CAPSTONE PROJECT #41

Region 5 IEEE Robotics Competition

NOVEMBER 14, 2014

COLE GULINO | WILL MORELL | JEFFREY RIDDLE | MOHAMED SHEMY | BEN TULLIER
Advisor: Hsiao-Chun Wu

Executive Summary

Each year, the Institute of Electrical and Electronics Engineers (IEEE) Association Region 5 hosts a competition which pits students within the region against each other in a competition to see which group of students can design a robot that can perform the given task under certain constraints. Each year the competition changes giving students new and exciting challenges but retains the same theme of the designed robot being able to take information from its surroundings and autonomously make decisions as to its next course of action.

Louisiana State University has competed in this competition for a number of years, placing in a few of them. This year the team plans to do the same and win the competition gaining reputation for the school and grow its ever increasing list of academic accomplishments. The competition this year places the robot in a maze where it is expected to find its way out and determine the shortest or "critical" path. The maze will be randomized and increase in size each round culminating in a 7' by 7' maze. Upon the walls of the maze will be characters that, if recognized by the robot, can add to the final score. The robot will have a set time limit in which to solve the maze and scan the walls which will lead into the second part of the round where the robot will attempt the critical path without deviation.

The team researched many different options for our platform but ultimately decided that treads would best suit this competition given its ruggedness and its ease to overcome obstacles. By searching online, the team found a small tread kit that can easily be implemented in our design. These treads are cheap, provide good traction, fit within the size constraint, and would provide the means to overcome speed bumps present in the

maze. The next step was to find a motor that would be propel our robot through the maze. After extensive research, the team decided to use the Pololu 150:1 Micro Metal Gearmotor since it will provide the needed torque and a quick speed. It also allows each motor to act individually so that our robot can turn.

In order to take in the various inputs, control the motors, and process the data necessary, a powerful controller and processor must be used in order to ensure that the robot can navigate the maze successfully. In order to accomplish this, we have chosen the Raspberry Pi B+ as our controller. The Raspberry Pi B+ has a fast processor for its size and speed. The Raspberry Pi B+ also has numerous GPIO ports and a dedicated camera port. This allows it to meet all of our I/O needs. In order to run our robot, the team also needed to choose a programming language compatible with the Raspberry Pi B+ that has enough packages to run the functions necessary. The Python programming language was chosen. Python is a flexible and readable language that makes it great for collaboration. It is also compatible with numerous packages necessary to control the robot such as: OpenCV, RPi.GPIO, and PiCamera. The Raspberry Pi B+ running Python will meet all of our controller needs.

In order to maneuver the robot around the maze, it must be able to detect its surroundings. For this, the team developed a sensor package consisting of 3 infrared proximity sensors to detect the walls, and an array of 5 reflectance sensors to detect the lines bisecting each square of the maze. One issue with the sensors selected is that each sensor outputs analog information, and the Raspberry Pi controller has no onboard analog to digital converters. A circuit board using a microcontroller with onboard ADCs was designed in order to read each sensor and relay binary information about the walls and ground to the Raspberry Pi.

The robot requires vision capabilities to successfully capture, recognize and save characters on the maze walls. After comparing many alternatives in both hardware and software for the vision system, in categories such as power, speed, and overall cost, the team chose to use the Raspberry Pi Camera Module to implement the robot's vision system. The Raspberry Pi Camera Module is comparatively cheap, fast, and has a dedicated connection port to the Raspberry Pi B+. On the software side, the team chose to use the Python implementation of OpenCV to cover any image processing.

In order to run the various components mentioned above, the robot requires a robust and stable power system. After researching many batter technologies, the team chose rechargeable NiMH technology. The team designed the power system of the robot in such a way that we can use the single NiMH battery for every component by cascading the voltage level down in steps with regulators. The team created simple +12V and +5V rails for all our components to run from which in turn helped keep complexity down. We also designed a simple detection circuit for the battery that allowed us to see when the voltage had dropped too low below nominal voltage and needed to be charged.

All of these components will be housed on the chassis which we will develop in-house using the resources available in the machine shop. This chassis will be approximately 7 in. long and 5 in. in width. Each piece will be attached with either bolts or screws to ensure each piece will be securely placed.

Contents

Executive Summary....1
Problem Context:....8
Description of Document:....8
Background....9
Scoring....10
Rules....11
Pairwise Comparison Matrix for Robot....12
Functional Decomposition for Robot....13
Behavioral Model of the Robot....14
Processing....14
Arduino DUE....17
Propeller C3....19
Raspberry Pi B+....20
Raspberry Pi B+ Specifications Table....21
Controller Decision Matrix....22
Programming Language....23
Libraries....24
Motion....25
Wheels....25
Treads....26
Swerve Drive....26
Mecanum....27
Legs....27
Pairwise Comparison Matrix for Platform....28
Sensing....29
Wall Detection Sensor Package....29
Wall Detection Sensor Technology Comparison Table....30
Wall Sensor Design #1....30
Wall Sensor Design #2....31
Choosing a Specific IR Sensor....32

Aligning the Robot....32
Ground Reflectance Package Design #1....34
Ground Reflectance Package Design #2....35
Ground Reflectance Package Design #3....36
Choosing Specific Reflectance Sensors....37
Sensor Interface Board....37
Sensor Reading Algorithms and Calibration....38
Visualization....41
Related Competition rules overview....41
Character Recognition Functional Decomposition....43
Hardware....43
CMUcam4....43
The Raspberry Pi camera module....44
Technical Comparison between Raspberry Pi Camera Module and CMUcam4....46
Pairwise Comparison Matrix for Camera Hardware....46
Decision Matrix for Camera Hardware....46
Related Competition rules analysis....47
Related engineering analysis....48
Software....49
Power....51
Battery Technology....51
Lead Acid....51
Nickel Cadmium....51
Nickel Metal Hydride....52
Lithium Ion....52
Lithium Polymer....53
Alkaline....53
Comparison of Battery Technology Specifications....54
Single Vs Multiple Batteries....54
Voltage Regulation....54
Low Voltage Warning....56
Refined and Expanded Engineering Analysis....57
Motor....57
Field Analysis....58

Character Recognition Algorithm 59
Battery Lifetime 62
Final Engineering Specifications 63
System Description and Product Architecture 65
Algorithm 66
Material Selection 67
Chassis 67
Idler Bracket 69
Brackets and Spacer 69
Manufacturing 69
Chassis 69
Idler Bracket 69
C-Bracket, L-Bracket, and Spacer 70
Assembly 70
Tier 1 Assembly 70
Tier 2 Assembly 71
Final Assembly 72
Safety 72
Testing 74
Budget 76
Schedule 77
Phase One 77
Phase Two 79
Milestones 80
Appendix 82
A.1: Raspberry Pi B+ Data Sheet 82
A.2: Raspberry Pi Camera Module Data Sheet 82
A.3: SHARP GP2Y0A41SK0F IR Proximity Sensor Data Sheet 83
A.4: Example Code for Character Recognition in Python 84
A.5: Propeller C3 Specifications Table 89
A.6: Arduino DUE Specifications Table 90
A.7: Behavioral Model of the Robot 91
A.8: Behavioral Model of the Processor 92
A.9: Functional Decomposition of the Processor 93

A.10: Sensor Interface Board Circuit Diagram 94
A.11: Calibration Plans for the Sensor Package 96
IR Sensor Test Sheet 99
Reflectance Sensor Test Sheet: 100
A.12: CMUcam4 Specifications Table 101
A.13: Comparison Matrix and Analysis of Maze Solving Algorithm 102
A.14: Full Bill of Materials 105
Dual MC33926 Motor Driver Carrier 106
A.15: Wiring Diagram 108
A.16: Detailed Character Recognition Testing Plan 109
Works Cited 114
Websites 114
Books 115
Papers 115

Problem Context:

In 2013 robotic sales increased 12% over 2012, which is the highest number of robots ever sold. This growth shows that this industry will require educated young engineers. Each year Region 5 of IEEE holds a conference, which features a robotics competition between student teams in the region. A fully autonomous robot is needed to compete in this competition for LSU. The robot must be able to take in sensory data from the maze, dynamically determine a critical path for solving the maze, and relay maze information to the judges in the form of ASCII character "easter eggs" found on the maze walls. The robot will be given an amount of time to first explore the maze and map it, then a smaller amount of time to make a "critical run path" from the end point back to the start without deviation.

Description of Document:

This report is intended to report the design decisions and research that Team #41 has done thus far in the first semester of Senior Design. The problem was broken down into five components in the initial functional decomposition. These five engineering design requirements were researched and design decisions were made. This document will relay all of the information that has been researched by Team #41 thus far. It will analyze the requirements of each module, and then it will explain in detail how each module's requirements could be met. The document will provide the design alternatives that we examined, and the document will explain the reasoning why one solution is favored at the moment over the others.

Background

This year's IEEE R5 Robotics Competition is a maze solving challenge. The team's task is to build a robot that can autonomously navigate, extract information, and solve a maze in under a specified amount of time. The competition is composed of 3 qualifying rounds and a final round. The competition maze will come in three sizes 5'x5', 6'x6', and 7'x7'. Each successive qualifying round the maze will increase in size, and the final round's maze will have a size of 7'x7'. Each maze will have a specified starting and ending point, and the rest of the maze will be randomly generated. There will only be one critical path that will solve the maze. Each round is further broken into two parts: the first part being where the robot explores the maze and learns about it, and the second being when the robot attempts to solve the maze by running the critical path. In order for the robot to be able to compete in the final round, it must qualify by completing any critical path run from one of the three qualifying rounds.

The first part of each round is primarily focused on exploration and discovery of the maze itself. Points are awarded for this round by moving into new spaces and moving along the critical path, while losing points for repeating moves and bumping into the walls.

Scoring

Here is a detailed breakdown of the scoring system for the qualifying rounds:

- Entering New Square (Part 1) +1
- Correct Critical Path Move +2
- Lighting A Correct Light +1
- Repeating A Path Move -1
- Touching Wall (per square) -1
- Moving Off Critical Path (part 2) -1
- Easter Egg Number +1
- Easter Egg Alphabetical +2
- Easter Egg Symbols NA

For the final round, the winner is decided by a total adjusted time for the critical path run during the final round. Runtime is lowered for each character identified during the final round of the competition. In addition to the qualifying round's system of scoring, the final round also adds extra scoring opportunities:

- Adjusted Critical path Execution Time – Winning Criteria
- Search (part 1) Execution Time – Tiebreaker
- BTB – Bonus Time Base – Fastest Critical Path Time in Finals
- Easter Egg – Number - +1 plus 5% of BTB off critical path time
- Easter Egg – Letter - +2 plus 15% of BTB off critical path time
- Easter Egg – Other – No score

By identifying Easter eggs on the walls of the maze during the final round, the team can lower its final runtime. However this bonus time we can eliminate from our run time is capped at 35% of BTB.

Rules

Along with scoring rules, there are also special rules in place for the competition.

- No robot kits allowed, only original designs
- Damaging walls or maze will result in disqualification
- Robots can never extend above 7.5" tall throughout the competition
- Robots cannot split into separate parts, as well as cannot fly, jump, or climb over walls
- Robots will be sequestered between rounds by the judges and powered down
- Robots that stop moving for more than 10 seconds will have their current round ended
- Robots must have an easily accessible start and stop button
- No wireless communication is allowed with the robot during the competition rounds

Conceptual Design

The objective of this project is to create a fully autonomous robot. The robot must be able to gain information about and from the maze environment in the form of recognizing maze walls, characters, and location on a grid. It has to be able to determine the critical path between the start and the finish of a randomized maze. It must also be able to work for at least 30 minutes on its own without human interaction and without losing power. It must also be able to move without human assistance quickly and effectively through said maze without contact with maze walls. A third party must be able to safely initiate the robot.

Pairwise Comparison Matrix for Robot

	Size	Battery Lifetime	Low Cost	Fast	Accurate Maze Algorithm	Weight
Size	1	1/3	1/2	1/3	1/4	0.07
Battery Lifetime	3	1	2	1	1/2	0.22
Low Cost	2	1/2	1	1/2	1/3	0.12
Fast	3	1	2	1	1/2	0.22
Accurate Maze Algorithm	4	2	3	2	1	0.37

Table 1) Pairwise Comparison Matrix for Robot

The size constraints come from the fact that the robot must be 7”x7”x7.5” for the full duration of the competition. The robot cannot gain height. This means that the robot must be a certain size. The reason why this was ranked the lowest was that the robots size is important, but it is something that will be easier to implement. Speed and battery lifetime we analyzed the same, because they both are major constraints that affect how well the robot can complete its task. Cost was important, but lower than most, because we are well within our budget. Accurate Maze Algorithm was ranked highest, because the whole competition is about traversing the maze. More than movement and size, the traversal algorithm will determine how well the robot completes its task. The algorithm is also what sets our robot apart from any other robot one could make.

