Robotics Report

Group Coursework

Riccardo Mangiapelo
Jack Alexander Moore
Prakash Waghela
Inthuch Therdchanakul
Jason Owusu-Afriyie
Matthew Turner
Matthew Coates

16 December 2016

Part A

Reactive Braitenberg behaviours

Design

In this part of the coursework, we have attempted to recreate four Braitenberg behaviours: aggression, fear, love and curiosity. Such behaviours can be achieved by controlling the speed of the robot (i.e E-Puck) based on the sensor detection values. In fact, as explained by Braitenberg (1984), it is possible to simulate 'simple' behaviours by increasing/decreasing the speed of the motors for high value sensors. By taking into account a vehicle with two front sensors, one on each side, and two motors, right and left (Figure 1) it is possible to have two kind of vehicles, depending on whether the sensor to the motor is connected on the same side (a) or on the opposite side (b). Using such design, it is possible to make the vehicle respond in different ways in the presence of a stimulus. In fact, as Braitenberg (1984) illustrates, by having a positive influence sign, the vehicle will tend to accelerate when the sensors are excited. If the source is directly ahead, it may hit the object. However, if the source is on one side, one sensor is excited more than the other, resulting in different behaviours (observable in Figure 2). Vehicle (a) would accelerate the wheel parallel to the activated sensor, eventually running away from the detected object. This behaviour is named *Fear*. On the contrary, if the object is on one side of vehicle (b), this will turn toward the object and ultimately hit it. Such behaviour is called *Aggression*.

By switching influence sign from positive to negative, the vehicles respond differently to a stimulus, resulting in a new set of behaviours. By having a negative influence sign, the motor attached to the activated sensor will slow down in the presence of a stimulus. As it can be observed in Figure 3, when approaching the source, vehicle (a) will decrease the speed of the parallel sensor, eventually stopping close to it. Such behaviour is called *Love.* On the other hand, by having the sensor-

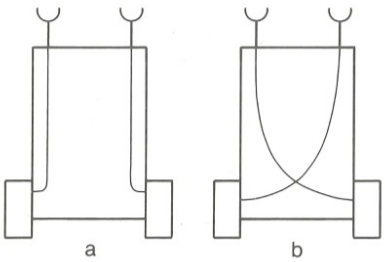


Figure 1: *Vehicle 2* (Braitenberg, 1984, p.7) with two motors and two sensors.

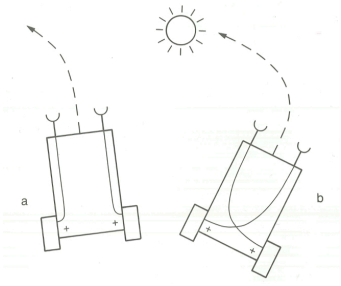


Figure 2: *Vehicle 2a and 2b* (Braitenberg, 1984, p.8) simulating Fear and Aggression.

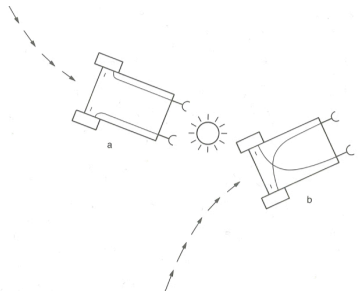


Figure 3: *Vehicle 3a and 3b* (Braitenberg, 1984, p.11) simulating Love and Curiosity.

motor control crossed, the vehicle decreases its speed in the presence of the source while turning away. This will result in an increase of the speed as the vehicle gets away from the source (Braitenberg, 1984). This is referred to as *Explorer* (or *Curiosity*).

All the four behaviours have been successfully implemented in the E-Puck robot. In order to make the robot exhibit the behaviours, we used four different selectors:

- selector 0: fear;
- selector 1: curiosity;
- selector 2: aggression;
- selector 3: love.

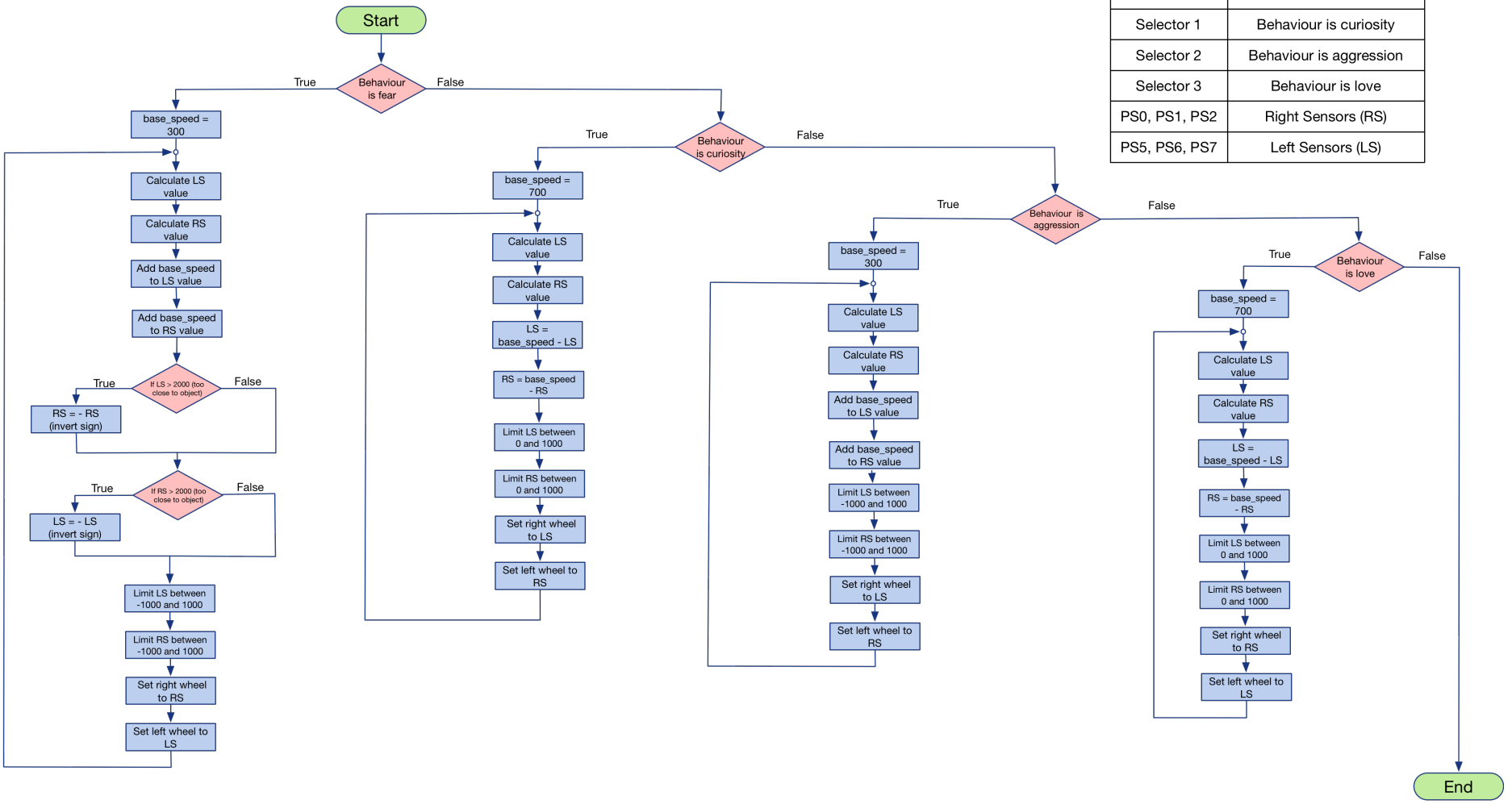
Consequently, the robot is only capable of representing one behaviour at the time. This results in the robot equally treating every detected object according to the selected behaviour.

