

HAPR Report

Word count: 4803

1. Abstract

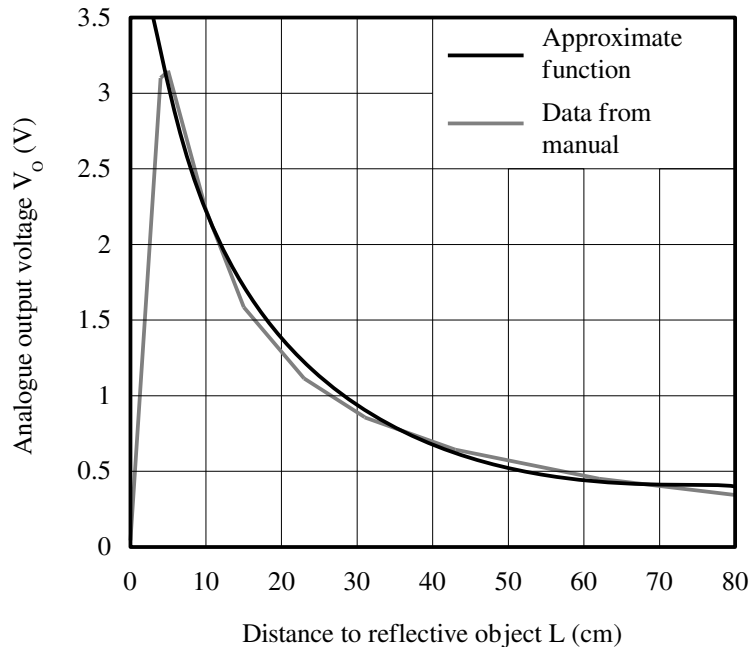
The main aspect of the project is revolved around the ARM Cortex-M3 based mbed NXP LPC1768 micro-controller and it's integral application to a mobile robot system. The basic hardware configuration was already set-up using a Pololu m3pi which is a dual layer chassis which houses two motors on the base. Both layers have a wide variety of I/O capabilities which allow connectivity with other digital or analogue components, the top layer has a socket to interface itself with the mbed micro-controller. There are additional components that have been added to the package which include: four analogue infra-red distance sensors, a digital collision sensor and a makeshift USB mouse sensor appended to the rear of the robot to measure displacement. Making use of the entire configuration to achieve a set of tasks will be done by implementing abilities and actions for the robot to perform. This is done by first designing a system that can tackle these defined problems by considering the constraints and abilities of the hardware available in addition to possible environmental confounds. Implementation was done using the language C as it offers low level control, as no operating system is available due to very constrained resources drivers for the chip were used. This was so that we do not have to set registers manually and have extensive knowledge about the inner workings of the mbed chip which would take a long time to learn and would reduce development time as some of the hardware was abstracted away.

We were required to achieve a set of basic of tasks which involved the robot to be able to move parallel to walls and follow lines on the ground as well as determine its position and direction. These high level tasks would be split into lower level ones such as adjusting the speed of each individual motor to remain on top of a line or reading infra-red sensors and making dynamic decisions based on returned values. Existing developed abilities of the robot can be improved such as being able to slow down and prevent a collision with an obstacle then resume navigating when it had been removed.

2. Group Project Functional Overview

2.1. Infra-red Sensors

There are two alternative methods to implement effective use of the analogue infra-red sensors: a lookup table or an approximate function. The latter was chosen because testing revealed that all four sensors produced results with insignificant variation between them. A



second factor was that noise was causing problems which can only be alleviated by having a massive sample to produce a reliable lookup table. It was realised that sensors follow a pattern that follows a 1 over x graph, so this was used at the base of the function. Various transformations were made and a similar function was found:

$$f(x) = \frac{1}{2} \cdot \left(\frac{100}{x+8} \right) - 0.25$$

This has been mapped on a graph (adapted from the manual) to show

its similarity to sensor specification from the manufacture. In order to make use of the

discovered function it must be rearranged to make distance the subject so that it can be found based on voltage. To the left is the final function where y is voltage as on the graph. The sensors are only usable between 5 and 40cm, so if the robot is moving towards something and is less than 5cm away it will report that it is getting farther away. This is a limitation of the sensor, which can only be handled in software if this event happens. Although the sensors are usable above 40cm they are not reliable due to noise, as noted from the graph a small change of 0.1V can lead to a ~15cm change between 50-80cm, elsewhere this will result in a much smaller alteration of distance.

