for learning how to code within the context of the visual arts” [12]. The software was used to manage the visual elements of the simulation, with pre-built methods allowing for easy manipulation of the interface. However processing doesn’t contain any in-built physics libraries, to solve this a plugin called jbox2D was installed. This is the java version of box2D which is an open source physics engine for simulating rigid bodies in 2D [13]. With this a more realistic simulation environment was used in the project as complex physical interactions such as restitution were simulated.

3. DATA REQUIRED

A comparison between the results of the simulation to those found in [3] is a requirement of this project. This is the only data required for this project and as it lies within the public domain there are no ethical concerns with using it. No human participants were involved with this project.

4. DESIGN

4.1 Summary of Proposal

Swarm Robotics is an approach to dealing with the coordination and collective behaviours of multi-robot systems. Using many simple robots, instead of one complex robot, a system utilizing swarm robots should be designed to be robust, flexible and scalable. To achieve this the following constraints should be imposed on the system:

- Each robot should be autonomous, it will act independently.
- No robot should have any access to a centralized control system.
- Each robot should only have local sensing and communication capabilities.
- Robots should act cooperatively.

The main aim of this project is to simulate the occlusion based transportation method found in [3]. The formulation of this problem is defined as:

“ A bounded environment contains a convex-shaped object, a goal, and a number of robots. The environment is otherwise free of obstacles. The aim is that the robots, which are initially placed in arbitrary locations, push the object to the goal.”

The following assumptions are made about the environment in this scenario:

- The object and goal are recognisable by the robots.
- The object is large enough to occlude the robots view to the goal.
- The robots can perceive the goal from any location except if occluded by the object.
- There are no obstacles in the environment excluding the boundary.

Other aims of this project include:

- A comparison of the simulation to the real world implementation.
- Automatic calculation of angular error and path efficiency.
- Changeable environment variables.
- A user interface displaying the simulation in real-time.
- An evaluation of how environment variables affect the algorithm.

My current research into swarm robotics can be found in the background section of this paper; it includes different transportation methods, an overview of swarm robotics and their real-world uses.

4.2 Environment Variables

This section describes each of the environment variables implemented in the simulation and what their effects are.

- Robot Number - How many robots are spawned within the initialization region at the start of the simulation.
- Robot Torque - This is the maximum torque to be applied to the robot at each step during its random walk behaviour.
- Robot Max Speed - The maximum speed a robot can achieve during the run.
- Object Shape - The shape of the object to be pushed to the goal, includes: circles, triangles, pentagons, squares, rectangles and a scalene triangle.
- Density - The mass per unit of an object, bigger values will make the object heavier.
- Friction - The resisting force applied when two objects move across each other.
- Restitution - The energy transferred between objects during a collision a value of 1 represents a perfectly elastic collision while a value of 0 represents a perfectly inelastic collision.
- Linear Damping - The rate of reduction of linear velocity.
- Angular Damping - The rate of reduction of angular velocity.
- Size - The diameter of the object.

These have been changed from the original design as variables such as simulation speed are irrelevant in terms of their effect on the simulation. Additionally box2D can assign variables to objects that were not originally planned to be added. Implementing these was relatively easy as it only involved assigning a value, examples of this include friction, density and restitution.

4.3 Class Diagram

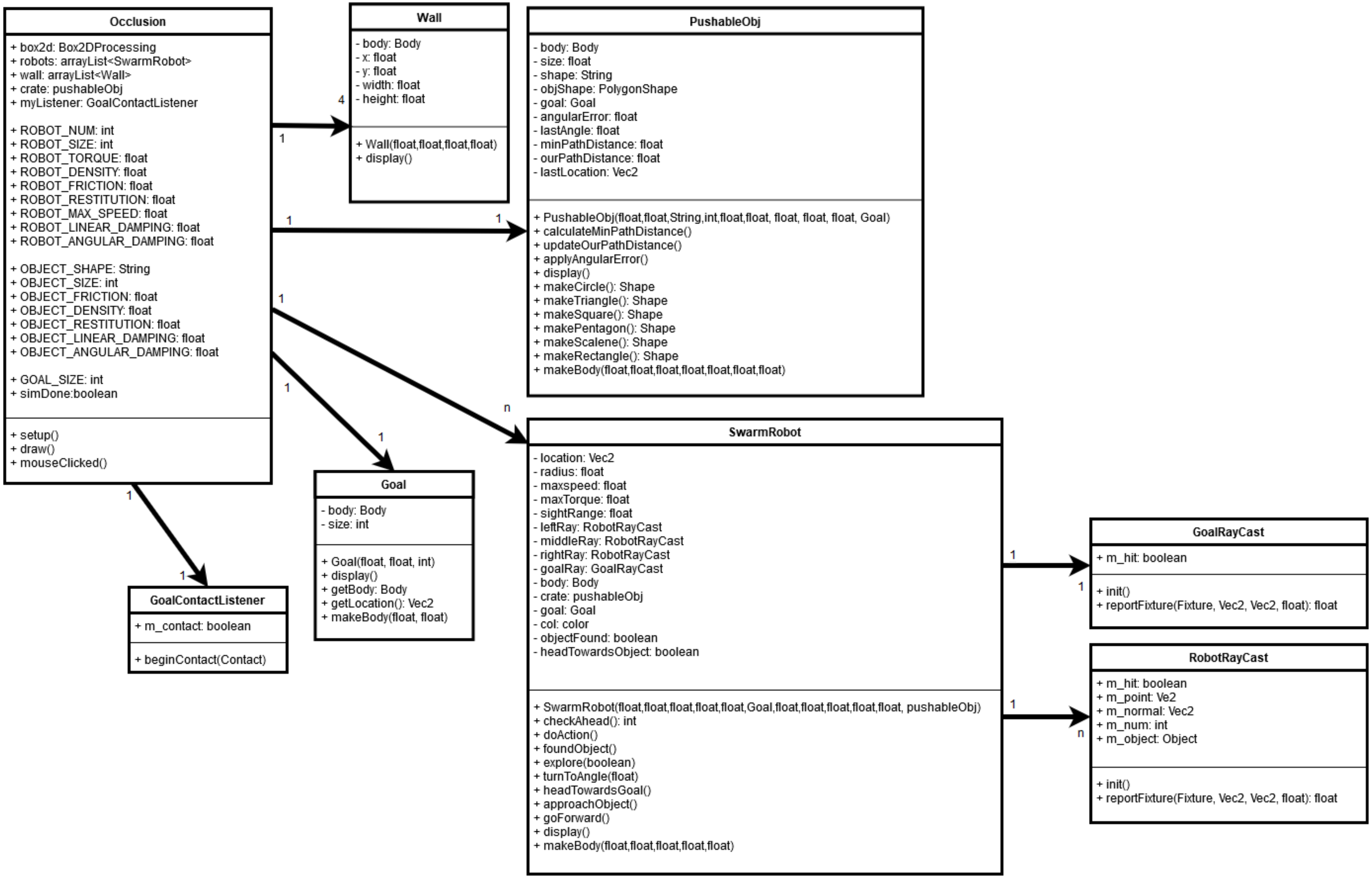
The current design contains eight classes; of these classes the occlusion, goal, wall and robot classes are present from the original design. Due to time constraints the class for the variable entry screen has been removed and the results calculator has been merged with the main occlusion class. Additional classes have been added to support the raycast and collision detection features, these classes are known as callback classes and process the data sent to them by the raycast or collision.

The Goal class represents a small static circular object which the pushable object must reach. A static object in this case is one that cannot be moved.
The Wall class represents a static line used to enclose the arena.

The PushableObj class represents the object which is to be transported to the goal. It contains a variety of methods to change its shape, calculate path efficiency and determine its angular error.

The SwarmRobot class represents an agent in the system, it uses subsumption architecture to determine which behaviour to run.

The raycast methods are callback classes used when the raycast command is called from the robot. A line is drawn between two points and any intersecting objects are reported to the raycast class which determines what to do with the data. The raycast to the goal only checks if the pushable object is intersecting between the robot and the goal whereas the robot raycast represents the front sensor and reports all objects seen.
The GoalContactListener class is a callback class used whenever a collision occurs within the simulation, the class is given the two bodies which have collided and if these are the goal and the pushable object it sets m_contact to true which signals the simulation has completed.