Based upon Braitenberg's approach, we classified E-Puck's proximity sensors into two categories: "right sensors" and "left sensors". Under this scheme, we considered the top-right proximity sensors of the E-Puck (PS0, PS1, PS2) to be "right sensors" and the top-left ones (PS5, PS6, PS7) to be "left sensors". The reason for neglecting to use the rear sensors (PS3, PS4) is to prevent the robot from behaving eccentrically while moving away from the source (for instance, being attracted back to the object). Logically, this could be explained by thinking that the robot could only sense what is in front of it.

The left and right sensors are used to constantly monitor the adjacency of the robot's surroundings, returning high values when in proximity of an object. To such values is then added a basic speed, which is set differently for each behaviour. The result is then used to adjust the corresponding wheel's speed, according to the behaviour. In the case of *Fear*, these values are further used to examine whether the robot is excessively close to an object. In such cases, then the sign of the values are inverted, allowing the robot to move backwards slightly, exhibiting the behaviour correctly. A more detailed insight of the process can be observed in the flowchart overleaf.

Flowchart and function description

E-Puck Hardware	Flowchart terminology
Selector 0	Behaviour is fear
Selector 1	Behaviour is curiosity
Selector 2	Behaviour is aggression
Selector 3	Behaviour is love
PS0, PS1, PS2	Right Sensors (RS)
PS5, PS6, PS7	Left Sensors (LS)



Description of the main program functions for Part A:

fear()

This function contains the *Fear* behaviour. The basic speed, the speed of the wheels when there is no sensory input, is set to 300. As a result, when the robot meets an object, the increase on the speed is more noticeable. The maximum speed of the wheel is set to 1000, while the minimum is set to -1000 (this represent the fastest speed the wheels can go in reverse). After having detected the proximity values for each sensor set (left and right), each side's sensor value is set to be the sensor value itself added to the basic speed. The right wheel speed is then set to the newly calculated right sensor value; similarly, the newly calculated left sensor value is assigned to the left wheel. However, if any of these two values are over a limit of 2000 (meaning that the object is extremely close to the robot) the sign of the value is inverted. This allows the robot to move backwards slightly, exhibiting the behaviour correctly. Hence, as the sensory value for a sensor on one side increases, the motor for the same side of the robot will increase in speed.

curiosity()

This function contains the *Curiosity* behaviour. The basic speed is set to 700, in order to make the robot relatively fast when it is not detecting any objects. By doing so, the reduction of the speed is more noticeable when the robot approaches an object. The maximum speed of the wheel is set to 1000, while the minimum is set to 0 (this represent the fastest speed the wheels can go in reverse). After having detected the proximity values for each sensor set (left and right), each side's sensor value is set to be the sensor value itself subtracted by the basic speed. The right wheel speed is then set to the newly calculated left sensor value; similarly, the newly calculated right sensor value is assigned to the left wheel. Hence, as the sensory values increase at the front, the robot will slow down. This will make the robot repel the object once close enough, speeding up as it moves away.

aggression()

This function contains the *Aggression* behaviour. As per the fear() function, the basic speed is set to 300, while the maximum and minimum speed of the wheel is set to 1000 and -1000 respectively. After having detected the proximity values for each sensor set (left and right), each side's sensor value is set to be the sensor value itself added to the basic speed. The right wheel speed is then set to the newly calculated left sensor value; similarly, the newly calculated right sensor value is assigned to the left wheel. As a result, as the sensory value for a sensor on one side increases, the motor for the opposite side of the robot will increase in speed.

love()

This function contains the *Love* behaviour. As per the curiosity() function, the basic speed is set to 700, while the maximum and minimum speed of the wheel is set to 1000 and 0 respectively. After having detected the proximity values for each sensor set (left and right), each side's sensor value is set to be the sensor value itself subtracted by the basic speed. The right wheel speed is then set to the newly calculated right sensor value; similarly, the newly calculated left sensor value is assigned to the left wheel. Therefore, as the sensory values increase, the robot is 'attracted' to the object, causing the increase. The robot also slows down as it gets closer. After getting close enough, the robot will stop and perform a signal of 'love' (lighting up the body lights).

Test procedure and result analysis

In order to test the correctness of each behaviour, no special environment's set up was required. It has been sufficient to place the robot on a straight surface (e.g. a table) long enough for the robot to safely navigate on it. A handful of objects were then positioned on the robot's path in order to observe the response of the vehicle to the object.

The results have fulfil expectations. Each behaviour is exhibited correctly from the robot, as per behaviour explanation in section Design.

Conclusion

The experiment in this Task simulated four Braitenberg's behaviour (fear, curiosity, aggression, love) on an E-Puck robot. By adjusting the wheels speed according to the sensor values and by changing the influence sign from positive to negative, we have been able to create the set of behaviours required. Upon the correct choice of the selector, the E-Puck is now capable of reacting differently when approaching obstacles.

Part B

Goal seeking and obstacle avoidance

Design

Our task for this section of the coursework was to program a robot that would find its way to a specified goal whilst avoiding obstacles.

Algorithm Design

For this task we decided to implement the Bug-2 algorithm as explained in the lectures. The idea behind the Bug-2 algorithm is to plot a line from the start point to the goal (i.e. the m-line) and move along that line towards the goal. If the robot encounters an object, it would follow its perimeter until it reaches a closer point of the m-line than it was before. When this happens, the robot will stop following the perimeter of the object and continue navigating along the m-line towards the goal (Figure 4). The Bug-2 algorithm is a greedy algorithm and generally outperforms the Bug-1 algorithm. However, this is not always the case. For instance, as it can be observed in Figure 5, if there is one object intersecting the m-line, following the perimeter would cause the robot to circle the object (Figure 5).

Position Tracking

The method used to track the position of the robot was to implement a coordinates system in the horizontal (x) and vertical (y) plane. This allowed us to specify the start

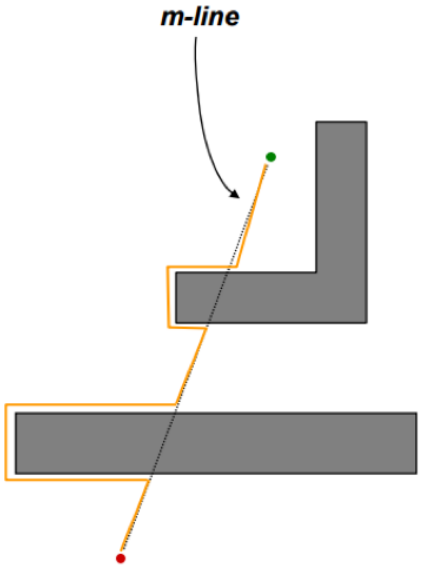


Figure 4: *Bug-2 Algorithm.*

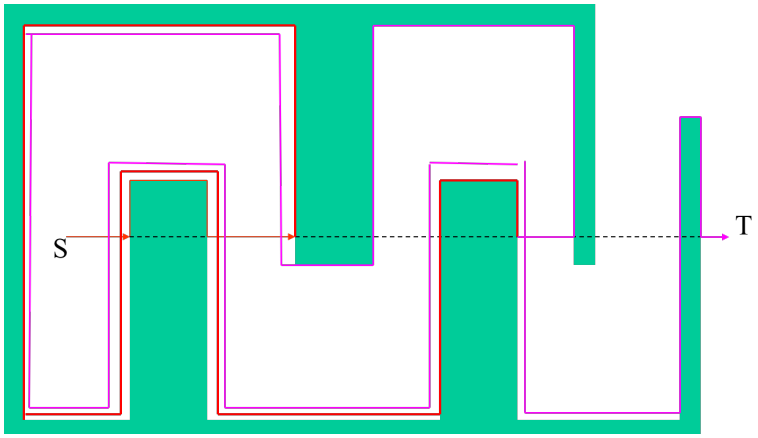


Figure 5: *Bug-2 Algorithm with m-line from start (S) to goal state (T).*

and goal position as any arbitrary (x, y) co-ordinate point. For simplicity, we assumed that the start position would always be the point (0, 0).