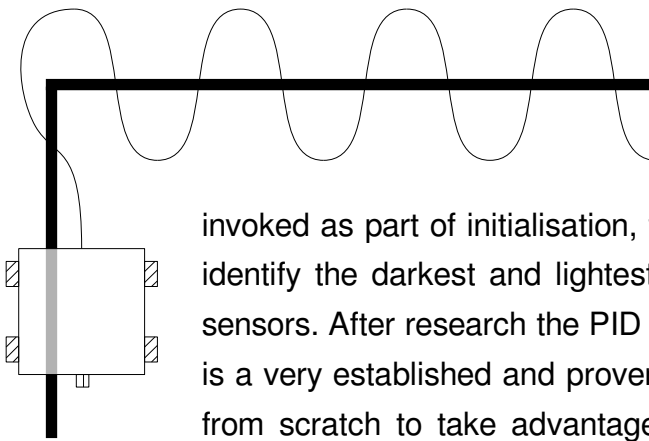
The digital front sensor is at its default configuration, when it triggers it will fire an interrupt that can be handled by the mbed as it sees fit. If activated then the robot will stop until the hazard is removed then the robot will continue as normal.

2.2. Motor control

All commands to the motors are sent via UART to the controller on the Pololu, this component is the most developed and important component. In order to move in a straight line there were two competing methods both of which were implemented. It is known there are small mechanical differences between the motors which typically leads one to being more

powerful. This can be handled with motor biasing which manipulates data before it is sent to the motors to send more power to the weaker one. This can be time consuming to set up as each robot is different and calibration needs to be accurate which is also time consuming. An alternative method exists in the form of utilising the wheel encoders. Each wheel is painted with six white stripes spaced equally, a sensor can detect if there is a change in reflection, hence there are twelve ticks per revolution of a wheel. The rate at which each wheel ticks compared to the command it has been sent can determine if a motor is not moving quickly enough. If the tick count does not match the speed issued the motor then the speed is increased or reduced depending on the case, to help maintain straight movement. The wheel encoders have a second useful ability, as the wheels have a diameter of 3.2cm which can be used to get the circumference: $3.2\pi = 10.05\text{cm}$. When used in combination with motor RPM the distance travelled can be calculated as after time frame each motor RPM is averaged and divided by 12 and multiplied by 10.05, so after a revolution $\sim 10\text{cm}$ should have been covered. This is still a form of dead reckoning and subject to accumulated errors but is much more reliable than a purely time based implementation which requires calibration based on different surfaces on a per robot basis. Rotating is based on a 180° set using motor bias, if a parameter of 270° is passed then the robot will rotate -90° because it is quicker and a shorter rotate will have a smaller percentage error.

2.3. Line Following



Line following uses the five sensors on the base of the Pololu to identify dark and light spots on the floor to identify lines.