Functional Decomposition for Robot

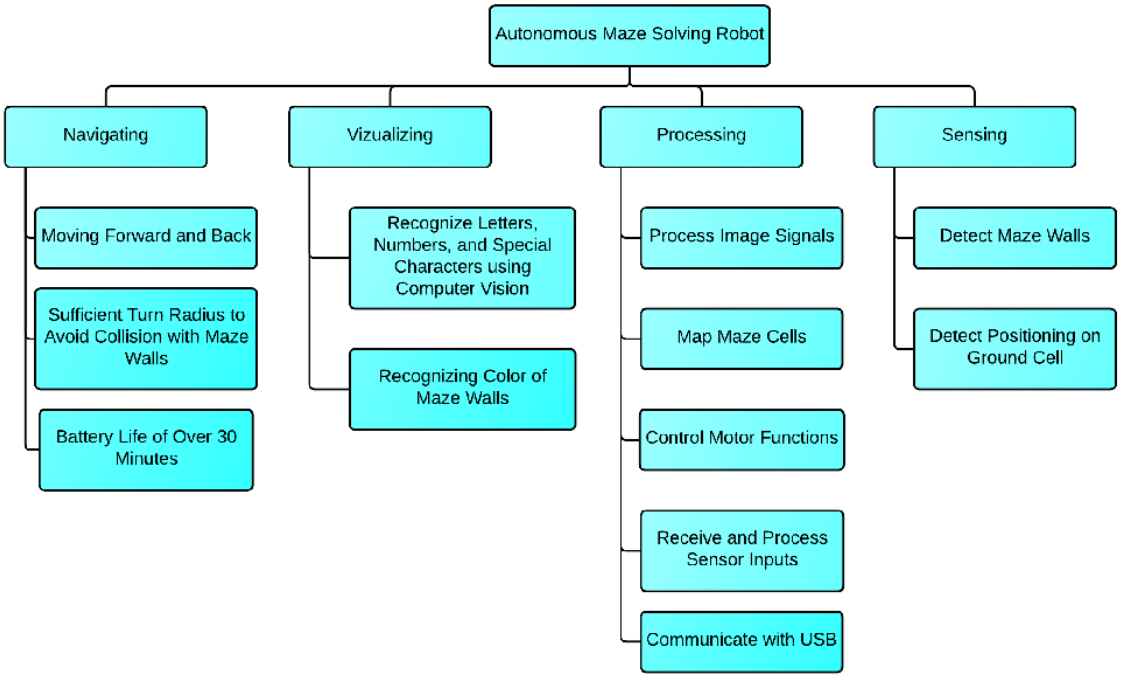


Figure 1) Functional Decomposition for Robot

The functional decomposition shown in Figure 1 breaks the robot into four main functions: navigating, visualizing, processing, and sensing.

The breakdowns into each functional group included the various options and different aspects of each module that must be undertaken by the design team.

Behavioral Model of the Robot

The behavioral model is centered on the microcontroller as it acts like the robot brain. It gets the feedback from the sensors and the pictures from the camera to process it and must importantly it runs the algorithm to calculate the critical maze path from square one to the end square. We have to choose the efficient microcontroller taking in consideration the processing power and the cost and some other categories.

See Appendix A.7 for the diagram of the behavioral model of the robot.

Processing

A controller/processor must control the movement of a robot, process information from sensors, including a camera. A processor must be able to process these simultaneously while having enough processing power to execute these quickly enough to move, map the maze, process and analyze images, and determine a critical path under specified time constraints. The processor must also be able to many inputs and outputs.

A microcontroller is a piece of hardware that has a CPU and a fixed amount of memory: RAM and/or ROM. Certain microcontrollers, such as the Arduino, have other hardware devices built in such as analog to digital converters. Certain boards are also built with connections like dedicated motor drivers, dedicated camera ports, and other

hardware meant to facilitate specific tasks. A microcontroller is made for a specific purpose: controlling motors, microwaves, cars, washing machines, etc. The CPU of a microcontroller has a specific type of architecture. Some examples are: ARM, Intel x86, MIPS, etc. The boards that a microcontroller is built with has a specific design made by the company that produces the controller. For example, the Arduino microcontroller can have an Intel or ARM based CPU architecture, but the board was made and designed by the company that produces the board. The computer architecture firm will usually source out their CPU architectures to multiple different board makers. Architectures are built around the number of bits that their instructions contain. Most architectures run on 32-bit instructions, but recently 64-bit instructions have become vogue. This allows for unification of software designs across multiple platforms. This allows compilers to run on multiple devices without having to be specifically built for certain computer architectures.

Certain microcontroller producers will design proprietary software that will only be able to run on their devices. This software can facilitate certain functions like accessing board pins and analog to digital converters. Other boards which are not programmed with proprietary programming languages are able to run multiple types of software like: C++, Java, Python, etc. These non-proprietary programming languages can have libraries built for them that run similar features on other microcontrollers that a proprietary programming language would run. A controller that can run multiple programming languages other than just proprietary software allows for flexibility in the design of software systems.

Microcontrollers communicate with a computer in multiple ways. This hardware is designed to either communicate in serial or in parallel with the computer. Some ways

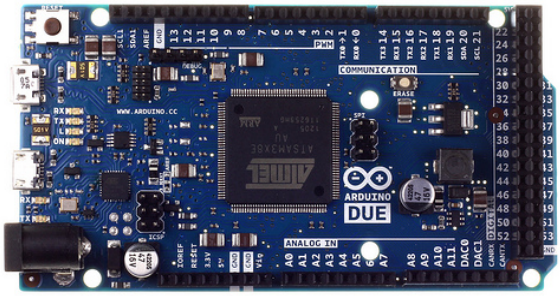
that the microcontroller can communicate with the computer is through: UART, USART, or TTL serial communication. UART stands for universal asynchronous receiver/transmitter. It is usually part of the integrated circuit and part of the computer architecture design. The form that the data can take is variable and controlled by a driver circuit which is external the UART, probably on the IC. The UART takes bytes of data and transmits them serially to the computer. A second UART reassembles the bits into full bytes which can be read and understood by the computer. Each UART contains a shift register and one wire for communication. Serial transfer is less costly than parallel communication requiring multiple wires. USART stands for universal synchronous/asynchronous receiver/transmitter. The USART is similar to the UART, but it offers the option of a synchronous mode, which requires that each end of an exchange respond in turn without initiating a new communication. Synchronous mode also requires a clock. Serial mode also allows for a higher data transfer rate than asynchronous mode. Finally TTL serial communication is a form of serial communication (UART or USART) which transfers one bit at a time. This communication is based on 0s being ground and 1s being Vcc. This Vcc is based on the power supply of the controller.

See Appendix A.8 for the Behavioral Model of the Processor.

See Appendix A.9 for the Functional Decomposition of the Processor.

Whenever we were looking at processors we were concerned about programming languages usable on the controller, speed of processor, power needed, number of input and output ports, size, and cost.

Arduino DUE



arduino.cc

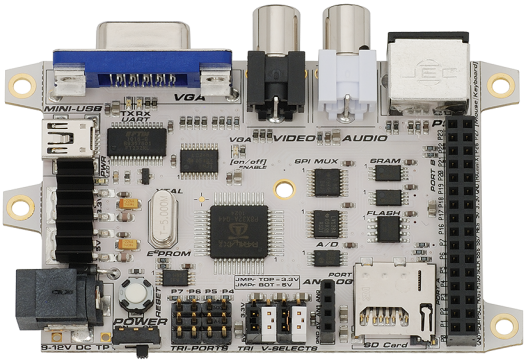
The Arduino DUE is a microcontroller is based on the ARM CPU architecture. It can communicate through the computer through one UART, two USART. The TTL communication is based on 3.3V. The Arduino DUE is programmed with proprietary software. This software is based on C++ and has a large development community which builds libraries to support pin I/O, servo motors, stepper motors, cameras, LCD screens, etc. The Arduino DUE has two USB ports: one for programming and one native port. The native port runs a "soft erase" in order to reset the programming on the Arduino board.

The Arduino DUE is a more capable machine than a more typical model like the Arduino UNO. The price is very competitive, and the size and I/O pins are adequate for our needs. Where the Arduino UNO comes up short is in clock speed and optional programming languages. With a clock speed of 84 MHz, the Arduino DUE is not competitive with the other microcontrollers that it was compared with. The other shortfall of the Arduino DUE is that it only runs the Arduino proprietary programming language. This does not give enough flexibility to run libraries that are not compatible with this proprietary software.

See Appendix A.6 for the Arduino DUE Specifications Table

Another class of microcontrollers are those that run full computer operating systems such as: Linux, Unix, Windows Embedded, etc. These microcontrollers have the full suite of programs as regular computers, but their architecture is made for a smaller embedded system that is made to run more complicated software than a simple microcontroller, but is made to be run on a specific and limited hardware in comparison to a full computer. An operating system is the low level software of a computer or microcontroller that operates as an interface between the hardware of the computer and the high level software from applications or computer operators. Operating systems like Windows and OSX, which is a form of Unix, are highly developed software systems that are made by large corporations. The Linux operating system was designed as an open-source version of the Unix operating system. Linux has been developed collaboratively since the 90s undergoing many iterations. There are numerous variations of Linux such as: Ubuntu, Debian, RHEL (Red Hat Enterprise Linux). There are also many operating systems that are repurposed versions of Linux. Some of the main examples are Google's Mobile System, Android, Google's Desktop System, Chrome OS, and Raspbian which is a repurposed version of Linux Debian designed to run on the Raspberry Pi. Microcontrollers with full-fledged operating systems are sometimes termed "microcomputers".

Propeller C3



parallax.com

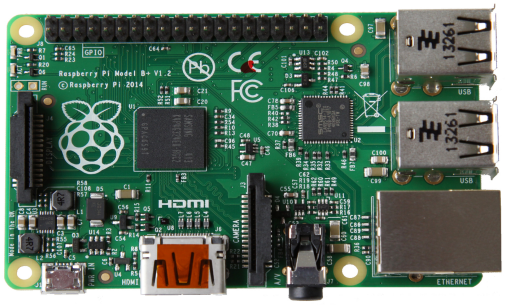
The Propeller C3 is a "Credit-Card-sized-Computer" made by Parallax Inc. which is designed as an add-on to the standard Propeller CPU which allows it to function as a microcomputer beyond what a regular microcontroller can do. The Propeller C3 has an SPI (Serial Peripheral Interface) which allows for serial sequential data. This is an industry standard interface which allows the Propeller C3 to work with many different types of sensors and interfacing apparatus. The Propeller C3 also comes with a 2 channel SPI analog to digital converter. It also comes with four three wire ports for dedicated servo motors. The Propeller C3 runs on a 7.5-12V DC power supply which is much higher than the Arduino DUE, but this is the same voltage range that the motors would run on, which would allow us to be able to use one battery for the motors and the microcontroller without having to step down the voltage with a voltage regulator.

The Propeller C3 works as a full-fledged computer which gives it the advantage over the Arduino DUE in that it can run multiple high level languages instead of just proprietary software. The other advantage it has is that it has dedicated servo wire ports

which simplifies the wiring designs. It also frees up other pins for sensor input and output. It is slightly smaller and faster than the Arduino DUE. The other advantage is that the Propeller C3's architecture is a proprietary multicore processor. The processor has eight "cogs" which can all run separate tasks. These "cogs" have their own small memory caches and reach into a single pool for main memory. This parallel architecture allows for faster processing in a parallel environment. The operating system may be more flexible than the Arduino DUE, but it is not as flexible as a full-fledged computer. The main drawback to the Propeller C3 is that it is much more expensive than the Arduino DUE.

See Appendix A.5 for the Propeller C3 Specifications Table.

Raspberry Pi B+



raspberrypi.org

The Raspberry Pi B+ is the newest version of the Raspberry Pi credit-card-sized Computing module built and designed by the Raspberry Pi Foundation in the UK. This microcomputer's architecture is based on the ARMv6 32-bit architecture. The Raspberry Pi B+ is a Linux based machine. The most popular operating system to run on the Raspberry Pi is the Debian based Raspbian, which sports an easy to use GUI (Graphical User Interface). Other Linux operating systems that run on the Raspberry Pi are Fedora and Arch Linux ARM which are lighter weight versions of Linux that do not have an

extensive GUI like Raspbian does. The Raspberry Pi B+ microcomputer runs on 5 VDC. It also has an HDMI cable which can facilitate sound and video from the Raspberry Pi to a monitor. It also sports four USB 2.0 ports which has many uses. Because the Raspberry Pi B+ is a microcomputer, it does not need to interface with another computer in the normal way that a microcontroller must, and it can communicate through the internet or with an Ethernet port just as any other computer can.

Raspberry Pi B+ Specifications Table

Module:	Raspberry Pi B+
Inputs:	Power: 12 A @ 5 VDC (Micro USB) GPIO: 26 HDMI 1 x mini CSI-2 Camera Adaptor 4 USB Ports
Outputs:	GPIO: 26 HDMI 1 x mini CSI-2 Camera Adaptor 4 USB Ports
Flash Memory:	Variable SD Card
RAM:	512 MB
Clock Speed:	700 MHz
Dimensions:	85mm x 56mm
Price:	$35.00

Table 2) Raspberry Pi B+ Specifications Table

The Raspberry Pi B+ model is around the same size as the Propeller C3 module and a little smaller than the Arduino DUE. All three microcontrollers researched will easily be small enough for the robotic design. The Raspberry Pi B+ model also has enough GPIO pins to meet our needs for sensors and motors. The Raspberry Pi B+ model has a dedicated camera port which will make the wire layout much easier. The team needs a camera module for the robotics project, and the Raspberry Pi camera module is a great choice. This camera will work easily with the Raspberry Pi B+ microcomputer. The Raspberry Pi B+ model is the fastest and cheapest microcontroller that have been researched. The other great advantage of using the Raspberry Pi B+ is that not only does it have flexibility in the programming languages that can be used, but there is also flexibility in the number of operating systems that can be run on the Raspberry Pi B+. This flexibility allows for numerous variations of robotics applications.

Refer to Appendix A.1 for the Raspberry Pi B+ Data Sheet.

Controller Decision Matrix

	Propeller C3	Raspberry Pi B+	Arduino DUE
Fast (0.4)	0.114	1.000	0.120
Size (0.2)	1.000	0.880	0.774
Cost (0.4)	0.389	1.000	0.764
Score	0.401	0.976	0.508

Table 3) Controller Decision Matrix

Programming Language

For a programming language, speed measured in CPU time is certainly important. The benchmark used to measure CPU time was calculated using math oriented programming. The benchmark looked at loops, dynamic arrays with numbers, and basic math operations. The program used for the benchmark was a program that calculates prime numbers within a specified range. This program was run 10 times in order to ensure that there were no outliers in the program execution.

It is worth noting that there are extensions that can increase the performance of Python. One such extension is called Psycho Python, which can increase performance up to three times the performance shown here.

The speed of the programming language, however will not drastically improve the performance of the robot due to the fact that the main bottleneck in the speed of the robot will be the time lost from analyzing the sensors. This analysis will not be dramatically affected by the speed of the programming language. The main speed loss from the programming language will come from processing of images. Because the character recognition portion of the contest is only to augment speed, the team could conceivable win the competition without recognizing a single character. This is why the team has put other aspects ahead of CPU time when analyzing the effectiveness of a programming language in our project.