Occlusion
+ box2d: Box2DProcessing
+ robots: arrayList<SwarmRobot>
+ wall: arrayList<Wall>
+ crate: pushableObj
+ myListener: GoalContactListener
+ ROBOT_NUM: int
+ ROBOT_SIZE: int
+ ROBOT_TORQUE: float
+ ROBOT_DENSITY: float
+ ROBOT_FRICTION: float
+ ROBOT_RESTITUTION: float
+ ROBOT_MAX_SPEED: float
+ ROBOT_LINEAR_DAMPING: float
+ ROBOT_ANGULAR_DAMPING: float
+ OBJECT_SHAPE: String
+ OBJECT_SIZE: int
+ OBJECT_FRICTION: float
+ OBJECT_DENSITY: float
+ OBJECT_RESTITUTION: float
+ OBJECT_LINEAR_DAMPING: float
+ OBJECT_ANGULAR_DAMPING: float
+ GOAL_SIZE: int
+ simDone:boolean
+ setup()
+ draw()
+ mouseClicked()
Wall
- body: Body
- x: float
- y: float
- width: float
- height: float
+ Wall(float,float,float,float)
+ display()
PushableObj
- body: Body
- size: float
- shape: String
- objShape: PolygonShape
- goal: Goal
- angularError: float
- lastAngle: float
- minPathDistance: float
- ourPathDistance: float
- lastLocation: Vec2
+ PushableObj(float,float,String,int,float,float, float, float, float, float, Goal)
+ calculateMinPathDistance()
+ updateOurPathDistance()
+ applyAngularError()
+ display()
+ makeCircle(): Shape
+ makeTriangle(): Shape
+ makeSquare(): Shape
+ makePentagon(): Shape
+ makeScalene(): Shape
+ makeRectangle(): Shape
+ makeBody(float,float,float,float,float,float,float)
SwarmRobot
- location: Vec2
- radius: float
- maxspeed: float
- maxTorque: float
- sightRange: float
- leftRay: RobotRayCast
- middleRay: RobotRayCast
- rightRay: RobotRayCast
- goalRay: GoalRayCast
- body: Body
- crate: pushableObj
- goal: Goal
- col: color
- objectFound: boolean
- headTowardsObject: boolean
+ SwarmRobot(float,float,float,float,float,Goal,float,float,float,float,float, pushableObj)
+ checkAhead(): int
+ doAction()
+ foundObject()
+ explore(boolean)
+ turnToAngle(float)
+ headTowardsGoal()
+ approachObject()
+ goForward()
+ display()
+ makeBody(float,float,float,float,float)
Goal
- body: Body
- size: int
+ Goal(float, float, int)
+ display()
+ getBody: Body
+ getLocation(): Vec2
+ makeBody(float, float)
GoalContactListener
+ m_contact: boolean
+ beginContact(Contact)
GoalRayCast
+ m_hit: boolean
+ init()
+ reportFixture(Fixture, Vec2, Vec2, float): float
RobotRayCast
+ m_hit: boolean
+ m_point: Ve2
+ m_normal: Vec2
+ m_num: int
+ m_object: Object
+ init()
+ reportFixture(Fixture, Vec2, Vec2, float): float
1
4
1
1
1
n
1
1
1
1
1
1
1
n

4.4 Pseudocode

The swarm robots use subsumption architecture to determine which behaviour to execute at each timestep. This type of architecture uses a hierarchy of behaviours each with a given trigger. The table below gives a list of these behaviours and their triggers. The higher the behaviour in the table the greater its priority is.

Behaviour	Trigger
Head Towards Goal	Goal is occluded by object
Approach Object	Object seen but too far away.
Circle Object	Object seen and is close
Avoid Obstacle	Obstacle detected ahead
Random Walk	None

4.4.2 Circling the Object

The Circle Object behaviour travels around the object to find a position where the goal is occluded, during this phase the robot doesn't want to get too far away from the object and get lost or bump into the object. The pseudocode for this method is as follows:

```
Start
    Use sensors to check ahead.
    Calculate Robot Trajectory.
    Check If Trajectory is Taking us too far from the object.
    If Object Sensed Ahead
        Apply Torque Away From Object
    Else If Trajectory taking us too far from Object
        Apply Torque Towards Object
    Else
        Travel Forwards
```

```
End
```

This behaviour originally just calculated the distance from the object instead of using a trajectory however when the robot got too far away from the object it would get stuck turning as it couldn’t go forwards.

4.4.3 Random Walk

The random walk implementation has gone through three different realisations. Initially a random velocity between two thresholds was applied in a random direction, this caused very jittery motion as a new velocity would be chosen every step. The second realisation applied a random acceleration at every step, this was a lot smoother however the robot didn’t explore the arena in a reasonable timeframe. The final realisation always travels forward while applying a small random torque at each step.

```
Start
    Check For Obstacle Ahead
    If Obstacle Ahead
        Select High Torque To Apply
    Else If No Obstacle Ahead
        Select Small Random Torque To Apply
    Apply Chosen Torque To Body
    Go Forward
End
```

This method allows the robot to explore the arena at a reasonable speed and allows the front facing sensor to be used to a greater extent. There is also rudimentary obstacle avoidance during the random walk.

4.4.4 Angular Error And Path Efficiency

Two of the variables used to evaluate the occlusion transportation methods effectiveness are angular error and path efficiency. Angular error is the amount of unnecessary rotation applied to the object whilst path efficiency is the ratio between the distance of the optimum path and the distance of the actual path.

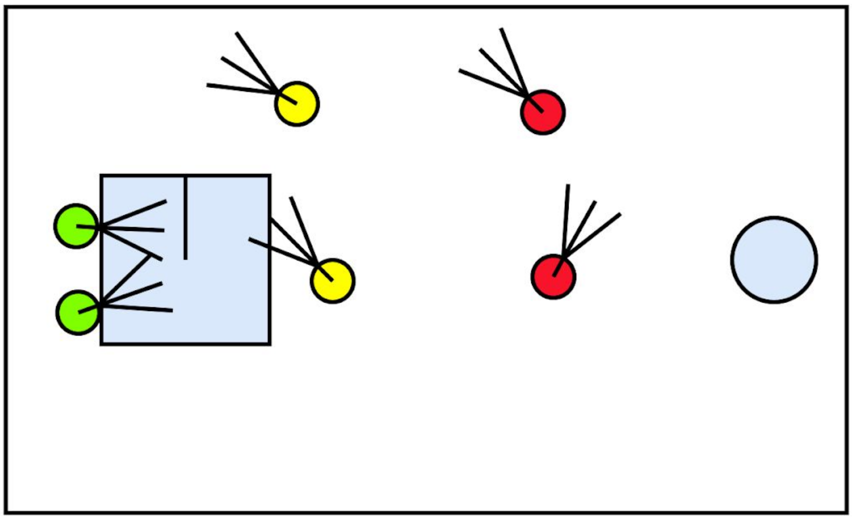
To calculate the Angular Error:

```
Start
    Get Body’s Current Angle
    Get Body’s Last Angle
    Angular Error This Step = absolute(Current Angle - Last Angle)
    Total Angular Error = Total Angular Error + Angular Error This Step
    Set Last Angle To Current Angle
End
```

To calculate path efficiency:

```
Start
    Min Path = Distance to Goal From Start Location- (Goal Radius + Object Radius)
    Get Body’s Last Location
    Get Body’s Current Location
    Length Travelled = Current Location - Last Location
    Current Distance = Current Distance + Length Travelled
    Last Location = Current Location
    Path Efficiency = (Min Path Distance / Current Distance) * 100
End
```

4.5 Interface Design



The current interface design is shown in the figure above, an additional screen displaying statistics for the run is shown after completion of the simulation. Unlike the previous interface design this model does not have a variable entry screen, this is due to time constraints on the project and the priority of other components.

The goal is represented by a circular, static object on the right of the arena.

The object is spawned on left of the arena and can take a variety of shapes (square in the figure above). Additionally a line is drawn down the center of the object to represent its location.

Robots on the interface are represented by circular objects with varying colour based on their current situation. They have indicators showing their cone of sight and like the object have a line denoting their rotation. There are currently three colours a robot can take: red, yellow and green. Red represents exploring the arena as the object hasn't been discovered yet, Yellow represents that the object has been found but the goal is not occluded and Green represents that the goal is occluded and the robot is ready to push the object.