First they need calibrating which is

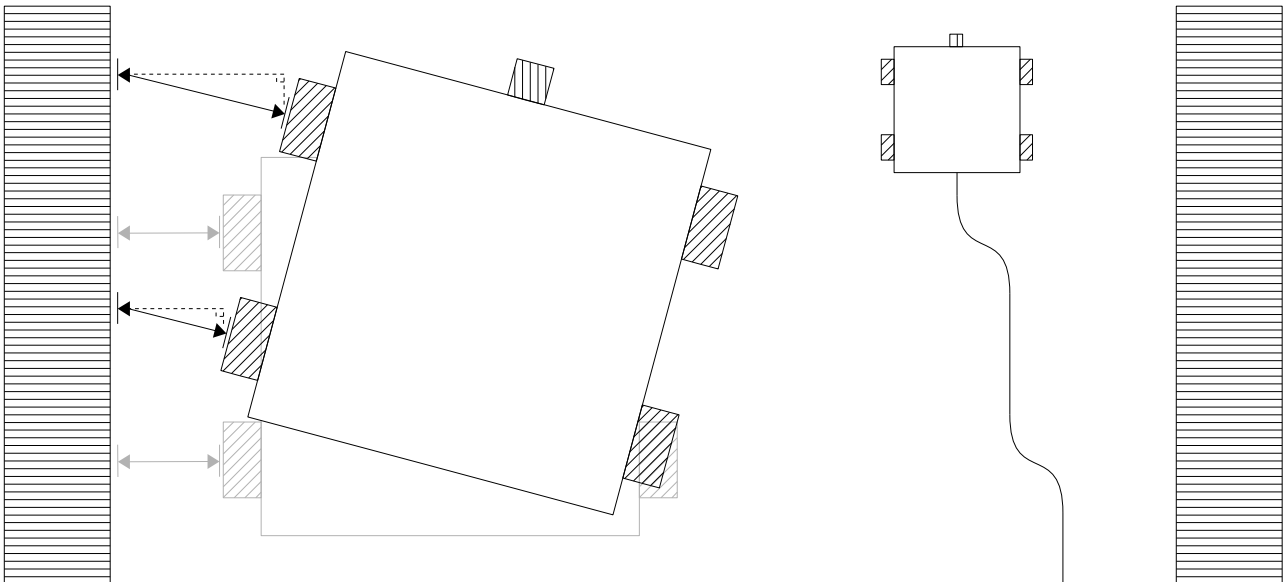
invoked as part of initialisation, the robot moves left and right over a line to identify the darkest and lightest parts in which to base thresholds for the sensors. After research the PID algorithm was chosen for line following as it is a very established and proven method, it was adapted and implemented from scratch to take advantage from all available sensors at disposale.

Calling the Pololu chip to ask for the bottom sensor data returns them in an array of five elements, each element ranges between 0-1000 these are used to determine the location of the line via a function. It returns a value in the range of ± 2000 , 0 meaning the robot is directly on top and 2000 for farthest right and -2000 for left. This is done by checking for the highest sensor and checking adjacent ones to see how dark they are to determine the position. If one is has a value of 800 and the one next to is 200 then the line is between them but closer to the first one. This is used by the our implementation of PID which essentially smooths out turning by small adjustment, the size of which is determined by how far away the line is. It follows a sine like wave like the diagram above (which has been exaggerated for

demonstrational purposes). If the parameters are too tight then the robot could turn too suddenly and lose the line, by having loose parameters turning round corners is not possible. A compromise was made as it was difficult to dynamically adjust the parameters because after a critical event had been detected it was already too late to be handled by PID. Instead it was deemed better to have loose parameters and detect whether a corner has been found and rotate accordingly.

2.4. Wall Following













Wall following is mainly devised of the cooperation of three algorithms, one of which tries to maintain a fixed distance from the wall while another is responsible for being parallel compensates and the last algorithm handles obstacles. The algorithm to remain parallel does so by reading the appropriate sensors and comparing the readings to each other. Should the difference between the two sensors be greater than a threshold then action needs to be taken. If the robot is veering away from the wall the inner motor is slowed down until the sensors are within threshold again, naturally should the robot be heading into the wall the outer motor is slowed. One aspect that needs to be considered is that when the robot is at an angle to the wall the sensors can over or underestimate readings. The left diagram shows a non-parallel robot superimposed on a parallel one, a slight angle can lead to big changes in readings due to the rules of Pythagoras theorem. If we take the sensor line of sight as the hypotenuse (c) then a change in the opposite (a) or adjacent (b) line will have a more significant effect on it as a result of $a^2 + b^2 = c^2$.