The number of supported libraries and packages for the Raspberry Pi has taken precedent over CPU time for our needs. Python has many libraries written which are made specifically to access the Raspberry Pi and the Raspberry Pi Camera Module. Moreover the libraries written for the Raspberry Pi Camera Module are also made to work directly with OpenCV, our computer vision software. By choosing a programming

language with a large support of Raspberry Pi, the team will increase the readability, flexibility, and speed that code can be written. This increase in human performance will out way any inherent speed increase that C++ offers over Python.

	Python	C++
CPU Time (0.1)	28,054 (0.1209)	3,392 (1.0)
Raspberry Pi GPIO Library (0.3)	1.0	0.5
OpenCV (0.3)	1.0	1.0
PiCamera Library	1.0	0.5
Total	0.91209	0.7

Table 4) Decision Matrix for Programming Languages

Libraries

RPi.GPIO

This library is fully supported by the Python Programming language. The team will use this programming language to access the Raspberry Pi GPIO Ports. The library is free and fully open sourced. It allows for one to control whether a pin is input or output. It allows the programmer to specify when to send a High or Low signal as output and when to take a signal input from one of the pins. It will make getting inputs from the interface board simple, and it will allow the team to control motors with ease.

OpenCV

This library is also open source and will be the team's main computer vision software. This language fully supports the operating system of the Raspberry Pi (Linux) and the programming language we have chosen (Python). The main applications for the project in OpenCV are the segmentation and recognition and artificial neural network algorithms present in the code. These applications will allow the team to segment off a

portion of the image captured by the camera and run it through recognition software from a trained artificial neural network in order to recognize which of the characters has been learned by the artificial neural network. The one closest match will be the output of the algorithm.

PiCamera

The PiCamera library is a library that is made specifically for the Raspberry Pi and the Raspberry Pi Camera Module. This library can directly access the Raspberry Pi's dedicated camera port to allow for seamless integration between the Python programming language, the Raspberry Pi, and the Raspberry Pi Camera Port. This language also allows the programmer to send videos and images directly to the OpenCV library.

The programming language, controller, and chosen libraries allow for seamless integration between the hardware and software. For this reason, they have been chosen together.

Motion

In order for the robot to solve the maze it's placed in, the robot must be able to move itself to the next square. The first step in the robotic design was to research and choose a robotic platform that would be best suited for the competition. There are a variety of platforms to choose from but the team decided which would best fit the robot.

Wheels

One of the most common platforms in robotics is wheels. Wheels are used throughout modern day life especially on cars, thus it is first choice for many due to its simplistic, familiar design. Some designs can range from using two wheels, four wheels, six wheels, or even more. There is also a huge variety in choice. Tires come in many

different sizes and treads to best suit whatever needs one may have. Wheels can also be a good economic choice since they are usually cheaper than many of the other available platform.

Treads

Another common platform used throughout the robotics industry is treads. Treads, much like wheels, are a great choice for its maneuverability and simple design. One of the advantages treads have over wheels is its ruggedness. Treads can easily overcome most obstacles it may meet. It has a great ability to climb over obstacles where wheels may have trouble. However there is a tradeoff. With this ruggedness comes the loss of speed, but since this is a small robotics competition this will not be an issue as it will be driving at relatively low speeds.

Swerve Drive

Swerve Drive is a system of wheels that have two separate motors per wheel. One motor drives the wheel forward and backwards while the other can turn the wheel on its side up to 90˚ to allow both forward and lateral movement without having to turn the entire robot. This allows for great maneuverability and a way to cut down on time wasted by turning. Since this a more complex system than most of the other choices, more time must be spent ensuring the system works properly. If one of the motors becomes misaligned from the others it can wreck the system resulting in a slower speed or it may travel in an unintended manner. The cost of a swerve drive system is high due to the amount of motors needed for each wheel.

Mecanum

Mecanum wheels are similar to the swerve drive system in terms of maneuverability, in that they allow forward, backward, and lateral movement. The key difference is that Mecanum wheels do not need to rotate 90˚ for the lateral movement. Mecanum wheels are wheels with a series of rollers angled in a way to produce a lateral force. This results in a less complex system and greatly decreases time spent turning. One of the major issues with Mecanum wheels is the cost. The wheels themselves are expensive and can be priced over $100 for a set of four and a motor would be needed for each wheel. Another problem is the ruggedness of the wheels. Since the wheel consists of rollers, it can become unaligned with the intended path after it climbs an obstacle.

Legs

Legs were a consideration in the early stages of development but were quickly dropped due to the lack of advantages. Legs would need a great deal of time to develop and more for it to work effectively. Legs would also be very slow compared to the other choices. The only advantage would be that it could easily climb the walls of the maze but since that is not allowed in the competition, other platforms were given consideration.

To choose the right platform, the team looked at several criterions. Maneuverability, speed, and ease of alignment were given top priority followed by complexity and uneven terrain. To easily compare these choices we developed a design matrix giving each criterion a weight.

Pairwise Comparison Matrix for Platform

Criterion	Weight	2-wheel	4-wheel	Swerve-wheel	Treads	Mecanum
Maneuverability	10	s	s	+	s	+
Speed	8	s	s	s	s	s
Easy to Align	8	-	-	+	-	+
Power Consumption	6	+	s	-	s	s
Robust	6	s	s	s	+	-
Lightweight	5	+	s	s	s	s
Size	4	s	s	s	s	s
Uneven Terrain	7	-	s	s	+	-
Complexity	7	+	s	-	+	-
Cost	5	+	-	-	s	-
Total +		4	0	2	3	2
Total -		2	2	3	1	5
Overall		2	-2	-1	2	-3
Weighted		-16	-13	0	12	-7

Table 5) Pairwise Comparison Matrix for Platform

From this matrix, it was found that treads would be the best choice for this competition.

Sensing

In order for the robot to learn from and react to the maze environment, it must have a set of sensors that take in information from this environment. Three problems must be solved by the robot's sensor package. The first is detecting the walls of the maze. The robot has incomplete information of the maze configuration at the start of the round and must fill in this information as it travels through the maze. The second problem is that the robot must move easily from square to square without hitting the maze walls. The third problem to be solved is that the robot must be able to center itself within each square in order to accurately detect the locations of the walls and to take photos of the characters on the walls. The sensor package must be able to provide enough information to solve all of these problems for the robot.

Wall Detection Sensor Package

The problem of wall detection can be solved in a variety of ways. For this application, three specific technologies were evaluated: IR proximity sensors, ultrasonic proximity sensors, and using the camera module with computer vision software. Using the camera module was discarded immediately as it would add a considerable amount of processing time for each step of the algorithm without an equally significant amount of cost reduction. If the maze was not made up of 90-degree elements, the camera would be a necessity for wall detection, but the geometry of the maze can be used to simplify sensor implementation. The other two technologies were evaluated for five categories:

Wall Detection Sensor Technology Comparison Table

Criteria	**IR Proximity Sensor**	**Ultrasonic Proximity Sensor**
Cost	$5 - $12 each.	$25 - $40 each.
Max Operating Range	~1.5 meters	~3 meters
Output	Digital or Analog	Digital Pulse
Relative Accuracy	Same	Same
Relative Ease of Use	Same	Same

Table 6) IR and Ultrasonic Wall Detection Sensor Comparison

The IR proximity sensor was chosen for this robot mainly based on the cost of the sensor. The proposed operating range for detecting the walls is on the order of 10 - 25cm, therefore the IR proximity sensor's max operating range is more than adequate to detect the walls of the maze.

Once the sensor technology was chosen, the next step was to determine how the sensors would be mounted on the robot and what they would detect.

Wall Sensor Design #1

The first design iteration of the wall sensor package involved four individual IR sensors, each mounted facing outward on each side of the robot. The exact position of each would be decided by testing as the detection range can be modified in software after mounting. Each sensor would detect an individual wall and give back an analog voltage signal detailing the exact distance away.

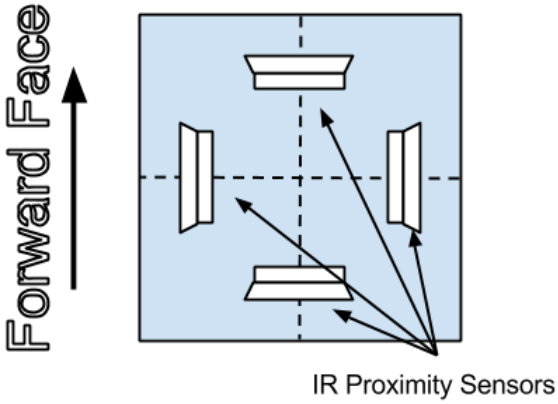


Figure 2) Wall Sensor Design #1

Wall Sensor Design #2

The second design is merely a simplification of the first design. The rear sensor can be eliminated as it faces a square that the robot just traveled from, the mapping algorithm can determine that there is no wall there.

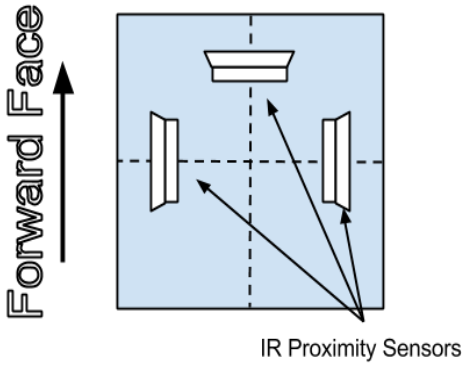


Figure 3) Wall Sensor Design #2

Choosing a Specific IR Sensor

Once the sensor configuration was designed, a specific model of IR sensor package was chosen to implement it. Based on the operating range for wall detection, the Sharp GP2Y0A41SK0F IR sensor was chosen.

Operating voltage	4.5 V to 5.5 V
Average current consumption	12 mA
Distance measuring range	4 cm to 30 cm
Output type:	analog voltage
Update period:	16.5 ± 4 ms
Size	44.5 mm x 18.9 mm x 13.5 mm
Weight	3.5 g
Unit cost	$7.40 ea.

Table 7) Sharp GP2Y0A41SK0F Feature Summary

Aligning the Robot

The other two problems to be solved by the sensor package, aligning the robot within each square, and aligning the robot in travel from square to square can be accomplished using the same sensors.

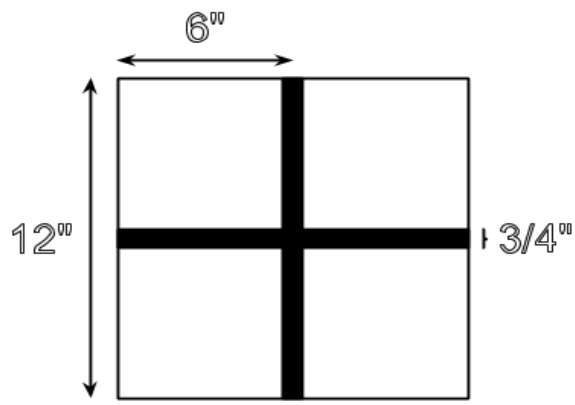


Figure 4) Maze Cell Center Lines

Each maze square itself is bisected on both the horizontal and vertical axes by a ¾" black line, separating each square into different black and white zones. The difference in light reflected from each zone can be exploited in order to determine where the robot is within each square or have the robot follow one line to the next square.

IR reflectance sensors can be used in order to measure this reflectance differential. IR reflectance sensors consist of a simple IR LED and phototransistor which, along with bias resistors, output an analog voltage as a function of light reflected back from the surface under test.

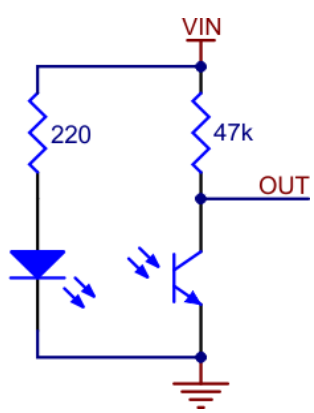


Figure 5) Reflectance Sensor Circuit

Using an IR reflectance sensor for this purpose is a bit of a solved problem for hobby and competition level robotics, as the implementation is straightforward and the hardware involved is very cheap.

Ground Reflectance Package Design #1

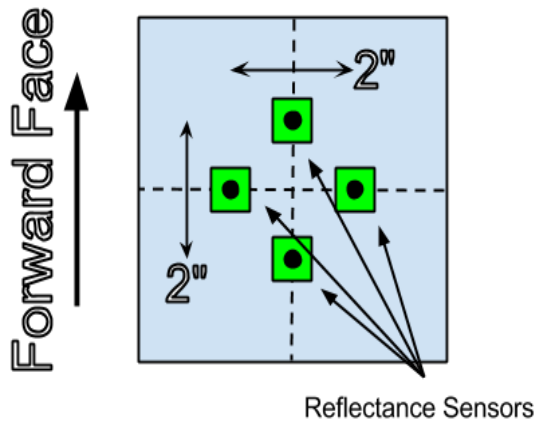


Figure 6) Reflectance Sensor Design #1

The first design for the ground reflectance sensor package involved mounting four reflectance sensors, two mounted on each center line of the robot with a 2" mounting pitch between each sensing element. This would provide information to the robot for whether or not each sensor was over a black or white surface. If each sensor was over the black center line then the robot would know that it was centered in a specific square (within an acceptable margin of error). However, this design falls apart when considering the problem of the robot moving from one square to another. Ideally, the robot would follow along a line to another square, and use this line to avoid contacting the walls, but with this first design, there is not enough sensory information to provide positional

feedback to the system. If the robot deviates from the line, it only knows that it has deviate, not how it deviated. Thus, a design revision is required.

