MAPPING A PHYSICAL ENVIRONMENT - FINAL REPORT

May 3, 2012

Tom Hancocks - 0930818
Supervisor - Frank C Langbein
Moderator - Dave Marshall
School of Computer Science and Informatics
Cardiff University

Abstract

The goal of this report and project is to explore the methods in which an agent can be implemented to take control of a robot in order to explore and navigate an unknown environment. The robot used by the agent is one that was constructed for less than £100, making use of cheaply available components, and low powered prototyping systems.

In this report, various aspects of how to represent the physical world upon digital mediums will be covered and how well each of them will work in regards to hardware available. Leading on from this, the agent and robot software will be produced and how they were implemented will be covered.

Additionally the calibration of the robot, in order to find the best operation ranges for components will be checked. The results of the entire system will be evaluated to find out why such robots are not yet feasible, but with the recent developments made in low powered computers, how this may change in the next few years.

Contents

List of Figures

1 Introduction

It is not always possible for humans to enter an environment with the purpose of gathering information about it. This may be down to the environment being too small, hostile or quite simply infeasible for a human to enter it. Typically people have sent in remotely controlled drones to remotely view these inaccessible areas, but increasingly (especially in high expense operations), autonomous robots are starting to be used instead. These robots are capable of making decisions themselves in order to successfully navigate their way around the unknown environment without need for human intervention.

The goal of this project is to find out how feasible it is to produce a robot capable of exploring an unknown physical environment for a budget of under £100. The interim report showed that it is certainly possible to get a bare minimum set of components required for the robot, but still needs to be determined how well these components will perform.

However, it is not simply enough to be able to build the robot and expect it to be able to function. For it to be able to function, a software layer needs to be introduced to manage, control and make decisions on what the robot should do. The Interim Report focused on the hardware aspects involved in the creation of the robot, and in this report the software aspects will be explored.

In order to be able to explore an unknown environment, a software system capable of learning through observations made of that environment needs to be developed. This software will be a rudimentary *AI Agent*.

There will be a number of problems which need to be addressed in order for this agent to function, not only correctly, but also efficiently. An initial problem that will need to be overcome is that of where to do all of the processing and computation. The robot is powered by an Arduino, which runs at 16MHz and has 32KB of memory, which has to hold the entire program code. It was determined in the Interim Report (Section 3.1.3, p.20) that sending information back to a computer in order to do any heavy computational work will be required. There still exists the issue of how to reduce the data being communicated between the robot and the computer to prevent overloading the USB channel.

Moreover, how will the agent represent the world it explores, and deal with different types of errors such as false-positives and false-negatives?

Finally, how will the agent determine where it needs to go in order to collect data? Does a basic naive Local Search work, or should a more complex path finding algorithm be employed to intelligently find a route to its next destination.

1.1 Goals & Aims

There are a number of goals that are specific to the software aspects of the project that were not outlined in the Interim Report. These aims are listed below.

- **Represent a physical world digitally.**
  This is an important aspect of the project as it is the means through which the robot will decide how and where it will conduct its search and exploration efforts. Additionally it will be used to present a visual map of the explored environment to the user.
- **Determine paths to explore.**
  How will the robot decide where to go? Once it has decided, how will it navigate its way through potentially unknown areas to get to that destination?
- **How will actions and exploration be co-ordinated between the agent and the robot?**
  Both of these systems need to work harmoniously together. How should work be segregated between the two to ensure optimum performance and stability?

In addition to these aims, the hardware of the robot will need to be calibrated. This calibration will be maintained and managed by the agent.

This report will focus primarily on the software behind the robot and agent, and touch up hardware when needed.

2 The Robot

2.1 Robot Specification

Since the Interim Report, the design and specification of the robot has been finalised. The following three tables all outline various aspects and information about the robot, including component costs, dimensions and processing capabilities.

Dimension Differences from the Prototype Robot		
	Prototype	Final
Wheel Diameter	6.3cm	3.1cm
Wheel Thickness	3.7cm	0.6cm
Robot Size (Width/Height)	16.6cm	12cm
Turning Circle Diameter	14cm	11.2cm

Table 1: *Details on how dimensions and measurements have changed from the prototype robot to the final robot.*

Component Listing		
Component	Quantity	Cost (Estimate)
Plasticard (A4 Sheet)	2	£1.20
Motor	2	£15.00
2 Wheel Set	1	£6.00
Arduino	1	£20.00
Motor Driver	1	£3.50
IR Range Sensor	1	£10.00
Total Cost (Estimate)		£71.90

Table 2: *Components listing of final robot, including the materials used to construct the chasis.*

Arduino Specification	
CPU Clock speed	16MHz
Memory	32KB Built in
Architecture	AVR ATmega328
Programming Language	More-or-less C/C++

Table 3: *Information about the Arduino used in the project.*

2.2 Motor Power

Perhaps most importantly for the accuracy of the robot, is the power supplied to both of its motors. It was determined in the Interim Report that both the Arduino board and motors for the robot would need to be powered by separate power sources. The Arduino would already be communicating with the agent via USB and would therefore, draw power from the computer that the agent is running on, whilst the motors would utilise an onboard NiCad battery.

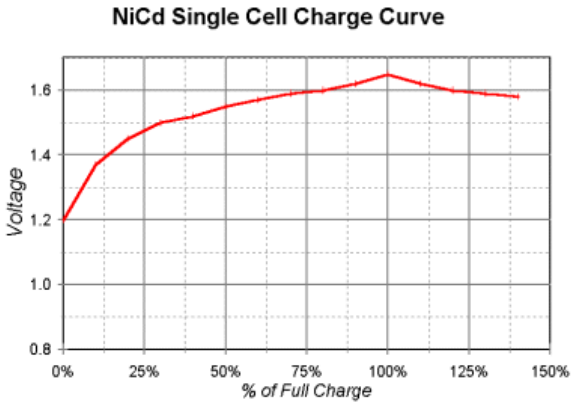


Figure 1: *The amount of current delivered by a NiCad cell and/or battery will vary depending on the amount of charge remaining. This will effect the speed and power of the motors leading to unreliable movement of the robot. Chart by ElectroDynamics (2011)*

However, upon investigation of initial movement tests in which the robot was unable to produce consistent movements when powering its motors with NiCad batteries, it was found that they deliver varying levels of charge which make them unsuitable for use in this application. Figure 1 shows the relationship between charge and current for NiCad cells.

An alternative to NiCad batteries are LiPoly batteries. Unlike NiCad batteries, LiPoly batteries do not diminish in the amount of current they deliver as the cells discharge, but instead deliver a consistent current over the course of their life.

Given that the robot would already by connected to the agent via a USB cable, it was decided that adding a second USB cable (that was modified to deliver only power rather than data) to the robot would be the quickest and cheapest method of powering the robot's motors. However, because the Arduino only has one USB port the additional cable had to be modified to to be able to deliver additional power to the robot without requiring a USB port. This was achieved by isolating the power lines inside a USB cable and connecting them to "jumper wires" which could be easily plugged into a breadboard. The finished cable is shown in Figure 2.

This source of power for the motors is guaranteed to 5V (stated in the USB Specification)

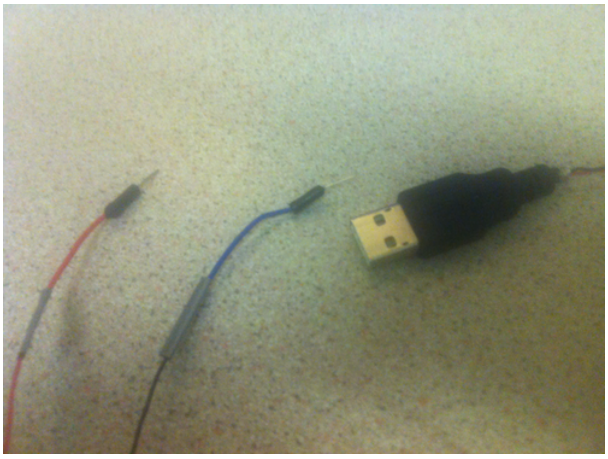


Figure 2: *A modified USB cable that has had the data wires and one of the plugs removed. The power lines have then been connected to jumper wires to make them easy to use with a breadboard.*

and means that any previous inconsistencies that were caused by varying current should no longer be an issue.

3 The Agent

The agent is the software that is responsible for interpreting any observations made by the robot and producing a map of the environment from them. In addition to this it also makes decisions on how to act upon this knowledge that it collects in order to collect more knowledge to help construct the map.

In order for the agent to work, it needs to have a mechanism to keep track of observations which can also be used to determine and visualise the structure of that environment. How will the agent decide on where to go, and how to get there? Finally, due to the agent and robot being separate entities, how will the agent communicate with the robot to co-ordinate and collect information about the environment?

3.1 Representing Knowledge

Before a solution to the problem of exploration can be designed, it is imperative that the agent has a means in which it can record and store the observations made by the robot and use them to build up knowledge of the environment. This knowledge will then be usable at a later time to help co-ordinate further exploration of the environment.

The most straight forward, and easiest to handle way of storing and accessing information about the world is through a matrix. In this, each cell of the matrix is taken to represent a specific section of the world. Figure 3 shows how the layout of an environment can be encoded into a matrix.

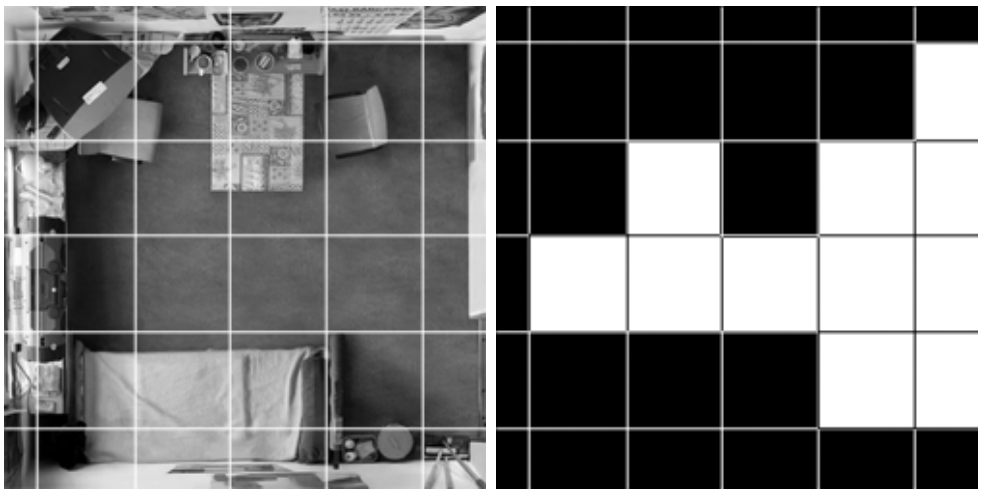


Figure 3: *Aerial photograph of a room (left) which illustrates how the room may be divided up into sections, which can be represented as a matrix (right). Room image by (Chill 2006)*

One of the simplest forms of matrix that can be used to accomplish this is the Bit Array. In this structure each cell is represented by a single bit, allowing only for the storage

of true or false values. These values can be used to represent whether a section of the environment with which, that value corresponds to, is obstructed or not.

A slightly more advanced form of this would be a Confidence Map. Rather than storing each cell as a single bit, each cell is stored as a byte, short (2 bytes) or integer (4 bytes). This allows for varying levels of *confidence* as to whether a cell is actually occupied or not. This can be referred to as the *Occupied Probability*.

Whilst analysing these methods of knowledge representation for the physical world we shall look at them each in regards to a 25x25 cell matrix.

3.1.1 Bit Array

The Bit Array is an extremely simple form of *matrix* and is data structure that which requires very little memory in which to be stored.

$$\text{Size}_{bytes} = \left\lceil \frac{M_{width} \times M_{height}}{8} \right\rceil$$

Where M_{width} and M_{height} are the width and height of the matrix respectively. By substituting in the matrix sizes into the equation, the minimum number of bytes required to store the Bit Array can be determined.

$$\begin{aligned} \text{Size}_{bytes} &= \left\lceil \frac{25 \times 25}{8} \right\rceil \\ &= \lceil 78.125 \rceil \\ &= 79 \end{aligned}$$

This low memory footprint is a strong positive point in regards to the Arduino, which has limited memory. However its limited capacity for storing data about a cell is a major negative point.

With the Bit Array structure, there is no way to determine which cells have not yet been visited. This makes it difficult to construct an effective search strategy that isn't just a fixed route. This can be overcome by creating a second Bit Array to store whether a cell is visited or unvisited. This effectively doubles the amount of storage required, however it is still less than that of a Confidence Map.

3.1.2 Confidence Map

The Confidence Map is a larger version of the Bit Array. Rather than using only one bit per cell, it uses an arbitrary number of bits, N. This allows it to store a range of values per cell, rather than just true/false values. The number of values capable of being stored in a cell depends on the number of bits allocated per cell. The maximum value that a cell can store can be computed by using the following.

$$\text{Value}_{max} = 2^N - 1$$

The reason for storing a larger range of values per cell is to allow the implementation of a "vote" system. This works by incrementing the value of a cell when the agent believes it has spotted an obstacle in the section of the physical world that corresponds to that cell. Cells with higher values (or more votes), are the ones which the agent has a higher confidence of something obstructing that cell. However, this ability to store a range of values comes at the cost of higher memory footprints.

$$\text{Size}_{bytes} = \left\lceil \frac{N\big(M_{width} \times M_{height}\big)}{8} \right\rceil$$

Assume that each cell is going to require 16 bits (2 bytes or a *short*) of storage.

$$\begin{aligned} \text{Size}_{bytes} &= \left\lceil \frac{16\big(25 \times 25\big)}{8} \right\rceil \\ &= \left\lceil \frac{10,000}{8} \right\rceil \\ &= \lceil 1,250 \rceil \\ &= 1,250 \end{aligned}$$

This is nearly 16 times larger than the Bit Array, but rather than allowing only 2 values per cell, it can take 65,536 values per cell. The size of N needs to be large enough to account for the number of observations that the robot may make of that section of the environment.