The correct distance algorithm checks at regular intervals if the robot is within its predefined spacing from the wall. If not then it takes control from the wall parallel algorithm and makes a shift away or towards the wall depending on the robot's distance from it. Control is handed back to wall parallel, later a check is taken again and if the distance is not correct then another shift is made. This produces a staircase effect as the diagram above shows until the

robot is in the correct position. This is used to correct from a bad starting position or the result of evading an obstacle which has left the robot off course a little. The last algorithm simply stops the robot when an obstacle is found ahead and continues when removed.

2.5. State Machine

		<i>Main</i>		<i>RIT Handler</i>	
Opmode		Current state		Trigger for next state	
Ø		Wait for button press		-	
1		Move to (x,y)		Detect wall? Detect line?	
2		Wall follow		Wall end?	
3		Move to (x,y)		Detect wall? Detect line?	
4		Wall follow		Wall end?	
5		Move to (x,y)		Detect line?	
6		Line follow		Detect dock?	
Ø		Hold and wait		-	

This is the state machine used to handle the track as part of the group demonstration. It uses a counter to determine which state it in, at regular intervals the RIT Handler fires an interrupt and based on the current state checks if the requirements for the next state are satisfied. To prevent falsely identified changes to the next state the condition needs to be satisfied for several continuous checks to compensate for random glitches and noise from sensors.

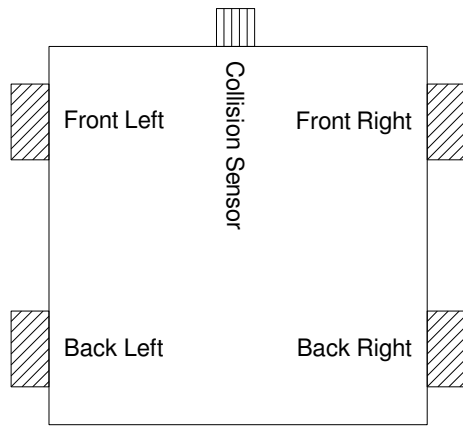
3. Individual Project Functional Overview

The goal of my individual project was to have the ability to map the robot's environment to a displayable image, this would be done using data infra-red sensors. The data is transmitted to a host computer at real-time which is logged to file. This is done by using the dedicated USB port on the mbed board providing a serial link between the robot and a computer. The host listens for data from the robot and writes the bytes directly to file. Also it is possible for the host to send commands for the robot to interoperate and perform actions such as move forward, rotate or stop also at real-time. This is achieved by the robot listening for another command while executing its current one, the only time it is not listening is when it is transmitting sensor data. It is not possible for the robot to transmit and receive data simultaneously because of the serial connection, but data is not lost if the host sends a command because it must internally buffer it until the robot gives up the line.

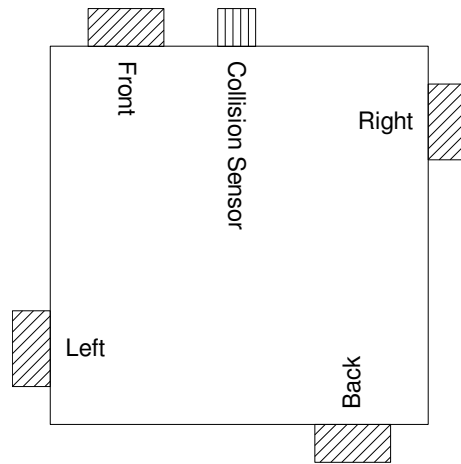
<i>Command</i>	<i>Char</i>
Move forwards	'w'
Move backwards	's'
Rotate anticlockwise	'a'
Rotate clockwise	'd'
Stop	'space'

This table shows the major list of commands the robot understands with a corresponding trigger. It is modelled using the WASD key control system which mimics the inverted-T configuration of the arrow keys popularised by video games. There is another command which is not listed as it is only used for configuring the motors, if the host transmits a carriage return the robot records all sent chars into a buffer until another return is sent. This string is parsed

and a more complex action is taken, for example is the user sends `[return]L20[return]` then the left motor speed will be set to 20. When a command is sent then the robot echoes the actions it has taken, if the character 'a' is sent then the robot should respond with `M -20 20` which indicates a motor command in which the left motor has been set to speed -20 and the right to +20 – the start of an anticlockwise rotation. This output is logged by the host and used to determine a change in speed or direction with respect to time. This can be determined as the robot sends 10 groups of all four sensor readings a second, at a constant time interval of 100ms between readings in conjunction with motor speed can determine the position of the robot relative to its origin. Sensor data is sent in the groups of all sensors using the following pattern: `I 30 45 55 20`, the order of sensors are front, back, left and right. This implies the front one is reading a distance of 30mm while the back is showing 45mm and so on.