The change in position was calculated using trigonometry. Assuming the robot has three distinct methods of moving (linear, rotating left, rotating right) where the speed on each motor is either identical or inverted, we are able to keep count of the number of rotational steps and the number of linear steps moved by the robot.

The rotation steps (s) can be converted into an angle with the following formula:

$$angle = \frac{\frac{s}{1000} \cdot Wd \cdot \pi}{Rd \cdot \pi} \cdot 2\pi$$

where *Wd* is the diameter of the wheel and *Rd* is the diameter of the robot. This formula calculates the distance travelled in steps and uses that in a ratio over the circumference of the robot to calculate the angle turned. Thus by keeping count of the number of steps rotated, we are able to track the current orientation of the e-puck. The linear steps (h) can be used to calculate the change in x and y position with the following formulae:

$$x = h \cdot \sin(angle)$$

$$y = h \cdot \cos(angle)$$

where *angle* is the current orientation of the robot, as calculated above. When moving in a linear motion, the robot will continually update its position. By monitoring this position, we are able to detect if the robot has moved to the goal state. Besides, by outputting these co-ordinate values, we have been able to visualise the movement of the robot (as shown in Figure 6).

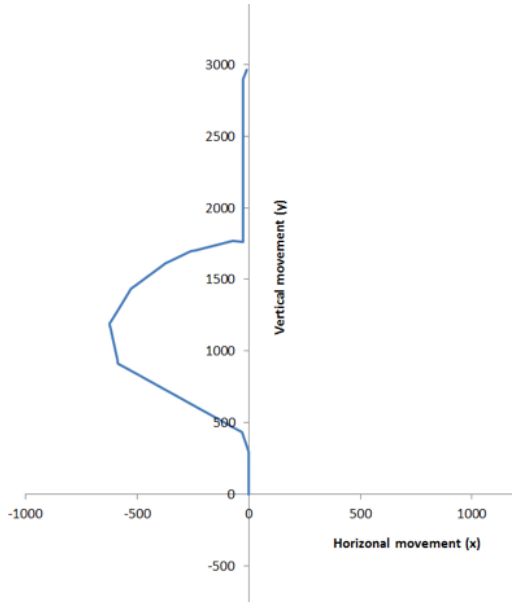


Figure 6: *Visual movement of the robot.*

Object Avoidance

To detect an obstacle while travelling along the m-line, we average the value of the front sensors. If such value is above a set threshold, an object has been detected. Hence, the robot begins rotating left since we opted for a left-leaning variant of the algorithm. The robot will rotate until it is roughly perpendicular to the object, at which point it will continue moving forwards until the average of the right hand side sensors no longer detect the object. The robot would then rotate right and move forwards, keeping roughly perpendicular with the object.

This process is repeated until either the object is completely circled or the m-line is detected. If the latter happens and the robot's position is closer to the goal than when it started tracing the object, then it moves on towards the goal. In the case that the obstacle completely blocks the path to the goal, the robot will circle the entire perimeter of the obstacle in an exhaustive manner until it has reached the position at which it entered the boundary. At this point it time, it will stop and report that no solution exists.

State Indication

Table 1 indicates the different states possible through the execution of this task and how the robot will indicate such states.

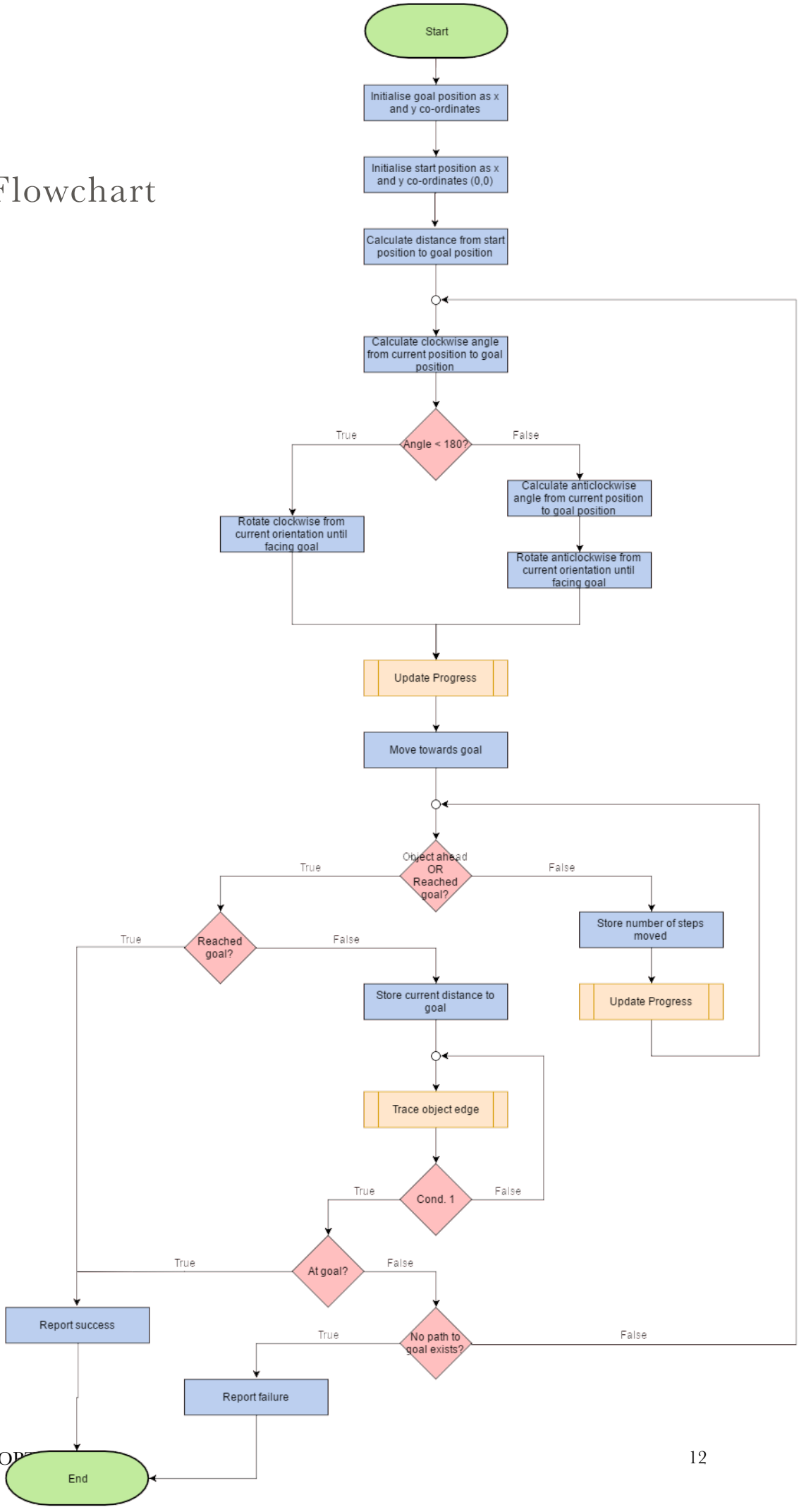
Table 1

State	Description	Indication
Moving toward goal	The robot is on the m-line, moving toward the goal. It is monitoring the space ahead of the robot for any objects.	No LEDS lit on the robot, robot moves toward goal.
Object found, avoiding	The robot has detected an object and is currently attempting to avoid it. It is monitoring its own position to check whether it intersects the m-line.	Back led (LED4) lit on the robot; robot avoids the obstacle by turning left at a boundary.
Object avoided, m-line detected	The robot is currently avoiding an object and has manoeuvred to a point that is on the m-line.	Back led (LED4) is lit and front led (LED0) will blink. This indication will be brief as the state will shift once the m-line is detected.
M-line point is improvement	The point on the m-line found is an improvement and the robot can continue moving towards the goal.	Front led (LED0) stays lit and robot will rotate toward goal.
Goal reached	The robot has manoeuvred to the specified goal.	All leds lit and robot is stationary.
Goal is not reachable.	The robot has manoeuvred around the object and cannot find a path to the goal.	Back led (LED4) is lit and robot is stationary.