In this structure, it is possible to record whether a cell has been visited or not. A single value can be taken to be a special identifying value that represents an unvisited cell. When a cell contains this value then it is treated as unvisited.

However, there is still a limitation in this structure. It can only record observances of obstructed cells. At first this is not an apparent limitation, as the purpose of the system is to explore and find obstacles in an environment. Further consideration of the problem reveals that it is equally important to consider the areas that are not obstructed, as it is these areas in which the robot will use to navigate the environment. Without a mechanism to record when a cell is unoccupied, it is entirely possible that a moving object within the environment may cause a number of cells to appear as occupied when in fact they are not (or at least not very frequently). By also recording the number of times a cell is not obstructed we can then account for motion and possible incorrect readings when exploring the environment, by generating probabilities of a section being obstructed.

To do this two Confidence Maps are needed, one for the observances of obstructions and the other for the observances of unobstructed sections. From these two matrices we can then calculate a probability of the cell being obstructed.

Let α be equal to the count from the obstruction matrix.
Let β be equal to the count from the unobstructed matrix.

$$\varphi_{x,y} = \alpha_{x,y} + \beta_{x,y}$$

$$P_{x,y} = \begin{cases} \text{unvisited} & \text{if } \varphi_{x,y} = 0 \\ \alpha_{x,y}\left(\frac{100}{\varphi_{x,y}}\right) & \text{if } \varphi_{x,y} > 0 \end{cases}$$

3.1.3 Knowledge Size & Resolution

An additional problem is posed in the form of how much space to allocate the knowledge for the agent? The problem arises from the very nature of the project... the environment is unknown. As far as we know at the beginning of the exploration, the environment could be tens of metres across, or just a few tens of centimetres. This poses the question of exactly how much space should be allocated for knowledge? Before this can be determined though, some additional information is required.

The first bit of additional information that is required is the amount of space that each cell actually covers. This largely depends on the actual robot, as it will need to be optimised to the capabilities of the robot. Assume the that size of a cell, as defined by the robot is

$$\text{Cell}_{size} = \pi r^2$$

Let r be equal to 1cm, where r is the radius of the robots wheel. We know that the size of a cell is then 3.142cm wide or high. Taking the matrix defined earlier (25 by 25 cells)

we can determine that it would map to an area $78.5cm^2$ in the physical world.

3.2 Exploration & Planning

> **Exploration** *noun*
> The action of traveling in or through an unfamiliar area in order to learn about it.
> - Oxford English Dictionary

The primary goal of the agent is to be able to explore and construct a map of the physical environment that it is in. From the definition of *Exploration*, this can be understood to require physically travelling through its environment and recording any observations made. These observations can then be used to build an understanding of the structure and layout of the environment.

However there are a number of different ways in which this exploration could be carried out, such as involving different paths and search strategies for determining the best path to take. Should the search strategy be dictated by a basic Local Search, checking for the best unobserved area overall or use search and path finding algorithms such as the A* algorithm?

What should the robot do at each point? Rotate around and observe all directions around it at each point? Or just make observations directly ahead of it at all times?

3.2.1 Local Search

Local Search is a meta heuristic intended to determine an optimal solution to a problem. This is accomplished by evaluating all of the potential solutions that could be used and determine which one is the best. However it does not consider any intermediate states of getting there.

In the context of mapping out a physical environment, and deciding the best route to take in order to fully explore the environment, we can consider that each cell represents an area of the world to be in a specific state. For example *Unobserved* or *Observed*.

However, these alone do not provide enough information; it is possible to have multiple candidate cells. In order to be able to weight each cell, so that the best choice is made, and not the first that happens to meet the criteria required, a cost function can be introduced. A cost function simply adds in the costs of considered properties of the world. Such properties may include distance, effort required, chance of death, etc. For this problem the following need to be considered:

1. Distance to candidate cell (further is better).
2. Probability of obstacle at candidate cell (lower is better).
3. Is it unobserved? (if it is, then decrease the cost.)

The distance to a candidate cell can be computed in two distinct ways, by using either the Manhattan or the Euclidean Distance. Weisstien (2012) defines the Euclidean Distance to be the ordinary distance between two given points and can be calculated by using Pythagoras' Theorem. This is a distance that assumes the agent is able to move along diagonals. However the acquired knowledge by the agent is built with the belief that it cannot travel along diagonals, only horizontal and vertical movements are permitted.

Russel & Norvig (1995, p.102) state that when an agent cannot move diagonally, its distance can be calculated through the sum of the horizontal and vertical distances, giving the City Block Distance or Manhattan Distance.

The probability of there being an obstacle at the candidate cell can be determined by using the function stated in Section 3.1.2. This function calculates the probability based on the number of times an obstruction has been observed and the number of times there hasn't been an obstruction observed at the given cell. It can also give a value for when the given cell has not yet been observed.

From this the cost of going to the candidate cell needs to be calculated. This is done by evaluating all the considered properties and parameters of the candidate cell, typically by summing them together. The agent will then select the candidate with the lowest cost as its preferred destination.

$s \Rightarrow$ maximum world width, as used by the knowledge of the agent.
$\lambda \Rightarrow$ distance from agent to candidate cell.
$\varphi \Rightarrow$ probability of obstacle at candidate cell.

$$C = (2s - \lambda) + \varphi$$

Once the agent has determined the cell to use, it will then move towards it either along the horizontal or vertical axis, before repeating the entire process again. This is inefficient and susceptible to becoming stuck in a particular spot due to obstructions in its path to the best candidate cell, as illustrated in Figure 4.

3.2.2 A*

The A* algorithm is a pathfinding algorithm that has faced widespread adoption due to being able to efficiently plot a route between two points. When tied with the Local

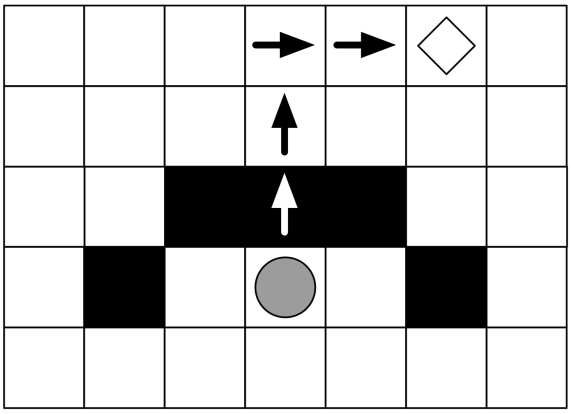


Figure 4: *The robot (circle) gets stuck on the environment as it tries to move towards its destination (diamond) due to only being able to move closer to its destination.*

Search, to provide a means of actually planning a route to the destination given existing knowledge about the environment, as opposed to *blindly* moving towards the destination, the major problem of becoming stuck in the standalone Local Search should be eliminated.

The A* algorithm was developed through two iterations of improvements upon Dijkstra's Algorithm by Nils Nilsson and Bertrand Raphael in 1964 and 1967 respectively. The improvements they both introduced focused on improving the speed of the algorithm and to use a heuristic means of determining the best route.

The algorithm works by iteratively expanding traversing nodes in the known graph or world, by focusing on the current lowest cost path known at that point. This method is Dijkstra's Algorithm and guaranteed to find an optimal path, but is not guaranteed to do so quickly. This is where the heuristic component of the algorithm comes in. The algorithm will also estimate the remaining cost to the destination from each node and add that in to the cost. By doing this it helps to prevent the algorithm from unnecessarily searching nodes that lead *away* from the destination node and thus reducing the computational time of the algorithm.

Russel & Norvig (1995, p.96) describe that two different search strategies *Greedy Search* and *Uniform-cost Search* both individually employ aspects that are desirable, but each have negative consequences that are not. Greedy Search, denoted as $h(n)$ can cut the cost considerably, but is not an optimal solution nor is it complete, which makes it unreliable in many applications. Uniform-cost Search, $g(n)$, on the other hand is both optimal and complete, but is very inefficient in how it conducts the search. Russel and Norvig show that the favourable aspects of these two algorithms can be gained with ease by simply summing them.

$$f(n) = g(n) + h(n)$$

In this $g(n)$ is the total path cost to get to the node that is currently being explored from the starting node, where as $h(n)$ is the estimated cost of travelling the cheapest path from the current node to the goal. Tracy & Bouthoorn (1997, p.107) describe that $h(n)$ should be an underestimate of $h*(n)$ and admissible, where $h*(n)$ is the actual optimal cost for all currently explored nodes.

$h(n)$ is admissible iff $h(n) \leq h*(n)$.

This all makes the A* algorithm a good choice for making the agent's planning and search strategy more robust and effective. By merging the two and using the Local Search to initially select a new destination cell and then switching to the A* algorithm to produce a plan on how to reach the destination.

However the A* algorithm needs to be given an admissible heuristic in which it can estimate the remaining cost from any given cell to the destination. If the heuristic is not admissible, then the algorithm can not be guaranteed to find an optimal path. Fortunately, the nature of the world constructed in the agent's knowledge is a uniform matrix comprised of square cells making the task of finding a suitable heuristic relatively trivial. Additionally because the agent can only move in vertical and horizontal directions in respect to the world, it is simple to the calculate the shortest distance between two points.

$$d = \Delta x + \Delta y$$

Where d is the distance and Δx and Δy are the changes in x and y respectively. As shown earlier this is simply the Manhattan Distance.

The effect that this has on the agent's ability to handle obstructions is illustrated in Figure 5.

Using this information so far, the following algorithm can be produced to incorporate both the Local Search and A* algorithm aspects of the agent.

```
function DetermineBestCell
    bestCost = INTEGER_MAX
    for y = 0 → worldSize do
        for x = 0 → worldSize do
            c = EvaluateCellCost(x, y)
            if c < bestCost then
                bestCost = c
                bestCell =Point(x, y)
            end if
        end for
```

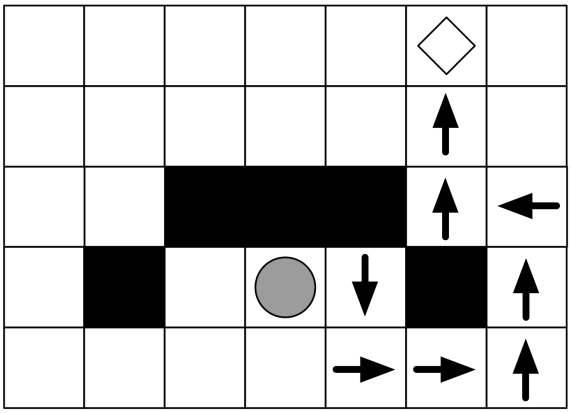


Figure 5: *The robot (circle) plans a route around the obstruction it encounters to get to its goal (diamond) using the A* algorithm to find the optimal path.*

```
    end for
     return bestCell
end function

function EvaluateCellCost(destination)
    dx =abs(current.x − destination.x)
    dy =abs(current.y − destination.y)
    distance = dx + dy
    p = OccupiedProbability(destination)
    if p = unoccupied then
        cost = distance
    else
        cost = p + distance
    end if
     return cost
end function
```

3.2.3 Dealing with Potential Problems

Even with both Local Search and A* algorithm working together to navigate the environment, they still have to work with, at least *early on*, incomplete information about the world. When the agent initially makes a decision on where to go, and then plans out a route, it has no knowledge of whatever obstructions may, or may not, exist along its chosen path.

These unknowns have to be dealt with as the new data is collected. At each point the agent visits it collects new information about its surroundings and *if* an obstruction is encountered in its immediate path that blocks it then one of two things can be done.

1. A new path can be worked out to its goal using the newly acquired knowledge.

2. A new destination can be chosen and a path planned to it.

Both of these actions will result in the agent changing its path to something which it believes to have no obstructions, given the present information that it has.

3.3 On-board System

Another aspect of the agent is the on-board system which will sit on the robot itself and communicate with the computer. There are four main things that need to be handled by the on-board system.

1. Forward Movement - controlling movement speed and time to move forward in a consistent manner so that mapping is reliable.
2. Rotation - controlling how the robot rotates so that it rotates at a consistent 90degrees.
3. Observation - making measurements to the nearest obstacle ahead of itself using its IR range sensor so that the agent can build the map.
4. Communication with Agent - send information to the agent, and carry out actions given by the agent.

3.3.1 Forward Movement

As determined by the way in which the agent builds up knowledge and a map of its environment, the robot will need to move forward in increments of a fixed distance. This kind of movement can be achieved in a couple of ways.

- Set the power of the motors and then activate them for a set duration of time before deactivating them again.
- Activate the motors, and use the IR range sensor to measure when the robot has moved a certain distance and then deactivate them.

The first option involves the most calibration, as tests need to be carried out to determine the relationship between how much power is delivered to the motors and the speed that the robot then travels. Once the speed is known the distance that the robot will travel in a specific period of time can be determined easily.

A major advantage to this approach is that it does not rely on any sensor which may be giving noisy results, and so can consistently be used without concern of failure in many different environments. The downside of it is, that many iterations of tests will be required to determine the speed of different power levels.

The second option involves repeatedly (and rapidly) measuring the distance to the the object directly ahead of the robot to determine when it has moved a certain distance.

The advantages to this are that the robot is then able to vary the speed at which it goes in realtime. Ease up the speed, and ease down the speed at the start and end of movement respectively. This would help diminish any jerking movements which can cause the robot to twist and lead to drift. However this is redundant as soon as the robot goes out of range of any objects. It is only able to detect objects within 10 to 80 cm directly ahead of its IR range sensor, making the method useless when in the middle of a large open space. Additionally noise from the IR range sensor will make it difficult to be precise in stopping the robot after the desired distance. However, due to the nature of the map and world constructed by the agent, this point should be a minor concern.

Due to the limitations of the second option, the robot will govern its movement using the first option, by activating the motors for a predefined interval of time at a certain power. Test data for determining this can be found in Section 5.1.