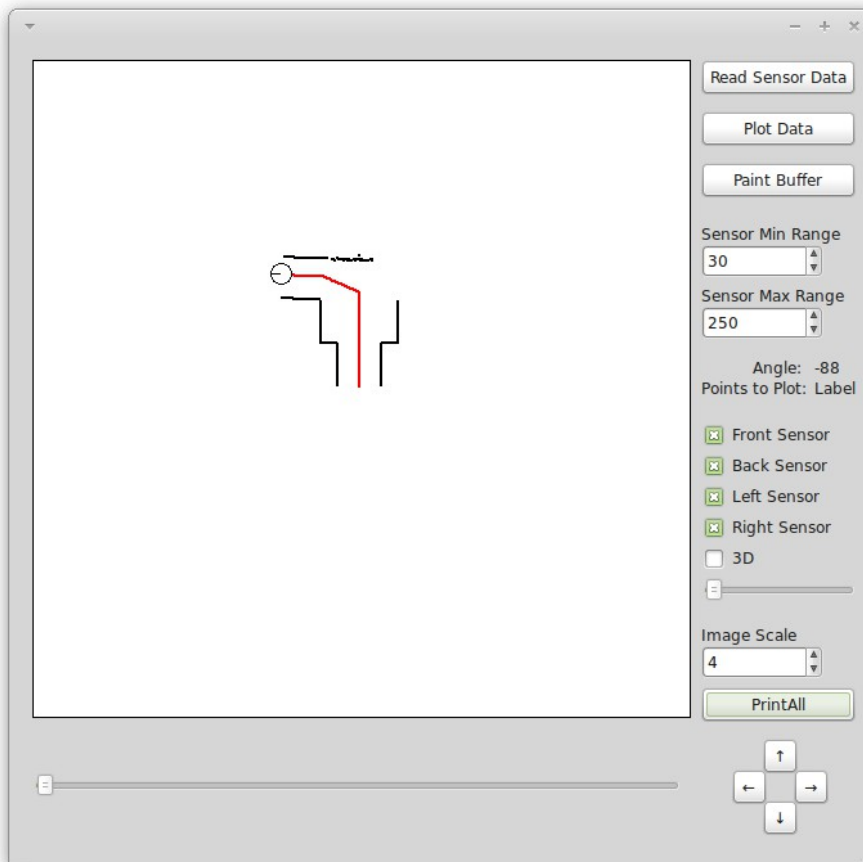


Original Layout



Modified Layout

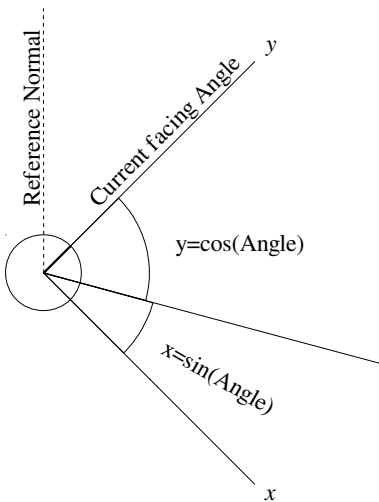
In order to get a 360° view the ordinal given sensor set-up was modified by rotating the front left sensor to face forwards while the back right was repositioned to look behind. It may have been better to have 8 sensors in total, 2 for each direction but this would have been a time consuming to implement because it would require a multiplexer as the mbed board does not have enough ADC ports plus it would also hamper the already short battery life of the robot. On this basis I believe the chosen implementation is a reasonable compromise which can be done very easily.