Ground Reflectance Package Design #2

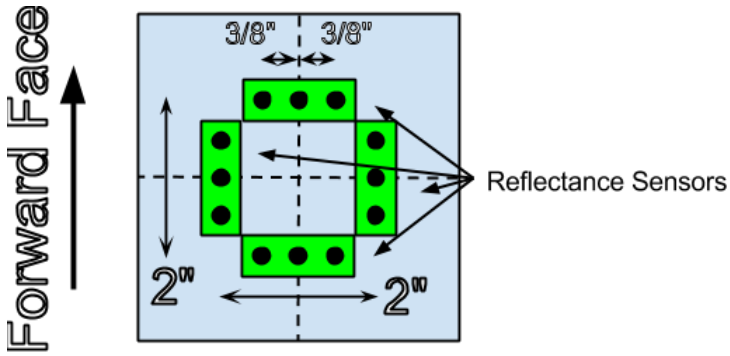


Figure 7) Reflectance Sensor Design #2

The second design replaces each sensor with a sensor array of three reflectance sensors. Each sensor array has a pitch distance of ⅜" which easily provides positional feedback for traveling from one square to the next. For instance, if the middle and right sensor of the front array detects black, and the left sensor detects white, then the robot knows to move right to align itself back to the line. This design gives both positional feedback while also facilitating alignment within a specific square. However, much of the sensory information cannot be adequately used, and the design is over engineered.

Ground Reflectance Package Design #3

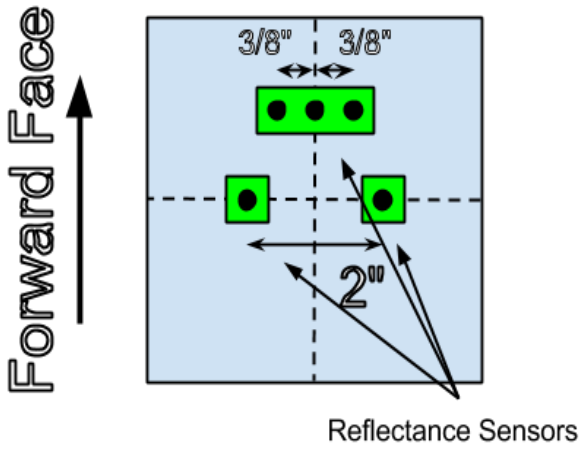


Figure 8) Reflectance Sensor Design #3

This final design combines the respective merits of both original designs. The front sensor is used to provide line following capability from square to square through positional feedback while the two side sensors give the robot ability to center itself in a square in order to accurately measure where the walls are and to photograph the characters.

Choosing Specific Reflectance Sensors

Criteria	**Single Sensor**	**Tri Sensor**
Part Number	Pololu QTR-1A	Pololu QTR-3A
Operating voltage	5 V	5 V
Average current consumption	17 mA	50 mA
Output type:	analog voltage	analog voltage
Size	0.3” x 0.5” x 0.1”	1.25” x 0.3” x 0.1”
Weight	0.2 g	0.6 g
Unit cost	$4.25 / 2	$4.95 ea.

Table 8) Reflectance Sensor Specifications

Sensor Interface Board

In order for the sensors chosen to communicate with the Raspberry Pi, some sort of interface must be created. This is because all the sensors chosen output information in the form of an analog voltage as a function of the input information (either object distance or object reflectance), and the Raspberry Pi has no onboard way of discriminating analog voltage levels. Using an Atmel 328p microcontroller, a board was designed to interface the sensors with the Raspberry Pi. The Atmel 328p was chosen for this job because of its 8 onboard analog-to-digital converters. Each sensor would be connected to a different ADC and the Atmel 328p would preprocess the sensor information for the Raspberry Pi.

The board was designed using EaglePCB, and can be printed by SeeedStudio for $9.90 for 5 boards.

See Appendix A.10 for the full schematic and layout.

See Appendix A.11 for calibration plans for the sensor package.

Sensor Reading Algorithms and Calibration

The algorithm that reads it sensor is relatively simple. Each sensor will have a specific calibrated range of operation as determined through a calibration test. The Atmel 328p will then read the analog voltage output from the sensor and determine whether or not it is in the calibrated range. If the inputs are within the calibrated range, it would output a digital TRUE result to the Raspberry Pi. For the reflectance sensors, this would correspond to the sensor being over the black line, for the walls it would indicate the presence of a wall.

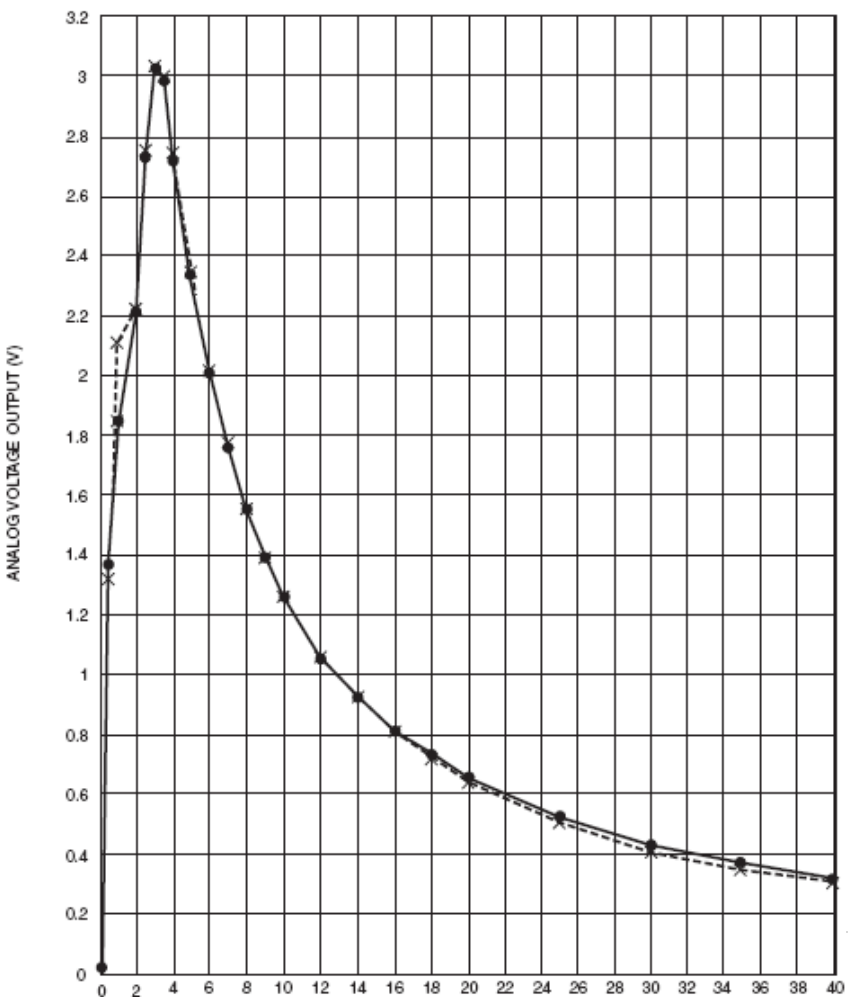


Figure 9) Example Wall distance vs Voltage curve:

Note: this is the manufacturer's example curve, each sensor will have to be measured and calibrated individually for actual implementation.

This curve was generated by placing the reflectance sensor ⅛" away from a rotating white disk marked with a radial black line. The peak voltage is 4.65 V and occurs whenever the black line passes under the sensor.

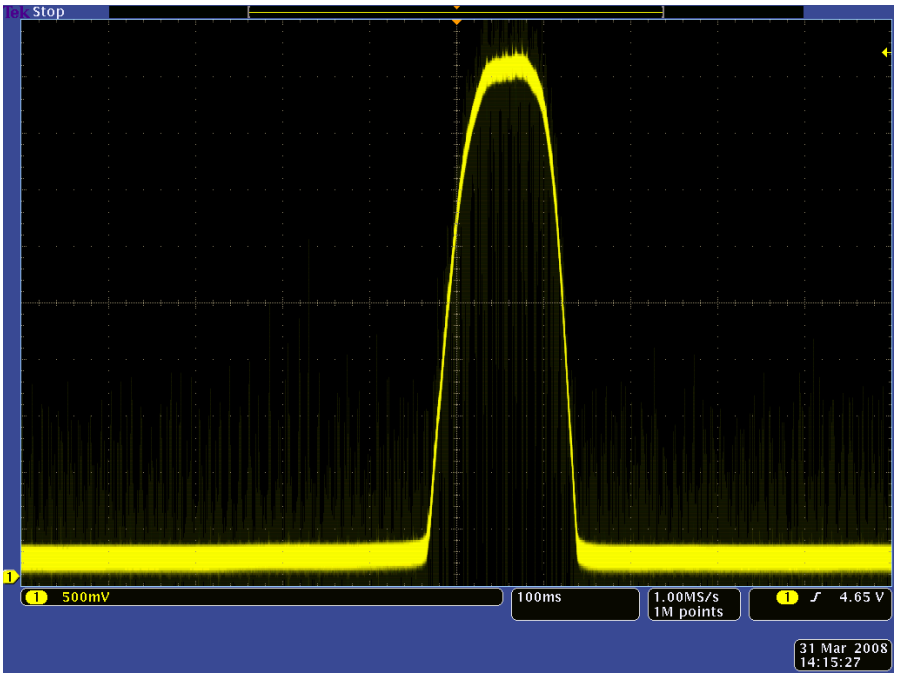


Figure 10) Example Voltage vs Reflectance curve for a ⅛" separation

Note: Just as with the wall sensors, the individual reflectance sensors must also be tested and calibrated separately.

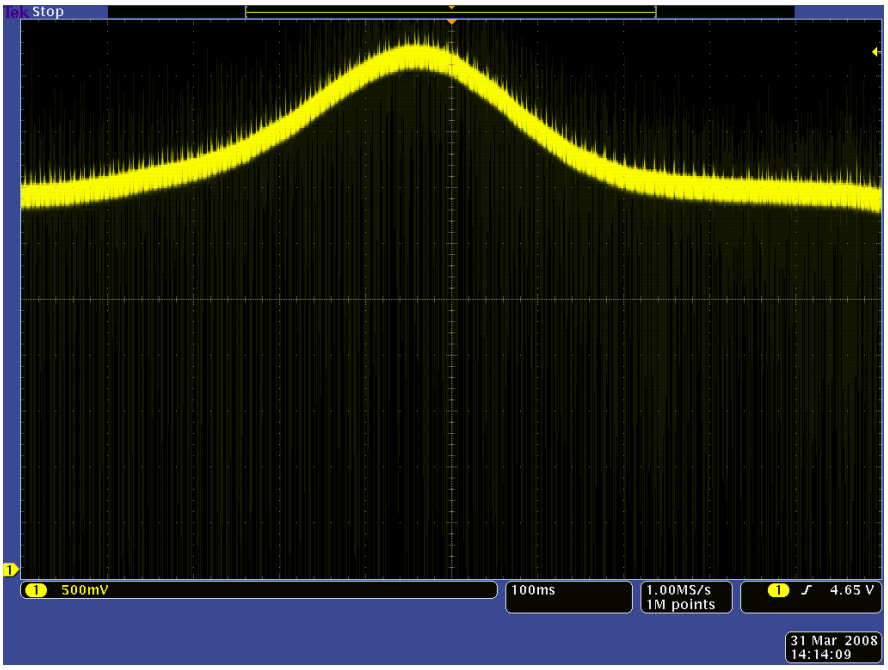


Figure 11) Example Voltage vs Reflectance curve for a ⅜" separation

This curve was generated just as the last one, but with a ⅜" separation. Even if the ideal ⅛" separation is not maintained in manufacturing, there is a large margin of error that can be made up for in calibration and software.

Visualization

Related Competition rules overview

According to the competition rules the robot can collect points for correctly identifying and storing characters that appear on the maze walls that are painted in pink and blue(Easter egg). The competition scoring rules for the first three qualifying rounds are as follows:

Scoring- Qualifying rounds (1, 2, 3)

1. Entering new squares (part 1) +1
2. Correct critical path moves (part 2) +2
3. Lighting a correct light (red or green) +1
4. Repeating a path move (both parts) -1
5. Touching wall (both parts) -1 max penalty per square
6. Moving off critical path (part 2) -1
7. Easter egg – numerical +1
8. Easter egg – alphabetical +2
9. Easter egg – other symbol no score except "Perfect Score Challenge +1"

Figure 12) Scoring for First Three Competition Rounds

For the final round the scoring the rules becomes as follows:

Scoring – Final

10.	Adjusted critical path (part 2) execution time – Winning criterion		
11.	Search path (part 1) execution time	2nd tiebreaker criterion	
12.	Entering new squares (part 1)	+1	
13.	Correct critical path moves (part 2)	+2	
14.	Lighting a correct light (red or green)	+1	
15.	Repeating a path move (both parts)	-1	
16.	Touching wall (both parts)	-1	max penalty per square
17.	Moving off critical path (part 2)	-1	
18.	BTB – Bonus time base	Fastest critical path time in finals	
19.	Easter egg – numerical	+1 plus 5% of BTB off critical path time	
20.	Easter egg – alphabetical	+2 plus 15% of BTB off critical path time	
21.	Easter egg – other symbol	no score	

Figure 13) Scoring for the Final Competition Round

From the highlighted points we can see how important the robot's ability to recognize characters on the maze walls. In the functional decomposition shown in Figure 14, the team considered the factors that will allow the robot to take and process pictures. For the robot to accomplish this, the right hardware and software must be chosen.

