UNIVERSITY OF CANTERBURY

ROBOCUP PROGRESS REPORT 2

GROUP 10

Jack Hendrikz
Peter Nicholls
Ryan Taylor

16 October 2015

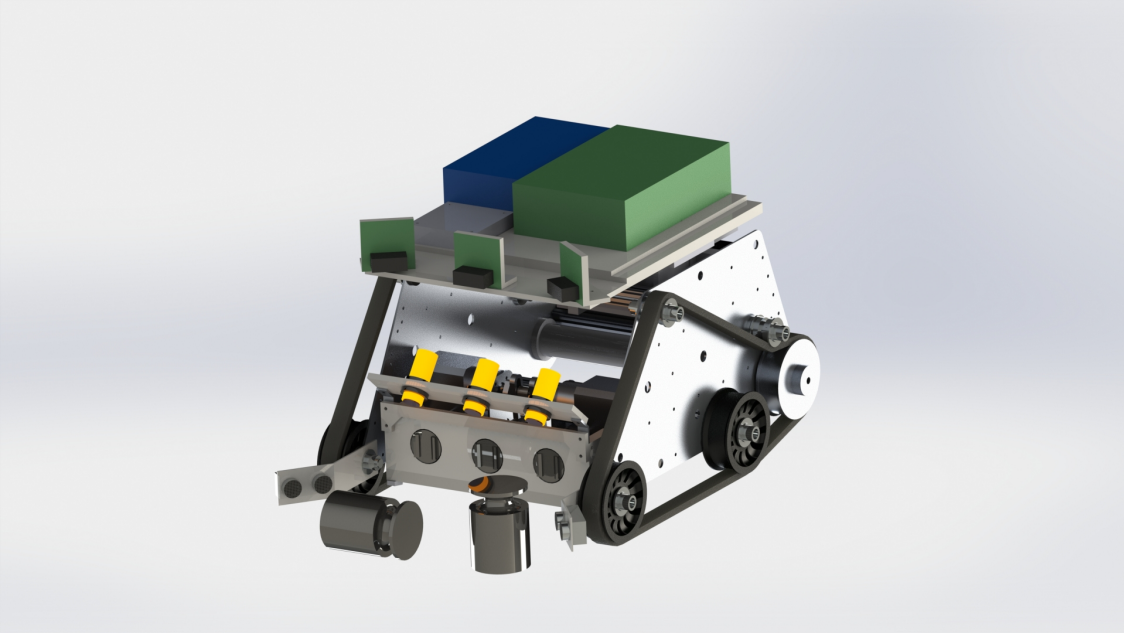


TABLE OF CONTENTS

0.0 EXECUTIVE SUMMARY

This report discusses the design of Group 10's robot that will be entered in the Robocup challenge in October this year. The robot has already performed well in the initial functionality assessment, achieving a perfect score.

Three modules are presented with a detailed overview in this report. The mechanical system module includes the robot chassis, locomotion method and package collection. Next the electrical system module includes an evaluation of sensors used, their position on the robot, and the circuits created for further functionality. Finally there is the software module which includes all the programming required to allow the robot to function autonomously.

An evaluation of each module is conducted to contrast the current functionality against the robot's requirements outlined in the initial report. The report also contains a future development section that discusses the changes that should be made to the robot before the final competition.

The robot has currently completed 17 of the 25 requirements and performed well in the functionality assessment. It can detect objects such as walls but has some issues with pole detection. It can identify, move to and collect packages around the arena but there is much to be improved. The code allows for wall avoidance, but pathfinding is yet to be implemented. Overall the robot is well on its way to performing as specified on the test date and will be significantly improved over the following months.

1.0 INTRODUCTION

In this year's Robocup Zombie apocalypse, teams have designed robots that search autonomously for food packages. The robot will compete in the arena against another robot for these packages. The battle will take place in an unknown environment which includes immovable obstacles, walls, and a series of rough sections of ground all unknown to the robot. The robot will need to return food packages to the team's headquarters. These food packages are represented by cylindrical weights made from steel weighing between 0.25 and 1 kilogram, and will be placed in both easy and hard to reach places.

This report takes our chosen design from Conceptual Design Report (CDR), and explains the technical details on how the robot is built and how it will complete the tasks. As a mechatronic system, the details are split into mechanical, electronic and software design, each of which are crucial for an effective, autonomous robot.

2.0 Mechanical Design

2.1 Chassis

The design will use the chassis supplied as it will take too long to design and construct a better one. The one supplied gives good mounting holes for the wheels in various positions. As discussed in the previous report, parts have been modified so the chassis is oriented up-side down.

2.2 Locomotion

Locomotion is one of many essential tasks for the robot to succeed. When designing the wheels several aspects must be taken into account: maintaining the tracks on the wheels, getting enough traction on the track from the motor, having enough ground clearance and where the tracks will contact the ground. The drive motors will be two 12V low noise 28PA51G DC motor, these will remain in their initial position on the chassis wall.

Opting for 3D printed wheels allowed the ability to have control over all these parameters. This also meant they could be designed in Solidworks and modelled appropriately to meet the parameters. Having the correct lip on the edge of the wheel is essential or else the track may come loose, this was modelled and a suitable lip designed. To get traction from the drive wheel to the track relied on getting the greatest amount of surface area with sufficient tension. A high amount to traction was achieved by having nearly 180 degrees of contact with the wheel. Since the maximum ground obstacles are 25mm in high, the radiuses of the wheels were made to be 30mm, also allowing for the mounting holes. By keeping the track-ground contact area small will improve manoeuvrability, but would result in decreased stability, and this was taken into account when choosing how much length of track would touch the ground. The kit was designed with a length of about 200mm, or 250mm when inverted as was our chassis. The length chosen was 200mm in order to counteract this disadvantage of the inverted chassis. Solidworks calculated the centre of mass of the robot to be in the centre of the track length which is ideal, seen in Figure 1.

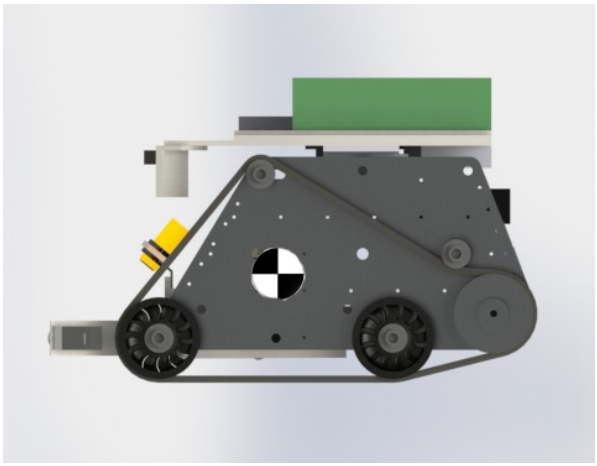


Figure 1, Shows the side of the robot with the wheel design and the centre of gravity.

2.3 PACKAGE COLLECTION

The mechanical element of the build requires a system to pick up and store three packages. The main principle for our design will be using magnetic force to pick and store packages. Three strong magnets in a row along the front of the robot will be ready if any packages are detected to pick them up. Once three packages are picked up then the robot will drive back to base and drop them off by retracting the three magnets though the perspex barrier. The force of the three magnets requires a greater force to retract them. This greater force has to overcome the 450N (worst case) from the attraction between the magnets and packages. To get this force two geared stepper motors, FIT0349, will be used in junction with a reciprocating mechanism. This mechanism changes rotary motion to linear motion and also gives a step up force simultaneously. Figure 2 shows the overall mechanism and Figure 3 shows more detail of the moving parts.

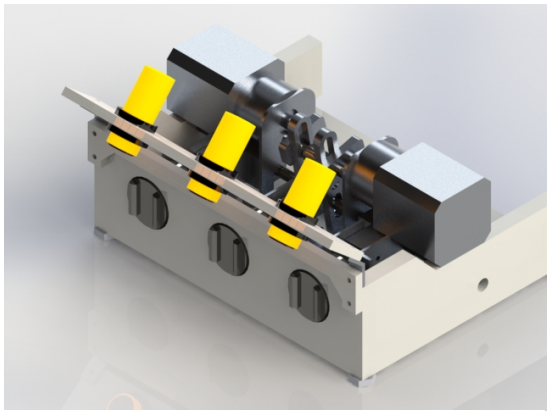


Figure 3, Shows the overall design for the pickup mechanism.

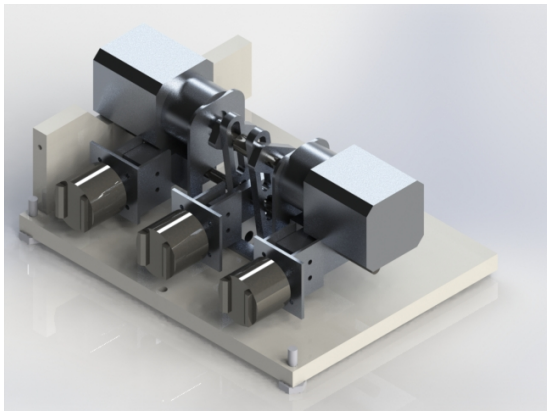


Figure 2, Shows a stripped away picture of the pickup mechanism.