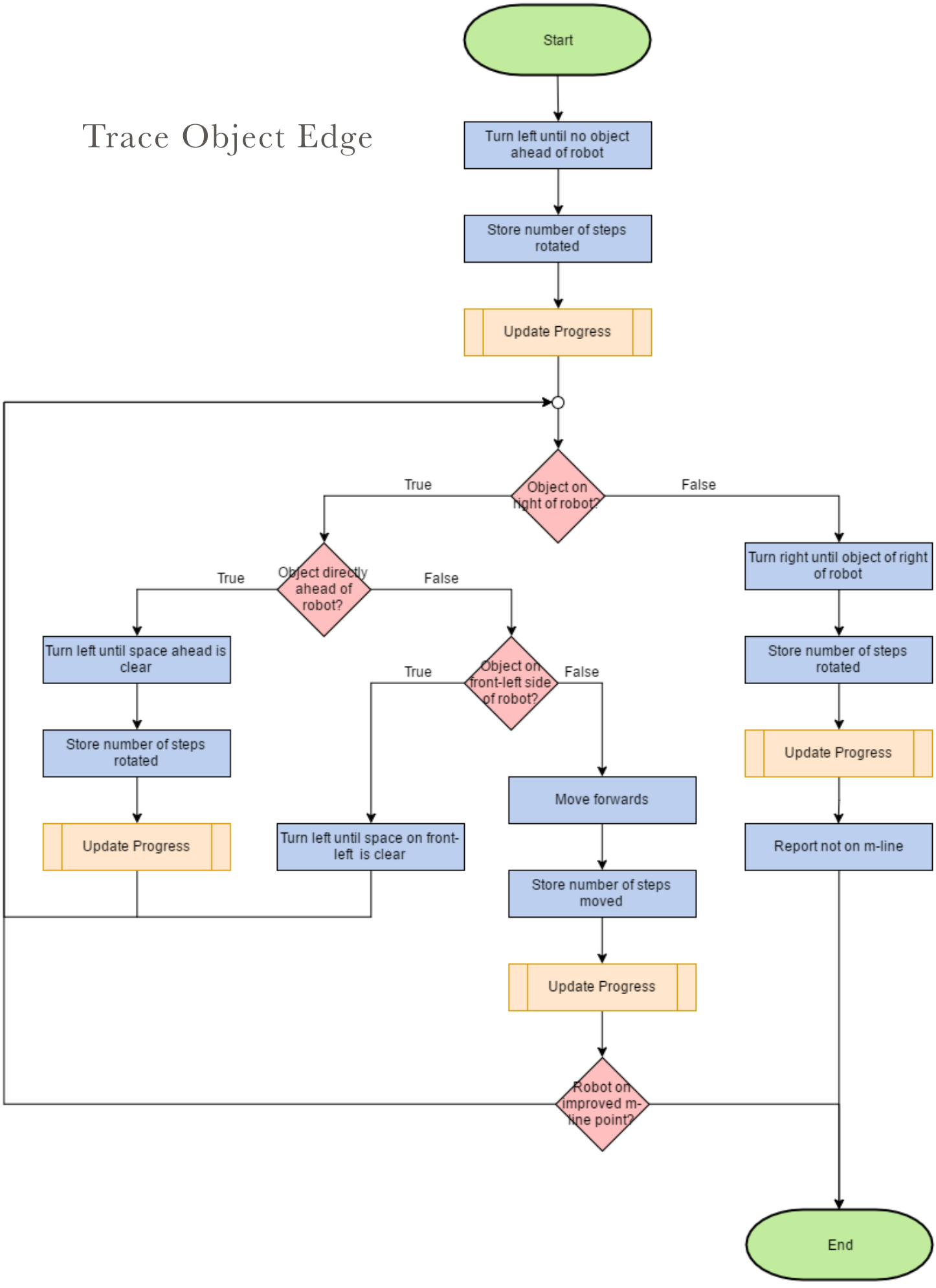
Flowchart and function description

Overleaf. The *Main Flowchart* is the flowchart describing the main process for Task B. In this flowchart, there are several sub-procedures which are described in separate flowchart (on following pages). Such sub-procedures are *Trace Object Edge* and *Update Progress*.

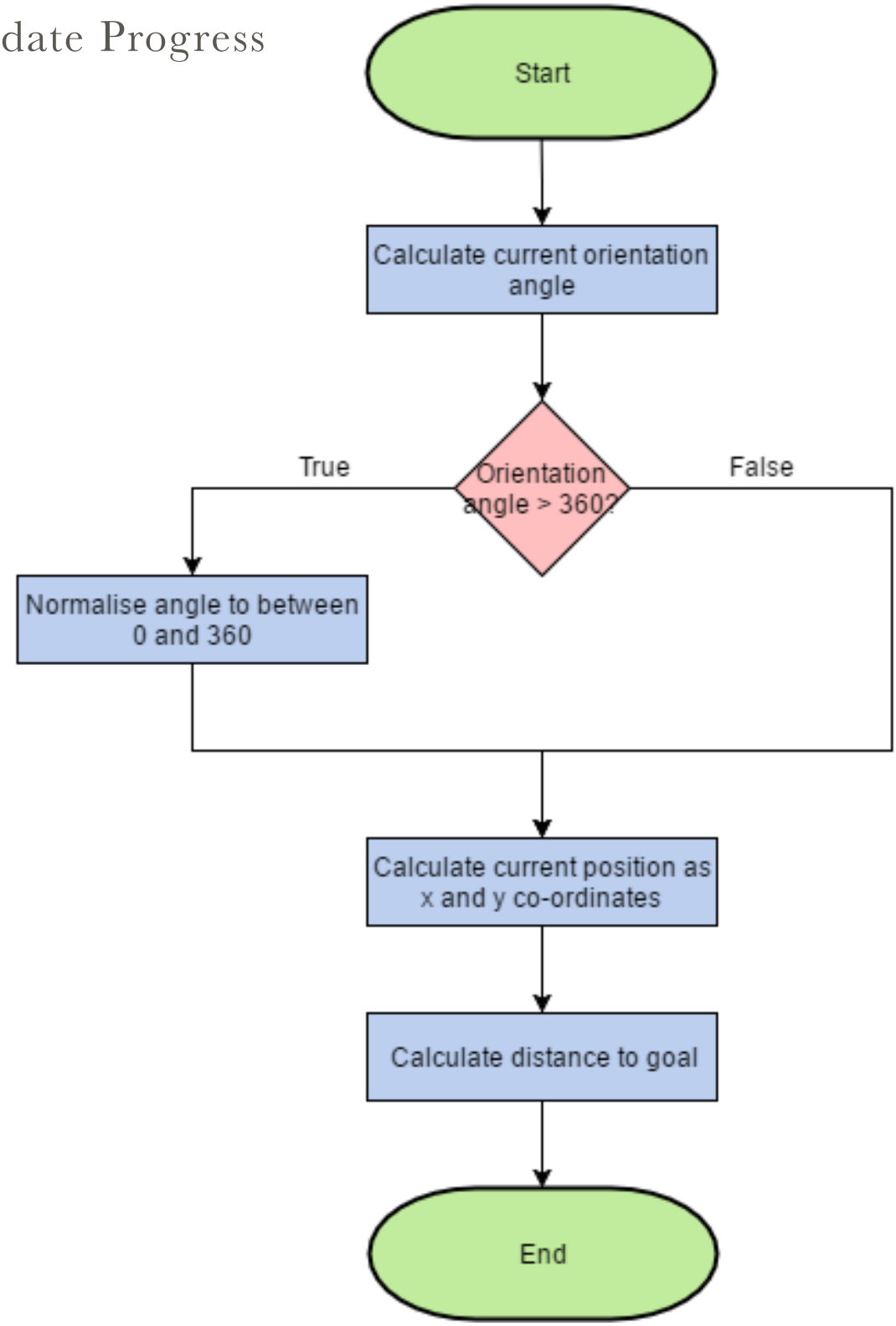
Main Flowchart



Trace Object Edge



Update Progress



Description of the main program functions for Part B:

distToGoal()