3.3.2 Rotation

The robot needs to be able to rotate in an accurate manner, and as defined by the way the knowledge and world are represented by the agent, in 90° intervals.

Rotation for the robot is achieved by activating both motors in opposite directions, thus causing one half to try to move forwards and the other half to move backwards. This causes a spinning motion around the centre point of the robot's wheel base.

Unlike with determining forward movement, which required many tests to accurately determine the speed of the robot at different power levels, rotation can fundamentally be seen as a motion in a straight line.

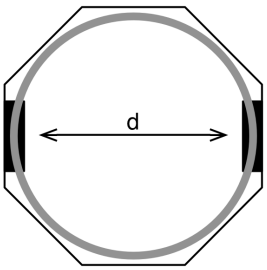


Figure 6: *The path (in gray) the wheels of the robot follow when rotating. d represents the distance between the two wheels, or the diameter of the turning circle.*

By measuring the distance between the two wheels of the robot, the circumference of the turning circle can be calculated. The circumference is in essence the distance that the

robot needs to travel in order to pass through 360°.

Further, we can then divide this distance (the circumference) by 4 to give the distance that the robot travels when rotating 90°.

$$\frac{C}{4} = \frac{d\pi}{4}$$

The distance between the wheels of the robot is specified in the Robot Specification in Section 2.1, as 11.2cm, which gives

$$\begin{aligned} d &= 11.2 \\ \frac{C}{4} &= \frac{11.2\pi}{4} \\ &= \frac{14\pi}{5} \\ &\approx 8.8cm \end{aligned}$$

This means that the robot will move through 8.8cm when rotating 90°. Using both this and the results collected in the speed tests in Section 5.1, it is possible to determine what power the motors will need to be set to and how long to activate them to achieve this.

3.3.3 Observation

Observation via the IR range sensor needs to be handled by the robot before information can be communicated back to the agent. In order for this to be done, an understanding of how the IR range sensor works is needed. Oomlout (2012) provide basic information on how the sensor outputs a varying voltage depending upon the distance of the object directly ahead, and how this relates to an actual distance.

The voltage provided by the sensor is between 0.4 and 3 volts, and is detected by the Arduino as a *10bit value*. Oomlout provide a formula to convert this 10bit reading to an actual distance.

$$d = 12343.85 \times (\text{10bit reading})^{-1.15}$$

This then provides the robot with an actual distance reading to the nearest object straight ahead of the robot. For simplicity in sending the information back to the agent, a simple calculation can be done to turn the distance into "cells", so that the agent is given a distance in terms of the number of cells between it and the nearest object. Figure 7 illustrates how the distance relates to a number of cells.

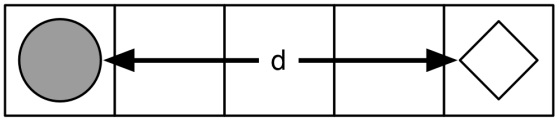


Figure 7: *The robot (circle) measuring the distance to the closet object (diamond) and getting a distance reading of d. The actual cell boundaries are shown beneath to show how the distance maps to a cell count.*

To show how this works, assume the following values.

$$\begin{aligned} d &= 33.4cm \\ \text{cell\_size} &= 10cm \end{aligned}$$

We can convert the distance, d, into a cell count very easily through simple division.

$$\begin{aligned} cell_{distance} &= \left\lfloor \frac{d}{\text{cell\_size}} \right\rfloor \\ &= 3 \end{aligned}$$

This gives a result of 3 cells, which is the number of whole cells that the beam for the IR range sensor passes through.

3.3.4 Communication

The final aspect that the on-board system needs to take into account is the communication back to the agent. All communication is handled via a serial link over USB. The Arduino contains a library for establishing and communicating easily via serial, which is aptly named *Serial*. In the case of the Arduino Duemilanove, which is what drives the robot, it has one Serial Port.

Arduino (2012) state that the library will take control of two of the digital pins that the Arduino offers. However given that the total number of required digital pins on the robot is 9, and the Arduino offers 14 to begin with, this is not a concern.

Additionally, the library requires a baud rate (pulses per second) to be specified. Whilst there is no universally agreed *best* baud rate, WormFood (2005) provides a collection of tables with supported baud rates for the AVR chips (which the Arduino uses) and how susceptible each baud rate is to errors for given CPU frequencies. The Arduino Duemilanove uses a 16MHz which gives the following suitable baud rates.

- 4800
- 9600
- 14400
- 19200
- 28800
- 38400
- 76800

The baud rate of 9600 also appears in the code examples provided in the Arduino references suggesting that it is the most suitable and ideal baud rate.

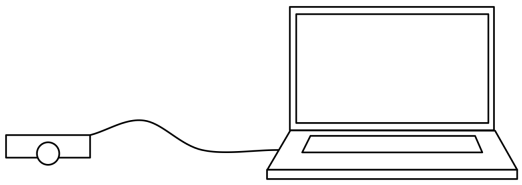
The reason for the onboard communication is so that the robot can receive commands from and send back information to the agent. The commands that are used are listed in Table 4.

Command Code	Command Value	Command Description
MOV	M	Move Forward. The value byte contains the power that the motors should be set at.
RCW	R	Rotate Clockwise. The value byte contains the power that the motors should be set at.
RCCW	r	Rotate Counter Clockwise. The value byte contains the power that the motors should be set at.
OBSV	O	Instruct the robot to make an observation. The value byte does not contain any information. The robot should make an observation and then send back an OBSV packet with the value set to the number of cells ahead that an object was detected.
SPWR	S	Set the power level of the motors. The value byte contains the power that the motors should be set at.
SDEL	D	Set the activation period of the motors. The value byte contains the time in tenths of a second that the motors should be set at.

Table 4: *Commands used by the robot and agent to co-ordinate exploration.*

4 Building the System Software

With the information that has been collected, and the decisions that have been made with regards to that data, the software for the system can now be produced. This software is what will be responsible for controlling and driving the robot.



(a) Onboard System (b) Agent Host Computer

Figure 8: *The on-board system is the software that runs upon the Arduino and the actual robot, whereas the agent host is the computer on which the actual agent will run.*

4.1 On-board System

The on-board system is what will run on the actual robot and be responsible for controlling the motors, sensors and communicating information back to the agent and carrying out actions that the agent tells it to. Figure 9 shows the basic overview of the system structure.

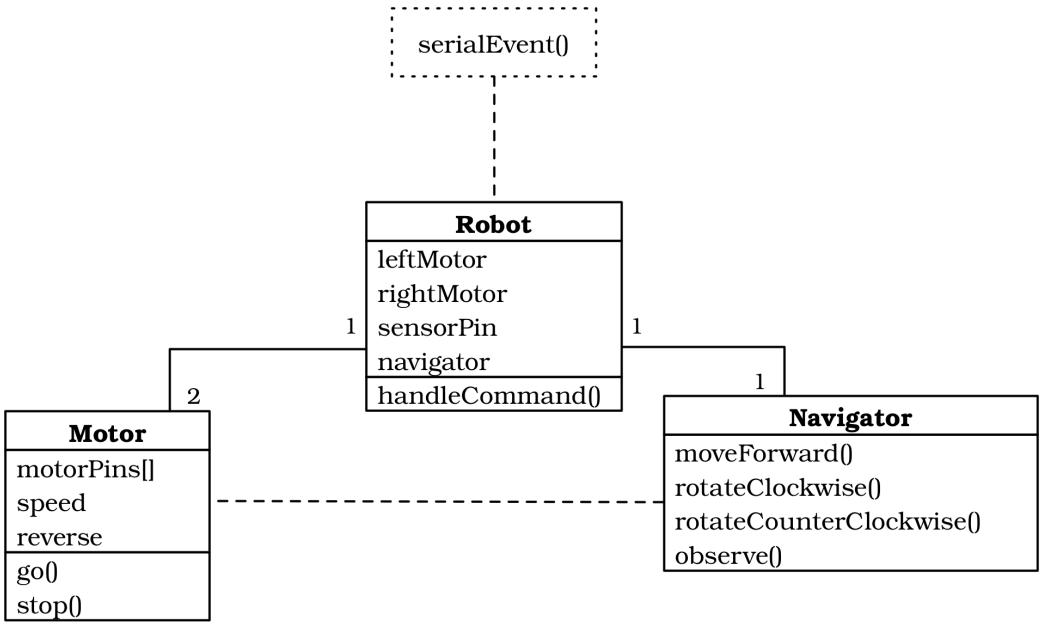


Figure 9: *The on-board system is split into three major components which each handle specific responsibilites.*

The overall design of the system can be split into three major components; the *Robot controller*, the *Motor controller* and the *Navigator*. Each of these components is responsible for managing just one aspect of the robot.

The first of these components is the Robot Controller, which is responsible for interpreting commands received from the agent and telling the Navigator what to do, before sending back confirmations to the agent. This component has one major function which is the `handleCommand()` function. This function is only called when there is available data in the USB buffer. Checking for available data in the buffer is handled by `serialEvent()` function, which is declared as part of the Arduino libraries. `serialEvent()` is a function that is called whenever the Arduino has detected activity on the USB serial buffer. By implementing the function in the system, USB Serial events can be used to trigger the `handleCommand()` function after checking the number of bytes available in the USB Buffer. As all commands in the system will consist of two bytes a check can be put in place to only call the `handleCommand()` function when two of more bytes are available.

Additionally the Robot Controller is responsible for initialising the motors and navigator when the robot is first powered on.

The second component is the Motor controller. This component is unique as there will need to be multiple instances of it in order to refer to each of the motors on the robot. When a Motor controller instance is created the robot will need to specify which digital pins it will be responsible for. The Motor controller can not directly detect or even control the motor, but instead turns the digital pins it is responsible for on and off. Provided the pins specified match up to actual motors correctly, then robot will move as expected. The digital pins it is responsible for control the activation and direction of the motors spin.

The first pin specified is the activation pin. This pin is responsible for the rotation speed of the motor. Normally digital pins only handle HIGH and LOW voltages, which in the case of a motor can be equated to full speed and stopped, or in terms of electronics 5V and 0V. However the Arduino provides Pulse-With-Modulation (PWM) functionality on several digital pins which can be used to alter the speed of the motor.

The second and third pins are used to specify the spin direction of the motor. By alternating the outputs of the pins between HIGH and LOW the motor spin direction can be set. PWM is not necessary for these pins.

The third component is the Navigator, and the most complex part of the on-board system. This component is responsible for carrying out instructions sent from the agent host and collecting information about the environment. This means that it needs to be aware of both motors and how to control them, the IR range sensor and how to convert the raw signal that it provides into a distance that can be equated to a number of cells.

4.2 Agent Host

The agent host is the computer and system that provides the bulk of the processing power and decision making capabilities for the overall system. It is connected to the on-board system via USB. Figure 10 shows what the core of the agent hosts structure is.

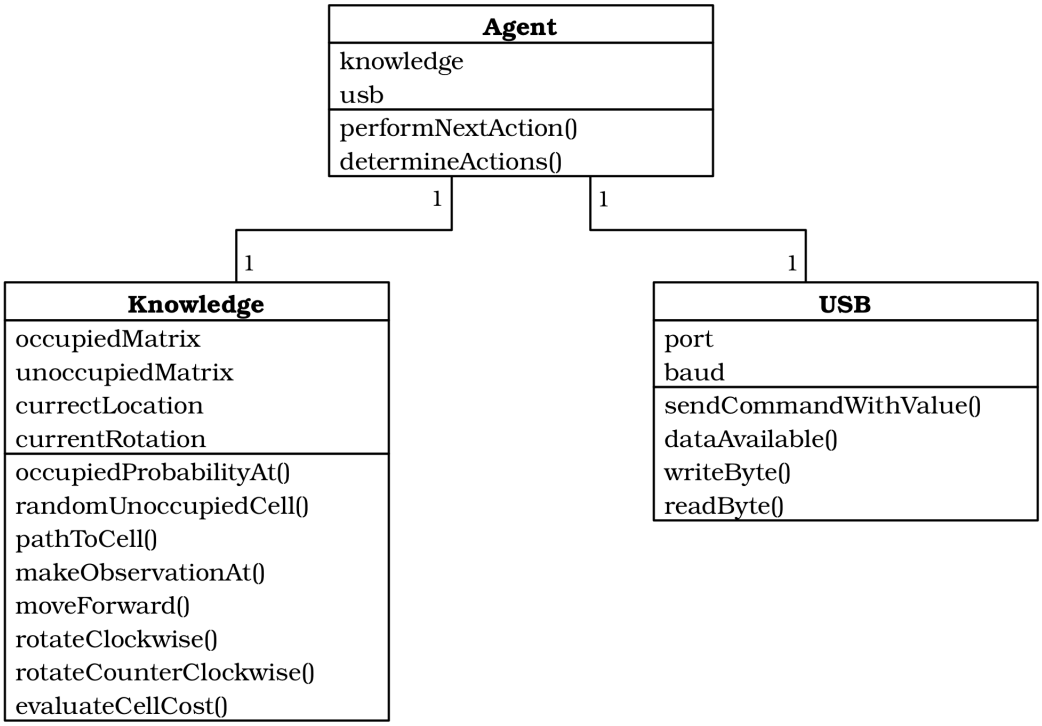


Figure 10: *The agent host is split into three primary components (not including UI components) that are responsible for determining what the robot should do and what the current environment knowledge is.*

These three core components of the agent host are responsible for co-ordinating the exploration of the unknown environment and controlling the robot whilst they do that exploration. The system will need additional components that enable user interaction and displaying information on screen, but the core functionality that will enable the system to function is in these components.

The USB component is used by the agent to co-ordinate the USB communications. It is responsible for opening a connection to the robot, sending commands and waiting for replies. In this regard the relationship between the agent and robot can be thought of as master and slave respectively. The robot will never be able to take control of the connection or even start a connection. Figure 11 demonstrates how a single command is handled by the USB connection.