Linear motion is provided by a simple slider crank idea, shown in Figure 4. Two geared steppers are mounted facing each other on a bracket holding them in line for direct drive. The two geared steppers are needed to provide enough torque. These motors have an eccentric shaft between them, this gives the torque. The eccentric shaft can slide up and down a link, the other end of the link is pinned on the bottom of the mechanism. Down the link a rod, parallel the eccentric shaft, is restricted to horizontal movement. A connection is made between the three magnets and the rod. The magnets are mounted in the front held in by a perspex barrier. When the link sways back and forth it push and pulls the rod thus moving the magnets in and out. All the custom parts have been machined.

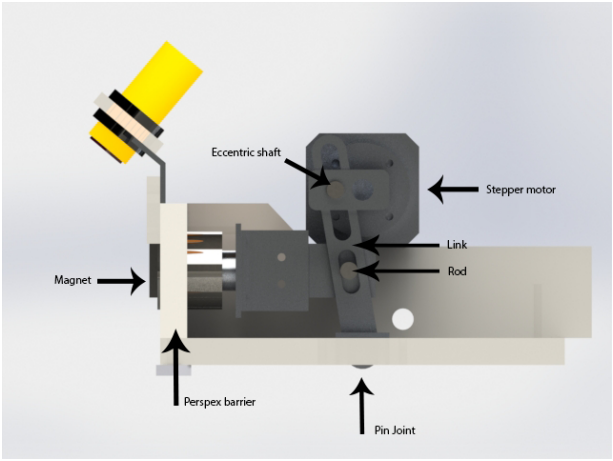


Figure 4, Shows a side view of the pickup mechanism with labels showing different parts.

Figure 5 is an exploded view of all the parts for the pickup mechanism.

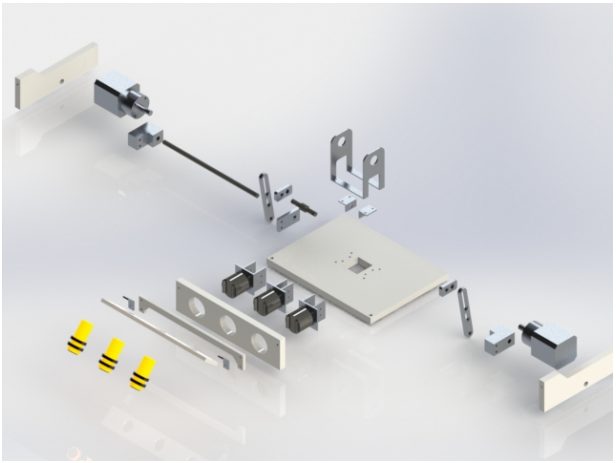


Figure 5, Shows an exploded view of the pickup mechanism parts.

Ensuring the packages fall off when the mechanism is operated is essential for the robot's correct operation. The mechanism uses two stepper motors with a torque of 1.78 Nm each. To make sure this happens, Matlab was used to find various lengths that would successfully retract the magnets, seen in Figure 6. The blue line shows the force needed to release the magnets and the red line is the calculated force applied to the magnets. The packages will be dropped off between 80 to 160 degrees. This is a small band of the force curve that the figure shows will be enough force to drop the magnets off. The reason the second peak of the curve is lower than the first is because when the eccentric shaft is on the lower side of the turn, less torque is transferred. See Appendix 1 for the code and hand calculations. This calculation doesn't take into account mechanical losses or the stall torque of the motors so the force produced will be less than shown. This shouldn't be a problem given that the maximum force is nearly double that required.

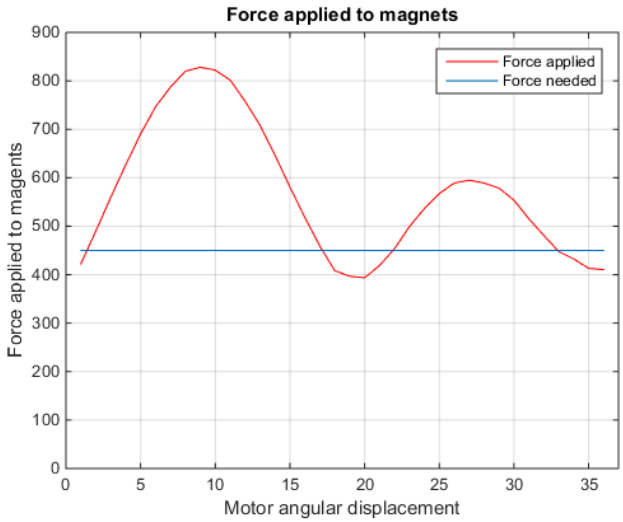


Figure 6, Shows the force produced by the pickup mechanism applied to the magnets.

The design focusses on overcoming big problems that could put a halt to package collection, which consisted of different orientation packages (fallen over), packages tight corners and fast and simple package collection. If the other team knocks the packages over then the packages can still be recovered due the high attraction force of the magnets, and if the opponent can't recover fallen over packages then it will be an easy win. Secondly, having the two of the magnets off centre of the robot will allow for corner collection easier. Finally, the actual collection of packages is fast, simple and reliable, as the robot merely needs to run into the package for it to be permanently on board.

Because it can pick up packages this easily, there needs to be some method of not collecting packages in the opponent's base. To ensure this is the case, the whole pickup mechanism tilts up and down, and also prevents the packages from getting jammed against a ground obstacle. The majority of the time the robot will drive with the mechanism raised, but will only lower to collect a package. Once the robot has driven into the package and confirmation the package is on board will raise the mechanism. Tilting will take about five seconds to complete, so the collection will be a total of just over ten seconds.

Drawings for the mechanical systems can be found in Appendix 7.

3.0 Electrical Design

3.1 Circuit Boards

In order to drive the sensors and motors with the correct voltage and to receive the desired signals for the sensors, the provided circuit boards must be implemented into the robot design. All the provided power and interface circuits were used except for the servo interface board. The Arduino and battery were placed on the top of the robot, while the rest were mounted on the underside of a Perspex sheet below the Arduino. Figure 7 demonstrates the connections between each of the boards and the components. It can be noted that the 'Power Safety Circuit' shown is also a distribution board.

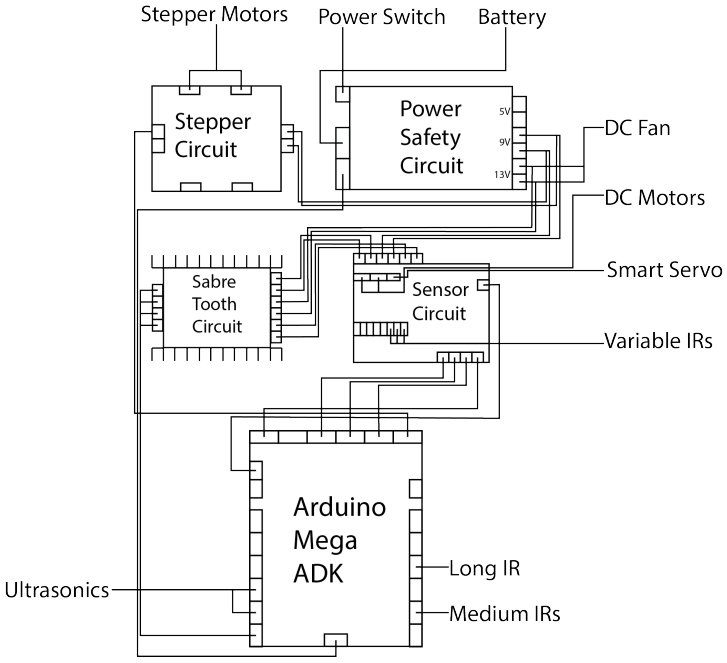


Figure 7, Shows the overall electrical circuit board connections.

Having the circuits on one piece of Perspex allowed the boards to be arranged in a compact design that could be easily accessible. The boards were arranged in such a way that they could be connected easily to one another as well as the Arduino. The cabling could be neatly run without the risk of interference with the pickup mechanism or the risk of potential damage by being exposed externally on the robot. Most importantly, every individual board could be accessed due to a through a quick release mechanism allowing the mount to swivel. Easy access of the boards is seen in Figure 8.

Figure 8, Shows the quick release mechanism rotating.

As part of the concept specifications, it was required that each group member design a circuit board to aid in the functionality of the robot. These three circuit board designs are presented in the following sections, and the schematics can be located in Appendix 2.

Circuit 1: Small speaker amplifier

This circuit is used to amplify audio signals sent from the Arduino as they are too weak to drive the speaker directly with significant volume. This circuit will consist of a LM386 IC chip, along with the necessary resistors and capacitors to drive a 1 watt speaker at 8 Ohms. The circuit will have a variable volume control via an adjustable potentiometer.

Circuit 2: Switch bank

There are several values and parts of code which can be enabled and disabled, but changing them requires the code to be recompiled and uploaded. To prevent this, Circuit 2 will have a bank of eight switches which can change Boolean values in the code during runtime. Such variables include whether or not to play sounds, send serial or to be controlled manually.

Circuit 3: IR LED Mount Board

This board will mount 6 IR LEDs which surround a separate IR camera. A switch will allow the LEDs to turn on or off manually. These LEDs will be used to light up the arena and due to the nature of the IR camera returning the location of the four greatest IR sources, can assist in detecting packages.

3.2 Sensors

The sensors are another vital component to the correct operation of our robot. The sensors are required to be placed in such a way that will allow them to detect their environment accurately and relay this information to the Arduino. The following describe tasks that must be fulfilled and the sensors chosen to accomplish them.