Character Recognition Functional Decomposition

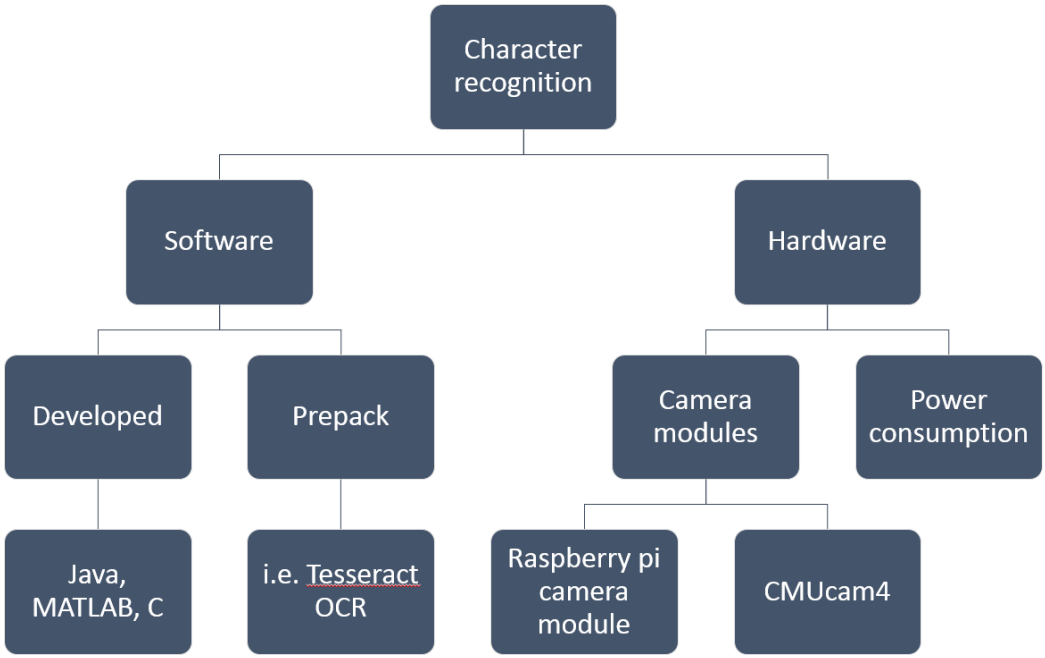


Figure 14) Functional Decomposition for Character Recognition

Hardware

The robot vision will depend on a camera to take a picture and store it for later processing to identify its content, to select a camera module the team took into consideration the following categories: size, price, power consumption, and the data transfer speed. After researching many camera modules, the team closely compared the CMUcam4, which is an Arduino based camera module, and the Raspberry Pi camera module. A regular digital camera was not considered, because it requires extra power and storage control circuitries and that will make it even more expensive than dedicated camera modules.

CMUcam4

The goal of the CMUcam4 project is to provide simple vision capabilities to small-embedded systems in the form of an intelligent sensor. The CMUcam4 is a fully

programmable embedded computer vision sensor. The main onboard processor is connected to an Omni Vision 9665 CMOS camera sensor module. It has a lot of features that makes taking pictures and processing it very interesting.

See Appendix A.12 for the specifications table of the CMUcam4.

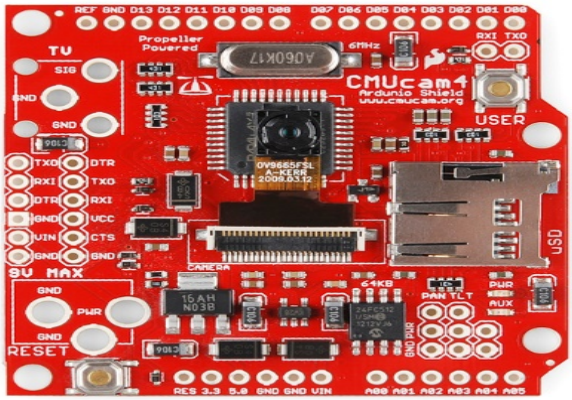


Figure 15) CMUcam4

The CMUcam4 is very powerful as it is capable of processing pictures and videos taking by, on board Omni Vision, camera sensor of high resolution. Also the two port servo controller allow for better quality pictures as it help in centering the camera to capture citrine scope. The only drawback of that camera module is the high cost and the low data transfer.

The Raspberry Pi camera module

The Raspberry Pi camera module is an Omni vision camera that can be connected directly to a Raspberry Pi through a dedicated CSI high baud rate bus. The camera module comes with many valuable features:

Raspberry Pi Camera Module Technical Specifications

Pros	Cons
Raspberry Pi compatible	No pan and tilt capability
Cost $25-$35	
High baud rate 4Gbps	
Power **200 mA** at **4.75V to 5V DC**.	
USB on Raspberry Pi Board	
Diminutions 25mm*20mm*9mm	

Table 9) Raspberry Pi Camera Module Technical Specifications

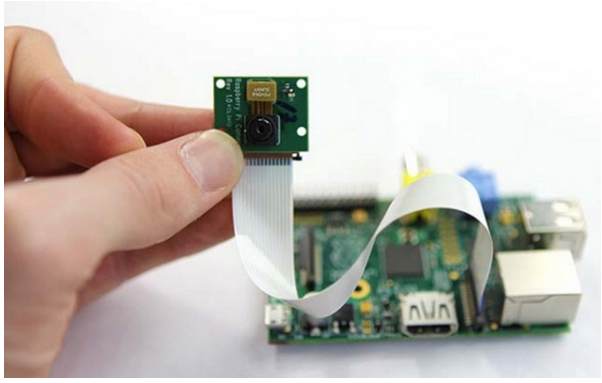


Figure 16) Raspberry Pi Camera Module

The Raspberry Pi Camera module is a better fit for our task as it is cheaper, faster, and smaller and it consumes less power than the CMUcam4, but it does not have a pan and tilt controller but we can easily implement one if we need it as the camera size will compensate for the pan and tilt feature. We looked at all of these differences then we have put the following comparison table and the decision matrix together.

Technical Comparison between Raspberry Pi Camera Module and CMUcam4

	Raspberry Pi camera module	CMUcam4
Size	25mm*20mm*9mm	52.25mm*53.50mm*13mm
Cost $	25-35	115-130
Connection	Serial Pin	CSI high bandwidth bus
Software interface	Arduino interface library	raspistill, raspistillyuv, raspivid
Power needed	200mA – 4.75V to 5V	250mA – 4V to 9V
Pan/tilt	No	Yes
µSD card	No	Yes

Table 10) Technical Comparison of Camera Hardware

Pairwise Comparison Matrix for Camera Hardware

	Size	Cost	Power needed	Data transfer	weights
Size	1	3	3	1/7	0.2
Cost	1/3	1	3	1/5	0.12
Power needed	1/3	1/3	1	1/3	0.08
Data transfer	7	5	3	1	0.6

Table 11) Pairwise Comparison Matrix of Camera Hardware

Decision Matrix for Camera Hardware

			Raspberry Pi camera module	CMUcam4
Size	0.2	-	0.3	0.7
Cost	0.12	-	0.18	0.82
Power needed	0.08	-	0.3	0.7
Data transfer	0.6	+	2.4	0.00015
Score			0.57	0.43

Table 12) Decision Matrix for Camera Hardware

In conclusion the Raspberry Pi camera Module is the best choice for our design because it is small and it has a very high data transfer rate that will allow us to process more images in shorter time.

Related Competition rules analysis

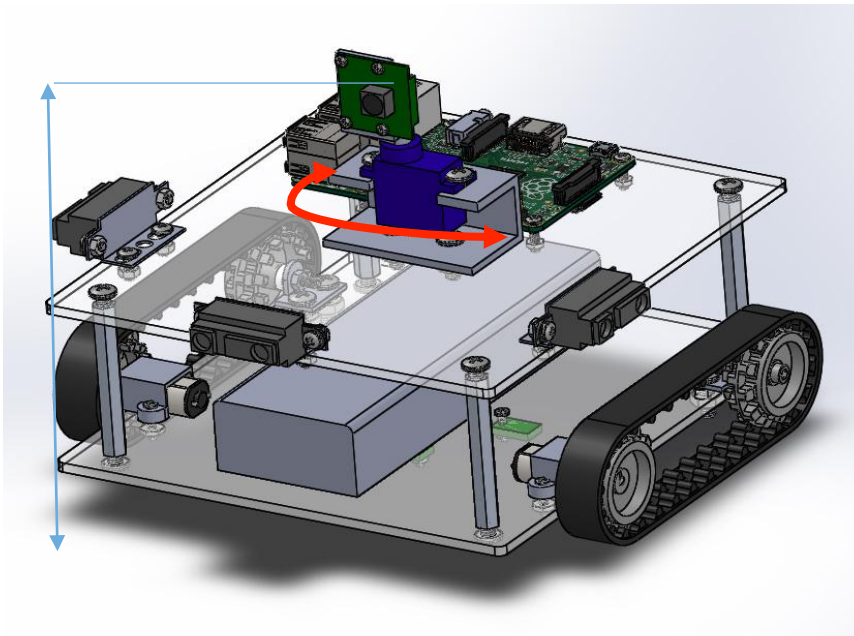
From the following pictures we can see that the maze walls appears in three colors blue, green, and pink. These colors have meaning as the blue and pink walls we have numbers, letters, or special characters. These are going to be numbers from 0 to 9, letters from A to Z, and special characters {!,@,#,$,%,^,&,*,(,) }.These will appear in the 6'x6' and the 7'x7' mazes in collection of 20 mixed characters, 8 numbers, 2 letters, and 10 special characters all called Easter eggs.

Figure 17) Example of maze walls

From the picture we see the character is 1" hight and 1" from the top of the maze wall that is 7.5" hight from the maze floor, meaning the charactyers are centered on the wall at 6.5" hight from the maze floor.

Related engineering analysis

The camera module has to be at 6” hight from the maze floor and it has to be allowed from maximum visability and centered on the fobot. All of these aspectes leed to the following desigen as we have a to tier platform that allows for larger space for our component and fanout. Also according to the competition rules the robot height of 6” inches is within the height constraints.



6”

Figure 18) Camera position and rotation

The camera is mounted on a servo motor to allow for color detection using the software libirary raspivid, and wider capturing scope for the camera. The camera will take pictuers of the walls if the it detectes pink or blue color and the robot is centered and not moving. The feedback system between the camera and the microcontroller and the sensor controller will determine when the robot is on the center of the cell using the black grid lines and the reflectance sensoers, and if the camera detectes pink or blue color the robot will stop on the grid and the camera will move in scope of 360° to take pictures.

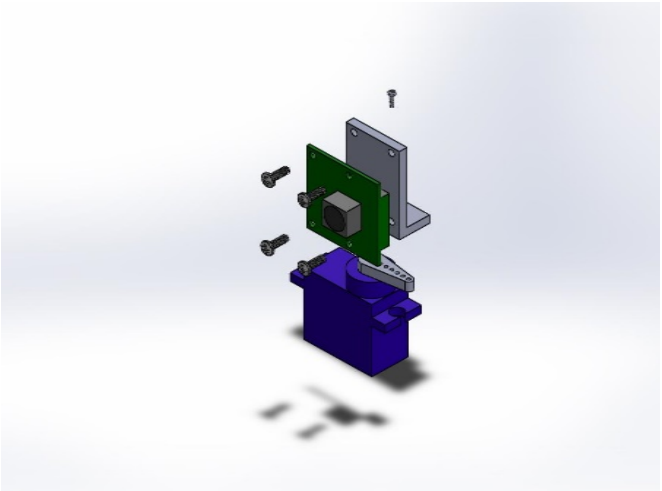


Figure 19) Camera Installation

Software

The robot needs to process the pictures it captures from the maze wall, to do that with high accuracy the robot has to not be moving while the camera takes a picture and the Optical Character Recognition (OCR) algorithm has to be accurate as well. From the previous engineering analysis we have solve the first problem of taking a picture while the robot is not moving by detecting if any pink and blue walls and the robot in the center.

There are many Optical Character Recognition software packages that are capable of subtracting a text of a given picture and save the results in a text file. Tesseract is probably the most accurate open source OCR engine available. Combined with

the Leptonica Image Processing Library it can read a wide variety of image formats and convert them to text in over 60 languages.

Tesseract works on Linux, Windows (with VC++ Express or Cygwin) and Mac OSX. It can also be compiled for other platforms, including Android and the iPhone, though these are not as well tested platforms. So it is a convenient tool if we use a Raspberry Pi board as it runs Linux by default. Tesseract is an open source software available on Google Drive, It has the ability of dealing with partial text and skewed pictures.

On the other hand, Python, JAVA, C++ or MATLAB can be used to develop an algorithm that allows the camera to take a picture when one of the Pink or blue walls has been detected. The algorithm then will store that picture in the Pi ram for processing, it will take the picture and divides it in pixels and then mark the pixels with Ones and zeros according to its shade difference then matching these patters with predefined patterns will allow the algorithm to identify the right characters.

According to the previous comparison between Python and C++ we have decided on using Python shell script to define our algorithm for character recognition. The robot has to learn the characters first in order for it to be able to identify each of these character looking at an example from Programming Computer Vision with Python by Jan Erik Solemn we see that we can use the same approach to design and implement an algorithm to identify each of the easter eggs correctly. The algorithm starts by training the classifier network.

Refer to Appendix A.4 for example of character recognition code.

Power

Battery Technology

To power the robot, a battery had to be chosen. While this seems fairly straightforward, there were actually a good number of decisions that had to be made. We started by looking at the various battery technologies available in the market and which would be the best fit for our robot.

Lead Acid

Lead acid batteries are an older technology, however still have a place in robotics. Lead acid batteries have respectable discharge rates, usually somewhere in the window of 10C-20C. This is lower than a few of the more modern technologies but is still very viable in today's tech. They also have decent capacities with the median capacity sitting around 4-10Ah. However they have major downsides as well. Lead acid batteries are extremely heavy and large, typically being larger than our entire robot would be and weighing on the upwards of 10-20 pounds.

Nickel Cadmium

NiCd batteries are also a fairly widespread option. Their high potential discharge rates combined with stable/flat discharge characteristic makes them an attractive choice for our robot. Many NiCd batteries are able to discharge at a rate up to 10C which is far more than enough current than we would need to run our motors. However, NiCd batteries are extremely susceptible to the "memory effect". This effect is caused by the growths of larger crystals in the cell formation of the battery when the NiCd is attempted to be charged without being completely drained first. Once the memory effect happens, the battery's capacity can effectively be destroyed and is very difficult to reverse.

Nickel Metal Hydride

Ni-MH batteries are a more recent technology (1989) that don't excel extremely well in most categories, yet don't fail in any areas either. NiMH batteries have a high energy density that are better than most technologies excluding Lithium battery types. Most NiMH batteries have charge cycles in the thousands, usually around 3000. They also don't have many issues being used in a deeply discharged state (~80%+). Their discharge rates are also relatively high, yet not as high as other types such as Lead Acid or NiCd. The NiMH battery does suffer from a minute memory effect, yet this is noticeably not nearly as bad as in the NiCd battery. They also deteriorate over time in storage rather quickly.