The Knowledge component is the most complex of the components that make up the

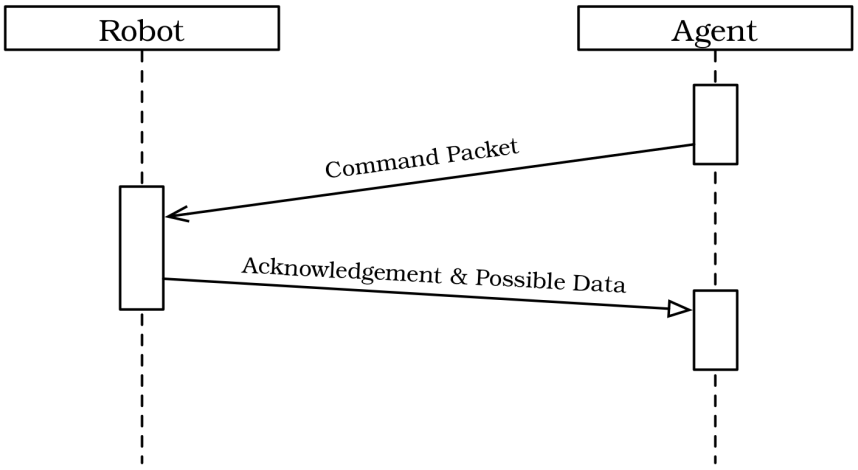


Figure 11: *An example of how single command packets travel between the agent and the robot.*

agent, and is responsible for managing and turning all collected data into structured knowledge about the environment. From this collected data and knowledge the Knowledge component is able to plan new routes and determine the probabilities of an object been at any given cell within the environment.

However because the robot does not directly relay any information about position to the agent, then the Knowledge component needs to be able to keep track of which cell it believes the robot is currently in and the current rotation of the robot. The UML structure shown in Figure 10 shows two properties associated with the Knowledge component; `currentLocation` and `currentRotation`. These properties are updated to reflect the believed position of the robot each time the agent issues a command to the robot.

Figure 12 shows how the process of sending actions and commands to the robot works. It begins by checking the *Actions Queue* for any existing actions that need to be performed. This actions queue stores all the actions that will take the robot along a certain path. This path is determined by the Knowledge component in the `pathToCell()` function by passing it a cell, which can be chosen by using the function `DetermineBestCell()`. The action of determining a new path should only be done when there are no outstanding actions to be carried out.

The agent will always select the next action in the actions queue and attempt to perform it. Before sending the action to the robot the agent will attempt to determine if the action will be successful or not. An action will be considered to be successful iff it will not result in the robot colliding with any objects. This is determined by checking the current knowledge of the environment and seeing whether the destination cell is occupied or not. If the agent determines that the action will be unsuccessful then it will not send the action to the robot, but rather, empty the current actions queue and redetermine a new path. The reason for determining a new path rather than continuing with the

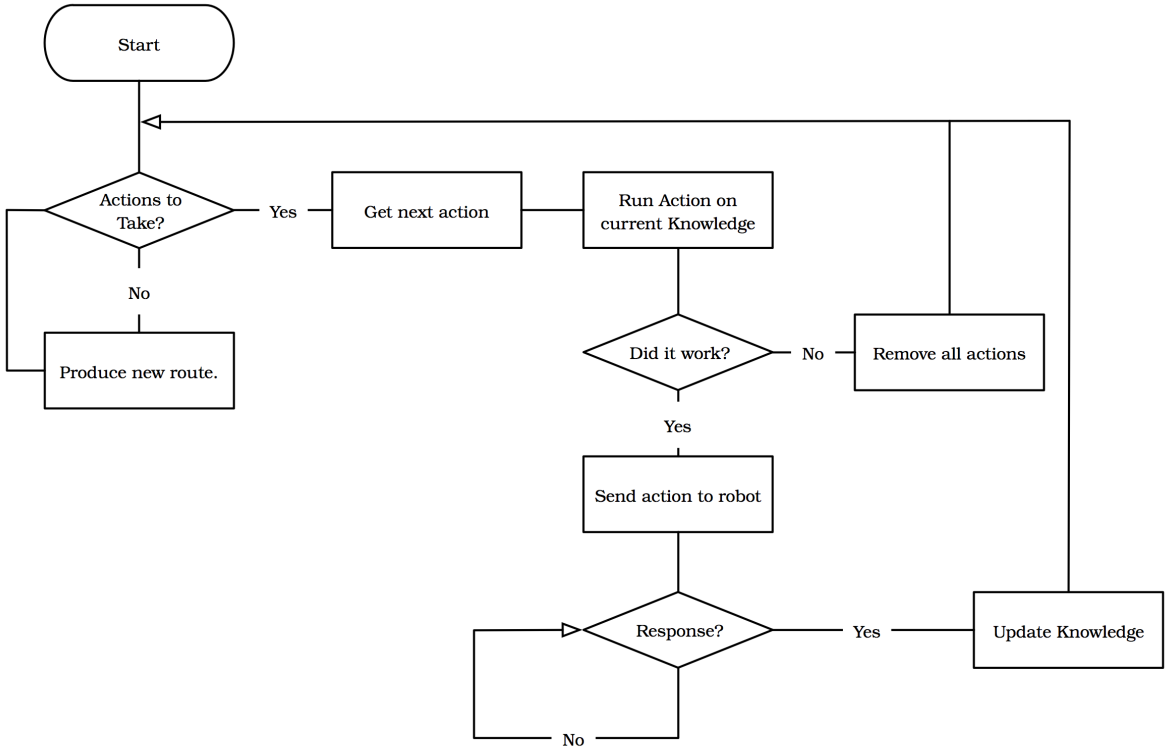


Figure 12: *This flow chart shows how the agent sends actions and commands to the robot, using the knowledge it has gathered to decide if the action will be successfull or not.*

next action in the actions queue is because the actions will no longer match up with the current knowledge the agent has, and will become increasingly likely to fail actions.

After the agent knows that an action will succeed it will then send the action command via USB to the robot and wait for a response acknowledging that the action has been completed. This process results in the agent being blocked from execution, which helps prevent both the agent and robot going out of sync with each other.

Once the agent receives the acknowledgement that the robot has carried the action, it removes the action from the actions queue and repeats the process once again. This process is repeated until no more paths can be found, at which point the agent assumes that it has fully explored its environment.

5 Testing & Calibration

Before being able to test the robot in an actual unknown environment to see the results it produces, a number of tests need to be carried out in order to determine the error ranges and rates that the robot experiences. These tests will be used to calibrate and determine the optimum values for cell sizes and speed for the robot.

5.1 Testing Motor Speed

The purpose of this test is to determine how different power levels provided to the motors, affect the speed of the robot.

The motor power is determined through a Pulse-With-Modulation (PWM) signal that causes rapid activation and deactivation of each motor every second. The reason for using PWM is because the micro-controller on the Arduino works purely on digital signals, and can only deliver either a *Low Signal* (0V) or a *High Signal* (5V). PWM allows the micro-controller to rapidly turn on and turn off the signal delivered by specific pins to *simulate* different voltages.

It is this simulated voltage which is referred to as the motor power, and what effects the speed of the robot. In the case of the Arduino the PWM can be set to a value between 0 (always off) and 255 (always on).

However, initial tests show that a minimum PWM value of 50 needs to be provided for the motors to able to turn over.

The test will be based on the robot moving forward in a straight line for 0.5 seconds, with the power being increased after every three tests run. The power will be increased by a value of 5, and at the end all measurements for like power levels will be averaged.

The results collected in this test are available in Appendix A. These results include information about drift, speed and distance travelled.

Figure 13 shows how the speed increases in relation to the power. Additionally Figure 14 shows how drift effects the robot in relation to the distance travelled.

The results for this test, show that there are several points in which the amount of drift experienced diminishes to little or none. These points include the motor power's of 50, 90, 110 and 130.

As stated in previously in Section 3.3.1 (p. 20), the robot can be made to move forward in steps of a fixed distance, with each of these steps representing the size of a cell. Table 5

Figure 13: *The increase in speed as the power to the motors is increase. Negative values show drift to the left, whilst positive values show drift to the right.*

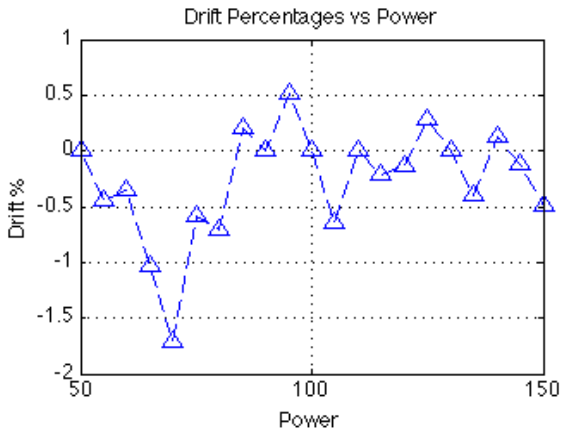


Figure 14: *The amount of drift experienced when travelling a specific distance*

shows how these motor power's relate to actual speeds and gives the distances attained using these speeds for 3 different durations.

Further, Figure 14 shows that as the motor power is increased the harshness of any drift that is experienced appears to drop off. This leads to the *possible* conclusion of 130 being a suitable motor power for travelling forward. This also fits nicely when taking the distance travelled in half a second at this motor power; it matches with the size of the robot.

However, additional tests with the IR range sensor in order to determine its accuracy and points in which it is the most accurate need to be conducted to determine if this speed and a cell size of 12cm are suitable.

It was also stated in Section 3.3.2 that the motor power used for rotation is dependant upon the speed of the motors and thus the distance in which is travel at that power. For the current robot, the "distance" that it needs to travel is 8.8cm in order to complete 90° of its turning circle. Again, from Table 5 there is only one motor power that provides an immediate and obvious closeness to this distance; 90, for a duration of 0.5 seconds. This

		Distance Travelled (m)		
Motor Power	Speed (m/s)	0.5s	1s	2s
50	0.06	0.03	0.06	0.12
90	0.18	0.09	0.18	0.36
110	0.22	0.11	0.22	0.44
130	0.24	0.12	0.24	0.48

Table 5: *Distances travelled by robot at different speeds and time durations.*

Motor Power	Distance
85	0.082m
86	0.0836m
87	0.0852m
88	0.0868m
89	0.0884m
90	0.090m

Table 6: *Stepping through motor powers to find the closest distance to 8.8cm possible from the robot's motors.*

does not give a perfect value however. Looking at the motor power of 85 within the test data we find that it delivers a speed of 0.164m/s which equates to 0.082m, which is now too low. This can be solved by using the following:

$$\begin{aligned}\alpha &= 0.09m - 0.082m \\ &= 0.008m\end{aligned}$$

This means that for an increase of 5 in the motor power, on average, the robot travels 8mm farther.

$$\begin{aligned}\beta &= \frac{\alpha}{5} \\ &= 0.0016m\end{aligned}$$

Now, stepping through each of the motor power's between 85 and 90, and incrementing the distance by 1.6mm at each step we can determine the best motor power for achieving a 90° rotation. These values are shown in Table 6, and show that the closest and most ideal value comes from a motor power of 89.

Fortunately two different speeds and durations can be used for both moving forwards and rotating the robot. The ideal values are listed in Table 7.

Aspect	Duration	Motor Power
Moving Forward	0.5s	130
Rotation	0.5s	89

Table 7: *Ideal motor powers and durations for moving forward and rotating the robot 90°.*

5.2 IR Range Sensor Distance Tests

The purpose of this test is to determine the accuracy of the IR range sensor which is used to measure distance to objects directly ahead, and thus find out its optimum operating range.

Oomlout (2012) state in their quick start guide sheet that the sensor has an operating range between 0.4 Volts and 3 Volts which equates to a range of 80cm to 10cm. They provide a chart of the relationship between voltage and distance which is shown in Figure 15.

However, the operating range of the sensor on the robot still needs to be tested incase of any manufacturing discrepancies.

Testing this is a very simple, and simply requires checking the output voltage of the sensor after moving an object in front of the sensor at a specific distance. This can be repeated for all integer distances between 1 and 80 cm.

Also shown in Figure 15, is the actual results of testing the sensors output voltage. The curve illustrates the same basic structure as what the manufacturer provides, but outputs lower voltages overall. Additionally the curve also illustrates some noise and not the perfect curve shown by the manufacturers.

The results for this test are available in Appendix B.

5.3 Error Rate Accumulation

In order to test the combined accuracy of different types of movement a simple test that makes the robot follow a predetermined path needs to be established. In this test all three major aspects of movement (moving forward, rotating clockwise and counter clockwise) will need to be tested, as the robot tries to complete the path. The rate of error is then measured in how far the robot is from its expected finishing point.

Unlike the previous speed testing in Section 5.1, this test will allow different types of errors to accumulate over the duration of the test in order to see how pronounced they may become when actually mapping out an environment for real.

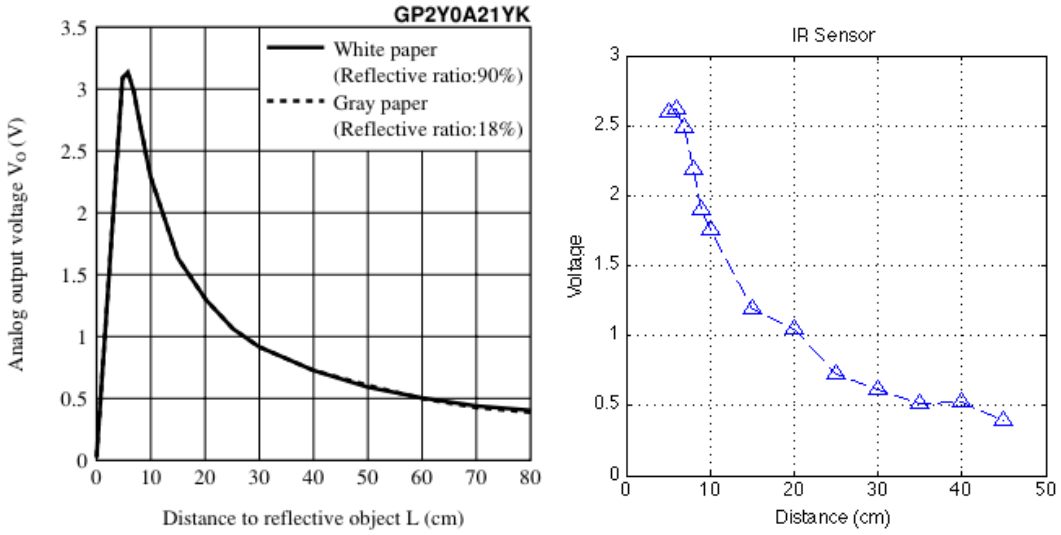


Figure 15: *The chart on the left shows the manufacturers specified operating voltages for the IR range sensor, whilst the chart on the right shows actual acheived voltages. Left chart produced by Sharp (n.d.).*

There are several paths in which the robot could be made to follow which would test these different aspects of its movement. The first test was mentioned in the Interim Report, and is called the UMBmark. J.Borenstein & L.Feng (1994, p.4-27) state that the test can be used to find the severity of different systematic errors in mobile robots by having them travel in a square path in both clockwise and counter clockwise directions.