Obstacle Avoidance

In order to navigate the arena, a combination of two medium and one long range IR sensors have been implemented for obstacle collision avoidance. IR sensors are much better suited to detecting obstacles as they have a narrower beam width and do not suffer from beam deflection.

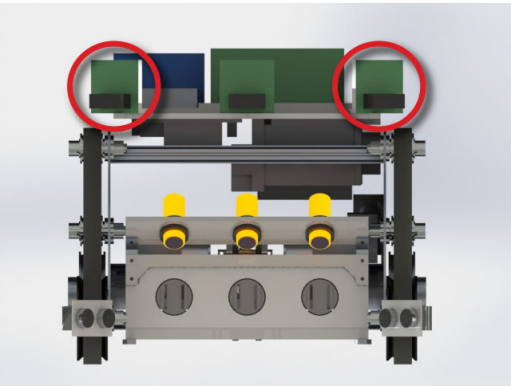


Figure 9, Shows the location of the medium range IR sensors.

The medium IR sensors were used for obstacle collision avoidance as they could operate efficiently with the range of 100-800mm. These sensors were mounted on the front of the robot and set at an angle of 35 degrees from forwards (facing inwards). See figure 9 for sensor position. This would allow the medium IR sensors to work in tandem with the ultrasonic sensors, allowing the robot to both detect obstacles and help distinguish packages.

For to assist in avoiding poles, the long IR sensor was used for the functionality assessment. It was placed in line with the medium IR sensors at the front of the robot. The IR acts to fill in areas that are not detected by the medium IR sensors. Figure 10 indicates the location of the long range IR.

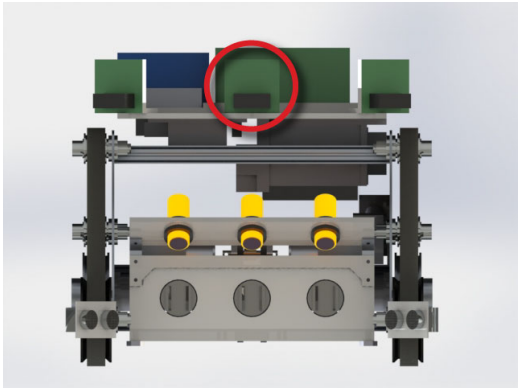


Figure 10, Shows the location of the long range IR sensor.

The IRs were required to be calibrated to allow the robot to know approximately how far obstacles are situated in relation to itself. This was done by measuring the signals from the sensor at varying distances and saving their value to the robot's configuration file.

Package Detection

It is obvious that the robot must have the ability to detect packages in the arena that it may bring them on in a permanent fashion. To achieve this, the ultrasonic sensors provided were utilised. Ultrasonic sensors are much better at detecting packages due to their broad beam width and ability to detect cylindrical objects. Two ultrasonic sensors were mounted at the bottom of the robot in front of the tracks. By comparing the signal difference between the top mounted IRs and the bottom mounted ultrasonic, the robot is able to distinguish between obstacles and packages. The ultrasonic sensors were set at an angle of 40 degrees to allow broad sensor coverage for the front of the robot. The mounting for the sensors are shown in Figure 11. Unlike the IR sensors, the ultrasonic were not required to be calibrated as they operate on a function that takes into account the speed of sound which determines the distance based on the delay. Figure 12 shows all these sensors' sight and their blind spots.

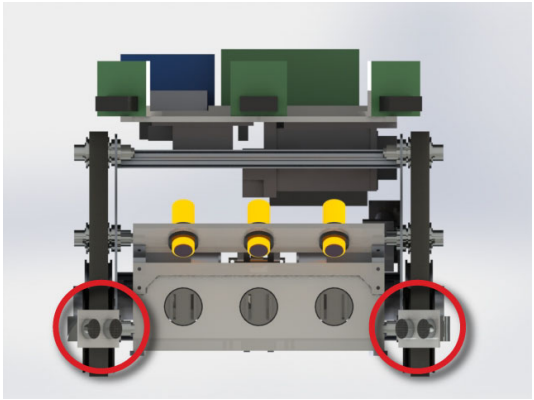


Figure 11, Shows the location of the ultrasonic sensors.

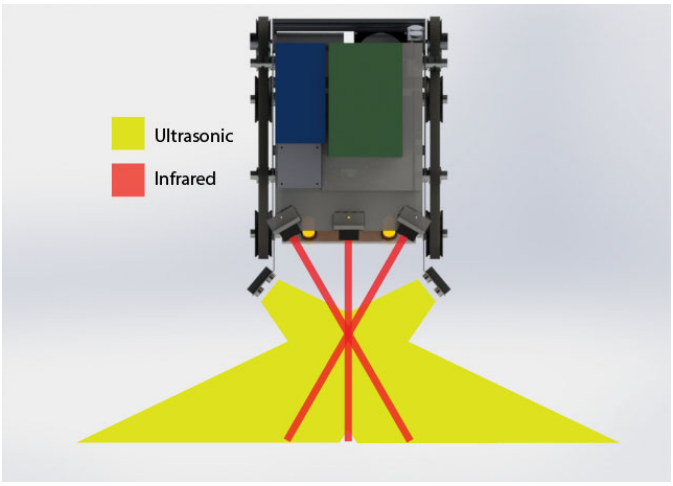


Figure 12, Shows the birds eye view of the sensor placement.

The reason the ultrasonic and IR sensors do not need to be straight above each other is due to the angle of the beam and the angle of the sensors. Since the ultrasonic has a beam width of 15 degrees, the IR sensors have been set to be pointing 15 degrees further forward (35 degrees rather than 50). This means that the IR beam will be travelling parallel with the 'front edge' of the ultrasonics' edge. The two areas this could cause problems is directly in front of the robot (which is solved by having a forward-facing IR), and right next to the ultrasonic sensors, in which case is close enough that the robot should collect it anyway.

Because the ultrasonic and IR sensors can be noisy, the supplied IR camera will be used to assist. This has the ability to detect four IR sources and give you the x and y coordinates in the perspective of the camera, and possibly from the perspective of the robot in the arena. Since it will be used in conjunction with IR LEDs (see Circuit 3), the packages would reflect the IR waves and appear as the brightest spot on the camera as it would reflect more than the surrounding darker obstacles. It does have some drawbacks including the risk of interference from the opposition robot, which is why it is only a secondary method.

Package Confirmation

Variable IR sensors will be used as on board package detectors as they had a simple true or false signal return. When packages are picked up by the magnets, the variable IR will trigger, allowing the robot to know the number of packages currently on board. These sensors are situated directly above the permanent magnets on the pickup module. This can be clearly seen in Figure 13.

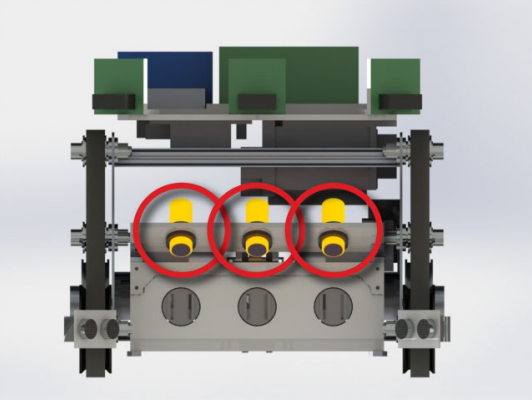


Figure 13, Shows the location of the variable IR sensors.

Distinguishing Bases

For the robot to detect the arena bases, the colour sensor must be implemented in the design. This sensor should be placed as near to the front of the robot as possible to allow for ease of detection. The exact location of this sensor is yet to be determined, but will likely be inside the Perspex shell of the pickup mechanism so that it is shielded from the ground.

Navigation

Besides using the IR sensors for avoiding obstacles, the robot must also know where it is on the map for our design. The other two sensors that will be used for this purpose are the digital encoder (attached to the drive wheels) and the IMU. The digital encoder will simply be used to predict how far it thinks it has moved, and the rest of the sensors will confirm or update the position using Simultaneous Localization and Mapping (SLAM).

4.0 SOFTWARE DESIGN

4.1 PROGRAM STRUCTURE

As an autonomous robot, the code is complex and needs to be well structured. Before writing anything, the different modules were created in order to keep everything tidy and fulfil R5.4. The program was implemented primarily from a bottom-up approach, creating the low-level sensor and actuator functions, followed by a top-down approach. This meant the robot did not have to be physically built initially, besides plugging in the components on a temporary basis to test code.

Each module is written to be as independent from others as is reasonable, with a level of abstraction for other modules to reference. Their interfaces with other modules are demonstrated in Figure 14 while the support functions are used by most other modules. While much code for a map and SD card referencing have been written, they were not used in the functional assessment.