This is the program I developed to which uses the sensor data and motor commands to generate an image representation. I used the wxWidgets library for the GUI, this demonstration is using generated data which was mainly used for debugging my point plotting algorithm. The slider at the bottom allows the user to time-step, which allows to see how the image progresses through as the robot moves. The image is of a T-junction with a wide

opening, the lines appear but are not solid, instead they made of smaller points that join to make a wall.

4. Noteworthy Design and Implementation



The diagram to the left is a representation of the robot as used in the GUI display program, in this instance the robot is facing 45° from its internal reference normal – the dotted line. As the robot moves with respect to time its displacement is calculated based on the motor commands, its current position on the map is referred as CurrentX (C_x) and CurrentY (C_y). All the four infra-red sensors are read together and are plotted on an image relative to the robots current position and angle on that image. Using the formula below we can calculate the X and Y coordinates of where the sensor readings should be placed on

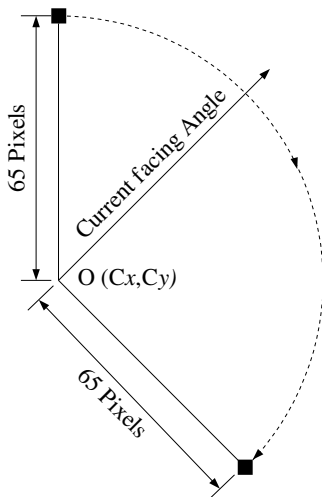
the map, this is derived from trigonometry of circles.

$$\text{Plot Sensor X Coordinate} = \text{CurrentX} + \text{Sensor Reading} \times \sin(\text{Robot Angle} + \text{Angle Adjustment})$$

$$\text{Plot Sensor Y Coordinate} = \text{CurrentY} - \text{Sensor Reading} \times \cos(\text{Robot Angle} + \text{Angle Adjustment})$$

$$\text{CurrentX} = \text{CurrentX} + \text{Speed} \times \sin(\text{Robot Angle})$$

$$\text{CurrentY} = \text{CurrentY} - \text{Speed} \times \sin(\text{Robot Angle})$$



A simple way to explain the algorithm it is with an example, first we know the robot's C_x, C_y coordinates on the image map. If the right sensor reports a value of 65, a we could plot this at $(C_x, C_y + 65)$ as shown by the diagram to the left. However the robot is facing 45° clockwise, this point needs to be rotated 135° , which is it's current angle of 45° and 90° because this reading is from the right of the robot or 90° clockwise from where it is currently facing. What is essentially happening it that the point is been rotated based on the robot's current angle and which sensor that value came from, the angle adjustment is based of the sensor. It is 0° for the front, 90° to

right, -90° for left and $+180^\circ$ for the back sensor.

This design was chosen because there was tangible alternative, unless the map is rotated instead of the robot. This will require heavy computation as every point will need to be moved per sensor reading interval, this will induce heavy rounding from multiple rotates unless a massive degree of accuracy is taken. The implemented algorithm is essentially using geometry based on relative positions, sensor readings which are relative to the robot which in turn is relative to the map. As the current angle of the robot is determined by time spent rotating which can lead to increasingly unreliable readings as this itself in an approximation is not the best design choice. This method was chosen due to time constraints but mouse based rotation would have been a better approach used in conjunction so that both data sets challenge each allowing more verifiable results.

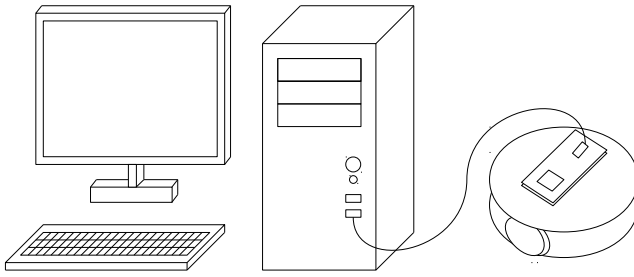
5. Testing

When designing an algorithm a good start is to do research on the problem you want to solve. An example involves line following, an algorithm already exists known as proportional-integral-derivative controller (PID controller) which can be adapted and applied to line following. This helps with testing as if one is developed from scratch then any faults with how the algorithm fundamentally works will be harder to debug rather than a well researched and established method.