4.6 Evaluation

As mentioned in the project aims an evaluation of the simulation and the project as a whole is required. To satisfy this requirement three types of evaluation are used: Experimental, Comparisonal and Self.

Experimental:

The Experimental evaluation focuses on how individual environment variables affect the results of the simulation. The variables selected for evaluation include:

- Object Shape.
- Object Size.
- Number of Robots.
- Object Density.

The effect of these are measured by taking the mean value of the time elapsed, angular error an path efficiency. The robustness, flexibility and scalability can be evaluated from these results based on the systems performance under different starting conditions and the effect of increasing the problems size.

Comparison:

As mentioned in the summary of proposal section, a comparison between the simulated algorithm to it's real-world implementation is to be included. To be consistent with [3], the performance of the algorithm will be measured using the following criteria.

- The number of successful trials.
- The mean and standard deviation of completion times.
- Mean and Standard Deviation of the Path Efficiency - a comparison between the path taken and the ideal straight line path .
- Mean and Standard Deviation of the Angular Error - the amount of unnecessary rotation applied to the object.

In order to make this a fair test we must use the same units as in the paper. Box2D by default uses meters and has an in-built function to convert the pixel unit used by processing into meters, this makes the conversion relatively easy. We must also make sure to place the objects in the same starting area, this includes setting up an initialization area which the robots will be randomly placed within at the simulations start. Any large discrepancies with the results found in [3] will be recorded and analysed.

Self:

A self evaluation on the project will be carried out to identify the strengths and weaknesses on parts of the project that would be impractical to analyse mathematically. The current plan is to perform self-analysis based on the following criteria:

- Have the project objectives been met satisfactorily.
- What changes could have been made to the project to improve its effectiveness.
- What challenges were faced and how were they overcome.

A critical analysis of the results and outcomes of the project is to be carried out alongside an evaluation of what I believe to be the strengths and weaknesses present over the course of the project.

4.7 Gantt Chart Analysis

The Gantt chart produced as part of the original design document (Appendix 9.3) was kept up with until late February, at this point a change in the development environment pushed everything back about a month as previous components had to be re-written. Towards the end of the project this led to a slight excess of work as the dissertation needed to be started early to leave a reasonable amount of time to complete it. The contingency time allowed me to begin to slightly catch back up with the original plan.

5. REALISATION

This section of the paper will give a description of how the design was implemented and how this was tested. Included are justifications on changes from the original design document and snapshots of some key methods in the program.

5.1 Design Implementation

To implement the design I used the techniques found in [14]. Firstly the world needs to be created, this contains all the elements in the world and knows everything about the coordinate space. The elements in the world need a body and a shape to exist, a body contains an elements velocity and position while a shape defines the geometry of an object. To connect the body and shape a fixture containing properties such as friction, density and restitution. With all these defined the object now exists in the world an can interact with other objects, we do however still need to draw the shape to the screen by converting the elements location to pixels and using processing's inbuilt methods.

Once the bodies have been added to the world they can be left for the physics engine to calculate their positions and velocities. The exception to this is the robots, these have to be proactive and thus require methods to manipulate their position and velocity.

5.2 Problems Encountered

One of the main problems encountered during the project was the limitations on the Netlogo IDE. The IDE doesn't support any physical interactions and constrains movement to a grid, these features affected the accuracy of the simulation so the decision was made to restart the project using the processing IDE with the box2D plugin. Overall the time-saved delegating the physics of the system to Box2D made up for the time lost after restarting the implementation.

A problem made evident whilst working with box2D and processing was the differing measurement units. A lot of errors during testing could be attributed to using processing pixel unit value instead of box2D's metre unit value. To attempt to solve this problem most of the program only uses the metre value to avoid confusion.

5.3 Changes to the Design

As discussed in the previous section a change to the development environment was done mid-way through the project.

The original interface design included a screen before the simulation where a user could enter specific values for each of the environment variables. Due to time-constraints on the implementation this screen has been removed however these variables can still be easily changed within the code.

Another change is the environment values which can be changed within the system. Those which have been removed include: Arena height, Arena width, Object distance from Goal, Simulation speed, Number of runs. During implementation it was found that processing does not allow variables to be used for the screen height and width, these have to be hard coded in making it impossible to implement these as user-changeable values. The other variables were not implemented as they were not believed to be statistically interesting based off the evaluation criteria.

5.4 Testing of Key Methods

Testing of the implementation involved checking that objects were instantiated properly and that the robot's behaviours were working correctly. Interactions between objects were managed by box2D which is a black box, as long as the objects are correctly instantiated these interactions should be accurate.

Below lies a table detailing an abbreviated version of the test results for the robots final iteration of its random walk, the full table can be found within the testing appendix.

Method	Expected Result	Actual Result	Pass/Fail
Random Walk V3	Robot goes forward and applies small torque at each step.	Robot goes forward and applies small torque at each step. Robot gets stuck when driving into wall.	Fail
Random Walk V3	Robot goes forward, applies small torque at each step or a large torque if boundary or another robot is detected.	Robot goes forward, applies small torque at each step or a large torque if obstacle detected. Robot get stuck spinning if it detects boundary in corner.	Fail
Random Walk V3	Increased angular damping should prevent robot from spinning when detecting boundary in corner.	Random walk works as expected.	Pass
Random Walk V3	Goal added to arena, robot should avoid this as well as other robots and the boundary.	Random walk works as expected.	Pass

The initial setup of the simulation involved a square arena which the robots would explore using a random walk algorithm. This algorithm drove the robot forward and applied a small random torque at every timestep. Testing on this algorithm showed that the robot would get stuck on the arena boundaries therefore collision avoidance was implemented. Using the robots front facing sensor this obstacle avoidance algorithm would apply a large amount of torque to the robot causing it to steer away from the obstacle. This worked to avoid the boundary however a robot could get stuck spinning in corners as the angular velocity wasn't decaying fast enough to prevent the robot seeing the another boundary and applying more torque. To fix this problem the angular damping on the robots was increased to make its angular acceleration decay faster.

Another method is required to circle the object once it has been found by the robot. This method underwent a number of iterations based off of the test data received from running the simulation.

Method	**Expected Result**	**Actual Result**
Circle Object V2	Robot should travel around the object remaining within a certain distance.	When robot gets too far away it gets stuck spinning.
Circle Object V2	Trajectory added to robot, torque applied towards object when this leads out of range.	Torque always being applied to robot.
Circle Object V2	Units of trajectory converted	Method works as expected.

	from pixels to meters.	

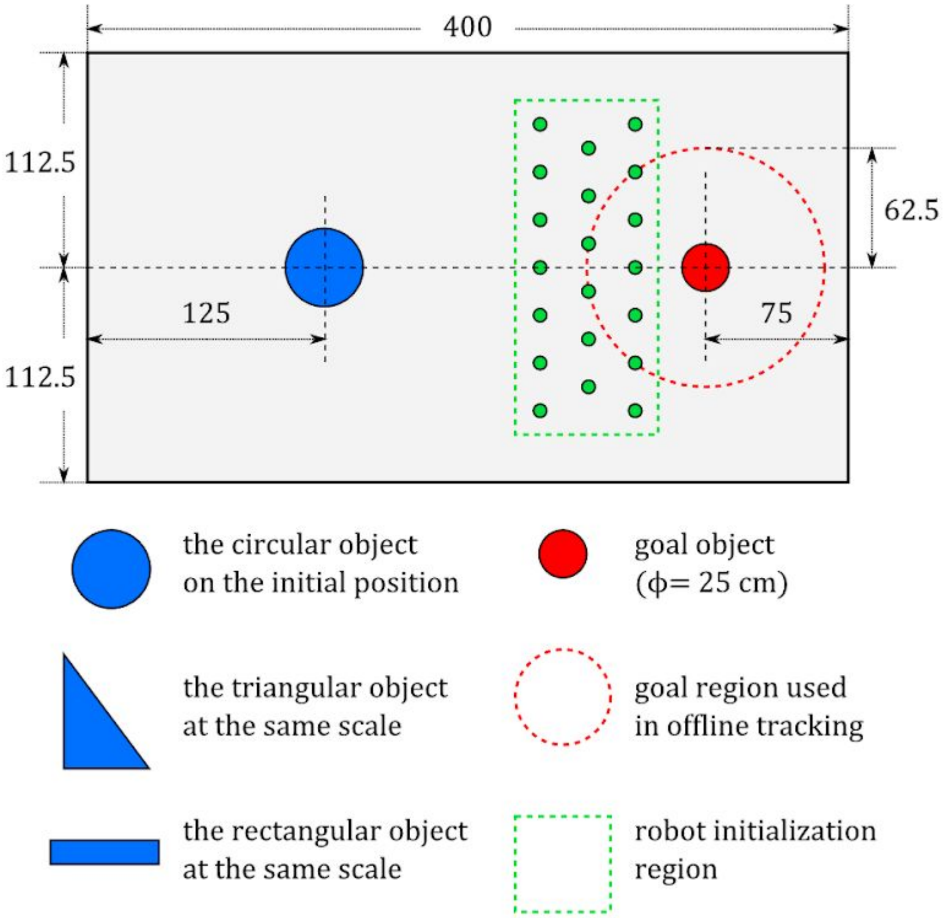
The circle object method is called when the robot has discovered the object and is within a certain distance. The original method used the equations for circular motion however this was reworked as it didn't have enough flexibility to account for a moving object. The second iteration applies torque towards the object when the robot gets too far away and away from the object when the robots is too close. Problems appeared when the robot was too far away from the object, torque was applied but as the robot did not move forward it got stuck. To solve this a trajectory was added, if this led outside the range then a torque would be applied. This trajectory was using the wrong units causing torque to be applied at every step, this was solved by using an in-built method to convert these units.