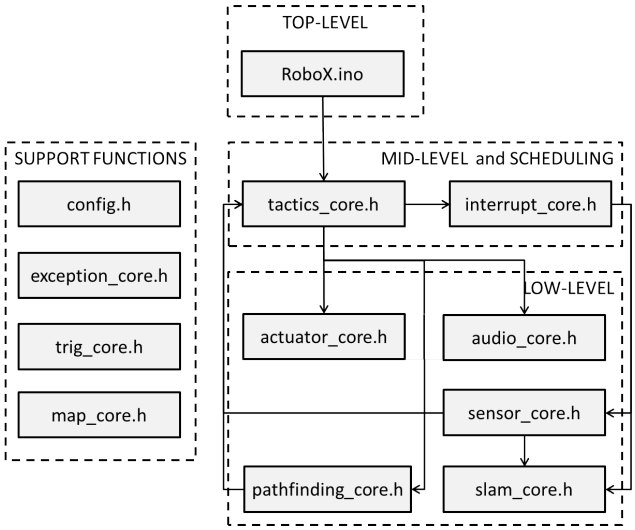


Figure 14, the files and how they interact

4.2 SUPPORT FUNCTIONS

config.h

To make adjustments to the robot simpler and all in one place, a configuration file was written. It contains no functions or classes, but only macros. This includes the components pin-out, sensor calibration and various program parameters. This file only has values that are subject to change, other macros are also defined in the relevant modules to keep the program clean. Several definitions are there to enable or disable various aspects of the program in order to save processing time. These include whether or not to print serial, play sounds or use the map.

exception_core.h

The exception core is a design decision intended to make it easier to code ways around potentially crippling errors. It contains a class which is essentially an error flag with a wrapper. If there is some problem (such as the motors do not initialise), then it raises the flag and prints the error (or plays a sound). Another section of code detects the error and takes a different course of action. This is intended only for functionality issues rather than normal program operation such as detecting a base.

trig_core.h

Since the intention of the program is to eventually use mapping and localization, trigonometric functions will be an essential set of tools. This module includes two classes and several macros for radian and degree conversions. The two classes are a two-dimensional Cartesian vector and a polar vector, and use overloaded functions to make it as easy to use as possible.

Since the vectors would added multiple times throughout the code, the efficiency of two different methods were compared. When adding two Cartesian vectors, it is simply a matter of adding components, but when adding two polar vectors, the sum can either be calculated directly or added as polar vectors. The comparison can be found in Appendix 4, which showed that converting to Cartesian coordinates, adding and converting back is more efficient than using trigonometric functions. This led to the decision to use Cartesian form as often as possible to make it even more efficient.

map_core.h

Since both the SLAM and pathfinding will constantly be referring to a map, another module was created for it. Initially, a single file was going to be created on the SD card containing the map information, but there were going to be issues with reading, writing and particularly changing values in the map. Instead, each node (point) on the map is located in a different file which, although is slower on average, is more consistent in speed, and is much more versatile. The robot could theoretically map out any size of arena if the SD card is large enough.

4.3 Top-Level Design

Since the robot is autonomous and will have no human intervention (R1.1), the code must be as robust as possible, and have backup systems in case of failure. To accommodate this, there is a loop even higher than the main routine. There are a total of four 'main loops' which the program can fall into depending on the state of the robot. Each of these is found in tactics_core.h.

Idle Loop

When the robot starts, each module is initialised and the program falls into the 'idle' loop. This is simply so that the robot can wait for user input that the round is starting. The robot will not fall back into this loop unless there is human intervention (the round has finished) or everything else has failed.

Primary Loop

When the round has started, the program begins the Primary Tactic. Under normal circumstances, the robot will perform SLAM, pathfinding and high level decision-making. If there is any crippling error such as losing the SD card, then it will drop to the Secondary Tactic.

Secondary Loop

This routine is meant to pick up packages with minimum complexity. This means there is no mapping or planning, but only on-the-spot decision-making. Since there is still a colour sensor, the robot is still capable of returning packages to base and avoiding the opponent's base, the only disadvantage over the Primary Tactic is that it will be less efficient. It should be noted that this is the code that was developed for the functional assessment so that there is a fall-back for the Primary Tactic.

Manual Loop

There is a fourth loop which has been implemented, and gives full control to the user. The user can send commands through the serial terminal to make the robot manoeuvre and move other actuators,

and the terminal prints the sensor readings. This mode has been used extensively for testing, though will be less useful now that the development is focussed on the other tactics.

4.4 Mid-Level Design

Each of the main loops have the same basic structure - when the function (tactic) is called, it initialises relevant variables and sets the correct interrupt. This interrupt will read the sensors as well as performing other foreground tasks at regular intervals. After this, it enters the while loop containing background tasks. The idle loop and manual loop have no further design, as they simply wait for user input, so only the two autonomous loops will be discussed in this section.

Primary Tactic

Because the Secondary Tactic was developed for the functional assessment, there are few completed parts of the program. The flow of the program has been written and is presented in Figures 15 and 16.

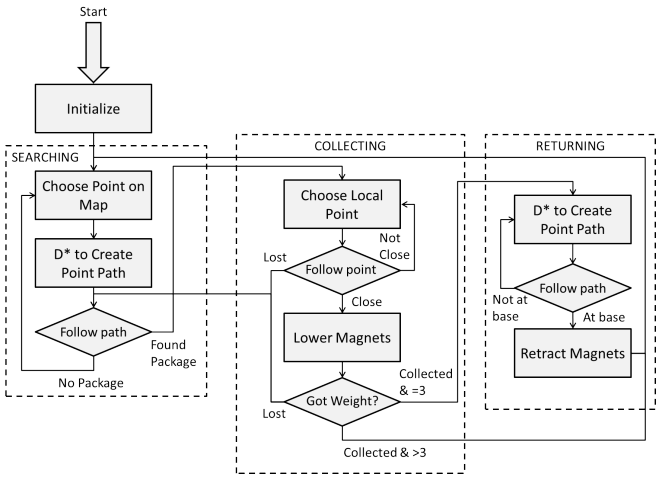
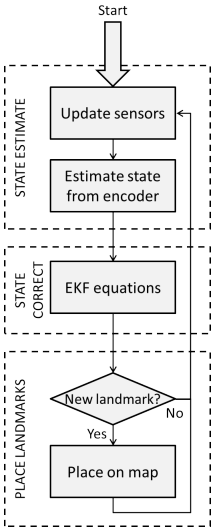


Figure 15 >, Primary Tactic foreground tasks

< Figure 16, Primary Tactic background tasks



Secondary Tactic

The overall structure of this tactic is very similar to the primary one since it still intends on completing the same task. The main differences are that the navigation is basically random (whilst avoiding walls), and the robot will drop off packages whenever it can. Unlike in the previous tactic, no processing is done in the foreground, but only updating of the sensors.

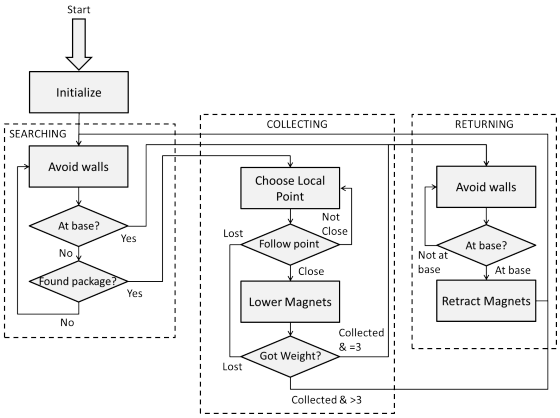


Figure 17, Secondary Tactic background tasks

4.5 LOW-LEVEL DESIGN

actuators_core.h

Each of the different actuators (DC motors, servos, stepper motors) were implemented as classes. Since they each have different functions, they have similar, but not the same public functions. The stepper motors are given a relative position, servos are given an absolute position, and the DC motors are given a scaled speed and rotation.

It seemed that each time new electronics were installed the motors would have a different 'zero' velocity in code. Because in many cases this resulted in the robot moving backwards faster than forwards, a self-calibration routine was written. When the robot is initialized, it moves forwards a minute amount, moves backwards and centres the position.

audio_core.h

Much effort was put into reading .wav files from the SD card, as well as playing audio in the foreground so that the robot could simultaneously play music and pick up packages. Unfortunately this could not be achieved. Instead, .rtttl files (ringtones) can be played, but only in the background, and not yet from the SD card.

sensor_core.h

Although there are many different types of sensors, they were all implemented as classes with the same wrapper. This means that elsewhere in the code, the function calls are the same. Each of the range sensors (IR and ultrasonic) have five public functions:

- void initialize(parameters)
- void update(void)
- CartVec cart_read(void)
- PolarVec polar_read(void)
- bool is_valid(void)

In the foreground, each of the sensors is updated. This is a straight reading from the sensor with no processing. When this is done, a flag is set to say the value has not yet been processed. When the sensor needs to be read, it processes the value into another register, and decides whether or not the reading is valid based on the sensor type. If called again, the function will not process it a second time, but just returns the value. The reason there are two different types of reading is because they are useful in different scenarios. polar_read() returns a distance (mm) and an angle, whereas cart_read() returns the detected object position relative to the centre of the robot (in Cartesian coordinates).

As for the other sensors, they only have three public functions:

- void initialize(parameters)
- void update(void)
- type read(void)

These are the same as for the range sensors, only they return various different values depending on their function.

pathfinding_core.h

The pathfinding method that will be implemented is D* Lite (or Dynamic A* Lite). This is a method of finding the fastest place between two nodes using heuristic values to optimise the speed (A*). Because the map is constantly being updated, there may be obstacles that were not taken into account. Instead of recalculating the route again, it recalculates it only based on what has changed (D*). This can be found in Appendix 6.

slam_core.h

While none of the localisation has yet been implemented, some tests have been performed to find out some ways to optimise. There are two reasons for trying non-conventional methods for SLAM. Firstly, current methods are commonly done with a 2D rangefinding laser which returns a 2-dimensional array of distances, and that is unavailable within the budget – the comparatively crude and more discrete IR and ultrasonic sensors are all we have. Secondly, the landmarks that will be created will not just be used for localisation but also pathfinding, which is best done in a 'grid' method rather than list of obstacles.