Our main testing method was to first design a rough algorithm and then implement it, when finished the code is checked for if is robust and there are no logical operational mistakes. Now we testing it is the field to see if does what it is designed to, multiple runs are taken with small changes in operational parameters such as environment and starting position. How well the system performs is noted and can be useful it determining what parts worked correctly, not so well or not at all from an implementation point of view. One useful tool was the debug framework that is part of mbed drivers, it allows communication to and from the robot usually in the form of text to the terminal. This can help get an overview of what the system is doing at any given time and the actions taken to tackle them.

During debugging a flaw in the design of how the robot operates was discovered for my individual project. Initially this issue was thought to be with the point plotting algorithm, this assumption was based on that some data would display odd results and sometimes the program would crash. No faults could be found so the next step was to look at the data that the robot transmitted, which is where a discrepancy was found. The problem was that in mid transmission of a motor command infra-red sensor was also been sent, the two where pieces of data where becoming jumbled which was causing the problem. A look at the robot code showed that it is possible for the robot to be echoing a motor command and midway the read sensor interrupt would take control and send its own data, when finished it would return to transmitting the rest of the motor command. A solution is to disable the interrupt listener while sending motor commands, interrupts are still generated but are ignored. One problem is that this can result in the loss of a batch of sensor readings, but a motor command is much more important to the point plotting algorithm and a loss of one sensor reading is trivial as there are many others available to construct the image. A better solution would be to buffer the infra-red data and send it when the motor command has finished, this would require significant changes to the handling of information by using a more complex priority queue. My fix is quick and cheap without affecting the performance or reliability of the system which is a good trade off as limited lab hours were increasingly becoming a problem.

6. Technical Specification



Prices have been sourced from the Google Shopping service with VAT included, shipping has not been included because when set to be mass produced this will be based on the volume of production. The prices stated are for single units and should

reduce when bulk buy discounts have been negotiated with suppliers.

<i>Components</i>	<i>Cost</i>
mbed NXP LPC1768 micro-controller*	£37.44
Pololu 3mpi smart base and mbed expansion	£96.82
4x SHARP GP2Y0A21YK analogue infra-red sensor	£25.58
SHARP GP2Y0D805Z0F digital infra-red sensor	£8.77
Optical mouse	£4.00
TT Electronics OPB608 Reflective object sensor	£1.93
4x AAA rechargeable Batteries	£5.00
iMAX-B6AC Balance Charger and Discharger**	£32.25
Total	£213.81

* A desktop computer is needed for reprogramming the robot but is essential for my individual project, currently only the Linux operating system is supported but the code can be ported.

** Not necessary but removing or replacing batteries requires disassembly of the Pololu chassis.

The system has rebuilt functionality which utilises the five sensors on the base of the system in conjunction with the PID algorithm to follow lines underneath the robot. It can also remain parallel to walls while moving by using the side infra-red analogue sensors and evade moderately sized obstacles on its path. The robot has an 8×2 character LCD which can be used for debugging or user interfaces, pre-built demonstrational use is to show features such as displaying battery life and playing sounds though the Piezo buzzer, which can all be cycled using GPIO control buttons. There are Bluetooth, Xbee and Wixel sockets to add additional functionality. Other additional sensors include the optical mouse which can be used to determine displacement with respect to time and determine speed and direction of the robot. The device is fully reprogrammable which can be done easily with the mbed SDK or SMSIS libraries to suit any custom requirements. Existing algorithms and be extended as the source code is provided or you can write your own to re-purpose the system to your needs.