6. EVALUATION AND EXPERIMENTATION

This section of the paper will evaluate the success of the criteria based on the project aims and the criteria presented in the design section. Additionally a comparison to the results found in [3] will be produced alongside an experimental comparison of individual environment variable effects.

6.1 Comparisonal Experimentation

In order to begin a comparison to [3] the arena must first be set up in the same way. Below is the setup used in the real-world implementation of the occlusion based transportation method, all units are given in cm.


400
112.5
112.5
125
75
62.5
the circular object on the initial position
goal object (ϕ= 25 cm)
the triangular object at the same scale
goal region used in offline tracking
the rectangular object at the same scale
robot initialization region

Three shapes are used to test this implementation: a circle, a scalene triangle and a rectangle. All of these will be used in the comparison.

The table from [3] gives the mean and standard deviation of the completion time, path efficiency and angular error of the real-world implementation.

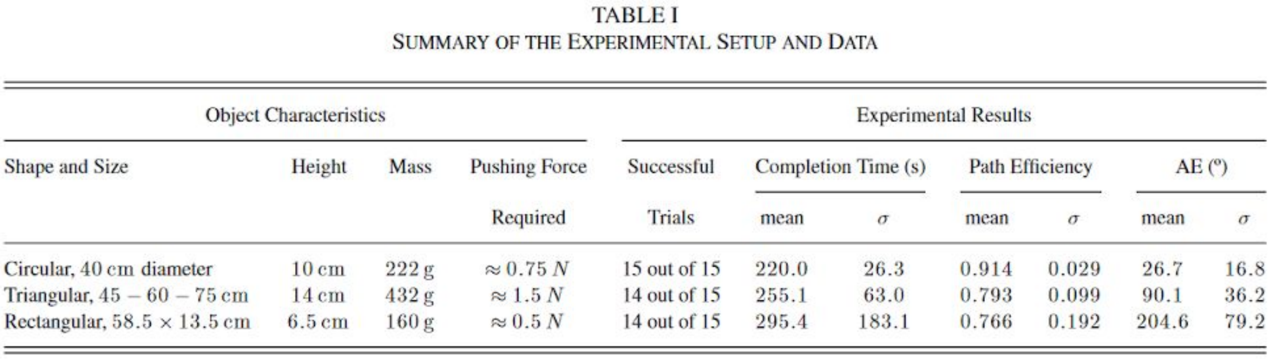
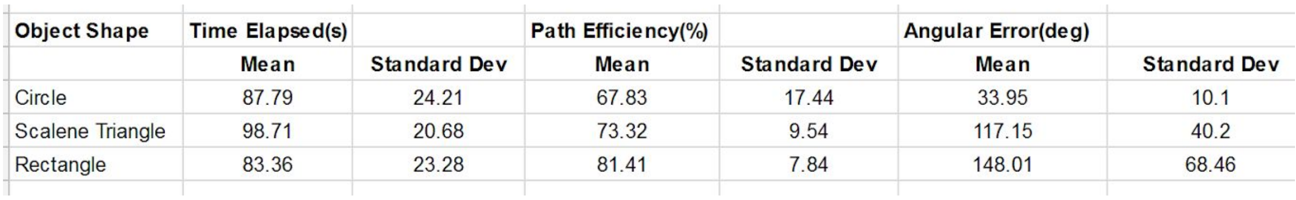


TABLE I
SUMMARY OF THE EXPERIMENTAL SETUP AND DATA

Object Characteristics				Experimental Results						
Shape and Size	Height	Mass	Pushing Force Required	Successful Trials	Completion Time (s)		Path Efficiency		AE (º)	
					mean	σ	mean	σ	mean	σ
Circular, 40 cm diameter	10 cm	222 g	$\approx 0.75\ N$	15 out of 15	220.0	26.3	0.914	0.029	26.7	16.8
Triangular, $45-60-75$ cm	14 cm	432 g	$\approx 1.5\ N$	14 out of 15	255.1	63.0	0.793	0.099	90.1	36.2
Rectangular, 58.5×13.5 cm	6.5 cm	160 g	$\approx 0.5\ N$	14 out of 15	295.4	183.1	0.766	0.192	204.6	79.2



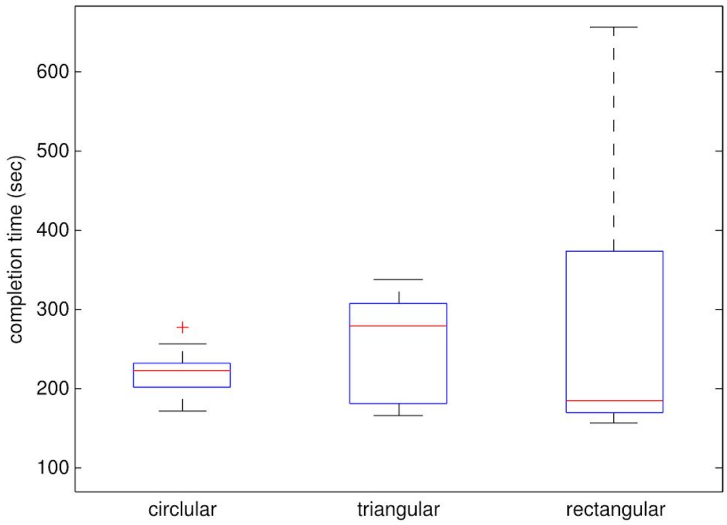
Object Shape	Time Elapsed(s)		Path Efficiency(%)		Angular Error(deg)	
	Mean	Standard Dev	Mean	Standard Dev	Mean	Standard Dev
Circle	87.79	24.21	67.83	17.44	33.95	10.1
Scalene Triangle	98.71	20.68	73.32	9.54	117.15	40.2
Rectangle	83.36	23.28	81.41	7.84	148.01	68.46

Above is the results from the simulation, for each object 15 trials were run of these none of them failed however they do provide some interesting results.

The biggest difference between the two implementations is the completion time, the simulation seems to get the object to the goal much faster. This is would suggest that the path taken by the object is much better however based on the results for path efficiency this cannot be the case. If the path efficiency of the simulation is at best equal to that of the real-world implementation there must be another cause for this discrepancy. I currently have identified three factors which could account for the difference:

- The simulation’s robots could explore the arena and find the object faster than the real-world implementation, this will most likely be caused by the random walk algorithm. The exact random walk algorithm is not given in the paper and thus could have factors which slow it down, for instance: having to stop to turn.
- Once found the object may be harder to lose in the simulation. Unlike the real-world implementation, the simulation keeps memory of where the object is and only considers itself lost if it gets a certain distance away. On the other hand the real-world method has a camera tracking the object and it is counted as lost when this camera no longer detects the object; this means an object could be lost due to a invalid sensor reading or something blocking its view.
- Other variables not mentioned in [3] could affect the algorithm. The surface of the arena could have a low friction coefficient causing wheel slips or the lighting conditions in the room could affect the range of the colour sensors.