Another possible path that the robot could be made to follow in order to test for the rate in which errors are accumulated is a triangular path, which is depicted in Figure 16.

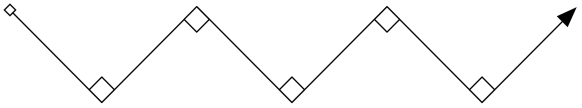


Figure 16: *A triangular path that would test clockwise, counter clockwise and forward movements of the robot.*

With this path the robot will alternate between clockwise and counter clockwise rotations after moving forwards by a single cell.

Finally a path which has the same basic shape as a square wave can be used to determine the accumulation of errors in a similar way to the triangular path, but using a different pattern. An example of the path is shown in Figure 17.

The results for this test are available in Appendix C. Figure 18 show how the X and Y offset distances are distributed pseudo-randomly away from the destination. The robot would always fail to reach the target and be to the left of it.

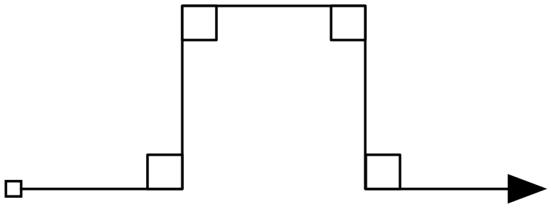


Figure 17: *A square path that would test clockwise, counter clockwise and forward movements of the robot.*

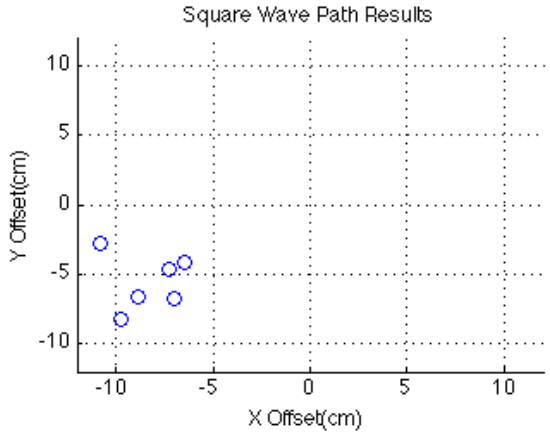


Figure 18: *The distribution of finishing locations in relation to the intended finishing location as offsets along the x and y axis.*

5.4 Simulation of Mapping

In order to be able to test the actual software independently of the robot, a simulation needs to be produced which will test all aspects of the software for the robot, whilst mimicking the actual movement and exploration of the robot. There are a number of advantages to performing a simulation in order to the test the software, over using the actual robot.

1. **Precise and accurate movements**
   This is arguably the most important aspect of a simulation. All of the physical components are not used in a simulation, and thus the errors and inaccuracies that they can introduce into the system will not be affecting the software. This is crucial in testing the software because any errors that do appear can then be said to be in the software and not the hardware.

2. **Less time consuming**
   When testing with an actual physical robot it can take far longer to carry out a single exploration to determine how well an algorithm is working. In a simulation however any physical movement such as translation and rotation can be carried out as simply changing the value of a variable. This means the time required to explore an environment in a virtual world is much less than required in a physical world.

Before any simulations can actually be run, a method for actually running them needs to be created. This method needs to take into account as many of the actual components of the system as possible in order to produce the closest result possible to the end result, without introducing the errors of the motors and sensors.

By disabling certain aspects of the on-board system on the robot and introducing a ready made map of the "environment" the vast majority of the overall system will be running as if it is an actual real world test, and not a simulation. This ready made map can be quite easily built using a Bit Array (explained in Section 3.1.1, page 12), with a set bit representing an object and a zero bit representing empty space. In turn each bit can also represent a single cell.

The on-board system on the robot can then be made to move itself around as normal using the position reference numbers to store the x, y co-ordinates that it believes itself to be at. This is the same as what happens in the real system, with the exception that it doesn't activate the motors. Therefore, as far as the robot is concerned, it is actually moving.

The code responsible for taking an IR range sensor reading can also be replaced with simulation code to simulate the IR range sensor in the ready made map. This can be done by moving along the beam projected by the "IR range sensor" and checking at each cell it passes through if there is an obstacle. It then measures the distance by incrementing it by the cell size at each step.

```
1   // Simulated World
2   uint8_t world[] = {
3       0,0,0,0,0,0,0,0,
4       0,0,0,0,0,0,0,0,
5       0,0,0,1,1,0,0,0,
6       0,0,0,1,1,0,0,0,
7       0,0,0,1,1,0,0,0,
8       0,0,0,1,1,0,0,0,
9       0,0,0,0,0,0,0,0,
10      0,0,0,0,0,0,0,0,
11  };
12  uint8_t worldWidth = 8;
13  uint8_t worldHeight = 8;
```

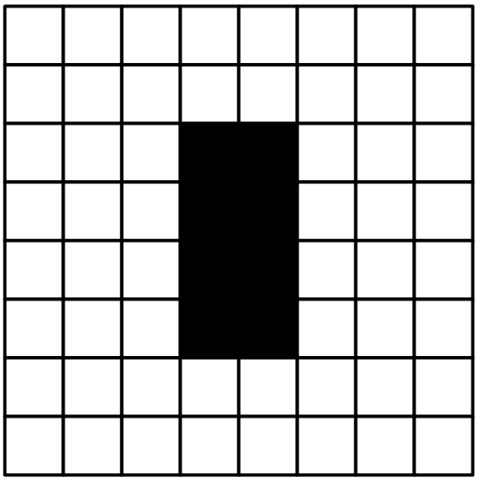


Figure 19: *A virtual world is defined as an array in the robot code, but can be visualized as 2D map.*

Figure 19 shows how the simulated world can be represented in code. The output of the simulation is shown Figure 20. This result shows the algorithms and software function

as expected.

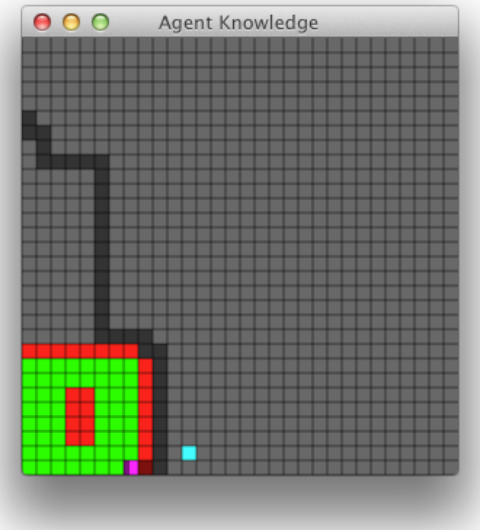


Figure 20: *The result of exploring an simulated world.*

6 Evaluation

This section will evaluate the overall performance of the robot, as well as evaluate the performance of independent aspects of the robot and look at the reasons why things have or have not worked as expected.

6.1 Robot Hardware

The performance of the hardware is perhaps one of the most critical aspects of the overall system. If the robot is unable accurately provide data back to the agent then the agent will be immediately running at a disadvantage. There are six main aspects to the hardware that need to perform correctly.

1. **Motor Speed**
   The speed of the motor needs to remain consistent from moment to moment in order to be able to produce a dependable movement.
2. **Travel Distance**
   Does the robot consistently travel a set distance when no power or timing settings are changed. This builds upon the Motor Speed.
3. **Drift**
   How susceptible is the robot to drift when travelling?
4. **Rotation**
   Is the robot able to consistently rotate an expected amount?
5. **Accuracy of Sensors**
   Does the IR range sensor of the robot produce the information expected?
6. **Reliability of the USB Communication**
   Do commands get dropped, or are there any latency and delay issues?

6.1.1 Motor Speed & Travel Distance

The motor speed is a fundamental function of the robot, and in order for the expectations and assumptions made by the agent and other software aspects, it needs to be accurate and consistent. Figure 21 illustrates how an inconsistent motor speed can affect the movement of the robot in the real world, causing the assumptions of the agent and the real world to go out of sync.

This variation in speed and movement will affect the actual distance achieved by the robot. This is has a negative impact on the *entire* system, as the robots own on-board system now assumes it to be in the centre of a different cell, as does the agent. However, in the real world, the robot has not moved far enough to be in the centre of that cell.

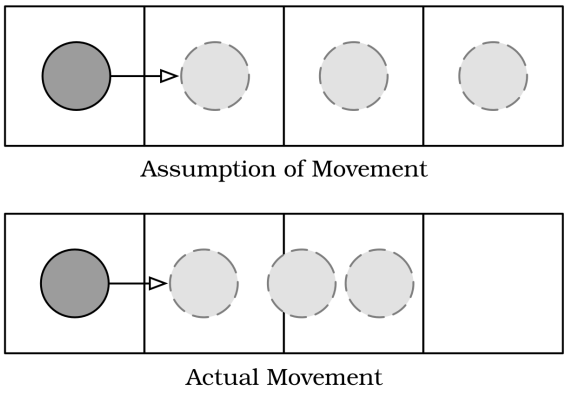


Figure 21: *The agent and software have to make an assumption of how the robot actually moved and how far it movement, which will differ to the actual movement in the real world. This difference needs to be reduced as much as possible for the accuracy of the results to increase.*

To illustrate this assume that every cell has a size of 10cm. This means that in order to get to the next cell the robot needs to move 10cm. However, internal to the software, cells are just represented via a simple co-ordinate such as `(2, 5)`. If this co-ordinate is changed to `(3, 5)` through an action that the robot has carried out, then the system assumes that the robot has moved 10cm along the x-axis, and now sits at the centre of the space represented by that cell. However, if the robot only travels 5cm then it is now out of sync with where the software expects the robot to be. Because of this variance between the real world and the assumptions of the software, any obstacles will be placed incorrectly around the robot when the software tries to determine where it is safe for the robot to travel.

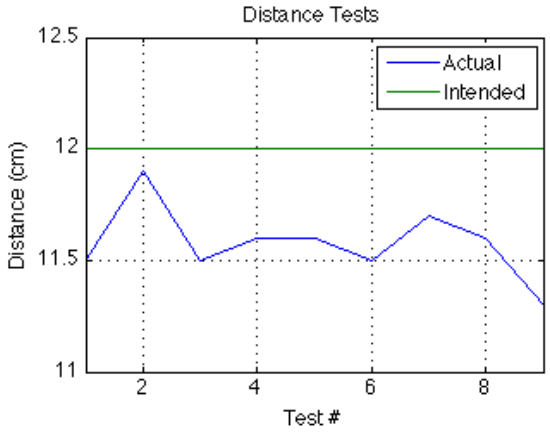


Figure 22: *The distance acheived by the robot is consistently lower than the intended distance, and varies across a 6mm range.*

Figure 22 shows how the achieved distance in the physical world varies and is consistently lower than what the agent assumes. The data for this test can be found in Appendix D. In this test the motors were set to a power level of 130 and an activation time of 0.5 seconds, as specified in Table 7 (p. 33).

6.1.2 Drift

Another important factor in the long term accuracy of the robot and its ability to provide reliable results is the amount of drift that it experiences. Drift is a negative effect caused by a power imbalance between each of its wheels and/or motors. This power imbalance might be caused by a difference in the amount of friction between the wheels and ground, actual voltage delivered to the motors, the motors being slightly out of alignment and other slight variances on the robot and in the environment. The combined sum of all these variances can cause the robot to experience a pull or push in a perpendicular direction to the direction in which the robot is travelling.

In Section 5.1, Figure 14, during the calibration of the motors the amount of drift that the robot experienced was measured and plotted as a percentage of the distance that the robot travelled. Additionally this chart shows that the robot tends towards a leftward drift, as opposed to a rightward one. This may suggest a couple of a possibilities.

- Tension from the USB cable connecting the robot to the agent host exerts a tugging force on the robot pulling it to the left.
- The right motor receives a higher voltage that the left motor.

The affect of the agent host in regards to the amount of drift can be determined by positioning it to either the left or right of the robot as it moves and checking which direction it pulls.

A number of possible solutions exist to be able to resolve this problem.

The first solution is to remove any physical connection between the agent host and the robot. Currently the robot requires 2 USB connections to the agent host, one for communication and power to the Arduino and the other for power to the motors. The USB connection used to power the motors was introduced in the improvement mentioned in Section 2.2. By switching back to an onboard power supply for the motors then the tension from the second cable will be removed. However, the reason for introducing the second cable was to provide a more stable and reliable power source for the motors to make the Motor Speed and Travel Distance consistent.

The reason for the increased reliability in the voltage delivered by a USB connection is due to a combination of voltage regulation circuitry which is used to maintain and convert a noisy voltage into a fixed level voltage. This circuitry could be moved onto the robot to help balance out the voltage delivered by a battery to help maintain a consistent motor speed and travel distance, whilst reducing the amount of tugging and drag caused by physical connections to the agent host.