7. Individual Project Limitations and Improvements

In order for my individual project to work several assumptions need to be made, firstly that motor commands issued to the robot can always be performed and done correctly. The current implementation on the mbed side can be described as essentially a dumb terminal, it obeys commands and sends back what it 'sees' with no real processing. If the robot is issued a command to move forwards but is not able to do so because the path is obstructed then the host computer will continue to believe that the robot is moving forwards when it isn't. Should any simulated or real world action become inconsistent this will result in the calculated relative position of the robot becoming incorrect, meaning all subsequent results are inherently unreliable. This happened somewhat in the demonstration in which low batteries were the culprit, I agreed to demonstrate my individual project after Ollie as his wall following project required the sensors to be in their original position while I need to have two of them sensors rotated. In the end he did not present a demonstration and I was last in my team to do so, we overrun and another team was waiting so there was no time to do a preliminary dry-run test. The problem occurred when the robot was issued a command to rotate anti-clockwise, but was unable to due to insufficient power sent to the motors. This issue was minor because it occurred at the end of the run so only the last few data points were unreliable and did not compromise the rest of the data, overall the result image was a good effort and recognisable from the surroundings of the robot.

The rendering algorithm of the image is sound but it is limited when the provided data conflicts with what genuinely occurred creating an inconsistency. To solve this issue, the algorithm needs more data, this can be sourced from other sensors such as the mouse or the wheel encoders. By having these extra sources then better actions can be taken when contradictory data is provided, for example, if the wheels are turning but the mouse doesn't show any movement it can be assumed that the robot is not moving. The same principle applies to people, we need multiple sensors to perform a task. Try walking with your eyes closed, you can feel your relative position but it may be challenged when you open your eyes. There are many reasons why the mouse wasn't used, the main is due to time constraints, essential functionality was implemented first to allow some displayable results. Which later could be built upon to make more accurate and reliable by refining algorithms and utilising different types of sensors. These additions could be implemented in two distinct ways, firstly the extra sensor data could be processed on the mbed and it will only transmit its new location and sensor data. This will not require any change to the point plotting algorithm but require dealing with the complexities of the mbed hardware to process the information. I believe a better alternative is to have the robot transmit all sensor data as bandwidth is not an issue and the data can be easily processed using a high level language on the host computer much more trivially.

8. Application of Mobile Robot Systems

If the original design for my individual project was successful then it could be possible to remotely control the robot purely by using the environmental image that could be generated at real-time. On the Pololu there is a dedicated Bluetooth socket meaning it is possible to attach a compatible chip which will then allow data transmission between the host and robot wirelessly. This communication method has many advantages over a cable mainly because they are not practical over longer distances due to physical tug and possible tangling in complex junctions. This will allow remote control with usable feedback in the form of the rendered image if data is processed at real-time. This would allow lots of flexibility in movement and could be used for applications such as mapping unused and redundant tunnels or pipes to see how they span and determine if they can be reused or should be removed. It could also help determine leaks or breakages in these underground systems without digging up roads which is costly and will likely cause a public disturbance. The remote control aspect means that it has a wide level of possibilities but this may not always be useful. Scouting out tunnels does not require a human controller as the robot can be reprogrammed to search all possible routes or prune others to get an overall map automatically. When finished they could backtrack and return to their starting position altering users via the connected host. As wireless communication usually has a short range in might be a better idea to store the data on the mbed micro-controller, this is viable for medium sized treks due to memory limits. This could be expanded but there are also power usage concerns which need to be addressed as the batteries which we used tended to have a short lifespan.

Should the useful lifespan of the robot come to an end it will be better to repurpose the device rather than dispose it. This is due to the difficulty and high cost of safe and environmentally friendly disposal of electronic devices. There are several alternatives such as donating to charity, there are several charities such as the Computer Aid International who collect hardware, refurbish and distribute to universities, hospitals and other places of need in developing countries. They mainly receive general purpose computers such as desktop and laptop machines, while the robot is a very specialised device which requires a high level of knowledge to operate. In this situation the charity should be contacted prior to donating to determine if the robots are useful. Alternatively the robot could be disassembled and each component could be reused in another system which will save money and reduce hazardous waste. The mbed itself which is of most interest because it has can be useful in other embedded system applications as it is reprogrammable, plus developers will not need to learn a new architecture which will save money on training.