Another interesting result of this experiment is the difference in how a rectangle is transported. Based on my results it seems to be the best object to transport (based on completion time), this is not true in [3] where it is seen to be the worst. However looking at the standard deviation of the completion time for the real-world rectangle shows that there is a large standard deviation, this implies that the mean is skewed by larger outliers and thus I will be using the median value to compare to.



As seen in the figure above the median is the lowest for the rectangle. This lines up with the results from the simulation. Based on the range of the data from [3] I do believe that the mean gives a misleading figure.

The angular error of the different objects in the simulation lines up with the results in the paper. While generally slightly higher than the real-world results I believe this difference is not significant enough to draw any conclusions as to why the two implementations don't directly match. One observation based on both results is that it seems the greater the difference in size between the different edges of the shape, the greater the angular error.

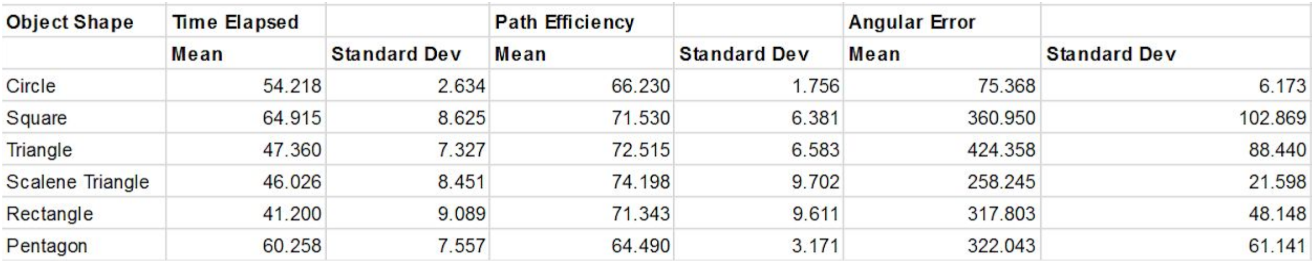
6.2 Experimental Evaluation

This section of the evaluation will determine how each environment variable can affect the simulation. As previously mentioned in the design section, the variables chosen for experimentation are: object shape, object size, number of robots and object density.
In order to make testing fair a standardized environment has been created, this will be the same for all the variables being tested. Each variable will be tested independently and its effects on angular error, path efficiency and time elapsed will be recorded and any changes will be discussed. In order to more accurately gauge these effects, each value the variables takes will be tested four times and its average and standard deviation will be taken.

Full test results and details on the standardized environment can be found within the accompanying spreadsheet within the zipped file.

6.2.1 Object Shape

There are six object shapes in the current implementation, these include: a Circle, an Equilateral Triangle, a Scalene Triangle, a Rectangle, a Square and a Pentagon. The structure of these shapes can greatly alter the effectiveness of the transportation method. Below are the results from each test.



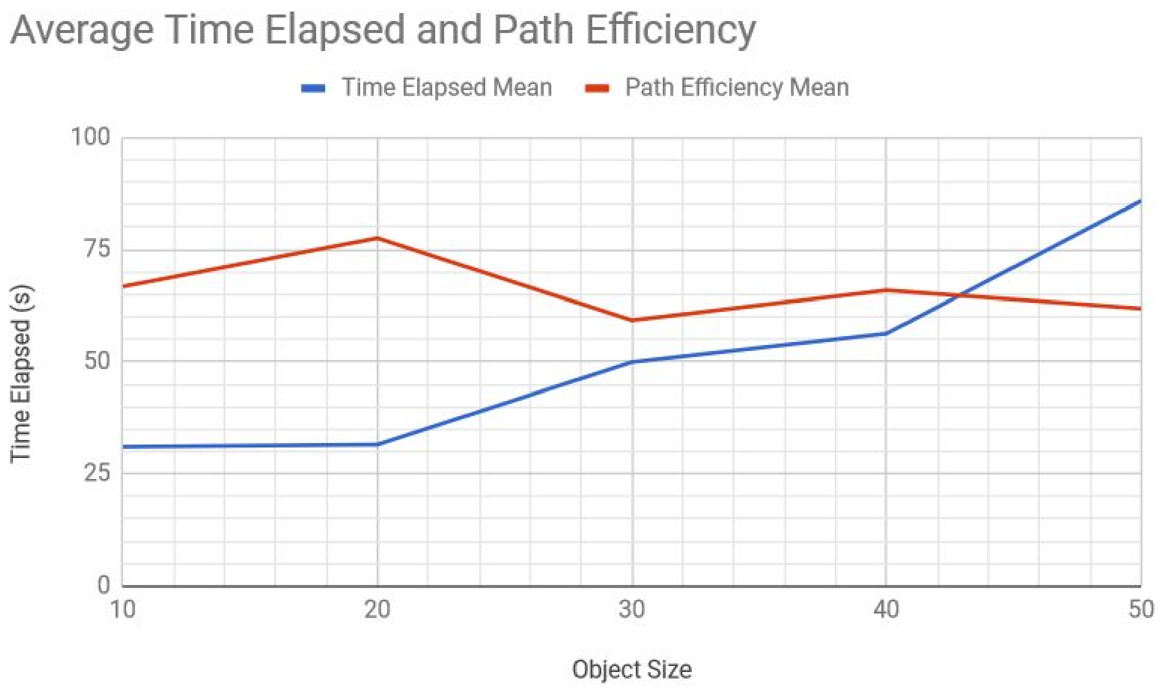
Object Shape	Time Elapsed		Path Efficiency		Angular Error	
	Mean	Standard Dev	Mean	Standard Dev	Mean	Standard Dev
Circle	54.218	2.634	66.230	1.756	75.368	6.173
Square	64.915	8.625	71.530	6.381	360.950	102.869
Triangle	47.360	7.327	72.515	6.583	424.358	88.440
Scalene Triangle	46.026	8.451	74.198	9.702	258.245	21.598
Rectangle	41.200	9.089	71.343	9.611	317.803	48.148
Pentagon	60.258	7.557	64.490	3.171	322.043	61.141

From these results it is apparent that a rectangle is the easiest object to transport in terms of the time elapsed, this is consistent with the results from the comparisonal evaluation. Surprisingly the square takes the longest time to transport, this is interesting as it is a similar shape and has a similar path efficiency to the rectangle.

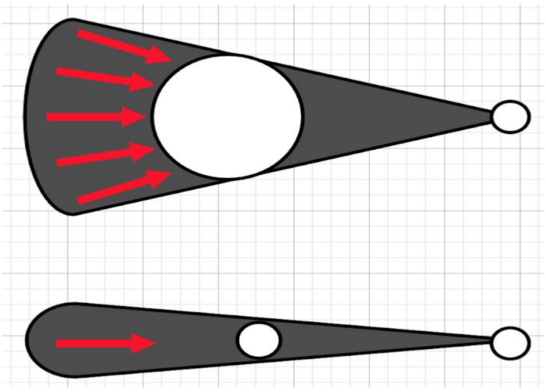
Watching the simulation transporting a square, it seems that the robots can sometimes get in the way of the object while it is moving. This is a problem which mostly affects the square and triangle shaped objects as the way they’re set-up means the distance from its center point to a vertex is greater than the size its given as that’s measured from the center point to the top of the square. This causes robots executing the circle object behaviour which to sometimes get caught on these vertices as the radius of the circles they travel in is based off the size of the object.

6.2.2 Object Size

The object size determines the distance between the middle point of an object to the first edge encountered when drawing a line straight up. Bigger sized objects will occlude a larger area meaning robots will find the object faster and have more space to push. There is a tradeoff however, as the surface area of the object increases so does its mass, meaning the object requires a larger force to be pushed.