5.0 DESIGN EVALUATION

5.1 GENERAL

R1.1 The robot will be fully autonomous ✔

R1.2 The robot will be controlled by the Arduino Mega ADK supplied ✔

R1.3 The robot shall be able to move, identify and collect packages ✔

R1.4 The robot shall operate until all 11 packages are claimed or the time limit is reached ✘

Currently, the robot can act fully autonomous with the provided Arduino Mega ADK. It can move around the arena and collect packages consistently. It is yet to be tested with collection of more than three packages as this requires returning to base to drop off the collected packages.

5.2 IDENTIFICATION

R2.1 The robot shall be able to identify food packages ✔

To test this functionality, an experiment was setup to observe the detection of packages within the arena. The packages were placed at 0, 30, and 60 degree angles on both sides of the centre line of the robot at a distance of 600 mm. The time taken to detect and retrieve the packages was measured, and it took an average time of 3.93 seconds when it was detected. It could not detect the packages at 60 degrees, but it found 90% of the other packages successfully. Results can be found in Appendix 3.

Based on the findings from this experiment, we can see that packages are easily detected and collected when they are at relatively small angles from the robot's centre line. Beyond this the ultrasonic sensors are limited by their beam width and set angle. The robot will eventually detect and collect the packages in the arena at any position from the centreline, but it requires the robot to approach from a shallower angle. It should also be noted that the robot produced false detections during the experiment when driving close to the arena walls. It is suspected that the skirting around the arena gives false positives but this needs to be investigated further.

R2.2 The robot shall be able to identify obstacles it cannot move over ✔

This was implemented using the medium and long range IR sensors as discussed in the previous report. As a result, the robot performed adequately during the functionality assessment. Its ability to detect poles still lacks accuracy and consistency and should be significantly improved before the final assessment.

R2.3 The robot shall be able to distinguish home HQ and the opposition HQ ✘

The colour sensor example code functioned at an acceptable standard, but would not operate in conjunction with the rest of the program. This has not yet been solved, so the robot cannot distinguish between bases. It should be noted that the robot can distinguish the bases as a side-effect of the digital IR sensors – when on a base, all three sensors 'detect a package' due to the change in specularity.

R2.4 The robot should rely on a range of navigational sensors ✔

As discussed previously, the robot uses four different types of sensors based on their varying specifications. The range of sensors used is expected to increase by about two times.

5.3 MOVEMENT

R3.1 The robot shall be able to move over obstacles at least 25mm in height ✔
The 3D printed wheels discussed in Section 2 give the robot a ground clearance of over 25mm. They were designed this way and measured using Solidworks.

R3.2 The robot shall be able to fit through gaps of at least 500mm in width ✔
When placing the robot in between two walls, it successfully drives though the gap.

R3.3 The robot shall be able to manoeuvre around obstacles it cannot move over ✘
The robot can manoeuvre around and follow walls. Unfortunately due to the false detection of packages, it occasionally turns towards and collides with the walls. The robot, while it can detect cylindrical object, currently does not attempt to avoid them.

R3.4 The robot shall not leave the designated arena during the competition ✔
It is impossible for the robot to climb the walls.

R3.5 The robot should not get stuck in any algorithmic loops for longer than 1 minute ✘
No watchdog timer or other method has yet been implemented to avoid this. When mapping and D* are functional, it may not even be necessary to use a timer.

5.4 COLLECTION

R4.1 The robot shall be able to pick up a package so that it is under the robot's control ✔
At this point the design can collect packages fast and effectively, as confirmed in the functionality assessment, and discussed under R2.1.

R4.2 The robot shall have a way of carrying, at most, three packages without hindrance ✘
Three packages are able to be picked without trouble. The tilt operation has not yet been implemented, so it is unable to prevent the packages from interfering with level obstacles.

R4.3 The robot shall not collect any packages within the opposition's HQ ✘
The tilt operation has not been implemented and thus will collect packages in the opposition's HQ.

R4.4 The robot should be able to release any packages it has on-board to HQ (Erogenous Zone) ✘
Much work was put into a release mechanism, as discussed in detail previously, but is unable to release more than one package. Minor developments and fine tuning is required to get enough force on the magnets.

R4.5 The robot should be able to pick up packages in any orientation and any part of the map ✘
The magnets attach the packages in any orientation, but is unable to pick up packages against a wall.

5.5 CONSTRUCTION

R5.1 The cost of additional items shall not exceed $50 (except for R5.2) ✔
The additional items as priced in Appendix 5 sum to $36.20.

R5.2 Each member shall design their own PCB for use on the robot, not exceeding $5 ✔
While the circuits haven't yet been created, they have been designed, and are within budget.

R5.3 The robot should be built with less than 200g of 3D printer plastic. ✔
The four 3D printed wheels, each printed in two halves weighed a total of 152 grams.

R5.4 The robot shall be easy to maintain and disassemble ✔
By mounting all the circuit boards on a single sheet of perspex, it allowed for ease of total removal from the robot by the removal of the three mounting screws. For simple rewiring, the quick release with its swivel design could allow access to all the circuit boards in 12 seconds.

5.6 SAFETY

R6.1 The robot shall not cause any deliberate damage to anything or anyone ✔

R6.2 The robot shall have an accessible 'off' switch ✔

R6.3 The robot shall use the battery safety circuit provided ✔
The on/off switch and battery safety circuit were both used, and the switch was mounted at the back of the robot so it could easily be accessed. It will be necessary in the final assessment to relocate the switch as it could be disabled by the opposing robot.

6.0 FUTURE DEVELOPMENT

6.1 MECHANICAL DEVELOPMENT

Locomotion

The tracks stay on the majority of the time but still come off occasionally, so this will need further development. An extra bearing will be added to support and help guide the track on the bottom of the robot. Consideration are also being taken for adding a spring to the bearing to account for when that section of the track goes over a bump.

Pickup Mechanism

The pickup mechanism needs fine tuning to make sure all the force is transferred to the magnets. This will be done by reinforcing the structure of the mounts and by connecting all the magnets together. For a greater force on the magnets two rubber bands may be added. This will give more force on the back the stroke of the mechanism where it needs the greatest force. When extending the magnets back out the mechanism will put energy into the rubber bands.

Having the pickup mechanism rotate up and down design also has to be finalised and implemented. At this stage two servos will be mounted above to raise and lower it.

A limit switch is needed to tell when the stepper motors have rotated around past the point of knocking of the packages. This will ensure the packages have indeed disconnected from the robot.

6.2 ELECTRICAL DEVELOPMENT

Currently based on the sensor and circuit design, the robot functions on a basic level. It has the ability to navigate most obstacles and detect packages the majority of the time. It is now necessary to improve the reliability of package and obstacle detection. There are a number of potential design paths that we are currently considering adding or modifying for environment detection and they are discussed in brief below.

Sonar

If a central mounted Sonar sensor was used instead of the long range IR that is currently utilised on the robot would be better at detecting poles. The advantage is that it produces a large beam width (similar to the ultrasonic sensors), and 'fill in' the whole blind spot in front of the robot. They would, however, be much less useful for mapping as there would be a large range of points that a value could represent. They are better for determining if something is detected rather than where it is.

Swivel mounted IR

Because the IR sensors show a limited selection of the environment, it would be possible for one or two IRs to be mounted on a servo. These motors could be driven back and forth to produce a sweeping motion for the IR sensors. An array of values could be produced from this, but could potentially be more processor intensive. The other disadvantage is that the IR would only see each point once every one or two seconds.

Side mounted IR

This would give a permanent view of a limited spot, almost the opposite of the swivel mounted IR. This would be good to help prevent collision from turning into walls from the side, as it would never be 'looking away', but only works because the majority of obstacles are large.

6.3 SOFTWARE DEVELOPMENT

Currently the robot is just able to fulfil the requirements of the functionality assessment, but there are still five specifications which need to be fulfilled by the software - over 50% of the remaining requirements.

Final Sensors

Several of the requirements would be resolved by getting the remaining sensors functioning. The only sensors that do not currently work are the I^2C sensors. Each of the example codes work, but none function when implemented with the rest of the code. Some test has shown it is not conflicting libraries, but it may be due to the power electronics involved.

The IMU will have additional issues as well, since the pickup mechanism's permanent magnets will interfere significantly with the magnetometer.

Tactics

Neither the Primary or Secondary tactics are currently fully implemented, each of which will take a significant amount of time. The secondary tactic has been written just well enough for the functionality assessment, but still needs much work before it can fulfil the specifications. Even though it is secondary, it will still be written and tested so that it works well so there is a fall-back if the primary does not work. This was intended to be fully complete by this time, so it is very possible there will not be sufficient time to accomplish the full SLAM and pathfinding. This is not a huge problem because even with just the secondary tactic the robot will achieve each specification.