The second USB connection however poses more complications than the first. To begin with, the connection is carrying a second power source, used to power the Arduino. The

reason for this second power source is to prevent voltage spikes, that are caused by DC motors, causing the Arduino to reset. This means that both the motors and the Arduino can not directly run of the same power supply.

Moreover the connection is used to carry commands and information back and forth between the agent and the robot. Without this physical connection, an alternate solution needs to be found. Two possible solutions to this are outlined below.

1. **Bluetooth or WiFi**
   Wireless technologies certainly address the problems of tension and drag induced by the USB connection(s), but they introduce another problem themselves. This is one of increased power requirements, which will reduce the life of the robot for a single charge. This can be addressed though, by using a larger battery on the robot. Another minor concern is that they also have an increased latency which may cause slowdown to the entire system.

2. **More powerful and capable onboard computer**
   The sole reason for sending back information to the agent host and determining commands there is due to the lack of power and severe memory constraints of the Arduino. By upgrading to a more powerful onboard computer the need to send back information to the agent host reduces. Some boards such as the *Beagleboard* offer an increased power and memory size, but also increase in cost quite drastically. However the recent development and availability of the *Raspberry Pi* makes it possible to cheaply put a much more powerful computer into the robot. The Raspberry Pi offers a 700MHz processor and 256MB of RAM, which is massively more powerful than the Arduino. As with the use of the wireless technologies, this is likely to reduce the lifespan of the robot.

Whilst both of these solutions do address the problem of physical connections to a computer whilst exploring, they both come with their own disadvantages. An immediate concern is that of power consumption. The first solution, Wireless, is an additional component to be placed upon the robot resulting in the battery having to be shared out between a larger pool of components. The second solution, a more capable onboard computer, will require more power to run. Another consideration to take into account with regards to the more powerful onboard computer is to do with seeing the result of the exploration. Neither the Beagleboard nor the Raspberry Pi have built in wireless capabilities and without any physical connection back to the agent host, they will require a wireless module to be added in order to communicate the currently constructed map back to the user.

In order to remove both USB connections between the agent host and robot, the onboard power systems need to be enhanced to deliver more reliable power. Wireless technologies will need to be added to the robot so that it has a means of communicating with the user (or in the case of keeping the Arduino as the main onboard computer, communicating with the agent) and possibly upgrading the onboard computer from an Arduino to something more powerful such as the Raspberry Pi or Beagleboard.

The second solution to reducing the amount of drift experienced is to alter the voltage delivered to each wheel to compensate for the drift. The idea is that when the robot begins to drift in a particular direction, one of the motors can have the voltage delivered to it either increased or decreased in an attempt to bring it back into a straight line. Immediately a problem with this is apparent; the robot has no means of detecting drift by itself, as it uses only a single fixed point IR range sensor positioned on the front of the robot. This solution would require additional sensors in order to be affective. These sensors would be able to detect the distance to objects either side of the robot and thus detect when drift is occurring. Even with these additional sensors detecting distance either side of the robot, they still have an effective range and if no obstacles are in range either side of the robot, then it will be unable to determine if any drift is happening. This makes the system only really suitable in small enclosed environments rather than wide open spaces.

6.1.3 Sensors

The robot was built with just one IR range sensor in order to keep the components cost to a minimum. However, during actual mapping tests it quickly became apparent that this single point sensor is unable to perceive enough information to be able to get a clear picture of the environment.

The problem of the limited perception offered by a single sensor is illustrated in Figure 23. In this example the robot is depicted to be "seeing" a cylindrical object directly ahead. The side view is given to show that without the larger context of the environment it is impossible to know anything about the surface that has been detected. What kind of rotation does it have? What is the size of it? These are details about the object that the robot has no means of detecting. It is only capable of detecting the distance to the point directly ahead of the IR range sensor.

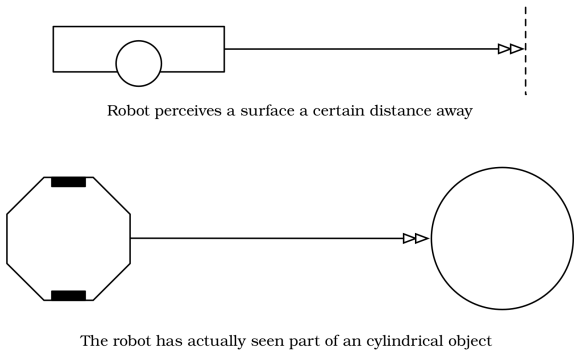


Figure 23: *The robot is only capable of perceiving a very limited amount of information from the environment when using only a single IR range sensor.*

The problem of limited perception becomes immediately obvious when an object *partially* obstructs the path of the robot, but does not cover enough of that cell to break the sensor beam of the robot. Figure 24 shows how the IR range sensor beam can pass by a closer obstacle without interacting with it and report a distance to an obstacle further away.

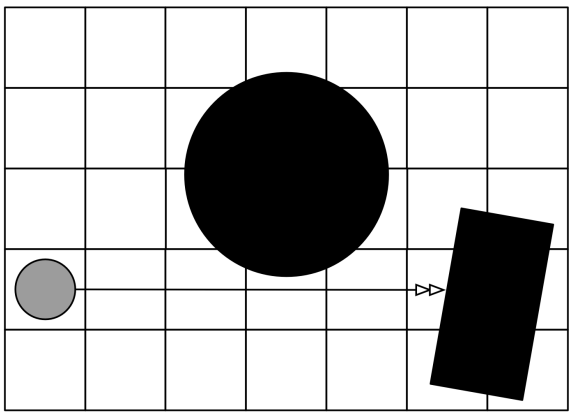


Figure 24: *The single beam projected by the sensor* must *be broken in order for the robot to detect the obstacle. This makes it possible for objects to be immediate obstacles to the robot without ever being detected.*

In order for the robot to be able to reliably detect objects and build more precise maps of the environment that it is exploring a more sophisticated object detecting system needs to be implemented. Below, several different types of sensors and setups for the sensors have been outlined as potential methods of overcoming the disadvantages of just one single point sensor.

- **Multiple Sensors**
  By increasing the number of sensors on the robot a more complete picture of the surrounding world can be gathered.
- **Sensor and Servo**
  By attaching the IR range sensor to a 360° servo, the sensor can be spun around to different angles to able to get a complete 360° view of the environment around it.

The first and most immediate solution is to simply add more sensors to the robot in order to increase the robots ability to perceive the world. Depending on the positioning of these sensors, immediately the problem of being able to detect the rotation of an object can be addressed. Figure 25 shows how placing three sensors across the front of the robot allows it to detect three distances to the obstacle in unison, and thus get an idea of the rotation of the surface that has been detected.

Additionally, this configuration of multiple sensors at different points along the robot will allow for a more comprehensive detection of obstacles within a cell. The initial problem came from the sole sensor on the robot was positioned in the middle of the robot and thus would only detect objects that passed through the centre of the cell. By adding

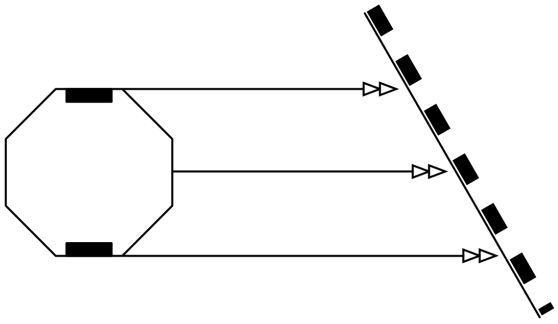


Figure 25: *By measuring the distance to an object from 3 different points on the robot in unison, a hint of the obstacles true rotation can be determined.*

additional sensors to the sides of the robot obstacles that only partially occupy a cell can also be detected.

However, the increase in sensors will put a greater demand on the power supply of the robot. Assuming a migration to a completely wireless robot this would require increasing the power capacity of the robots onboard batteries.

The second potential solution doesn't increase the number of sensors, but instead adds a servo and mounts the sensor upon it. Servos are components that can be rotated precisely to specific angles. By using a servo capable of rotating through 360° the robot can then move the sensor to look in any direction and measure the distance to it. This allows the robot to take many different measurements without actually increasing the number of sensors.

This approach has a number of advantages over the previous one. Firstly it does not increase the power requirements as much as adding several more sensors. There is the power requirement of the servo, but unlike the approach of multiple sensors, which requires power for each sensor, this approach only ever has a single sensor. This means that for each direction you wish to take a measurement, the first solution requires an additional sensor and thus more power is required, but the second solution still stays at the power needed for one servo and one sensor.

The second advantage is that it does not require as many I/O pins on the robots onboard computer. Each sensor requires a single I/O pin to be able to send information to the computer. The Arduino offers five analog pins, meaning a maximum of five different measurements can be taken. However by using the servo and sensor solution, many different measurements through 360° can be taken whilst only requiring a single analog pin.

6.2 Software

The software is responsible for automating the process of mapping the unknown environment without the need for human intervention. This software is comprised of a variety of different algorithms which work together to help the robot navigate, explore and build up an image of its environment, each of which need to work and respond in an expected and logical manner.

6.2.1 Determining Destination & Path Finding

In any autonomous system that is required to move from point to point, there needs to be a method for it to find a route between those two points. The goal of any pathfinding algorithm is to find the shortest path from the starting point to the ending point in the quickest time possible. The agent uses the A* algorithm in order to find the shortest path to an unexplored cell that has been selected through a Local Search algorithm. This cell might be anywhere in the world, from far away to right next to the robot. These cells should be further away from the robot at the start of the exploration, and gradually become closer to it as further cells become either obstructed or observed.

However, is this method of selecting a destination effective? Or could it be made more effective.

When selecting a destination the agent demonstrates a tendency to repeatedly select cells that it has no way of reaching. This in itself is largely unavoidable, due to the need of selecting the cell in order to be able to test to see if that cell is reachable. However the agent doesn't mark that cell as obstructed, but leaves it in the unexplored state. This has the negative effect of allowing the algorithm to repeatedly return to that cell and keep testing it, regardless of the fact that it has already been found to be unreachable. This oversight in the algorithm allows for it to take a potentially infinite amount of time in trying to select a reachable cell, as it repeatedly loops through unreachable unexplored cells.

This clearly is not an effective or efficient way of finding a new destination, as potentially large amounts of processing time are used up trying to find a new route to explore. Additionally anytime the agent is trying to search for a new route, the robot is idle and consuming power.

The most apparent solution to this is to mark those unreachable cells as obstructed, thus narrowing the search scope down with each successive failed attempt to find a new path. This does not address the problem completely though, as the agent still may potentially scan through large amounts of the unreachable environment before finding a new route to take.

Another solution that would help reduce the amount of search time for new routes, is to set the remainder of a path to obstructed if the agent is forced to cancel and redetermine a path. Currently the agent will redetermine the cell with the best cost as its destination if it encounters an obstacle in its path, and thus recalculate its route. Prior to choosing this new destination, the remainder of the path could be immediately traversed by the agent and each cell it passes through set as obstructed. If the agent later manages to directly observe those cells that were automatically set to obstructed the probability of the cell being occupied will be reduced, making it possible for the robot to be pass through those cells. However, the number of unobserved cells will have diminished resulting in a smaller search space to test when searching for a new route.

The current performance of actually calculating the path is very good. Figure 26 shows how the agent is able to determine the route through an incomplete maze to reach its destination.

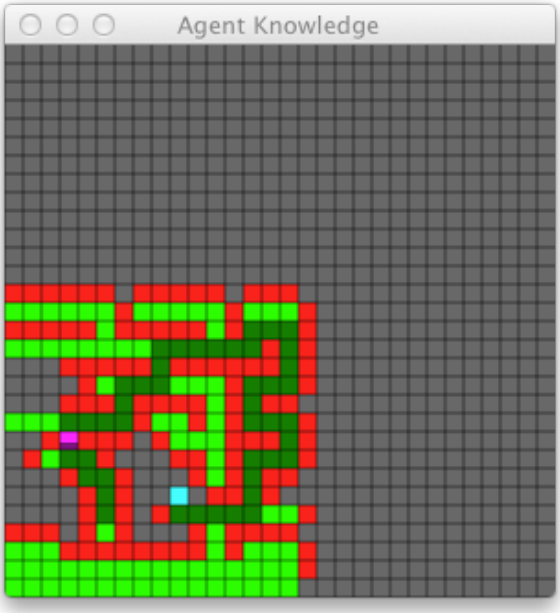


Figure 26: *This illustrates how the path finding algorithm is able to determine the shortest route through an incomplete environment to its destrination, successfully determining that it can travel through unoccupied cells and not occupied cells. The cyan cell represents the destination and the magenta cell represents the robots current location.*

Ultimately this shows that even though the agent is able to determine the shortest path to take, due to potentially checking the same unreachable destination cell repeatedly, it may never find a valid path to explore.

6.2.2 Exploration

One of the core premises of the project was to find a way in which a robot could explore an unknown environment to construct a map of it. In order to accomplish this task it had to be able to explore that environment in such a way that it would not miss any important aspects of the environment.

To do this the robot needed to employ a search strategy that would not only collect as much information about the environment as possible, but also compensate for the limited sensor capabilities of the robot. The strategy employed by the agent currently, is to rotate through 360° at each cell it visits and measure the distance to the nearest obstacles at each 90° interval. This allows the robot to build up a more complete picture of its environment in a single pass rather than having to repeatedly return to places to complete its exploration.

There are a number of problems in this approach however. The first major one is that the problem of drift experienced by the robot when rotating is largely compounded in doing this. For each observation the agent wants to make, the robot needs to rotate 90°, leading to a drift error being added four times in the course of one set of observations. Add this up over the course of its exploration of the environment and this could very quickly become large enough to make results meaningless.

The second problem is with the USB cables connecting the robot to the agent host. As the robot is made to spin around to take measurements the cables will be twisted together making them increasingly taut, leading to a increased pulling on the robot and increasing its drift.