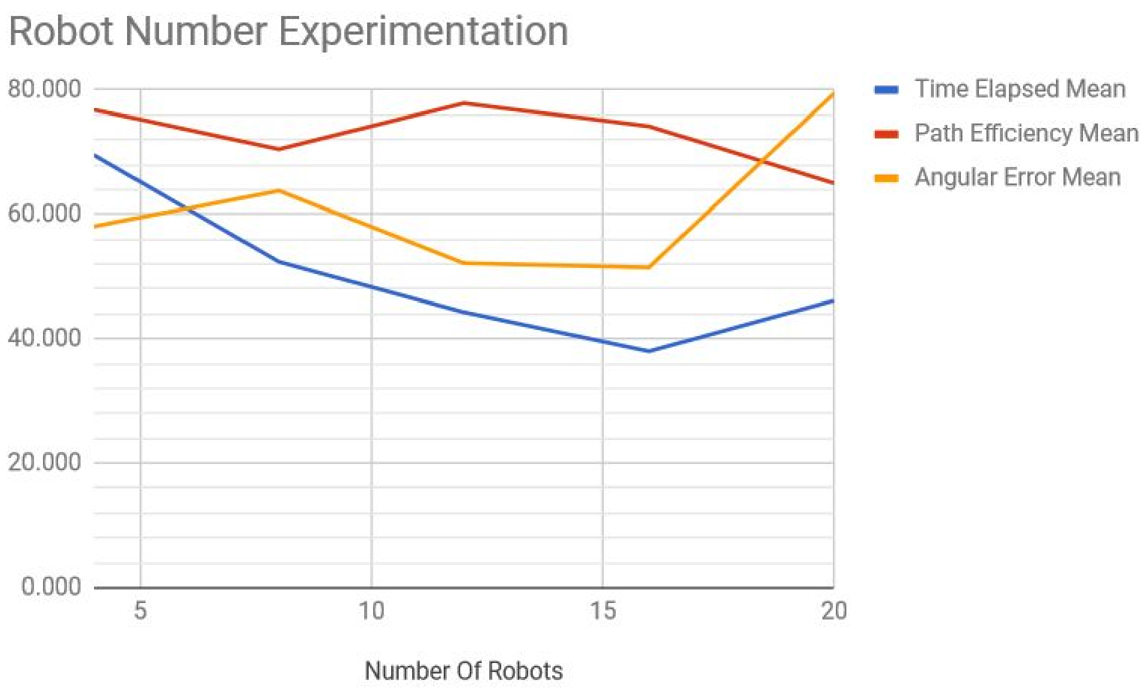
Above is the chart showing the average time elapsed and the average path efficiency. A trend in the graph shows that as the object gets larger the time taken to transport it gets higher, this suggests that the mass of the object is a more significant factor than the area it occludes. Most interestingly is how path efficiency changes, there is a slight negative trend as the object size increases. I believe this to be caused by two factors: fewer robots pushing the object and the occluded area being smaller which causes robots to push from more favourable positions (see fig below).



The angular error was also measured however this remained mostly constant through testing, with the exception of the first test where the angular error is at least four times higher than the others. This is because of the objects small mass which meant even a single robot could greatly impact the objects angle. I believe that the other relatively small angular errors are to do with the circular shape of the object in the tests, as this has no vertices it becomes much harder to turn.

6.2.3 Number Of Robots

The number of robots in the simulation was experimented with and below lie the results.

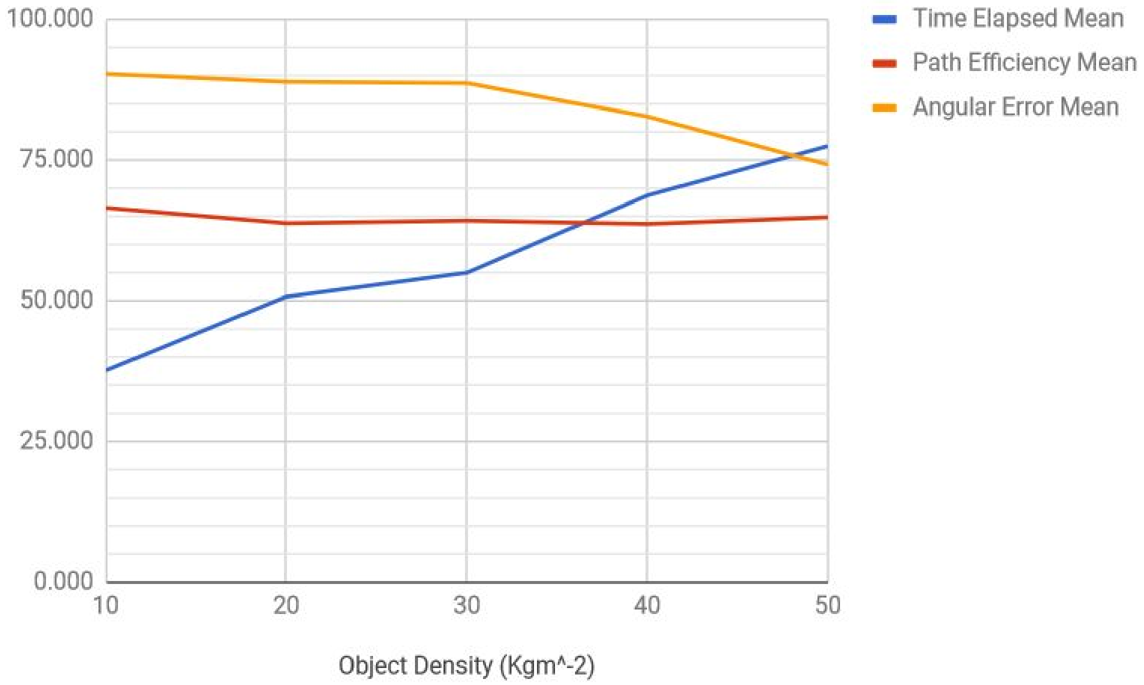


The time elapsed seems to decline to a certain point at which it once again begins to increase. This is most likely caused by the obstacle avoidance built into the system, if too many robots are in the arena they may spend more time avoiding each other rather than exploring for the object. A similar although weaker trend is seen with the angular error, having too many robots on too small an object can cause problems when a robot tries to leave the pushing formation, this can lead it to apply unintentional forces to the object. The trend of the path efficiency doesn't seem strong enough to infer any relevant information.

Overall these experiments have shown that there is an optimum number of robots which is dependant on the arena and object size. This should be considered when trying to improve the running time of the algorithm as adding too many robots may lead to an adverse result.

6.2.4 Object Density

The object density is the mass of the object per unit size. In box2D this unit is defined as kilograms per meter squared. Increasing this value will cause the object to be heavier and thus will require more force to move, this should have an adverse effect on running time of the algorithm but the angular error and path efficiency are to be determined.



From the resultant graph it is clear that the time elapsed follows the predicted trend, the angular error is shown to decrease as the density increases. These two values are both impacted by the density as more force is required to apply velocity, this applies both to the linear and angular velocities of the object. For this test case the path efficiency doesn't seem to have a correlation to the object density.

6.3 Self Evaluation

This section of the evaluation will give an evaluation of the entire project. This will include a review of the project aims and will attempt to identify the strengths and weaknesses of different components.

The project aims as laid out in the design are:

- To simulate the occlusion based transportation algorithm.
- A comparison of the simulation to the real world implementation.
- Automatic calculation of angular error and path efficiency.
- Changeable environment variables.
- A user interface displaying the simulation in real-time.
- An evaluation of how environment variables affect the algorithm.

The simulation of the occlusion based transportation algorithm was achieved, the simulation gives a visual representation of the working algorithm. While considered a success there are still some things which could be improved regarding this aim. I believe that the simulation could of been made to be more accurate, certain components from the real-world implementation were not added due to time constraints. An example of this would be the sensors, in the simulation only the front facing colour sensor is implemented, the other infrared sensors from the real world implementation were not able to be added.

The aim to produce a comparison to the real-world implementation was a success, a review of the differences in results between the two implementations was produced and reasoning was made to ascertain why any of these differences appeared.

The calculation of the angular error and path efficiency along with the time elapsed is displayed after clicking once the simulation completes.

The environment variables were originally planned to have their own screen to be setup before the simulation begins. Due to time constraints this screen was not implemented however these variables are setup in the main occlusion class so that they can be changed easily from within the code. Extra variables were also added which were not originally intended in the original design these include: friction, restitution and density.

The creation of a user interface was a success, the simulation is viewable in its entirety and has the robots change colour based on what behaviour they are performing to add extra clarity. Originally options to change the environment variables were planned to be added to the user interface however this is not yet implemented.

An evaluation of how each environment variable affected the system was produced in the previous section. It contains graphs showing the effect of these different changes and discusses why these changes occur. Originally I had planned to experiment with all the environment variables however towards the end of the project it became apparent that this was not feasible, as such I decided to only evaluate the variables which I believed were going to be statistically interesting.