We use this function to calculate the distance from where the robot currently is to the goal. To do this we calculate the distance away from the goal in the x-direction and y-direction, and then use Pythagoras's theorem to get the direct straight line distance.

stepsToRotate()

This is a utility function where we give the robot the angle in degrees we want to rotate clockwise, and it returns the number of steps the wheels have to turn to rotate that far. To do this we know that a full rotation of the wheel is 1000 steps, and we calculate how far 1000 steps will rotate us by using the diameter of the wheels and the diameter of the robot.

angleFromGoal()

This function gives us the angle we need to rotate clockwise by to face the goal, taking into account the angle we are currently rotated at. We work out the distance from the goal in the x direction and the y direction and then use trigonometry to get the angle we need to rotate by from 0. We then take away the angle we are currently rotated at to get the angle we need to rotate by in total. We can then pass this into our stepsToRotate() function to tell the wheels how far to turn.

distToMline()

We use this function to tell us if we are on the m-line from the start to the goal or not, if the distance away from any point on the line is less than a certain threshold then we say we are on the m-line.

setGoal()

This allows us to specify the goal position in x and y coordinates.

moveToGoal()

Called at the start of the program and again each time we have finished navigating around an object and have encountered the m-line again. Uses angleFromGoal() to find out how far it has to rotate to point towards the goal, taking into account if rotating clockwise or anticlockwise would be faster based on current

orientation, and then moves forwards towards the goal until it reaches it, or encounters an obstacle.

updateProgress()

This function is called after every time we have finished moving somewhere, such as when we encounter an object. It takes the distance we have travelled in steps since we last finished moving, and the angle we are currently rotated by, and uses trigonometry to calculate how far we have moved in the x and y directions, and adds that to our current position.

avoidBoundary()

This function runs as soon as the robot detects an object in the space ahead of it. It takes the distance to the goal as an input and will output whether or not the robot is on the m-line. First the robot is rotated left until it is roughly parallel to the object, then it will continue moving around the object until it reaches the m-line, or it has reached its start location again.

pathfinder()

Contains the main loop for the bug-2 algorithm. It will keep moving forwards towards the goal using the moveToGoal() function until it either reaches the goal or the front sensors detect an obstacle. If the robot reaches the goal then it stops and lights up all of the LEDs to let you know. If it reaches an object it uses our avoidBoundary() function to navigate around it. Once the robot has navigated around the object and has detected that it is on the m-line, then it will call the moveToGoal() function again to move towards the goal. If the robot does not detect a point on the m-line around the object that is closer to the goal than when it started, then it will stop as it cannot reach the goal.

Test procedure and result analysis

In order to test the correct behaviour of the robot, we tested each subpart of the program individually. This allowed us to partition the testing into subtests of each functionality. One of these included placing the robot on a flat surface, setting the goal to 1000 units away directly in front of the starting point (i.e. 1000 units in the y direction). By doing this, we observe the robot moving forwards towards the goal, stopping once at the goal position. This showed the correct behaviour of the functionality.

A further test we conducted was to evaluate the ability of the robot to move to a specified location while considering its current orientation. In order to do this, we tested the capability of the robot to move 1000 units up, right, down and left, ending in its initial location.

Further to the above, we tested a simple object avoidance behaviour by placing a single object in front of the robot. We observed that the robot would stop upon the detection (by the front sensors) of the obstacle. Next, the robot would turn and navigate around to the other side of the object.

Once the robot could reach a goal and successfully navigate around an object, we implemented the Bug-2 algorithm, combining these behaviours. We tested the Bug-2 algorithm using different sized obstacles in different orientations, and with the goal in different places. This showed the robot successfully navigating through the m-line, avoiding object(s) and finally stopping at the goal.

Below, Table 2 describes few tests that have been conducted in order to evaluate Part-B behaviour.

Table 2

Test ID	Description	Expected Result	Actual Result	Comments
1	Set the goal to the co-ordinates: (0, 1000)	Robot should move 1000 steps in the y direction.	As expected	Tests the position tracking of the robot.
2	Set the goal to the co-ordinates: (1000, 0)	Robot should move 1000 steps in the x direction.	As expected	Tests the position tracking of the robot.
3	Set the goal to the co-ordinates: (-1000, 0)	Robot should move -1000 steps in the y direction.	As expected	Tests the position tracking of the robot.
4	Set the goal to the co-ordinates: (0, -1000)	Robot should move -1000 steps in the x direction.	As expected	Tests the position tracking of the robot.
5	Combine the above 4 tests, running them one after another in a single program.	Robot should manoeuvre in a square, moving up, right, down then left eventually returning to the start position.	As expected	Tests the position tracking of the robot, taking into account the current orientation of the robot.
6	Have the robot moving toward the goal with an object blocking the path.	Robot should move towards the object and display a change in behaviour once the object is close enough to detect (in this case, stop moving).	As expected	Tests the object detection of the robot and change of state.
7	Have the robot moving toward the goal with an object blocking the path.	Robot should move towards the object and once it is close enough to detect it, it should turn left then follow the object boundary. It should leave a sensible amount of space between the robot and object at all times.	As expected	Tests the change in state and object avoidance.

Test ID	Description	Expected Result	Actual Result	Comments
8	Allow robot to follow boundary until it crosses the m-line.	Robot should follow the boundary, keeping the object on its right. Once the m-line is found, it should turn on the front led.	Front led blinks when m-line is found, rather than staying constant	Tests the m-line detection. Result is caused by the rapid change in state when the m-line is detected, thus actual result is acceptable.
9	Allow robot to follow boundary until it crosses the m-line. The m-line intersection is an improvement.	When m-line is found, the robot should check whether the m-line is an improvement. In this case, it should blink the front led then turn toward the goal and continue moving.	As expected	Tests the m-line detection and checks whether it is an improvement. Also tests change in state.
10	Allow robot to follow boundary until it crosses the m-line. The m-line point intersection is not an improvement.	When the m-line is found, the robot should check whether the m-line is an improvement. In this case, it should blink the front led but continue following the boundary.	As expected	Tests the m-line detection and checks whether it is an improvement. Also tests change in state.
11	Set goal to (0, 3000) with a single square box blocking the path to the goal.	The robot should initially move toward the goal. When it detects the object, it should turn left and trace the boundary of the object. It should find the m-line on the opposite side of the box that it entered from), turn toward the goal and continue moving until it reached the goal. At this point it should stop and turn all the leds on.	As expected	Overall and complete program test for a successful outcome.
12	Set goal to (0, 3000) with two square boxes blocking the path to the goal.	Same as above but both objects avoided.	As expected	Overall and complete program test for a successful outcome.
13	Set goal to (0, 3000) with the robot position in an enclosed space, such that the goal position is not reachable (goal is outside of an enclosed area or goal is covered by object).	The robot should initially move toward the goal. When it detects the object, it should turn left and trace the boundary of the object. It should detect the m-line (blink led) but continue tracing the object as it is not an improved point on the m-line. It should return to the point where it originally detected the boundary then stop moving, turning on the back led.	As expected.	Overall and complete program test for an unsuccessful outcome.