Lithium Ion

Lithium Ion batteries are the frontrunners in consumer electronics' batteries. They have solved the major issue of the memory effect prevalent with NiMH and NiCd batteries, as well as most of the weight and size issues. Lithium Ion batteries can be discharged at up to 40C or even higher in special optimized batteries, and can even have capacities on the upwards of 5-10Ah. They have extremely high energy densities as well, even managing to easily beat out Lead Acid batteries. Lithium Ion cells do have the issue of being slightly more expensive than most other battery types. They also are very susceptible to losing capacity when overcharged and degrading if cell voltage ever drops below 2V (nominally 3.7V).

Lithium Polymer

Lithium Polymer batteries are an offshoot of regular lithium ion batteries, sometimes referenced as “battery in a sack” due to the housing of LiPo cells usually being a bag of some sort. They are a more extreme version of Lithium Ion batteries. They are even lighter and smaller than Li-Ion, flexible due to the lack of metal/rigid housing, and have even higher discharge rates. However their energy density and life cycles are lower than Li-Ions. A major downside to Polymers are the fact that if charged improperly, they can be damaged or even explode.

Alkaline

Out of all the other types we looked at, this is the one non-rechargeable option of the group. Alkaline cells run at a higher cell voltage (1.5V) compared to most of the other rechargeable types that have a nominal voltage of 1.2V. That means we would need to use less alkaline cells to reach our needed voltage of 12V. However, alkaline batteries are not well suited to high current applications such as DC motors as they do not have a very high capacitance and cannot run for very long under such situations. Alkaline cells would be well suited to running our microprocessors and sensors however. Cost would become an issue as well once replacement costs are factored in over the course of our project.

Comparison of Battery Technology Specifications

Technology	Capacity	Discharge Rate	Cost	Downsides
Ni-Cd	2000-5000mAh	Up to 10C	$20-50	Memory Effect
NiMH	2000-10000mAh	Up to 2C	$20-50	Self-Discharging
Lithium Ion	5000-10000mAh	Up to 20-40C	$100+	Expensive
Alkaline	(D) ~10000mAh (9V) 500mAh	1C 0.1-0.5C	~$2.50/cell ~$5.00/cell	Low capacity, not rechargeable

Table 13) Comparison of Battery Technology Specifications

Single Vs Multiple Batteries

The team next looked at how it should organize the power system for the robot. One of the big decisions to make was whether to use a single battery or multiple batteries. By using multiple batteries one could make sure each subsystem or module could have its own power source. This would make it easy to match output voltages of batteries to exactly what each component needed. However in doing this, it would quickly increase the cost along with weighing the robot down more and increasing complexity in organizing the wiring. With one battery for the whole robot, the issue arises of running high current DC motors alongside lower voltage and current electronic components like microprocessors and cameras.

Voltage Regulation

With the combination of motors and electronic devices on our robot, it is necessary to create a way to distribute the correct voltage level to the correct modules. Looking at all of the components on the robot, the DC motors run on 6V while all the electronics such as the sensors and controllers run on 5V. This provided an easy solution

of using voltage regulators to create 2 separate rails, one at 5V and another at 6V to power the whole robot from a single source.

After comparing the many different options available, the team ultimately decided on using 2 Hobbywing UBEC regulators. These are preassembled circuits containing a voltage regulator IC, filter capacitors, and a selectable voltage output switch. This specific model has a 3A throughput, and a selectable 5V/6V output.

By using these voltage regulators, two separate supply rails are created, isolating higher current and noisier motor power from the more sensitive control circuits.

Low Voltage Warning

The team determined that it would need an advance notice of when the robot's battery is about to run out of power. If the components are sent lower than rated voltages, there is a risk of damaging the electronics. The design is very simple and straightforward: Connecting 2 LEDs in parallel with an active zener diode and a resistive voltage divider. A threshold voltage is selected, based on what voltage would trigger the warning, and appropriate resistor values are determined in order for that threshold voltage to correspond with 2.5V across the active zener diode. Once 2.5V is hit, the zener will swap the flow of current into the LEDs, thus indicating that we need to charge the battery.

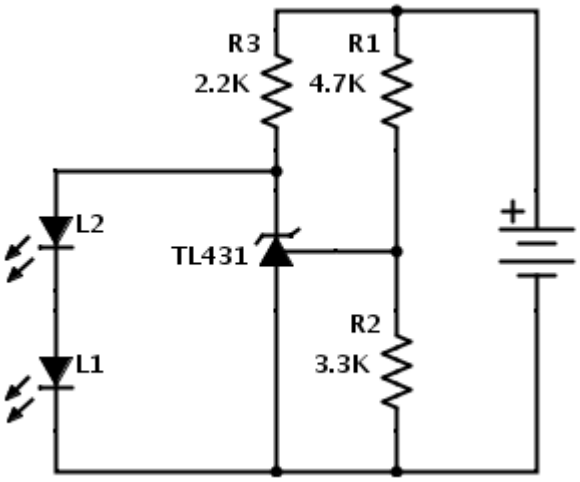


Figure 20) Low Voltage Detector Circuit

Refined and Expanded Engineering Analysis

Motor

To drive the robot, a motor with power and speed will be needed, but with so many motors on the market, it is hard to choose one that suits the robot's needs. To get an idea of what is requuired, a simple engineering analysis was done. Using simple realistic assumptions, a Free Body Diagram (FBD) of a cog with the tread overcoming a speed bump was used since this is where the robot will need the most power. By making a scale model, it was found that the cog wheel and speed bump make contact at a tangent line of 55°. Assuming a weight of 5 lbs, the following equation was derived:

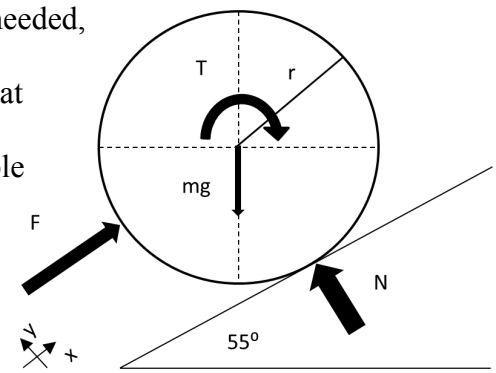


Figure 21) Free Body Diagram of Cog

$$\Sigma F_x = \frac{T}{r} - mg * \sin(55)$$
$$T = mg * \sin(55) * \frac{r}{N}$$
$$T = 25.229 \text{ oz-in}$$

N=number of motors

Where T is the torque, m is the mass, g is acceleration due to gravity, and r is the radius. At the time of writing, friction was not represented in this analysis, however, it shall be rectified in the future.

Field Analysis

The next step in the motion analysis was to analyze the playing field itself to define a boundary on speed. Since the maze walls will be randomized there is no way to predict the layout so we developed a few scenarios to estimate the time it would take to complete the maze.

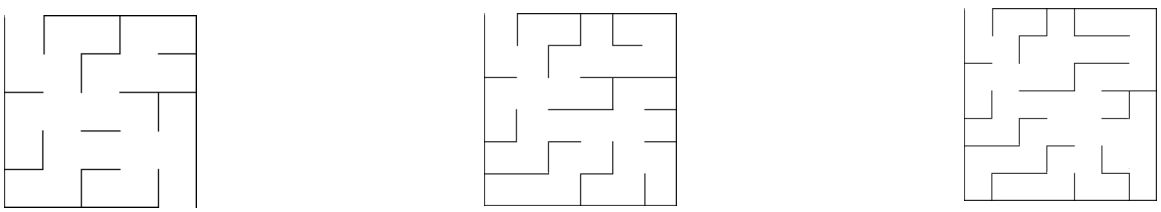


Figure 22) Sample Mazes

Each of these squares represent a sample maze for each size; 5'x5', 6'x6', 7'x7'. As mentioned earlier, each maze a certain time limit that must be met.

For the 5'x5' maze, there is a 3 minute time limit. In order to travel the entire maze, it must visit each square twice. In theory, the robot will travel a total length of approximately 50 ft not including turning. With this in mind, in order to complete the maze in the given time limit the robot will have to travel .278 ft/s. Taking the inverse of this value, one can compute the amount of time it would take to travel 1ft or 1 square. This value for the 5'x5' maze is approximately 3.6 seconds per square. To find the speed of the robot, simply use the angular velocity equation:

$$V = \omega r$$

Where ω is the angular velocity and r is the cog radius with the track. Doing the math this comes out to be approx. 1.34 ft/s or .74 sec/square. This would allow adequate time for turning and character recognition.

Using the same method for the other mazes, the following table was made:

Maze Size	Calculated Speed for Completion	Seconds per Square Allowed	Seconds per Square for turning and character recog.
5'x5'	.278	3.6	2.86
6'x6'	.3	3.33	2.59
7'x7'	.326	3.06	2.32

Table 14) Table of Speeds and Time for Maze Completion

Character Recognition Algorithm

The inputs are a segmentation of the greyscale image where "1" means filled and a "-1" means white. Images of each of the characters are taken by the camera and then filtered into a matrix. The Pseudoinverse of the matrix is taken to be:

$$P^{+} = (P^{T}P)^{-1}P^{T}$$

Where P is a matrix containing the inputs, the superscript, T, means the transpose of a matrix, and the superscript, -1, invokes the inverse of a matrix.

Because we want to get the original vectors back as an output, the "target" of our algorithm is just the input vector.

So in order to create the weight matrix, which will be what gives us the output, we use the formula:

$$Weight\ Matrix\ (W) = Target(T) * Pseudoinverse(P^{+})$$

Where the target is the original input matrix:

$$W = PP^{+}$$

Once we have a weight matrix that has learned all of the character inputs, the team can gather our output by multiplying the input the robot gets from the camera and the weight matrix:

$$Output = WP_{new}$$

Even if P_{new} is not exactly the same as a previous learned input, the algorithm will output the learned input that is the closest match. This allows the robot to recognize images that are slightly skewed compared to the ones it has learned before.

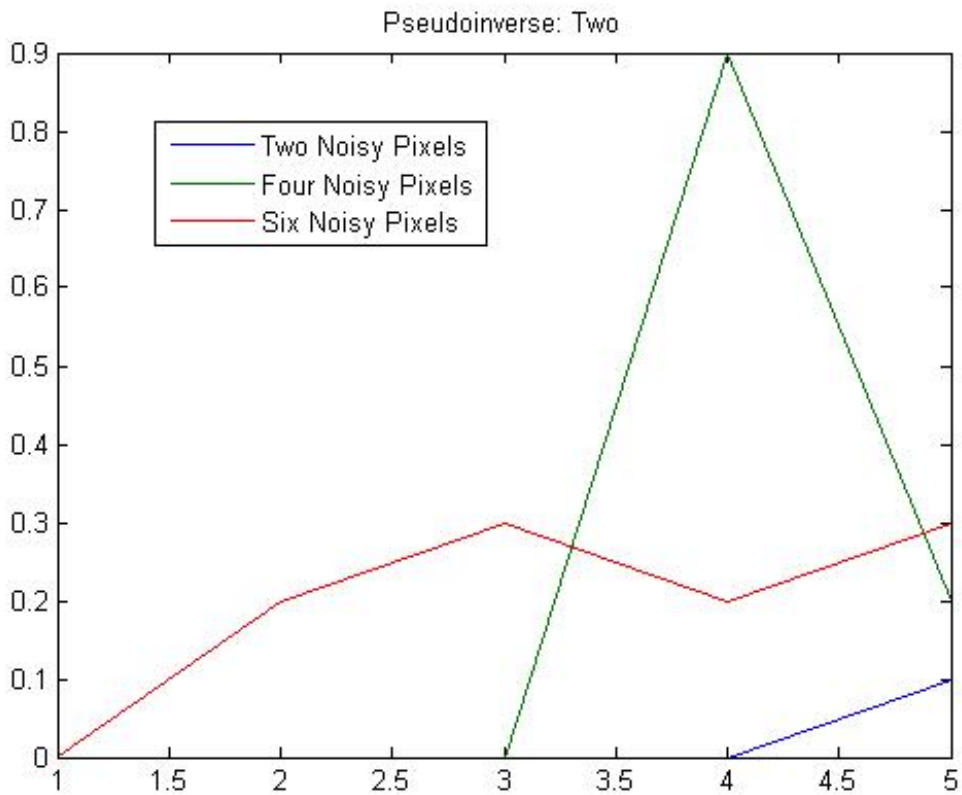


Figure 23) Pseudoinverse algorithm

This figure looks at how well the Pseudoinverse algorithm was able to recognize the number 2. The noisy pixels are how varying the number two shown the algorithm differed from the number stored. This algorithm used 30 segments or "pixels" to test the algorithm. It shows percentage of failures over ten runs. Even with 6 noisy "pixels" or 1/5 of the number skewed, the algorithm was able to detect the correct number over 50% of the time. There is one outlier where the algorithm only was able to predict the number 10% of the time. This outlier was due to the random nature of the randomized pixels. That randomization made the number presented to the algorithm similar to another number that was learned by the neural network. In order to ensure success, we must position the robot in such a way that its view of the characters remains the same to ensure

that the algorithm receives inputs that vary minimally. Even if this is not possible, the algorithm still has a chance to recognize the character.

The team believes that it is safe to expect an average of over 80% character recognition from this algorithm.

Battery Lifetime

In order to power the robot, the team decided to go with a 7.2V NiMH 3800mAh battery. The nominal voltage of the battery is very close to the component's voltage needs.