The third problem is that this spinning motion is very inefficient. In order to look around itself the robot is having to expend a lot of power rotating around just to take measurements. As discussed in Section 6.1.3, increasing the number of sensors on the robot will help to address this problem, particularly if the increase in sensors allows it to take measurements in all directions without needing to rotate. This could be accomplished by putting a single sensor on each side of the robot, or attaching a sensor to a servo.

An alternative method of exploring the environment is to simply take measurements from directly ahead of the robot. This approach however is not an optimal solution. Figure 27 shows how the picture of the environment doesn't change as the robot moves from one cell to the next when using only a single sensor, whilst when taking measurements in all directions around the robot it builds up a picture far quicker.

This shows that although the method of taking measurements from all around the robot is far quicker and thorough in collecting information about the world, unless the robot has multiple sensors, or a servo mounted sensor, it requires a lot of excessive spinning

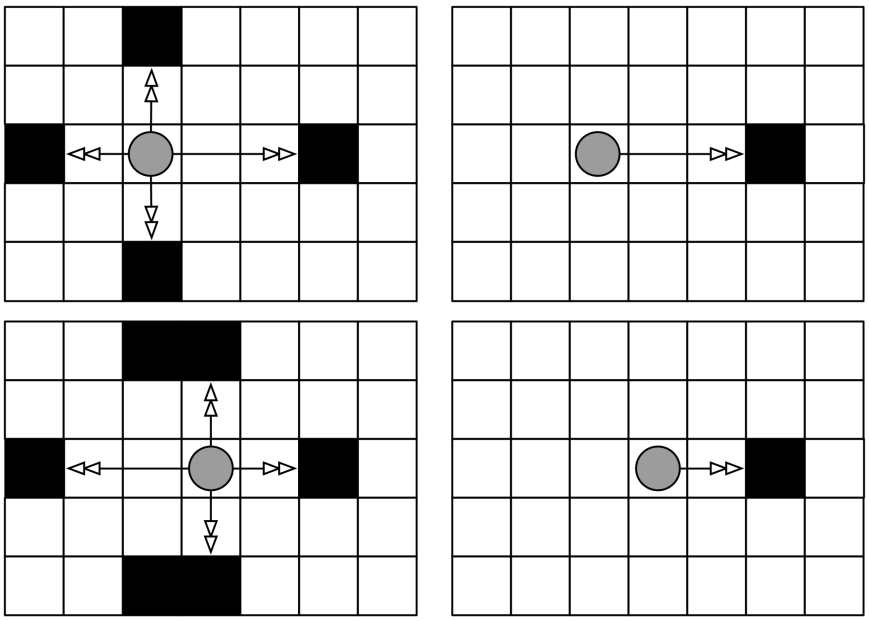


Figure 27: *By taking measurements in four different directions the robot is able to build up a picture of the environment far quicker than it would be able to if it just took measurements to obstacles directly ahead.*

which can lead to a larger build up of errors and inaccuracies.

6.3 Overall Performace

It was shown in Section 5.4 that the software for the agent all works correctly and as expected. However this does not demonstrate the performance of the entire system, including the movement and sensor functions.

Figure 28 shows the progress of the robot as it maps out a physical environment that is completely empty. Immediately it is obvious that the exploration is not happening correctly. Firstly, the robot is only ever meant to go horizontal and vertically, *not* diagonally. This movement error in the physical world can be see in the virtual world representation given by the agent, and shows how the problem of drift leads to completely unreliable results. Figure 29 further shows how the imprecision and inconsistencies of the robots movements prevent it from being able to produce a reliable result.

Additional images of explorations can be found in Appendix E.

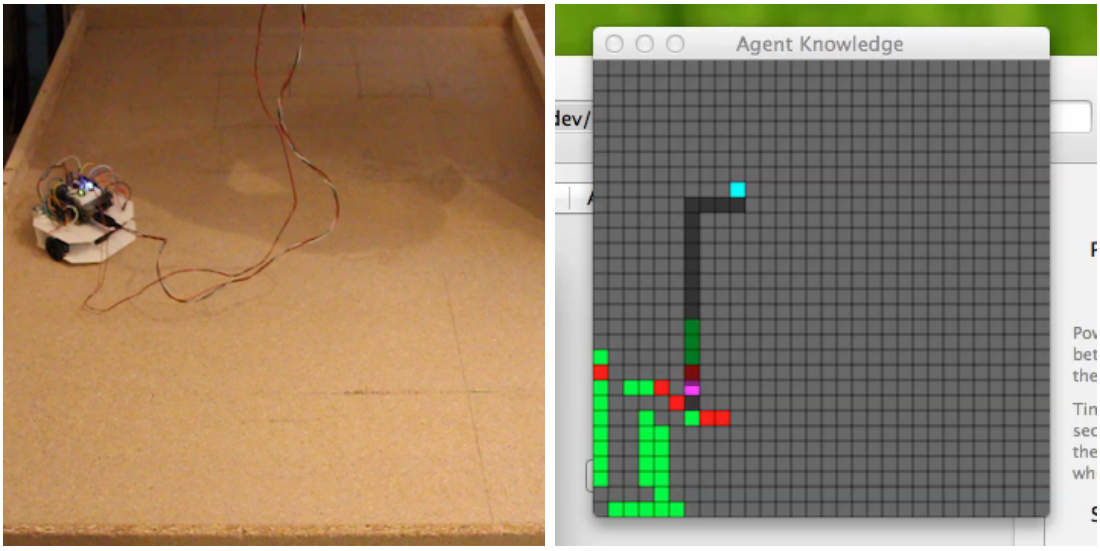


Figure 28: *An example of results produced by the robot as it maps out an empty environment. These results demonstrate how the movement functions of robot are imprecise and inconsistent.*

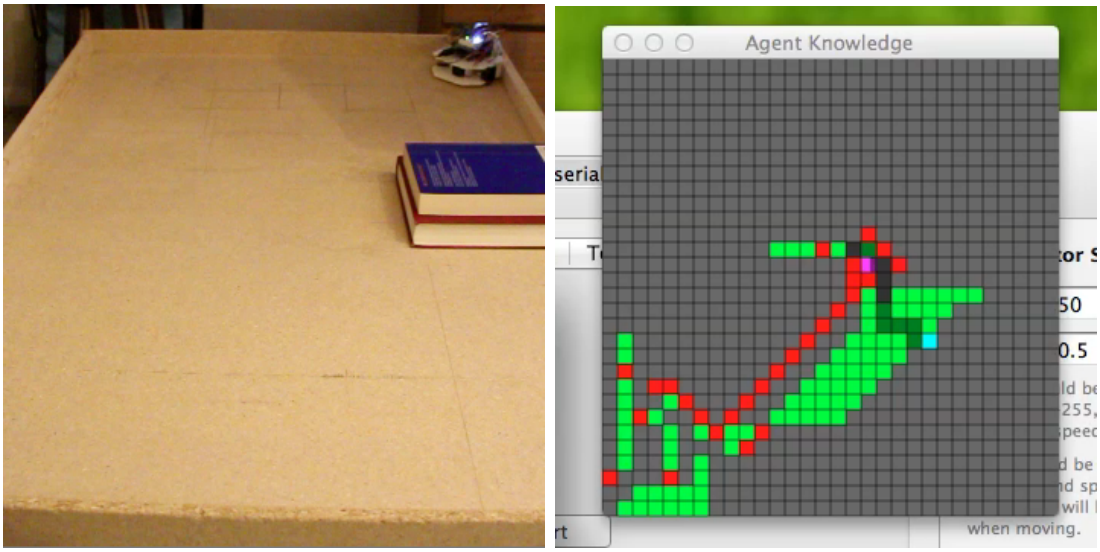


Figure 29: *An example of results produced by the robot as it maps out an obstructed environment. These results reiterate how the movement functions of robot are imprecise and inconsistent.*

7 Future Work

The scope of the project was to determine the feasibility of mapping unknown environments using a robot that had been assembled for an extremely low cost. Advancements in technology, even in the space of a year, have brought relatively large computational power to a small single board computer, the Raspberry Pi, that is available for a price similar to the of the Arduino. As discussed in Section 6.1.2, this alone will allow certain limitations of the current robot, such as drift to be overcome. The cheap availability of this increased processing power which can be put onboard the robot, opens the scope

to other techniques of perceiving the environment such VisualSLAM. VisualSLAM uses standard images captured by a camera (such as a Webcam) to determine where objects are in relation to the robot.

VisualSLAM algorithms are becoming increasingly sophisticated and as the available power, and small sizes of low end computers increases, the ability to incorporate such algorithms in to cheap robots becomes easier. This ultimately means that the dependancy of a large array of range sensors onboard the robot becomes less important, as the information these sensors gather can be inferred from images captured using VisualSLAM. This opens up the possibilities of integrating such techniques in to the robot to help it build a clearer and more thorough map of the environment than is currently possible with just a single sensor.

Even without going to VisualSLAM, the accuracy and efficiency of the actual robot has many aspects that need improving. Perhaps most importantly of all, a solution to the drift problem needs to be found. When running the exploration algorithms in both the simulator and real world, there is one factor that is not simulated, that happens in the real world; Drift. Currently the robot is unable to produce a map in the real world due to errors caused by drift adding up very quickly. This shows that drift is the most immediate and critical issue that needs to be resolved. In Section 6.1.2, numerous methods were addressed that may have the potential to reduce drift. These are outlined below.

1. Wireless communication to the user's computer. Bluetooth, WiFi or Radio?
2. The agent sits onboard the robot, rather than on the computer, meaning that only a map of the environment is sent back to the user.
3. On-board battery with regulated voltage supplying power to the onboard computer, motors and sensors.
4. More powerful onboard computer, such as the Raspberry Pi.
5. Additional sensors to determine if the robot is drifting.

Each of these possible solutions needs to be explored in further detail to determine exactly what benefit it will have in addressing drift. Is it possible to do so by only implementing one or two of these methods, or would all of them need to be implemented? Do any of them address the issue, or is something more complex needed?

There is also much room for expansion in the agent itself. The basic structure used works with a single sensor and can be improved with little effort to support several sensors measuring distances all around the robot, but it will not be able to handle input from a VisualSLAM algorithm should that be implemented. Moreover, if VisualSLAM should be implemented how will this effect world representation? Can VisualSLAM be made to work with the methods discussed in Section 3.1, or will new methods need to be explored?

Moreover, the search strategy of the agent needs to be improved to help prevent it from

getting stuck in loops, or carrying out large amounts of unneeded actions. As mentioned in Section 6.2.1, numerous alterations to how the agent handles paths that have become obstructed could be made to help reduce the search space available to the agent and thus help to prevent loops and getting stuck.

It is also clear that any future work conducted is going to require a larger budget than what was allocated for this project, as a larger amount of hardware is likely going to be required.

8 Conclusion

Autonomous robots that deal with exploration of unknown environments have historically been expensive to produce. Ranging from the Mars Exploration Rovers at $820 Million (AssociatedPress 2007) to cheap robot vacuum cleaners such as Samsung's NaviBot Robotic Cleaner at £330, the cost of this has not yet dropped below the £100 mark. The aim of this project was to determine if this was possible to do. From the tests carried out using a robot constructed for around £70 it certainly seems as though it is not.

However, as mentioned in Section 7 the recent advances in computation power available on cheap, small onboard computers is making what were previously harder to access algorithms (i.e, VisualSLAM) due to their higher computational demands more readably available. This means that it is plausible that such robots will become possible within the next few years for less than £100.

Many of the issues in robotic exploration lie in the hardware of the robot. Many factors can undermine the affectiveness of the robot, such as imprecision in alignments of components, sensor inaccuracies, unreliable voltages, weight distribution through the robot or even components failing. Each of these problems when identified is not as easy to correct as problems that occur in software. Very often when a problem occurs it may require a new part to be bought in order to fix it, or the complete design to be altered in order to correct the alignment of a single component that was out. The hardware issues can become very expensive and very time consuming. Many of these issues can be overcome with access to high precision tools, which in themselves can be costly.

It is far easier to make software precise however, as everything is dealt with in a digital and mathematical world where altering several values in source code can change the outcome of an algorithm instantly. This ability to rapidly develop and engineer the algorithms behind robotic exploration means that the software costs of such robots can be potentially much lower than the hardware costs.

9 Reflection on Learning

Now that the project is complete, I can look back at aspects of the project that have either posed challenges and complications, and how I would approach these differently in the future.

A major part of the project that I believe I could have done better, is the construction and designing of the robot. Much of the design and planning for the robot was done for the software aspects of it, rather than the hardware. The result of this was that many parts of the robot were either not constructed optimally, or had to be modified to correct shortcomings that were identified after robot construction had been completed.

Additionally, time should have been allocated in the project plan to learning various skills required to build the robot, rather than trying to learn whilst assembling it. This was particularly evident in the soldering required for some components. Several times whilst soldering components, the heat from the soldering iron damaged them to the point of not working. By learning the required skills to solder electronic components, prior to beginning construction of the robot, would help to overcome the problem of heat damage to components.

This ultimately meant that several components either did not work once attached (and thus required replacements where possible), or were intermittent, requiring modifications to be made to the robot.

The time management of the project was generally good, although due to some bugs and problems that arose in the software for the agent, a large section of it had to be redeveloped causing a delay to progress. Although this type of problem can not be 100% anticipated, better planning for the project can help to pin point where potential problems may exist.