Test ID	Description	Expected Result	Actual Result	Comments
14	Set goal to (0, 3000) with a single square box blocking the path to the goal. Ensure terminal capture is open so robot can communicate with the pc.	Once the program has completed, the terminal capture should contain co-ordinate data sent from the robot at regular intervals. Plotting this data on a scatter graph should produce a trace of the robots path.	As expected .	Tests the e-puck – PC communication and position tracking.

Conclusion

The results that we have captured from our test plans are very positive. Out of all of the tests, only a single test did not give the expected result but in this case the actual result was still acceptable as the indication was still present and clear.

During repeat tests, we have seen that the accuracy of the position tracking and goal seeking varies on each test run. This is due to minor differences in the test settings which have a small impact on the values returned. For example, the light available will affect the values returned by the IR sensors, which in turn could cause the robot to hit the object. Also, as object avoidance is triggered and controlled by the IR sensors, there is a limitation on the performance with certain objects. For example, we observed that objects that have a reflective surface are not detected as well as objects with an opaque surface. However, as this is a factor we can control there was no need to take it into account while testing.

We have assumed that the environment the robot is traversing will be static from the start of the execution until it terminates. In some cases, if the object is removed while the robot is following its wall, the robot will rotate endlessly. However, this implementation falls outside the scope of this task so was not considered while testing. Therefore, we can conclude that the implementation for this task is sufficient for the task definition. We have observed that the robot is able to traverse around a static environment, moving towards a set goal and avoiding any obstacles along the way. We have also observed that the robot is able to detect when the goal is unreachable and will indicate this and terminate in this case.

Part C

High level behaviour

Design

In this part of the coursework, we have attempted to implement a high level behaviour which simulates cooperation between two robots (i.e E-Puck) in order to reach a common goal. This behaviour is inspired by the possible cooperation between two animals (e.g. ants) in order to execute certain tasks (e.g. locate food and bring it back to anthills by combining effort). Alternatively, this behaviour could be associated to actions that multiple Mars explorer robots would perform. The objective would be to explore a distant planet to collect samples of precious rocks, bringing such samples back to the mother ship spacecraft in order to go back to Earth. In such situation, the location of the samples is not known in advance, but there is a knowledge of the appearance of such elements. Furthermore, no detailed map of the planet is available.

Similarly, we designed a simple environment, unknown to the robots, where a desirable object is placed. The assumptions made are:

- the presence of the object in the environment;
- the lack of obstacles different than the walls.

As a result, the robots are aware that there is at least one desirable object in the environment, performing the search without questioning the presence of such item. Furthermore, the robots are aware of the appearance of the item. They acknowledge it's colour (green) and the proportion of the box (two sides large enough to fit both robot on each side in order to push it simultaneously).

The starting position of the two robots is fixed to be the bottom-left corner of the environment. This position is treated as the eventual goal position (equivalent to a mother ship / anthill). Initially, the two robots would be positioned one in front of the other, undertaking two different roles: *leader* and *follower*. As the terms indicate, the leader would be the robot in front, performing the search, scanning the environment and eventually guiding the partner to the object. The follower robot has a reflexive behaviour, where actions are executed based on the perception of a stimulus. More precisely, it is a *taxes reflexive behaviour*, where actions occur in response to visual phenomena (i.e. infrared information sent by the leader to the follower robot). This could be associated to behavioural responses that orient animals toward or away from

a stimulus, as in the case of ants following pheromone trail. Both robots are born with these sequence of innate behaviours.

At the start of the simulation, the front robot starts exploring the environment for the desirable object, going straight until a wall is sensed. Upon detection of such obstacle (accomplished using Proximity Sensors), the robot would turn 90°, move upward for its field of vision's length, turning a further 90° in the opposite direction with the purpose of starting the search again (as observable in Figure 5). This set of action is repeated until the object is detected. Once the robot is back in position, it starts navigating straight again. The robots repeat this set of behaviours until the object is detected (Figure 6).

Upon detection of such item, the leader robot would signal the companion, triggering a switch in the behaviours. The follower robot would stop in the current position, waiting few seconds in order to allow the leader move towards the item. The latter would scan the item, in order to calculate its orientation in relation to the goal state as well as the optimal position to assume in order to push the object. After few seconds, the follower would perform the same set of actions, finding the second optimal position to carry the object. Upon the correct positioning of each robot, they would send a signal to each other, indicating the end of the arrangement behaviour.

The remaining part of the robot's task is to bring the object back to the goal state. In order to do so, as it can be observed in Figure 7, they advance toward the ahead wall, pushing the object until they both sense the item being attached to the wall (achieved using the accelerometer). Promptly after perceiving the wall in front of the item, the robots would signal each other. Only when both robots have sensed the wall do they reposition themselves a final time on the perpendicular side of the box to the goal state. When both are in position (signalling each) they move further in order to enter the goal state. The robots accomplish their task once they both detect and signal the companion their presence in the start/goal area.

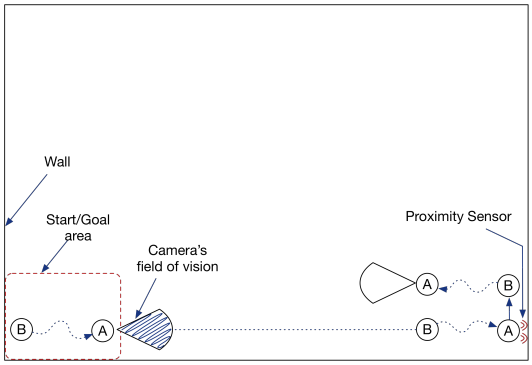


Figure 5: *Robot A* and *Robot B* starting in the Start/Goal area and navigating until sensing the wall.

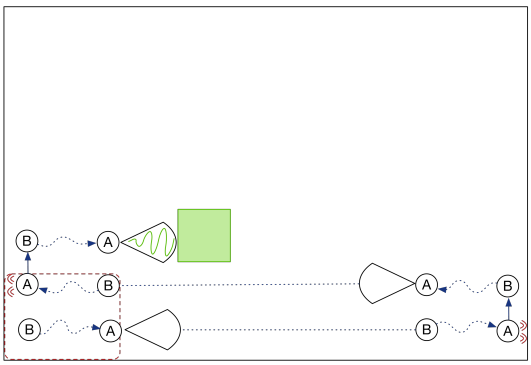


Figure 6: *Robot A* and *Robot B* scanning environment and finding object.

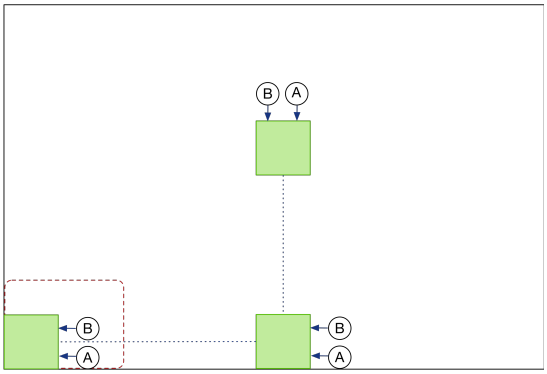


Figure 7: *Robot A* and *Robot B* driving the object back to goal state

In order to send information to each other, the robots use infrared communication. Initially, the objective was to make use of the Bluetooth transmission in order to have an increased range of connectivity. However, due to issues encountered in the use of the Bluetooth library (released by the manufacturer company but not transmitting information as expected), we re-planned some behaviours (i.e. searching and positioning) in order to make efficient use of the infrared transmission. As explained by E-Puck (2010), messages can be detected at a distance of up to 25 cm between emitter and receiver. As a consequence, the design of our behaviours maintain the robots within that range at every step of the simulation. A further constraint deriving from using the infrared transmission is the length of each message transferred between the robots. The latter is set to be 1 Byte of length for each message. As a result, we created an encoded scheme where each Bit of the message has a different meaning:

- Bits 1-2 are used to identify the type of command. In total, there are 4 types of commands (set state, broadcast X position, broadcast Y position, finish);
- Bits 3-8 are used as the payload for that command. The more important payloads are: no op state, successful acknowledge, unsuccessful acknowledge, propose leader behaviour, acknowledge leader behaviour, setup completed, tell other robot to follow on side, communicate other robot that you are left robot, communicate other robot that you are right robot.