CONTRIBUTION STATEMENTS

JACK

- Design and build of electrical module
- Circuit 1 design
- Report electrical design section
- Proof reading
- Report formatting

PETER

- Software design and implementation
- Circuit 2 design
- Part sourcing
- Report software design section
- Report formatting

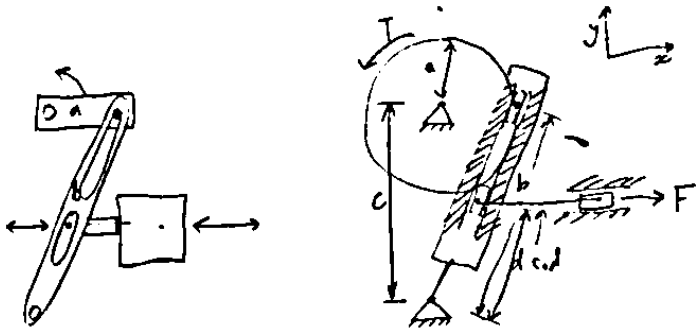
RYAN

- CAD modelling/drawing
- Pickup mechanism design, calculations, Matlab coding and build/machine (85 hours)
- Material/Parts sourcing
- Circuit 3 design
- Report mechanical design section
- Report formatting

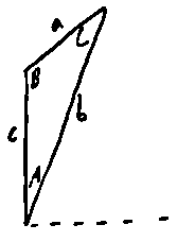
APPENDIX 1 – MATLAB CODE AND DERIVATION

```
clear, clc
close all

force_required = 150; %N
motor_torque = 1.76*2; %Nm

radius_gear = 0.010; %needs to be more then half the distance of our tavel
radius_rod = 0.030;
off_set = 0.050;
%           0    10   20   30   40   50   60   70   80   90   100  110  120
b_length = [61.3 63.2 64.8 66.1 67.4 68.6 69.4 70.5 70.6 70.6 70.5 69.4
68.6 67.4 66.1 64.8 63.2 61.2 60  58  56.4 54.7 53.6 52.3 51.4 51  50.7 51
52.3 53.6 54.7 56.4 58  60  60.8 61.3];

b_angle  = [9.5  9    8.4  7.6  6.6  5.3  4.0  2.7  1.5  0    2.7  4.0  5.3
6.6  7.6  8.4  9    9.5  9.6 9.4 8.9  8.1  7.2  5.6  4    2.2 0    2.2 4
5.6  7.2  8.1  8.7 9 9.2    9.5];
a = 1;

for i = 1:1:36 ;
    angle_c(i) =  asind(sind(i*10+90)*0.05/(b_length(i)/1000));
end
for i = 1:1:36
    force_b(i) = motor_torque*cosd((angle_c(i)))/0.01;
end
for i = 1:1:36
    torque_b(i) = force_b(i)*b_length(i)/1000;
end
for i = 1:1:36
    force_rod_per(i) = torque_b(i)/0.03;
end
for i = 1:1:36
    force_rod(i) = force_rod_per(i)*cosd(b_angle(i));
end
for i = 1:1:36
    force_needed(i) = 150*3;
end

plot(force_rod,'r')
grid on, hold on
plot(force_needed)
axis([0 37 0 900])
title('Force applied to magnets')
xlabel('Motor angular displacement')
ylabel('Force applied to magents')
legend('Force applied','Force needed')

motor_force = motor_torque / radius_gear
applied_force = motor_force * (off_set / radius_rod)
```

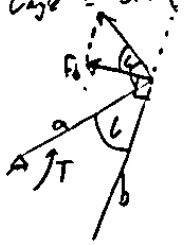
Calculation for PickUp Mechanism



rotation on solid works allowed for the recording on the changing length and angle of b