Overall I believe that the project objectives were met satisfactorily based on the number of objectives I outlined and completed and my general satisfaction with how each of them turned out

In order to improve the effectiveness of the project I could of better managed my time. Whilst this was originally not a problem, the setbacks incurred during the project meant I needed to work more in a shorter period of time which I sometimes failed to do.

One of the main challenges when working on this project was working with the new programming environments. I overcame this by spending extra time finding good tutorials and looking through the documentation when I was confused about how components were meant to function. Another challenge was writing the dissertation itself, i’ve never had to write such a detailed report on a project before and being able to constantly envision the bigger picture was difficult at times. To overcome this I had to use my time-keeping skills in order to ensure each section was completed in time and then went back over it to improve its quality and make sure it was relevant to the project as a whole.

7. LEARNING POINTS

7.1 Time Management and Planning

The amount of work and planning required to design and implement a project of this size was definitely a challenge which I have learnt a lot from. From the conception of the project to the final implementation a lot of time management was needed. Whilst I originally had a plan laid out problems during development led to me having less time than expected, though this made the project hard to manage at times I believe that I have learnt a lot about how to manage a large workload and how to adapt to changes that can occur during a project.

One mistake I made in the implementation was to keep going back to modules and modifying them. This led to me having less time towards the end of the project to work on the incomplete modules. Looking back on this I have learnt to ensure the system works as a whole before going back to improve on individual modules.

7.2 Research Skills

Throughout the project a lot of research was done to explore the different areas of the project. Going through these articles to collect and collate data was an interesting task which has improved my analytical skills. I have also learnt about how these articles are structured which has helped me to quicker understand them and to also structure my own writing.

Whilst I am quite happy with how the research towards the end of the project went I would, if doing the project again I would spend more time doing research during the planning phase. A lot of the problems encountered during this project were due to not having a full understanding of the limitations of the environments I was working in.

7.3 Writing and Presentation Skills

Writing the specification, design and the dissertation was a difficult task for me. My writing skills are not the strongest and being able to convey this much information in a professional manner was definitely a challenge. During my time writing these documents I believe that my writing has improved in both content and structure.

Two presentations were required for the project, one describing the project design and another demonstrating the implementation. My first presentation gave me an idea of how to setup a presentation and the kind of questions that would asked, this gave me more confidence in designing and presenting the second on which is reflected in my marks.

7.4 Technical Skills

During the project I had to work in development environments which I had no experience in. Having to learn new methods and properties of this environment was challenging and while it may have slowed development of the project down it did help me develop skills to deal with adapt to such a situation. When working in new development environments I am now much more equipped to find information pertaining to problems I may be having.

7. PROFESSIONAL ISSUES

The British Computer Society outlines a Code of Practice [15] and a Code of Conduct [16] to govern the responsibilities of an individual whilst developing software. Throughout this project I have followed these codes and this section will discuss the various standards this project conforms to.

Section 3.1 Code of Practice

"Make realistic estimates of the costs, timescales and resource requirements, wherever possible basing your estimates on recognised methods and/or experience of delivering similar solutions."

During the design of the project an allocation of time for different components was produced. Estimates of the time needed to develop these components was originally based on code found in the Netlogo library of projects based on how much code was needed to implement a specific feature. When the design was revised the programming the tutorials for processing and box2D found in [14] was used to estimate the time needed for each component.

"Provide early warning of any possible overrun to budget or timeline, so that appropriate actions can be taken."

Midway through the project a late design decision set the project back. To make sure the project could be completed on time, certain components had to be removed from the final implementation. I think this is an example of an appropriate action to take to prevent overrunning the deadline.

Section 2C Code of Conduct

"C. develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field."

As discussed in the section on learning points of this project, I believe that I have improved in a wide variety of skills; this includes learning how to develop in new programming environments, writing and communication skills.

Section 2E Code of Conduct

“E. respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.”

During the project I have received a lot of feedback from my project supervisors and engaged in fortnightly meetings which I have used to better my subsequent work. As the project progressed there was a discernible increase in the quality of work produced.

8. BIBLIOGRAPHY

[1] L. Bayindir and E. Sahin.
A Review of Studies in Swarm Robotics.
Turkish Journal Electronic Engineering, vol.15, no.2, 2007.

[2] M. Brambilla, E. Ferrante, M. Birattari and M. Dorigo.
Swarm robotics: A review from the swarm engineering perspective.
IRIDIA Technical Report Series, Technical Report No.TR/IRIDIA/2012-014, 2012.

[3] J. Chen, M. Gauci, W. Li, A. Kolling, R. Groß.
Occlusion-Based Cooperative Transport with a Swarm of Miniature Mobile Robots.
EEE Transactions on Robotics, Volume: 31 Issue: 2, 2015.

[4] D. Milner.
Swarm Robotics Algorithms: A Survey.
2007.

[5] Arai, Tamio, Enrico Pagello, and Lynne E. Parker.
Advances in multi-robot systems. *IEEE Transactions on robotics and automation* 18.5 (2002): 655-661.

[6] S. Nouyan, R. Groß, M. Bonani, F. Mondada, and M. Dorigo.
Group transport along a robot chain in a self-organised robot colony. In Proc. of the 9th Int. Conf. on Intelligent Autonomous Systems (2006), IOS Press, Amsterdam, The Netherlands, pp. 433–442.

[7] J. C. Barca, and Y. A. Sekercioglu.
"Swarm robotics reviewed." *Robotica* 31.3 (2013): 345-359.

[8] U. Wilensky.
NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. (1999).

[9] Z. D. Wang, and K. Vijay.
"Object closure and manipulation by multiple cooperating mobile robots." *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 1. IEEE, 2002.

[10] M. Rubenstein, et al.
"Collective transport of complex objects by simple robots: theory and experiments." *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[11] Y. Hasan. "Swarms Robots and their applications."

[12] Processing Foundation.
Processing, “Processing.org”. Accessed 02.05.2018

[13] E. Catto.
Box2D, “box2d.org”. Accessed 02.05.2018

[14] D. Shiffman.
The Nature of Code: Simulating Natural Systems with Processing. Daniel Shiffman, 2012.

[15] The British Computer Society. Code of Good Practice.
http://www.bcs.org/upload/pdf/cop.pdf

[16] The British Computer Society. Code of Good Conduct.
https://www.bcs.org/upload/pdf/conduct.pdf

9. APPENDICES

9.1 Testing

Random Walk

Method	Expected Result	Actual Result	Pass/Fail
Random Walk V1	Robot applies a random velocity at every step.	Robot applies a random velocity at every step.	Pass
Random Walk V2	Robot applies a random acceleration at every step.	Robot applies a random acceleration at every step.	Pass
Random Walk V3	Robot goes forward and applies small torque at each step.	Robot goes forward and applies small torque at each step. Robot gets stuck when driving into wall.	Fail
Random Walk V3	Robot goes forward, applies small torque at each step or a large torque if boundary or another robot is detected.	Robot goes forward, applies small torque at each step or a large torque if obstacle detected. Robot get stuck spinning if it detects boundary in corner.	Fail
Random Walk V3	Increased angular damping should prevent robot from spinning when	Random walk works as expected.	Pass

	detecting boundary in corner.		
Random Walk V3	Goal added to arena, robot should avoid this as well as other robots and the boundary.	Random walk works as expected.	Pass