Glossary

A* A widely used pathfinding and graph traversal algorithm due to its ability to efficiently find optimal paths between two points. 14–18, 44

Actions Queue A first-in-first-out (FIFO) queue in which the agent adds actions to ready for them to be performed at a later time. 27, 28

Agent Something the perceives and acts in a rational and logical manner. (Russel & Norvig 1995, p.7). 5, 6, 8, 10, 12–15, 17–28, 37, 38, 40, 44–47, 49

Agent Host The computer upon which the Agent runs. 24–26, 39, 40, 46

Arduino An open-source prototyping platform. 5, 7, 8, 11, 21–25, 29, 39, 40, 43, 48

Autonomous System A system that can operate and make decisions independent of any external control. 44

Baud Transmission speed, or how many symbols/pulses per second are used. 22, 23

Beagleboard Like the Arduino, the Beagleboard is a prototyping platform. However it is based on the ARM architecture and offers a 1GHz processor and 500MB of RAM. It is available for $150. 40

Bit Array An array in which every value is represented by a single bit. Typically accessing these values requires *Bit Operations*. 10–12, 35

Breadboard (Protoboard) Used for prototyping electronics without the need for soldering. 8

Confidence Map A form of matrix which holds values, or counts, representing probabilities of something. 11, 13

Cost Function A function that calculates the benefit of doing something, and producing a score (cost) as a result. 14

Drift An undesirable product of systematic and nonsystematic errors which cause movement, perpendicular to the desired direction. 20, 29, 37, 38, 41, 46, 47, 49

IR Range Sensor An electronic component which emits a single infrared beam which can be used to measure the distance to an object. 19–22, 25, 30–32, 35, 37, 41, 42

LiPoly Lithium-polymer is a type of rechargeable battery which is composed of several "secondary" cells in parallel to increase the discharge current capability. 8

Local Search A meta heuristic method for solving computationally hard problems. 5, 14, 15, 17, 18, 44

Matrix A rectangular array of values which can be stored as a 2-dimensional array. 10, 11, 13, 17

Motor Power The value given to the Pulse-With-Modulation (PWM) output of the Arduino chip to vary the speed of the robot's motors.. 29–31

NiCad Nickel-cadmium is a type of rechargeable and readily available battery, typically available as AA, AAA or 9V Batteries in shops. 8

On-board System The software that runs upon the Arduino on the robot. 19, 22, 24, 25, 35, 37

Plasticard High Impact Polystyrene Sheet (HIPS) is a material commonly used in model building. 7

Pulse-With-Modulation By rapidly alternating between 5V and 0V on a digital signal, an analogue signal can be simulated which allows for a varying voltage and current to be delivered to components. This method can be used to change motor speeds, the brightness of LEDs, etc. 25, 29

Raspberry Pi A small, cheap computer that is the size of a credit card. It provides a 700MHz processor and 256MB of RAM, and is available for £20. 40, 48, 49

Search Strategy The route constructed by the agent to find information about its environment that is currently unknown. 11, 14, 46, 49

VisualSLAM Visual simultaneous locatisation and mapping, is a technique used by robots to build up knowledge of its environment through the use of a webcam, or similar image capturing devices. 48, 49

Appendix A

These are the results from the Speed Tests for the robot. The purpose of the Speed Tests was to determine how fast the robot was able to travel at different levels of power delivered to the motors.

Motor Activation Time	Motor Power	Speed (m/s)	Drift (m)
0.5	50	0.031	0
0.5	50	0.029	0
0.5	50	0.031	0
0.5	55	0.034	0
0.5	55	0.035	0
0.5	55	0.037	-0.0005
0.5	60	0.042	0
0.5	60	0.048	-0.0005
0.5	60	0.042	0
0.5	65	0.048	-0.001
0.5	65	0.049	-0.0005
0.5	65	0.053	0
0.5	70	0.058	-0.001
0.5	70	0.056	-0.0005
0.5	70	0.059	-0.0015
0.5	75	0.0625	0
0.5	75	0.065	-0.00015
0.5	75	0.066	-0.001
0.5	80	0.071	-0.0015
0.5	80	0.0695	0
0.5	80	0.074	0
0.5	85	0.082	0.0005
0.5	85	0.082	0
0.5	85	0.082	0
0.5	90	0.088	0
0.5	90	0.09	0
0.5	90	0.091	0
0.5	95	0.096	0.0005
0.5	95	0.097	0.001
0.5	95	0.096	0
0.5	100	0.101	-0.0005
0.5	100	0.1015	0
0.5	100	0.0985	0.0005
0.5	105	0.105	0
0.5	105	0.1015	-0.001
0.5	105	0.106	-0.001
0.5	110	0.109	0
0.5	110	0.107	0
0.5	110	0.107	0

Motor Activation Time	Motor Power	Speed (m/s)	Drift (m)
0.5	115	0.1135	-0.0005
0.5	115	0.1135	-0.00025
0.5	115	0.114	0
0.5	120	0.119	0
0.5	120	0.1175	0
0.5	120	0.1185	-0.0005
0.5	125	0.1215	0.001
0.5	125	0.122	0
0.5	125	0.123	0
0.5	130	0.124	0
0.5	130	0.124	0
0.5	130	0.119	0
0.5	135	0.12	0
0.5	135	0.125	-0.001
0.5	135	0.126	-0.0005
0.5	140	0.126	0.0005
0.5	140	0.1285	0
0.5	140	0.1285	0
0.5	145	0.131	-0.00025
0.5	145	0.1335	-0.00025
0.5	145	0.135	0
0.5	150	0.133	-0.0005
0.5	150	0.136	0
0.5	150	0.1365	-0.0015

Appendix B

These results are for measuring the accuracy of he IR Range Sensor on the robot. The results were collected by measuring the output voltage of the sensor with an object positioned at different distances from it.

Object Distance	Output Voltage
5	2.6
6	2.62
7	2.49
8	2.19
9	1.9
10	1.76
15	1.19
20	1.05
25	0.73
30	0.61
35	0.52
40	0.53
45	0.39

Appendix C

These results come from the resulting offset of position of the robot after following a square wave path, from where it should have been.

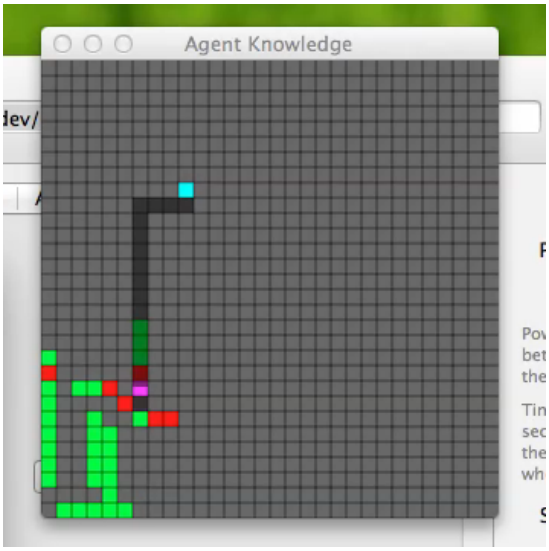
Offset Distance X	Offset Distance Y	Offset Distance
-10.8cm	-2.8cm	11.2cm
-8.9cm	-6.6cm	11.1cm
-7.3cm	-4.7cm	8.7cm
-6.5cm	-4.1cm	7.7cm
-7.0cm	-6.7cm	9.7cm
-9.7cm	-8.2cm	12.7cm

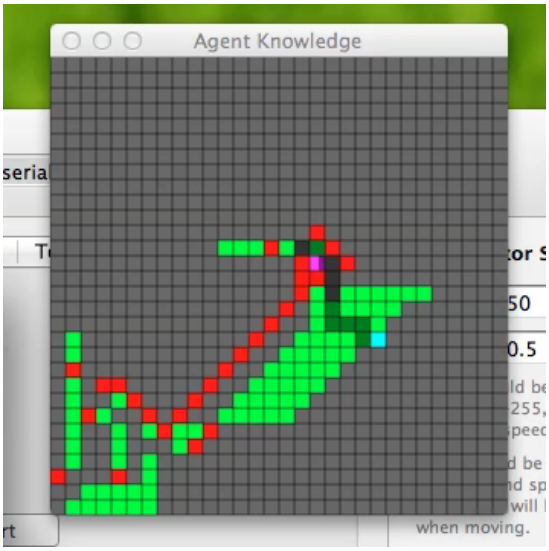
Appendix D

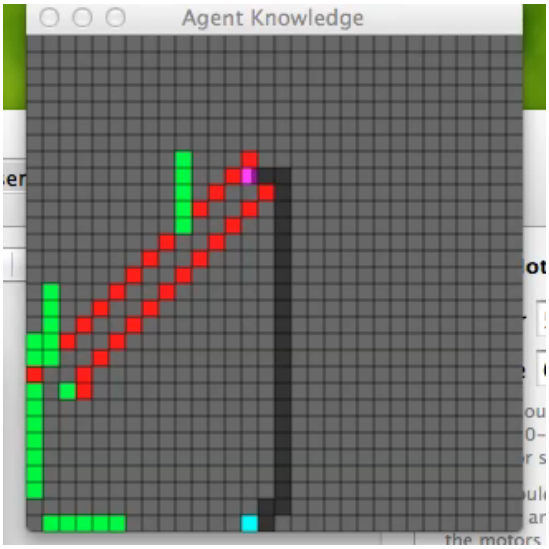
These are the results are from the Distance Tests to determine how accurate and consistent travel distance of the robot is. Negative values on the X axis represent an offset to the left, whilst negative values on the Y axis represent being short of the target.

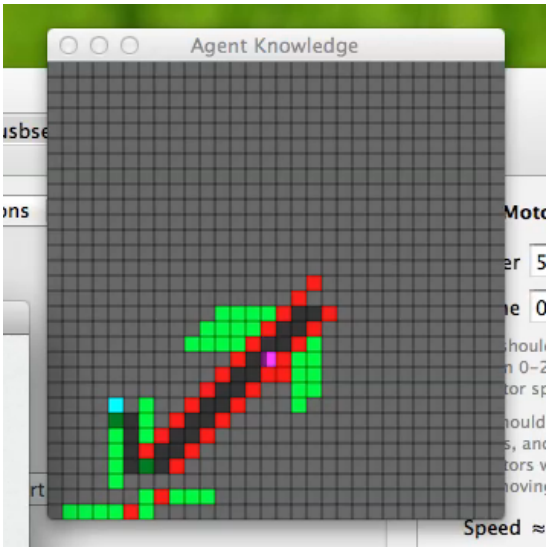
Motor Power	Motor Activation Time	Distance Traveled (cm)
130	0.5	11.5
130	0.5	11.9
130	0.5	11.5
130	0.5	11.6
130	0.5	11.6
130	0.5	11.5
130	0.5	11.7
130	0.5	11.6
130	0.5	11.3

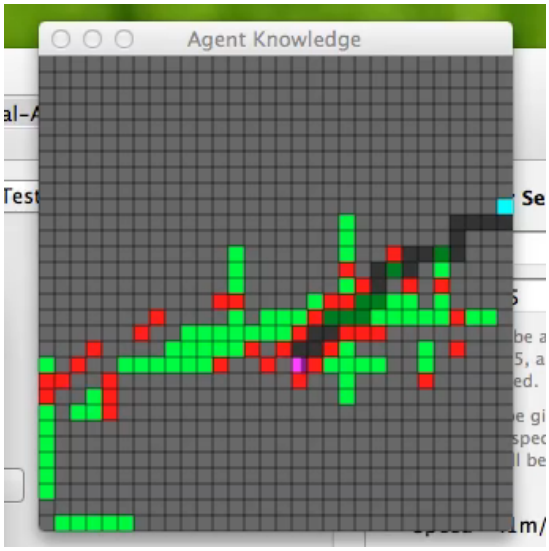
Appendix E

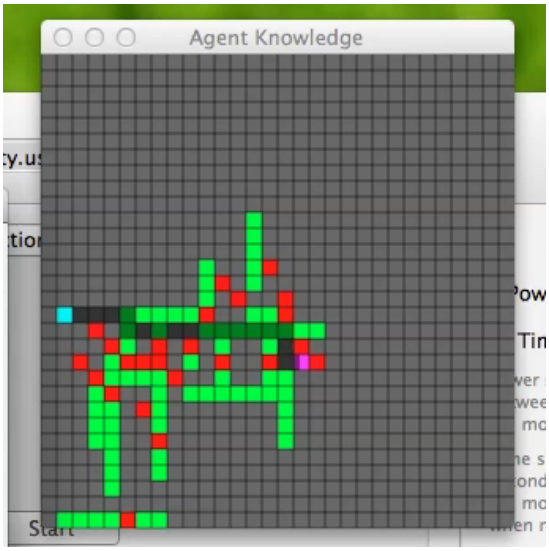

Agent Knowledge


Agent Knowledge
when moving.


Agent Knowledge


Agent Knowledge
Speed


Agent Knowledge


Agent Knowledge

References

Arduino (2012), 'Serial'.
URL: *http://arduino.cc/en/Reference/serial*

AssociatedPress (2007), 'Nasa extends mars rovers' mission'.
URL: *http://on.msnbc.com/rNvhRn*

Chill, J. (2006), 'Zellen aerial image of room'.
URL: *http://www.juergenchill.com/photography/zellen/z_08.htm*

ElectroDynamics, I. (2011), 'Demistifying nicd's part 3'.
URL: *http://www.electrodynam.com/rc/totm/totm1100.shtml*

J.Borenstein & L.Feng (1994), Umbmark: A method for measuring, comparing and correcting dead-reckoning errors in mobile robots, Technical report, University of Michigan.
URL: *http://deepblue.lib.umich.edu/bitstream/2027.42/3753/5/bac6477.0001.001.pdf*

Oomlout (2012), 'Sensing distance'.
URL: *http://oomlout.com/PROX/PROX-Guide.pdf*

Russel, S. & Norvig, P. (1995), *Artifical Intelligence: A Modern Approach*, Prentice-Hall, Inc, USA.

Sharp (n.d.), 'Gp2y0a21yk/gp2y0d21yk: General purpose type distance measuring sensors'.
URL: *http://oomlout.com/DATASHEETS/IC-IRPROX.pdf*

Tracy, K. W. & Bouthoorn, P. (1997), *Object-Oriented Artificial Intelligence using C++*, Computer Science Press, USA.

Weisstien, E. (2012), 'Distance'.
URL: *http://mathworld.wolfram.com/Distance.html*

WormFood (2005), 'Wormfood's avr baud rate calculator'.
URL: *http://www.wormfood.net/avrbaudcalc.php*