$C_{angle} = \sin^{-1}\left(\frac{\sin(B)\,c}{b}\right)$

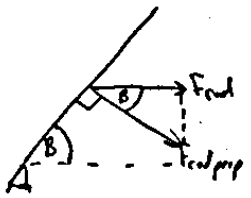


$$F_b = \frac{T\cos(C_{angle})}{a}$$

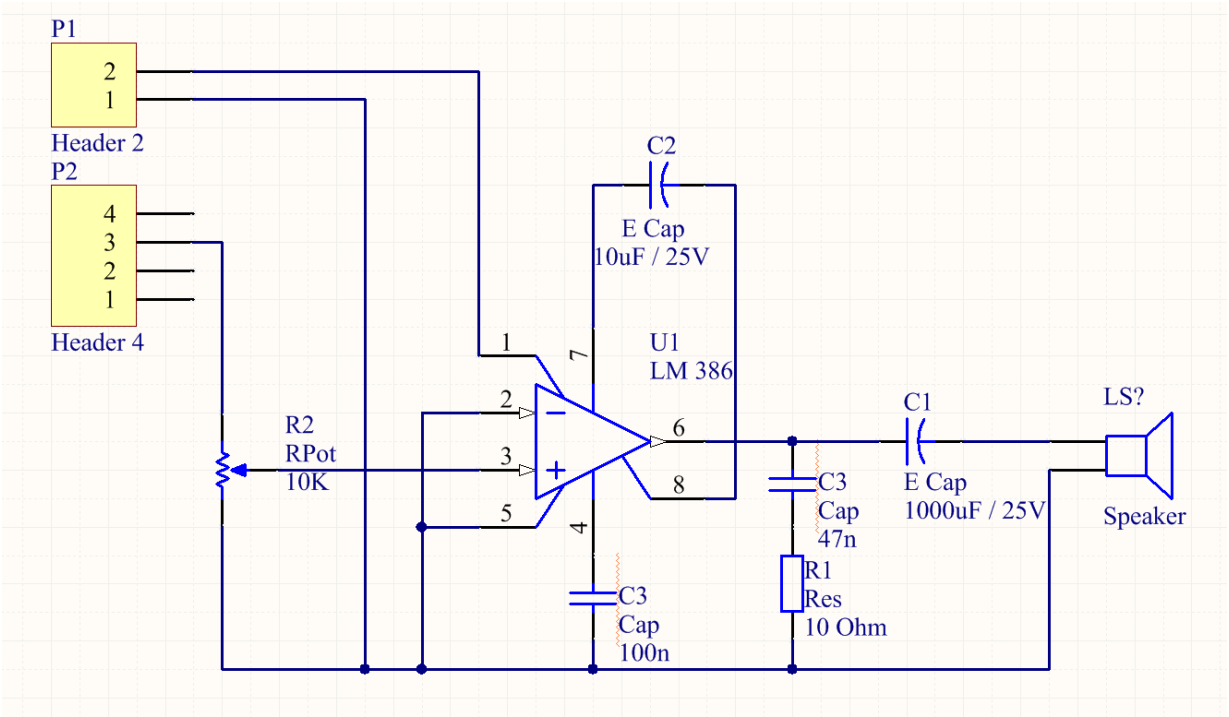
$$T_b = F_b\, b$$

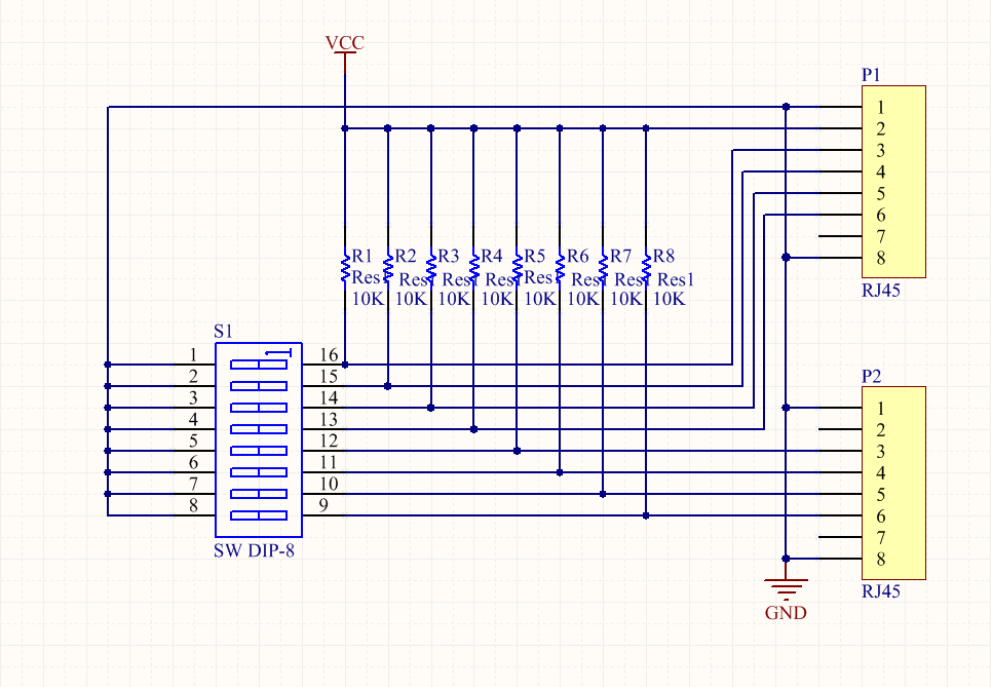
$$F_{rod\,perp} = \frac{T_b}{d}$$

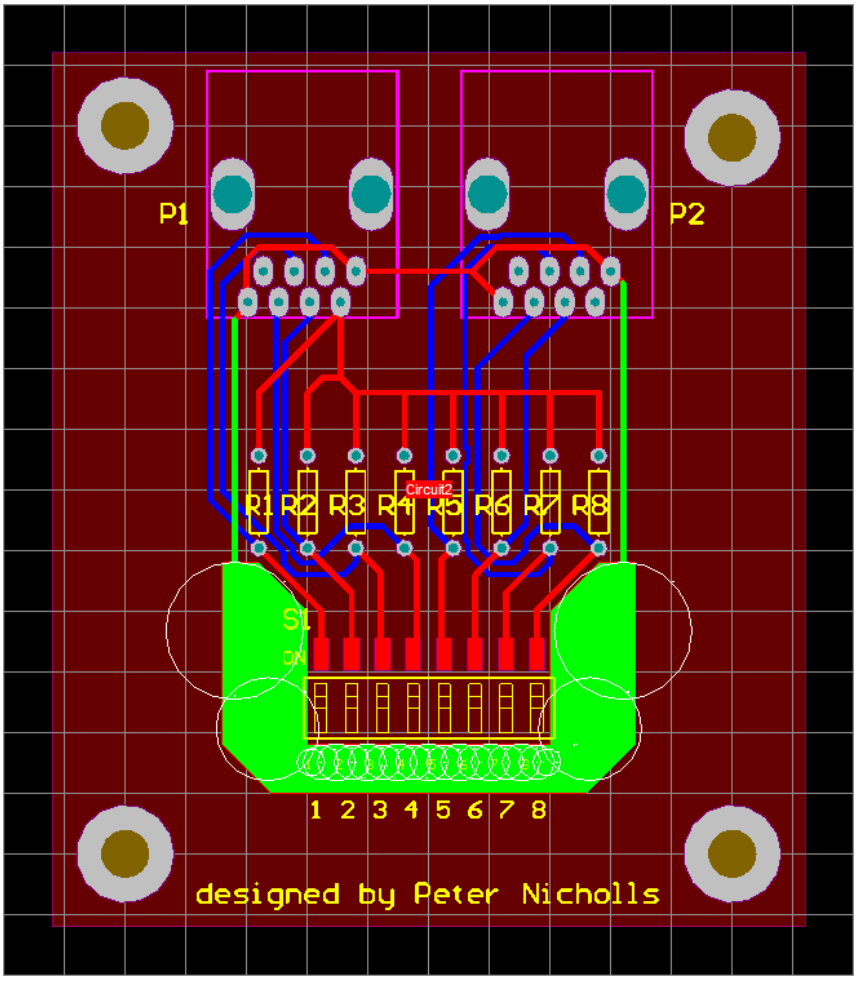
$$F_{rod} = F_{rod\,perp}\cos(B_{angle})$$

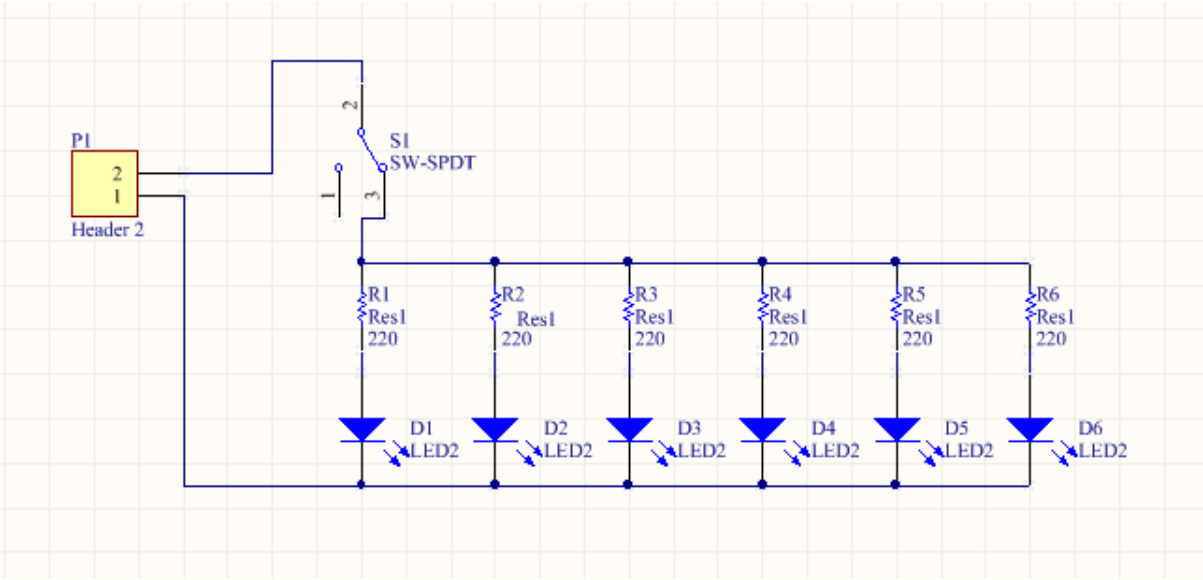


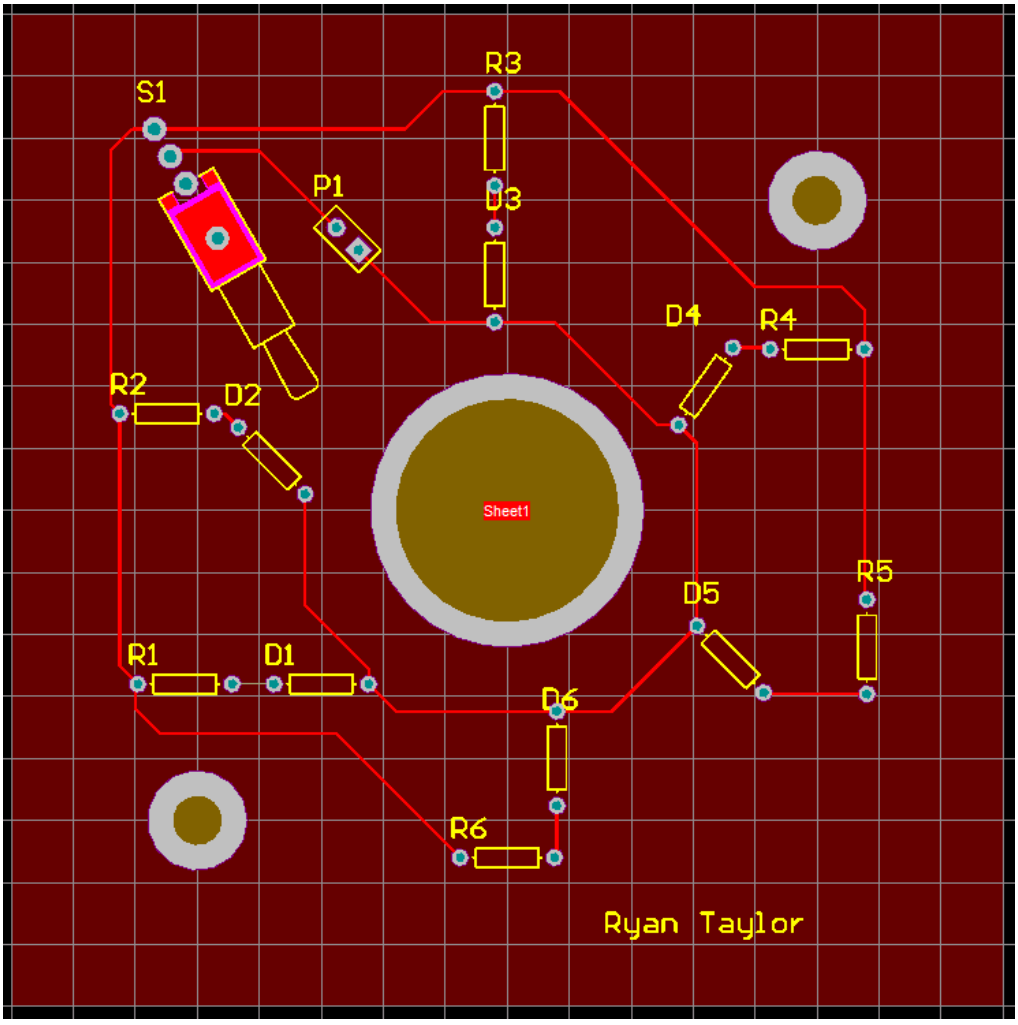
APPENDIX 2 – CIRCUIT SCHEMATICS AND PCB DIAGRAMS



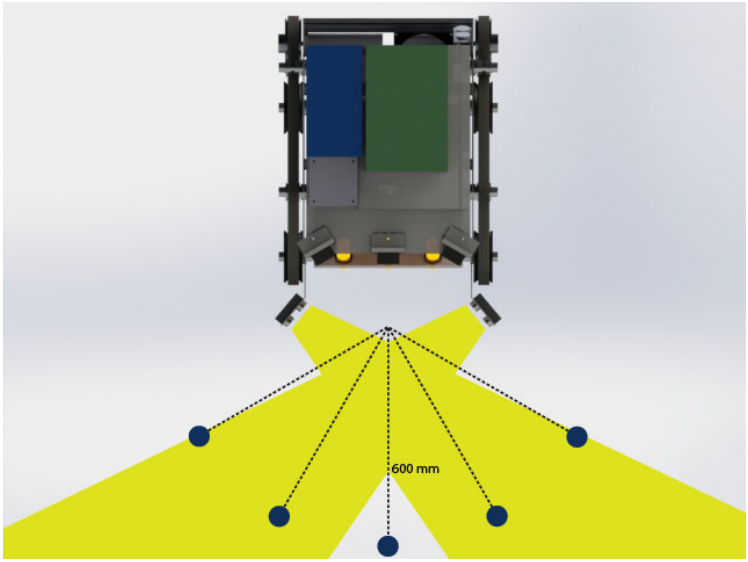

VCC
P1
1
2
3
4
5
6
7
8
RJ45
R1 Res 10K
R2 Res 10K
R3 Res 10K
R4 Res 10K
R5 Res 10K
R6 Res 10K
R7 Res 10K
R8 Res1 10K
S1
1
2
3
4
5
6
7
8
16
15
14
13
12
11
10
9
SW DIP-8
P2
1
2
3
4
5
6
7
8
RJ45
GND