Circle Object

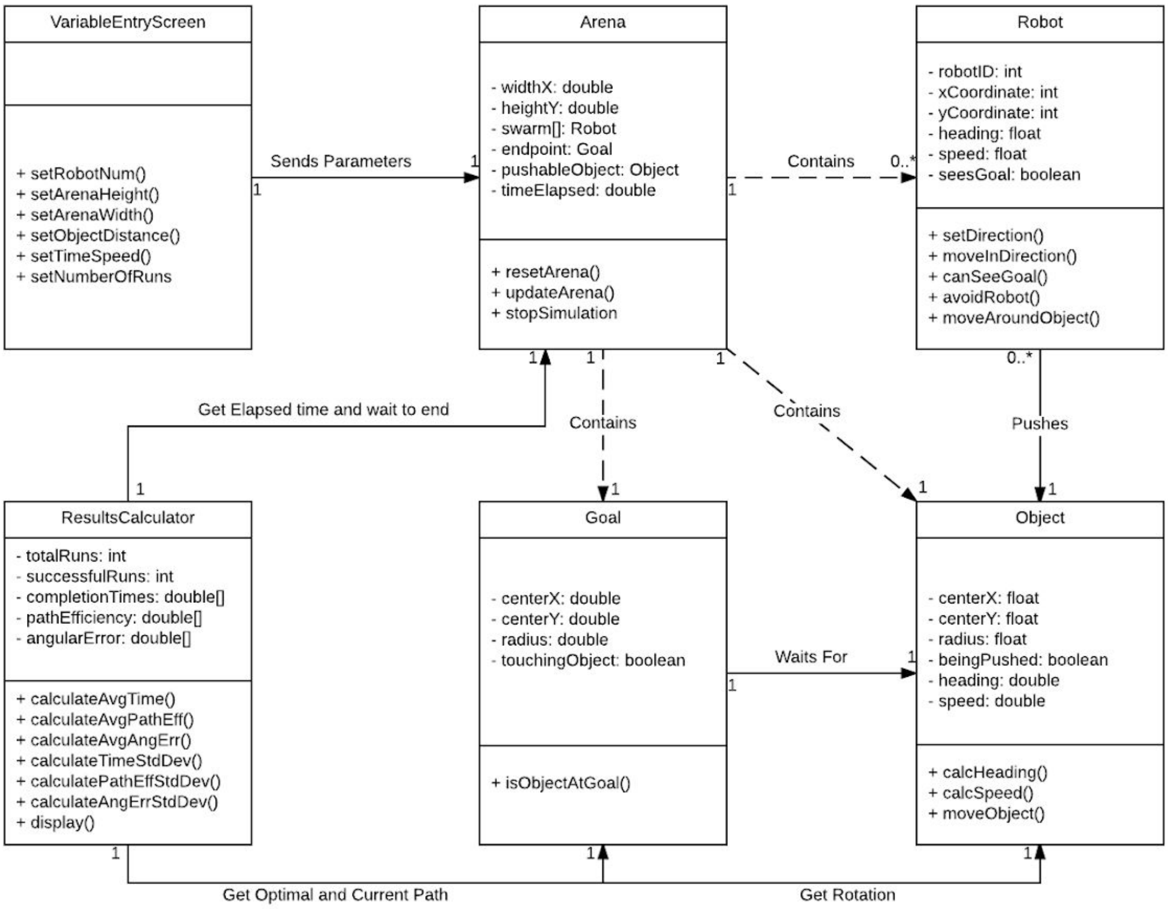
Method	Expected Result	Actual Result	Pass/Fail
Circle Object V1	Robot uses circular motion to travel around the pushable object.	Robot begins to travel in a circles however does not account for collisions which can send it off course	Fail
Circle Object V1	Robot uses circular motion to travel around the pushable object. And waits if an object is detected ahead	Robot travels in a circle correctly however the radius of this circles doesn't account for the movement of the object	Fail
Circle Object V2	Robot should travel around the object remaining within a certain distance.	When robot gets too far away it gets stuck spinning.	Fail

Circle Object V2	Trajectory added to robot, torque applied towards object when this leads out of range.	Torque always being applied to robot.	Fail
Circle Object V2	Units of trajectory converted from pixels to meters.	Method works as expected.	Pass

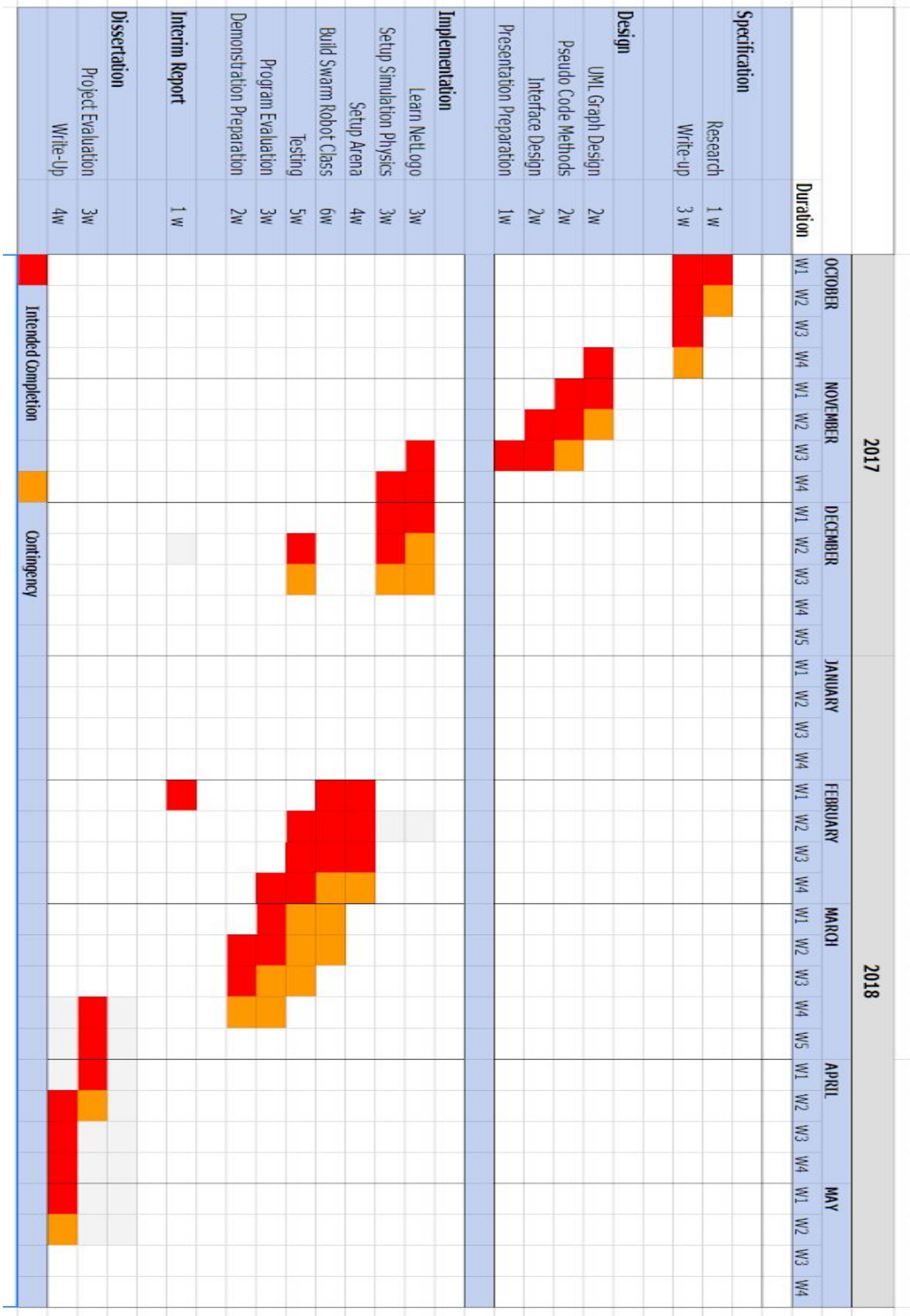
Line of Sight

Method	Expected Result	Actual Result	Pass/Fail
Robot Raycast	Robot should detect objects ahead and avoid them.	Robot always tries to avoid object even if nothing is there.	Fail
Robot Raycast	Robot should detect objects ahead and avoid them. - Units converted from pixels to meters.	Robot detects objects ahead and avoid them.	Pass
Goal Raycast	Robot should detect if its line of sight to the goal is obstructed by the object and head towards the goal if this true.	Robot should detects if its line of sight to the goal is obstructed by the object and heads towards the goal if true.	Pass

9.2 Previous Class Diagram



9.3 Gantt Chart


OCTOBER
NOVEMBER
DECEMBER
JANUARY
FEBRUARY
MARCH
APRIL
MAY
W1 W2 W3 W4 W5
Duration
Specification
Research 1 w
Write-up 3 w
Design
UML Graph Design 2w
Pseudo Code Methods 2w
Interface Design 2w
Presentation Preparation 1w
Implementation
Learn NetLogo 3w
Setup Simulation Physics 3w
Setup Arena 4w
Build Swarm Robot Class 6w
Testing 5w
Program Evaluation 3w
Demonstration Preparation 2w
Interim Report 1 w
Dissertation
Project Evaluation 3w
Write-Up 4w
Intended Completion
Contingency