Future Work

This high level behaviour could be extended further in order to receive optimal results. An improvement area could be the searching behaviour. Such action could be performed by both robots simultaneously, combining the camera's field of vision in

order to cover a broader area during the searching of the object. By doing so, the number of full scans (i.e. from side to side of the environment) would be decreased by half. Such potential behaviour can be observed in Figure 8.

A further improvement could be the methodology by which the robots bring the object back to the goal area. As shown in Figure 9, the robots could constantly adjust their speed in relation to distance and angle from their current position to the goal state. By doing so, they would drive the object into the goal state without having to reposition. As soon as any of the robots would locate itself in the goal area, it would signal the partner. By doing so, when both robots receive confirmation from the companion, they would determine the achievement of the task.

Flowchart and function description

The flowchart of the system for Part C is overleaf.

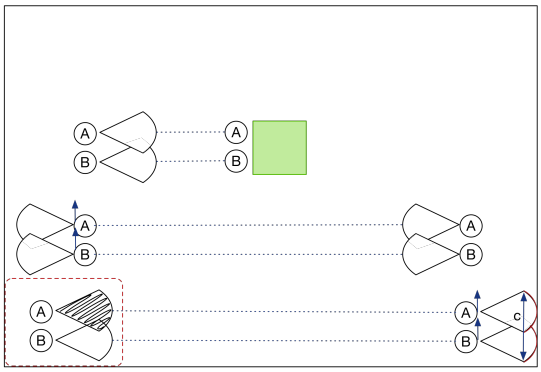


Figure 8: Possible solution for *Robot A* and *Robot B* possible combined search.

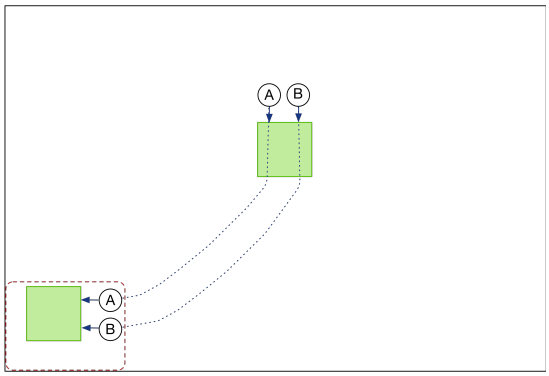
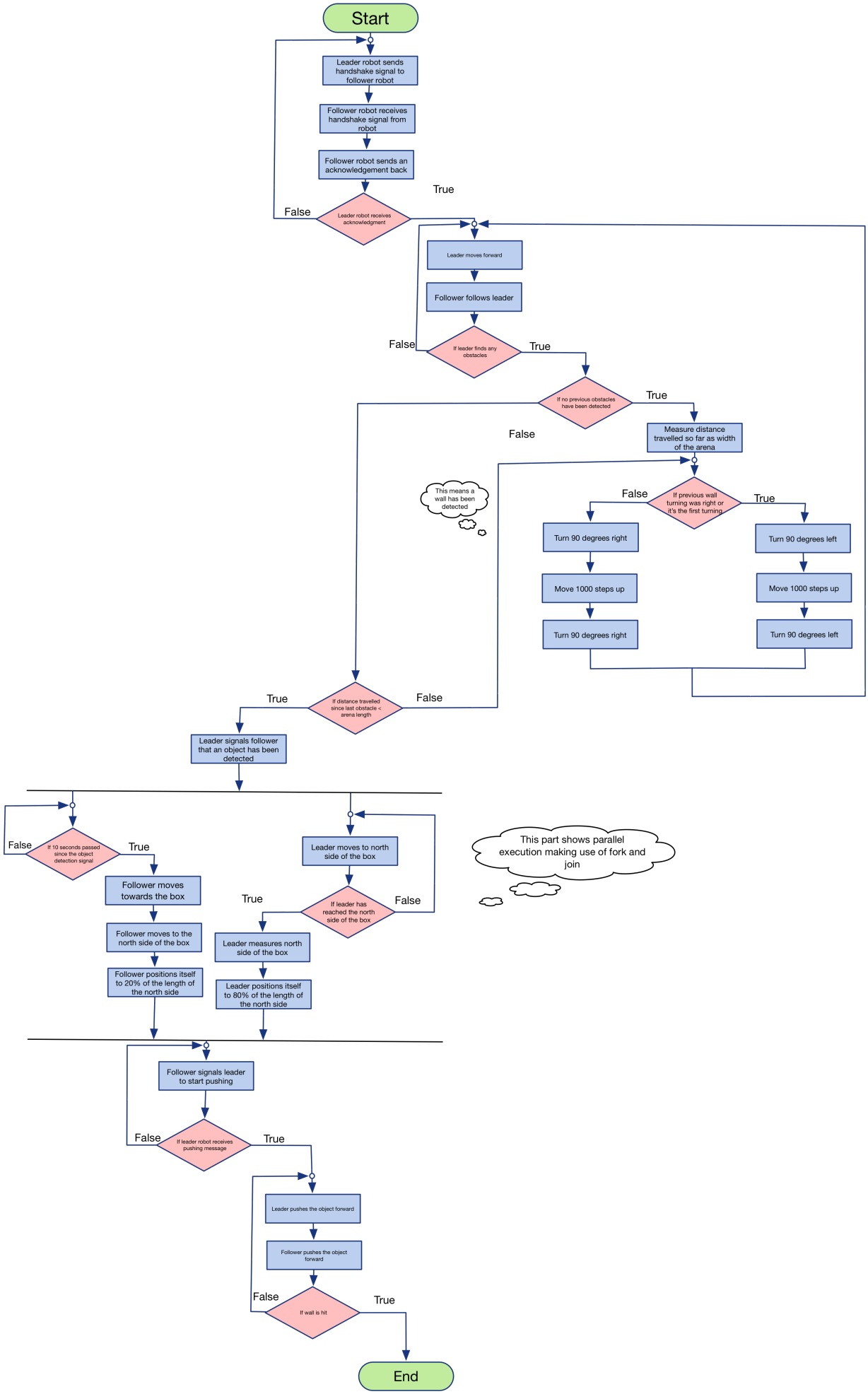


Figure 9: Possible solution for optimising the object return journey.