P1
P2
R1 R2 R3 R4 R5 R6 R7 R8
Circuit2
S1
ON
1 2 3 4 5 6 7 8
designed by Peter Nicholls


P1
2
1
Header 2
S1
SW-SPDT
2
1
3
R1
Res1
220
R2
Res1
220
R3
Res1
220
R4
Res1
220
R5
Res1
220
R6
Res1
220
D1
LED2
D2
LED2
D3
LED2
D4
LED2
D5
LED2
D6
LED2

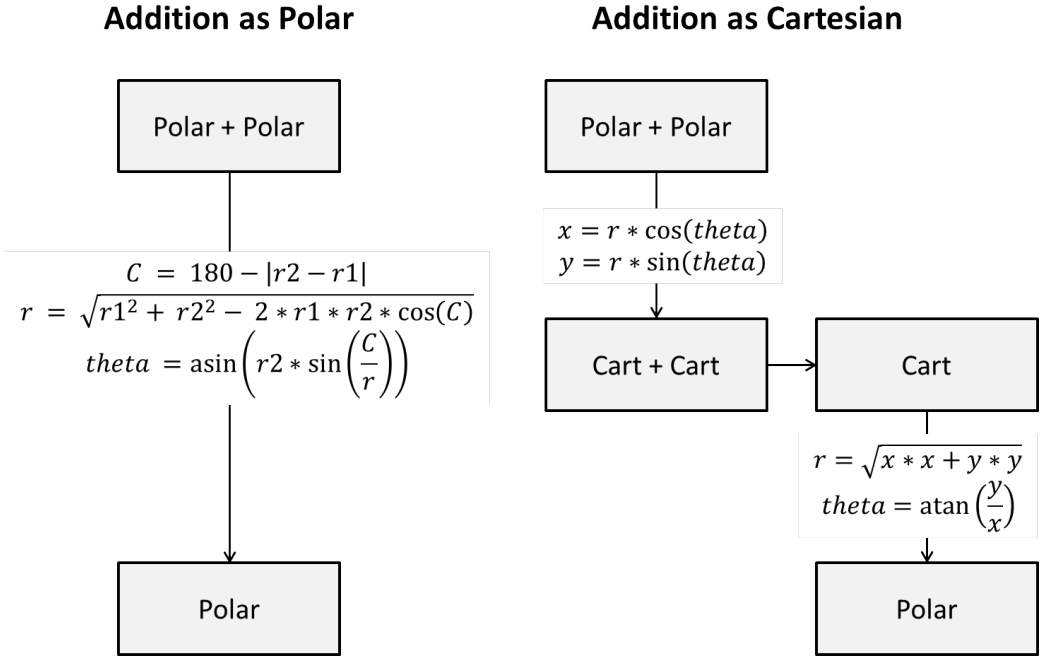

S1
R3
P1
D3
D4
R4
R2
D2
Sheet1
R5
D5
R1
D1
D6
R6
Ryan Taylor

APPENDIX 3 – PACKAGE DETECTION



Test #	Centre line	30 deg anti	60 deg anti	30 deg clock	30 deg clock
1	3.88	3.80	Not detected	NA	Not detected
2	4.00	3.96	Not detected	NA	Not detected
3	3.90	3.86	Not detected	3.96	Not detected
4	3.95	3.93	Not detected	NA	Not detected
5	3.90	3.82	Not detected	4.08	Not detected
6	3.92	3.80	Not detected	3.91	Not detected
7	4.04	3.88	Not detected	3.93	Not detected
8	4.14	3.84	Not detected	3.80	Not detected
9	3.91	3.93	Not detected	4.08	Not detected
10	3.64	4.13	Not detected	4.00	Not detected
Average	3.93	3.90	Not detected	3.97	Not detected

APPENDIX 4 – EFFICIENCY TESTS



Since the majority of vector addition will be adding an offset to the sensor, the offset can be saved in its Cartesian form. This would imply one fewer conversions.

	Polar	Cartesian	Difference	Efficient Cartesian	Difference
Add, Subtract	4	2	+2	2	+2
Multiplication	8	6	+2	5	+3
Division	1	1	0	1	0
Absolute Value	1	0	+1	0	+1
Tan Inverse	0	1	-1	1	-1
Sine	1	2	-1	1	0
Sine Inverse	1	0	+1	0	+1
Cosine	1	2	-1	1	0
If Statement	1	1	0	1	0
Square Root	1	1	0	1	0

APPENDIX 5 – BILL OF MATERIALS

BILL OF MATERIALS (PROVIDED)

Item	Quantity	Total Cost (NZ dollar)
Aluminium plate	2	32.00
Aluminium extrusion	4	Not priced
Sensor Circuits	4	Not priced
Arduino Mega ADK	1	60.00
Lithium Battery	1	Not priced
Ultrasonic Sensor	2	2.00
Medium Infrared Sensor	2	38.00
Long Infrared Sensors	1	17.00
Cables	NA	Not priced
3D Printed Wheel	4	Not priced
Smart Servo	1	50.00
DC motor	2	106.00
Drive Wheel	2	Not priced
Bearing (29mm)	16	Not priced
M3 Cap Screws	NA	Not priced
M8 Cap Screws	10	Not priced
Aluminium 15mm 40x100	1	5
Aluminium 5mm 120x100	1	10
Aluminium 2mm 300x200	1	5
Steel rod 1/4 inch 200mm	1	1
Steel rod 10mm 100mm	1	0.5
Spring 20mm compressive	1	0.1
Perspex 10mm 200x350	1	Not priced
Perspex 4mm 200x150	1	Not priced
M3	40	4
M5	4	0.75
Geared Stepper	2	30
Digital IR	3	15

BILL OF MATERIALS (ADDITIONAL)

Item	Quantity	Total Cost (NZ dollar)
SD Storage Card	1	4.09
SD Reader	1	2.11
Permanent Magnets	3	30.00
Total Cost		**$36.20**

Bill Of Materials (Circuit 1)

Item	Quantity	Cost (NZ dollar)
1000 uF/ 25V electro cap	1	0.22
10 uF/ 25V electro cap	1	0.04
100 n cap	1	0.05
47 n cap	1	0.40
IC LM386	1	0.08
10k variable resistor	1	0.33
10 Ω resistor	1	0.02
1 W speaker	1	0.51
Total Cost		**$1.65**

Bill Of Materials (Circuit 2)

Item	Quantity	Cost (NZ dollar)
SW DIP-8	1	2.51
RJ45 8P8C jack	2	0.50
Resistor 10K	8	0.02
Total Cost		**$3.67**

Bill Of Materials (Circuit 3)

Item	Quantity	Cost (NZ dollar)
IR LED	6	3
220 Ω resistor	6	1
Header pin	2	0.1
Switch	1	4
Total Cost		**$8.10**

APPENDIX 6 – D* LITE (PSEUDOCODE)

```
procedure CalcKey(s) {
return [min(g(s), rhs(s)) + h(s_start, s) + km, min(g(s), rhs(s))];
}

procedure Initialize() {
U = []
k_m = 0;
for all s in S rhs(s) = g(s) = inf;
rhs(s_goal) = 0;
U.Insert(s_goal, [h(s_start, s_goal); 0]);
}

procedure UpdateVertex(u) {
if (g(u) != rhs(u) AND u in U) U.Update(u, CalcKey(u));
else if (g(u) != rhs(u) AND u not in U) U.Insert(u, CalcKey(u));
else if (g(u) = rhs(u) AND u not in U) U.Remove(u);
}

procedure ComputeShortestPath() {
while (U.TopKey() < CalcKey(s_start) OR rhs(s_start) > g(s_start))
u = U.Top();
k_old = U.TopKey();
k_new = CalcKey(u));
if(k_old < k_new)
  U.Update(u, k_new);
else if (g(u) > rhs(u))
  g(u) = rhs(u);
  U.Remove(u);
  for all s in Pred(u)
  rhs(s) = min(rhs(s), c(s, u) + g(u));
  UpdateVertex(s);
else
  g_old = g(u);
  g(u) = 1;
  for all s in Pred(u) union {u}
    if (rhs(s) = c(s; u) + g_old)
      if (s != s_goal) rhs(s) = min_s'_Succ(s) (c(s, s') + g(s'));
    UpdateVertex(s);
}

procedure Main() {
s_last = s_start ;
Initialize();
ComputeShortestPath();
while (s_start != s_goal)
  /* if (rhs(s_start) = inf) then there is no known path */
  s_start = arg min_s'_Succ(s_start) (c(s_start, s') + g(s'));
  Move to s_start;
  Scan graph for changed edge costs;
  if any edge costs changed
    k_m = k_m + h(s_last, s_start);
    s_last = s_start;
    for all directed edges (u, v) with changed edge costs
      c_old = c(u, v);
      Update the edge cost c(u, v);
      if (c_old > c(u, v))
        rhs(u) = min(rhs(u), c(u, v) + g(v));
      else if (rhs(u) = c_old + g(v))
        if (u != s_goal) rhs(u) = min_s'_Succ(u)(c(u, s') + g(s'));
      UpdateVertex(u);
    ComputeShortestPath();
}
```

S Koenig, M Likhachev. (2002). '*Fast Replanning for Navigation in Unknown Terrain*' [online]. Available: https://www.seas.upenn.edu/~maximl/files/dlite_tro05.pdf

APPENDIX 7 – DRAWINGS FOR MECHANISM SYSTEMS