The approximate runtime of the robot on a single charge was calculated to see if it would be able to compete through the whole competition on a single charge. Using the worst case scenario current loads for each component we can do some simple math to determine an approximate runtime:

DC Motors (2x): 3200mA

Servo Motor: 175mA

Raspberry Pi: 500mA

Raspberry Pi Camera: 250mA

Sensors: 100mA

TOTAL: 4250mA

Battery Lifetime = Current Load / Battery Rating

3800mA/4220mAh = .90 Hours

Final Engineering Specifications

	Specifications Desired	Specifications Met
Battery Life	Over 0.5 hours	0.90 hours
Speed	Over 0.82 ft/s	1.344 ft/s
Character Recognition	80%	Over 85% on average
Wall Detection Range	0.5 ft	0.131 ft to 0.984 ft
Motor Torque	0ver 25 oz-in	40 oz-in
Size	8”x8”x7.5”	6.5”x6”x6”

Table 15) Final Engineering Specifications

The battery lifetime specification is based on how long the robot must be able to operate without needing a charge. Based on the competition’s number of rounds and time for each round, the robot must be able to run over 30 minutes without needing a charge. Based on the worst-case scenario of power consumption and the battery the team has chosen, the robot’s total lifetime is around 0.93 hours.

The speed is based on a worst-case scenario of the maze run. The analysis was done to decide what our max speed would need to be in order to ensure that the robot can successfully run the maze in the time allotted.

The character recognition percentages are based on Pseudoinverse calculations. The Pseudoinverse algorithm can get an average 80% character recognition from obstructed or imperfect character images. With perfect images, the camera can get very close to 100% recognition. 80% character recognition is a good target, because the competition rules place a premium on the total speed of maze completion versus number of characters recognized.

The wall detection range is based on the distance between the center of the maze tiles and the walls around the tiles. 0.5 feet is the distance between the center of the maze tile and the walls. This number represents the max number we would need to detect a wall to our sides even if the robot was imperfectly aligned with the tile center. The proximity sensor we chose has a range of up to 0.984 feet almost double the needed distance. This provides ample room for alignment error.

The motor torque is based on the torque needed to successfully clear the speed bumps present in the 7x7 maze, which create a 55-degree angle between the speed bump and the robot's treads. The motors The Team has chosen provide 40 oz-inch torque which is far over the 25 oz-inch calculated that the robot would need. This provides ample room for discrepancies between the reported and actual sizes of the speed bumps.

The size desired for the robot is specified by the competition rules. They also recommend the robot to be less than 7"x7"x7". Our design has the robot coming in at 6.5"x6"x6" which is under the recommended value of the competition. This design should provide enough clearance to maintain turns that avoids collision with the maze walls.

System Description and Product Architecture

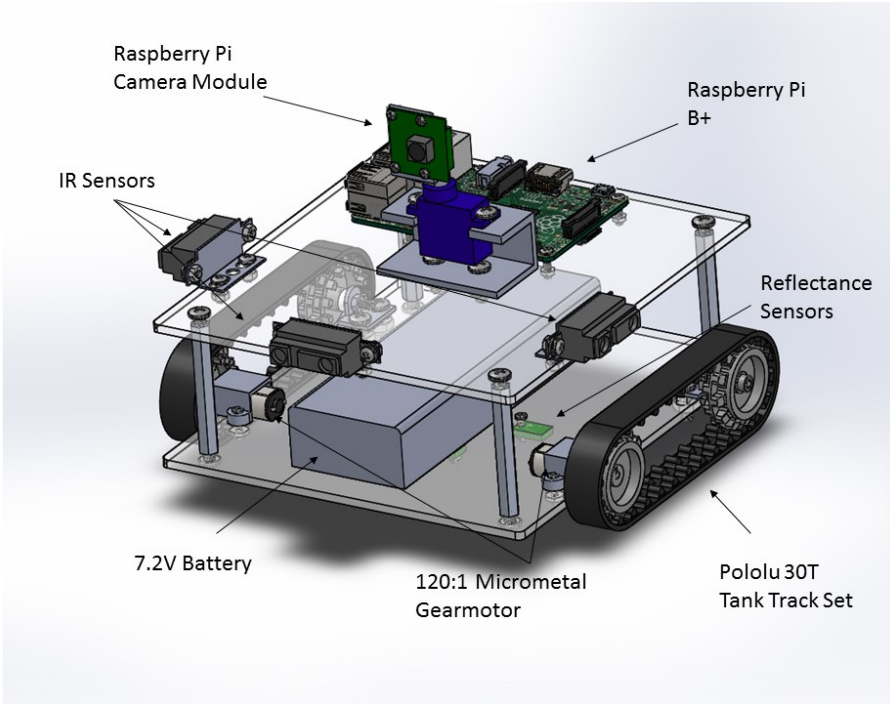


Figure 24) System Description

The reflectance sensors placed on the bottom of the robot will provide positional feedback to allow our robot to know where it is located on the playing field while the IR sensors placed on the front and sides of the robot will scan the surroundings to determine the proximity to the walls. These produce analog signals which will travel to the sensor board (not shown) which will convert these to digital signals and sent to the controller. In each square, the camera will rotate atop the servo motor and scan the walls searching for either the blue or pink colored wall. If there is a colored wall, it will take a picture that will be processed and stored in a USB drive.

After detecting its location and waiting for the camera to take pictures, it will determine the next best course of action and send signals to the two motors to travel to the next square. While it is in motion, the reflectance sensors will keep the robot on the

cross lines if it becomes misaligned. In the case that it does the sensors will send signals to the controller which will in turn control the motors to realign the robot. Every component will be powered using the 7.2V battery.

Algorithm

1. Create an unfilled matrix for 5'x5', 6'x6', or 7'x7' maze.
    a. Mark all squares around the grid as not part of the maze.
2. Indicate on matrix where the predetermined start and finish squares are.
3. Create a two-direction path from start to finish (up and left, up and right, down and right, or down and left) depending on where the finishing square is to the final square.
4. Proceed to move to the next square on the two-direction path.
5. Analyze walls around the robot.
    a. If the next square is blocked:
        i. Move to the second direction on the two-direction path.
    b. If both directions are blocked:
        i. Retrace steps until one of the two directions are not blocked and mark the path blocked.
    c. If the walls around the robot are colored blue or pink:
        i. Take an image of the wall and analyze it.
    d. If the next square is not blocked:
        i. Move to said square and mark square as part of the critical path.
    e. If the square is not blocked, but you have to back track
        i. Mark the square as not blocked, but not part of the critical path
    f. If the square you have reached is the ending square:

        i. Mark that you have finished the maze and wait for input.

6. Update the square visited as blocked, part of the critical path, not part of the critical path, or not part of the maze.
7. Repeat steps 4-6 until you have reached the final square
    a. Once you have reached the final square and have received the input from the judge, move through the maze only to squares you have marked as part of the critical path until you have reached the starting square.

See Appendix A.13 for an in depth analysis of maze solving algorithms.

Material Selection

Some materials that will be discussed in this report include the material for the treads and the base for the robot. These selections will be important as a wrong decision can cause the robot to fail or not work as efficiently as possible. Materials for other parts were not considered since these will be ordered from manufacturers.

Chassis

The chassis material must be strong enough to hold the weight of all the components of the robot and also be lightweight to reduce the strain on the motors. When researching, it was found that wood, aluminum, and acrylic plastic were the most widely used materials in making a robot chassis.

Aluminum was the first material considered as it is the strongest. Aluminum 6061 is the most common form of aluminum found in the US. With an ultimate yield strength of 45000 psi it easily outperforms the strength required for our design. The price of aluminum is $15.72 for a 1/8”x12”x12” sheet from *onlinemetals.com*. Since this is a metal, it will weigh a bit more than the other materials. With a density of .0975 lb/in^3 it is one of the lightest metals but compared to the other options, it is still the heaviest. The other con is that it will conduct electricity. If the chassis comes into contact with any electrical circuit board, it will short out the board and leave our robot useless and possibly present a safety issue.

The second material researched was wood. This material is both lighter and cheaper than aluminum and comes in a variety of sizes and styles. It can easily be cut and shaped into what is required of the robot. It also does not conduct electricity which means electrical components can be secured directly to the chassis. Since the plywood in stores comes in layers, our design would have to change to make the chassis thicker to have a higher strength. While in most cases this would be a sound design choice, there is a size limitation that must be met.

The final material researched for the chassis design is acrylic plastic. With a yield tensile strength of 10000 psi, it can easily hold all the components without failing. It can be easily worked and cut to fit the robot design and is relatively cheap priced at $5.64 for a 1/8”x12”x12” sheet. Like the wood, all electrical components can be fastened directly to it since the material does not conduct electricity. With a specific gravity of 1.2, it is also lightweight and won’t strain the motors much.

Comparing all the materials together, it is easy to see that the acrylic plastic is the best choice for the robot due to its light weight, cheap price, and strength.

Idler Bracket

The idler bracket must be strong enough to hold the weight of the robot and be strong enough to not fail while overcoming obstacles. Aluminum would best suit this piece as it is strong and light enough to handle most situations.

Brackets and Spacer

The brackets for the camera and servo motor will be made from the ABS plastic from the 3-D printer located on campus. These brackets will not hold much weight so plastic will be suitable and easy to manufacture. The spacer for the idler cog will also be made from plastic since it won't have any weight supported on it.

Manufacturing

Chassis

The chassis will consist of two 6.5"x6"x1/8" (HxLxW) sheets of acrylic. These sheets will come in 12"x12" sheets so they will need to be cut to size. After they are cut, one must simply drill the holes as specified on the design drawings.

Idler Bracket

To manufacture the idler bracket, a piece of aluminum must first be ordered. For the simplest machining, a rectangular tube would the easiest to work with. The first step would be to cut the tube to make two separate L-shaped pieces. Next simply cut these pieces to the designated length and drill the required mounting holes.

C-Bracket, L-Bracket, and Spacer

Since these pieces will be made from ABS plastic, a 3-D printer will be utilized to manufacture these pieces. Each of these components has been modeled using SolidWorks which then can be used in the 3-D printer. Using the 3-D printer ensures that each part will be accurate to a degree and will allow for easy fabrication. The design drawings can be located in the appendix.

See Appendix A.17 for detailed design drawings.

Assembly

Once all the components have been acquired, they must be assembled together to create a functioning robot. This section will detail the assembly process using simple steps and pictures.

Tier 1 Assembly

1. Secure the motors using the motor brackets and the motor bracket screws (included)
2. Secure the Idler bracket using two 2-56 threaded screws
3. Fit the driver cog onto the motor shaft
4. Using the idler spacer and washer and shoulder bolt from the track set, attach the idler cog to the idler bracket
    a. Ensure that the spacer and washer are between the cog and the bracket
5. Stretch the tread around the cogs and ensure that it is taunt

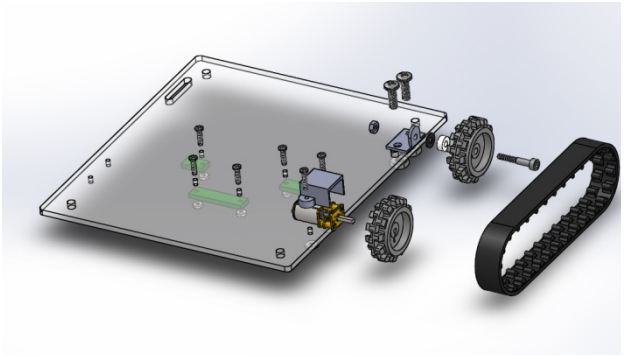


Figure 25) Tier 1 Assembly

6. Repeat steps 1-5 for the other side
7. Attach reflectance sensors to bottom of chassis using 2-56 thread screws

Tier 2 Assembly

1. Attach IR sensors to the IR sensor bracket using the screws provided
2. Secure IR brackets to the chassis using two M-3 thread screw
3. Repeat steps 1-2 for each IR sensor
4. Secure the servo motor bracket to the chassis using two 8-32 thread screws
5. Secure the servo motor onto the bracket using the screws provided
6. Secure the Raspberry Pi controller to the rear of the chassis with 4 M2.5 thread screws
7. Secure the Camera module onto the L-bracket using 4 M2 screws
8. Secure the camera module and bracket onto the servo horn using the screw provided

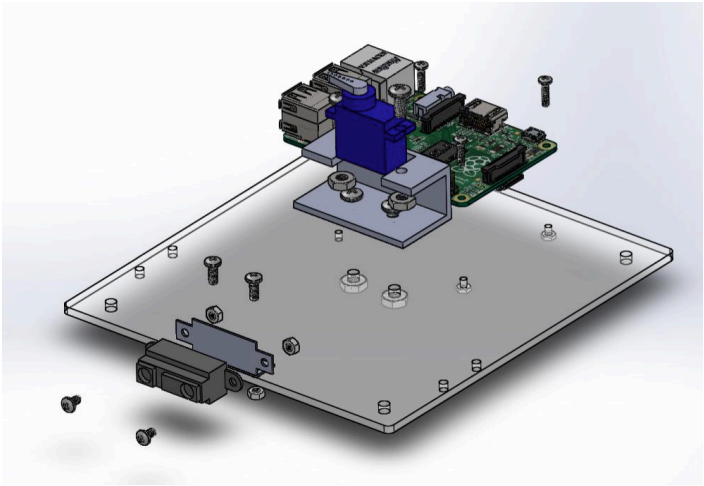


Figure 26) Tier 2 Assmebly

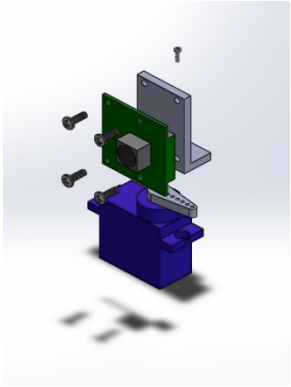


Figure 27) Camera Assembly

Final Assembly

1. Align each 2” standoff with the standoff holes located on each tier
2. Use 8 8-32 screws to secure the standoffs to each chassis tier

See Appendix A.15 for the wiring diagram.