Start
Leader robot sends handshake signal to follower robot
Follower robot receives handshake signal from robot
Follower robot sends an acknowledgement back
Leader robot receives acknowledgment
True
False
Leader moves forward
Follower follows leader
If leader finds any obstacles
False
True
If no previous obstacles have been detected
True
False
Measure distance travelled so far as width of the arena
This means a wall has been detected
If previous wall turning was right or it's the first turning
False
True
Turn 90 degrees right
Move 1000 steps up
Turn 90 degrees right
Turn 90 degrees left
Move 1000 steps up
Turn 90 degrees left
If distance travelled since last obstacle < arena length
True
False
Leader signals follower that an object has been detected
If 10 seconds passed since the object detection signal
False
True
Follower moves towards the box
Follower moves to the north side of the box
Follower positions itself to 20% of the length of the north side
Leader moves to north side of the box
If leader has reached the north side of the box
True
False
Leader measures north side of the box
Leader positions itself to 80% of the length of the north side
This part shows parallel execution making use of fork and join
Follower signals leader to start pushing
If leader robot receives pushing message
False
True
Leader pushes the object forward
Follower pushes the object forward
If wall is hit
False
True
End

Description of the main program functions for Part C

emit()

This function allows messages to be emitted by an robot via Infrared (IR) communication. The messages sent are 1 byte long, and are defined by a message protocol defined in the program. This protocol is upheld by usage of the *packet.c* and *messages.h* files. Once a message has begun sending, no other message may be emitted until that one has completed. Due to the erroneous nature of infrared communication (not every message may be received every time), we used this function in an agenda to allow us to constantly emit the same message in an asynchronous fashion.

receive()

This function allows received messages by the IR library to be processed. The IR communication library uses interrupts in the background to constantly sample for incoming IR messages, and then throw these valid messages into a bounded queue. This function checks if there is a message on the queue, pops the message off, and processes it. Messages are also processed in accordance with the protocol outlined in *packet.c* and *messages.h.*

setPacketToEmit(int command, int payload)

This function is used to assign a packet to constantly emit via IR. This packet will contain the given *command* and the *payload* the the *emit* function should send. This is stored as a global state variable by the robot.

traverse()

This function is called to perform the "traverse" or "search" behaviour. This behaviour is used by the "leader" robot to search for an object, and avoid any walls in the process. It uses several sub-behaviours in a hierarchy that allows them to be combined to create this more complex behaviour. These sub-behaviours are, in hierarchical order: **"Avoid wall", "Locate object" and "Cruise".** If an object is verified as found by the **"Locate object"** sub-behaviour, then we exit this behaviour and move onto the next phase of the Part C behaviour.

runWallFollow()

This function is used by the **"Avoid wall"** sub-behaviour to follow alongside a wall. This is used for a given amount of time to allow us to move up the arena, and create a "lawnmower search" approach that is seen in the "traverse" behaviour. This

works by using the proximity values of the robot to avoid the wall, but also not get too far away from the wall.

pushBox()

This is used by the robots to push the given object once both robots are correctly positioned. This incorporates the ideas of Braitenberg's aggression behaviour by tying the front left and front right proximity value to the increasing speed of the right and left wheel respectively. This caps at a max speed of 800 on each wheel to avoid the robots moving too fast.positionAroundObject()

This function is called when we have detected the object inside the environment, and start moving towards it. The end goal of this function is to place the Leader robot and the Follower robot on the North side of the box, ready to push.

The function constantly receives input from all 8 of the IR sensors around the robot, and places specific weights on the left and the right wheel for each sensor. The robot will then turn based on the input from the IR sensors, averaging out to make the left and right wheels turn different amounts. These weights on each wheel can be tweaked to change the behaviour of the robot, in ours the robot will follow around each side of the box, turning around the corners quite closely.

We then keep track of when the robot has done a turn while following the box, by monitoring the differences in the speeds of the wheels. For the leader robot, we monitor when it has done one turn, and then start tracking the steps of the wheels. When it starts its second turn, this tells us the length of the North side of the box. Since the robot will have travelled too far at this point, we reverse roughly 20% of the length of the North side of the box, and then turn to face it.

For the follower robot, it will turn and follow around the box the same as the Leader robot initially, but when it gets to the first corner and starts to turn, we start tracking the steps again but we stop the robot after it has moved a certain amount, which is roughly 20% of the box. The robot then turns to face the box. This will place both the robots roughly equal distance apart on the North side of the box.

This function will place the robot on the North side of the box each time, no matter if we approach from the left hand side or the right hand side, as we keep track of which direction we are currently facing in the main searching loop. The robot will then turn either left or right initially to always face the North side.

Test procedure and result analysis

Table 3

Test ID	Description	Expected Results	Actual Result	Comments
1	Have the robots established a connection as leader and follower via infrared.	A handshake agreement is made by the two robots and the follower begins following the leader as the leader moves.	Follower might miss the handshake, causing the follower behaviour not to be triggered.	This was to test the infrared communication between the robots.
2	Calculating the length of the arena when moving along the x-axis for the first time.	The robot reports the distance travelled as it begins turns to traverse the y-axis.	As expected.	This was used to differentiate between the walls and the object
3	Robot re-aligning itself when moving along the walls of the y axis.	Leader makes slight adjustments as it moves along the wall.	As expected.	We did this in order to keep the robot on track as they hardly move in a straight line.
4	Traversal of the arena in 'lawnmower' fashion.	Leader moves along the x-axis turns and goes up the y-axis a little bit and turns back on the x-axis in the opposite way.	A few difficulties with the wheels prevented the robots from going perfectly straight but it still worked.	Used to search the entire arena. Alignment needed to keep the journey on track
5	Second robot following leader.	Leader signals its location to the follower and the follower moves in the direction this signal came from.	As expected.	Used to guarantee that both robots went the same way.
6	Finding the object by checking if it met a wall quicker than it should have.	Leader stops to begin scan of object.	As expected.	The leader compares the actual length with the new obstacle to determine if its the object or not.
7	Leader signalling the robot to wait as it begins scanning the object.	The follower should wait for a certain time period once they object is found.	As expected.	This was done to avoid crashing between the robots.
8	Follower starts scanning the object once it's done waiting.	After waiting a certain amount of time the follower begins to scan the object as well.	As expected.	This was used to get it ready to align itself on the box.
9	Turning right on the object when travelling from right to left.	Both robots turn right on the object when travelling from right to left.	As expected.	We used this to always place the robots at the north side of the box.

Test ID	Description	Expected Results	Actual Result	Comments
10	Turning left on the object when travelling from left to right.	Both robots turn left on the object when travelling from left to right.	As expected.	We used this to always place the robots at the north side of the box.
11	Leader aligning itself on the second corner it traverses.	The leader reverses and faces the box before if it encounters a 2nd corner.	As expected. Difficulties encountered with tape's reflection on object.	This was used to place both robots at each end of the north side.
12	Follower aligning itself on the first corner it traverses.	The followers traverses the first corner it meets and faces the box.	As expected.	This was used to place both robots at each end of the north side.
13	Moving the object once both the leader and follower are in position.	Follower tells leader it is ready to push and they both push the box at the same time.	As expected.	Used to move object towards a goal.

Conclusion

Task-A, Task-B and Task-C exploit the capabilities of the E-Puck robots, using a wide range of hardware components in order to achieve different behaviours. In particular, the high level behaviour in Part-C deeply relies on the communication between the two robots. Such communication has not been achieved throughout the intended method (Bluetooth) but rather with an alternative method (through Infrared sensors). Due to challenges faced in the use of the accelerometer, the robots are not able to detect when the object is touching the wall. As a consequence, the robots are not able to return the object to the goal state. However, they are capable of finding the optimal position around the object, synchronising the movements and start pushing the object toward the optimal direction. Furthermore, the search has been limited to Proximity Sensors rather than the use of the camera (which would have detected the green object from further away) due to conflicts between the use of the camera and the Infrared communication.

Video

In order to demonstrate the behaviours described above, we created a short video which exhibits various scenarios in Part A, Part B and Part C. The video can be viewed simply by clicking on the placeholder below. However, should the video not play by doing so, please click this link. Alternatively, the video can be found in the same electronic folder as the report (submitted via CD-ROM) under the name "Robotics Coursework Group 7.mp4".

Bibliography

Braitenberg, V. (1984) V*ehicles: Experiments in Synthetic Psychology*. Massachusetts: The MIT Press.

E-Puck (2010) *IR communication with libIrcom*. [Online] Available from: http://www.e-puck.org/index.php?option=com_content&view=article&id=32&Itemid=28 [Accessed: 09 December 2010].