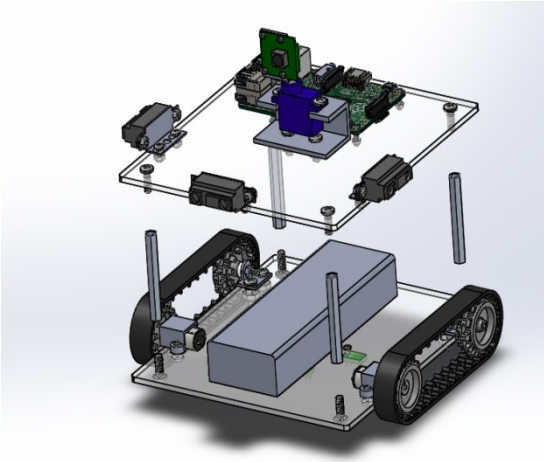


Figure 28) Final Assembly

Safety

There are some safety constraints that must be met in order to ensure that the robot is safe to operate. If the robot is deemed unsafe to operate by the judges, it will be disqualified.

The team’s design will require the use of many wires. These wires will be run from sources of varying voltages. In order to ensure that wires are not shorted, disconnected, or exposed, the wires should be tied down and secured. They also should be tied away from any moving parts and away from the sides in order to avoid clipping by outside edges. Exposed, disconnected, or shorted wires could lead to sparking, destruction of circuits, blown motor drivers, and burns from overheated components.

To further ensure that our robot is safe to operate, edges must be rounded and smoothed in order to ensure safe handling. Rough or sharp edges can penetrate the skin of the judges or the team as they handle the robot. This could lead the judges deciding that the robot is not safe to operate, leading to disqualification.

The other safety constraint depends on the turning radius and speed of the robot. During the competition, if the robot collides with a maze wall, points will be deducted from the final score. Furthermore, if the robot causes any damage to the maze itself, The Team will be disqualified. In order to ensure that the robot does not cause damage to the maze, The Team must establish a safe turning ratio in order to ensure that the robot will not unintentionally collide with the maze.

The team has also developed a low voltage warning system mentioned in the Power section. The system will ensure that the voltage supply does not go below a certain threshold. Going below this threshold without warning would cause the battery to discharge too far rendering it unchangeable and useless. This could also lead to the robot running out of power at an unintended time. Powering down prematurely could lead to the robot damaging itself.

Testing

The Team has a plan for testing the robot which begins with testing individual components. Then The Team will move to begin integration testing between components and the whole. Finally, The Team will begin testing the robot functionality as a whole in regards to completion of competition rounds.

A general testing outline is posted below:

- Motors
    - Test for individual motor output
    - Test for max speed with 2 motors installed
    - Test for turning clearance and angular velocity
    - Test for overcoming maze obstacles (speed bumps)
- Battery Runtime Test
    - Connect Battery to all motors and have them run at full speed.
    - Measure with Voltmeter and time to see how long till depletion
- Voltage Regulation Test
    - Connect both regulators to battery
    - Add devices to voltage buses one at a time while measuring voltage level of the buses to ensure stability under heavy load.
- Sensors
    - Calibration of microcontroller to output proper signal to Raspberry Pi
    - Test microcontroller outputs with dummy loads (LED indicators)
    - Software test of whether Raspberry Pi is reading the microcontroller data
    - Testing for Character Recognition
        - Camera-Pi connection working correctly.

  - Camera can take pictures and store them with relative square number.
  - Camera can record video and stored correctly.
  - Camera responses to the robot position by taking pictures if pink or blue walls.
  - System can recall a stored picture and process it for potential character identification.
  - Existence and correct configuration of the output file
- Testing for Maze Solving of 5x5 Grid
- Testing for Maze Solving of 6x6 Grid
- Testing for Maze Solving of 7x7 Grid

See Appendix A.11 for detailed testing plan for Sensor Calibration.

See Appendix A.16 for a more detailed character recognition testing plan.

Project Management

Budget

Part	Price
Raspberry Pi B+	$37.49
Camera Module	$27.21
Sensors	$68.63
Test Maze	$230.36
Sensor Board	$33.38
Chassis/Movement	$69.33
Power	$63.00
Total	$529.40

Table 16) Preliminary Budget

See Appendix A.14 for the full Bill of Materials.

Schedule

Phase One

Date:	Milestone:
August	Assigned Project
September 18th	Needs and Objectives Statement Written
September 23rd	Functional Decomposition Finished and Roles Delegated
October 1st	Designed Proximity Sensor Layout
October 5th	Designed Reflectance Sensor Layout
October 7th	Treads Chosen as Mobility Platform
October 8th	NiMh Chosen as Battery Type
October 9th	PiCamera Module Chosen as Vision Apparatus
October 10th	Raspberry Pi B+ Chosen as Controller
October 17th	Mid-Term Oral Presentation Given
October 20th	Conceptual Design Report Turned In
October 30th	PiCamera Library Chosen as Camera Manipulation Package
November 1st	OpenCV Chosen as Vision Software Package
November 2nd	RPi.GPIO Chosen as GPIO Manipulation Package

November 7th	Designed Sensor PCB
November 14th	Python Chosen as Programming Language
November 28th	Robot Model Completed in SolidWorks
December 3rd	Final Oral Presentation Given
December 3rd – December 10th	Final Presentation Written

Table 17) Schedule: Phase One

Phase Two

Date:	Milestone:
December 7th – February 13th	Algorithm Written
January 14th – January 16th	Parts Ordered
January 19th – February 13th	Robot and Test Maze Manufactured
February 16th – March 2nd	• Testing for Robotic Movement and Sensor Recogntion • Testing for Character Recognition • Testing for Low Voltage Warning Circuit
March 2nd – March 16th	Testing for Maze Solving of 5x5 Grid
March 16th – March 23rd	Testing for Maze Solving of 6x6 Grid
March 23rd – March 30th	Testing for Maze Solving of 7x7 Grid
March 30th – April 16th	Tweaking of Algorithm and Robot Design for Maximization of Competition Score
April 17th	IEEE R5 Robotics Competition in New Orleans, LA

Table 18) Schedule: Phase Two

Milestones

August

- Assigned project

September

- Functional Decomposition
- Obtained competition rules and constraints

October

- Torque Calculations
- Mobility Platform chosen
- Camera module selected
- Mid-term presentation
- Designed proximity sensor layout
- Designed reflectance sensor layout

November

- Track kit selected
- Motor selected
- Chassis design
- Assembled view
- Character recognition software chosen
- Designed the sensor PCB

December

- Fall final report edited
- Final presentation
- Test of the maze and camera software

January

- Assembly of robot
- Playing filed construction

February/ March

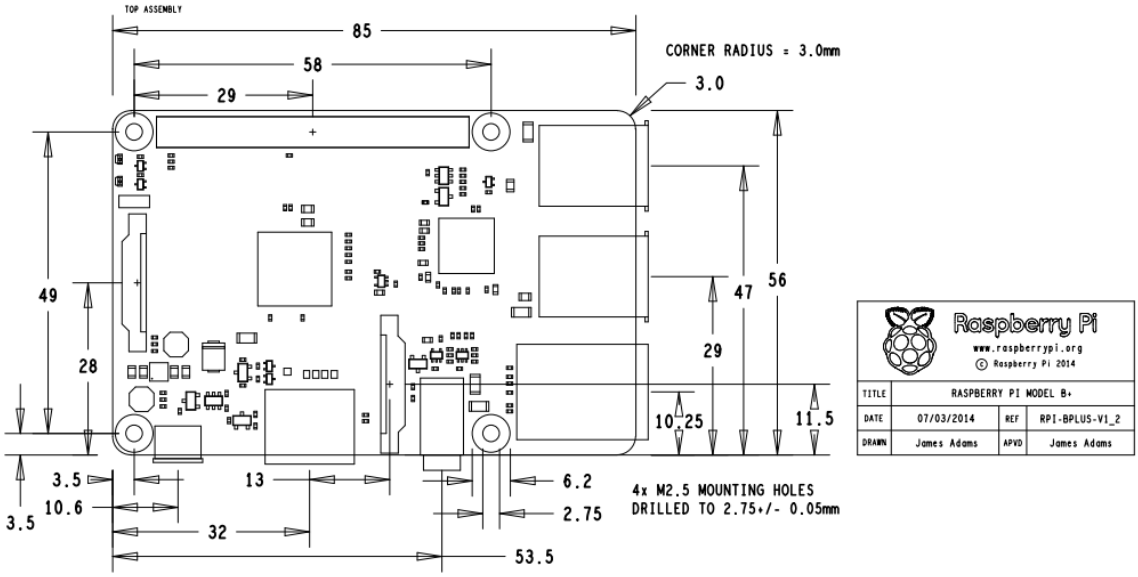
- Testing of robot different functional modules
- Enhancing the run time under up normal conditions

April

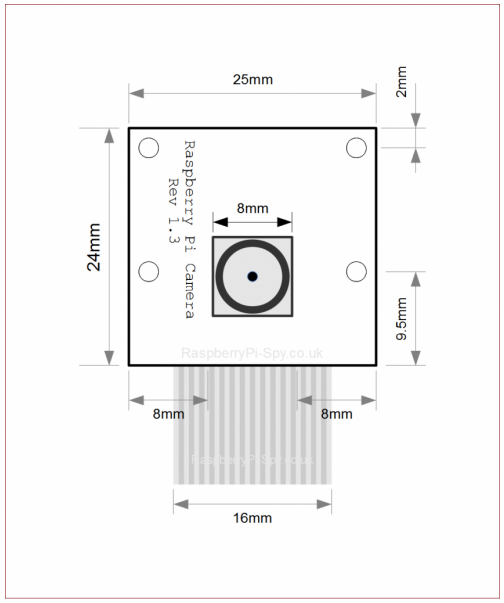
- Competition
- Final presentation
- Final project report

Appendix

A.1: Raspberry Pi B+ Data Sheet



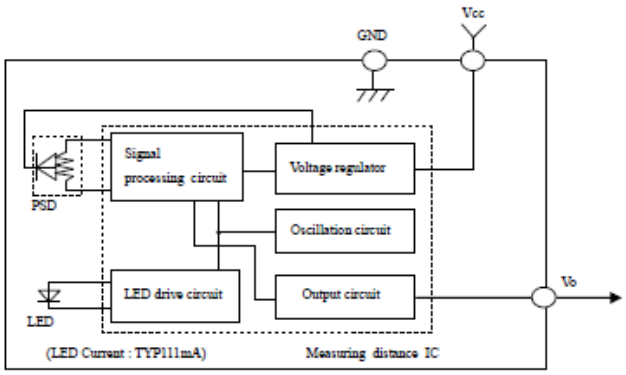
A.2: Raspberry Pi Camera Module Data Sheet



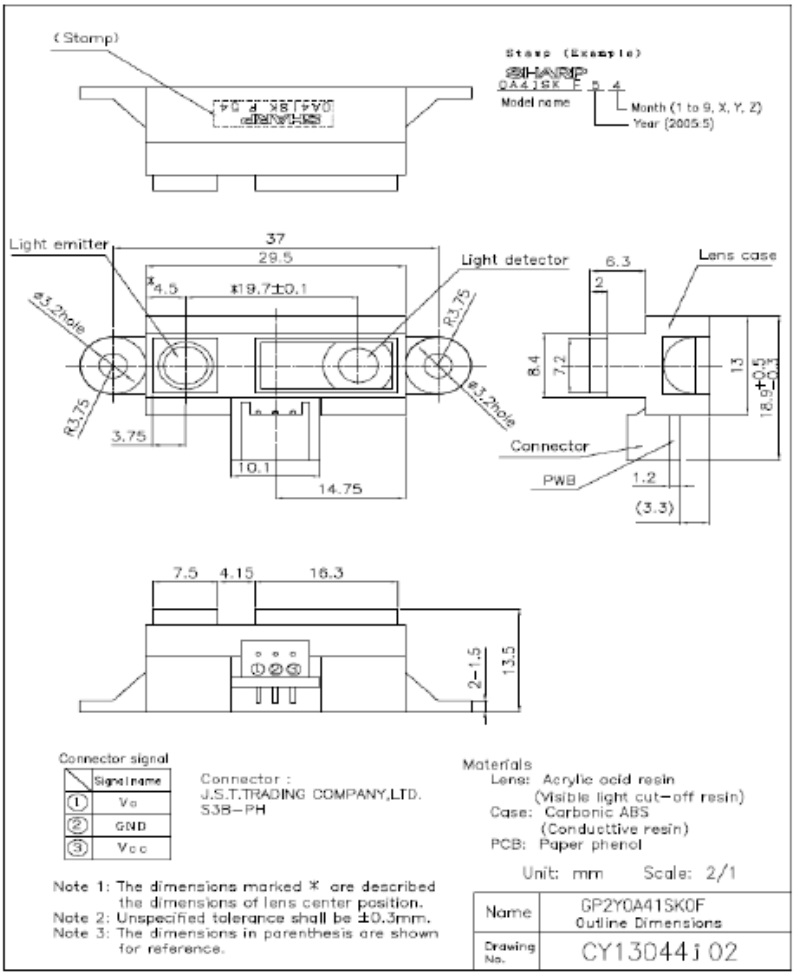
A.3: SHARP GP2Y0A41SK0F IR Proximity Sensor Data Sheet

SHARP GP2Y0A41SK0F

■**Schematic**



■**Outline**



http://www.robot-electronics.co.uk/datasheets/gp2y0a41sk_e.pdf

A.4: Example Code for Character Recognition in Python

Training a classifier

For this classification problem we have ten classes, the numbers 0 ... 9, so that our class labels are 0 ... 9. To train this ten-class classifier, we will use a dataset of images of cropped Sudoku cells. First we going to use a set of images for training. The images are named with the first character equal to the class (0 ... 9). The images are grayscale and roughly 80 ⇥ 80 pixels (with some variation). The following function takes an image and returns a feature vector of the flattened grayscale values.

```
def compute_feature(im):
""" Returns a feature vector for an
ocr image patch. """
# resize and remove border
norm_im = imresize(im,(30,30))
norm_im = norm_im[3:-3,3:-3]
return norm_im.flatten()
```

This function uses the resizing function imresize() from imtools to reduce the length of the feature vector. The picture also will be croped away about 10% border pixels since the crops often get parts of the grid lines on the edges.

Now we can read the training data using a function like this.

```
def load_ocr_data(path):
""" Return labels and ocr features for all images
in path. """
# create list of all files ending in .jpg
```

```
imlist = [os.path.join(path,f) for f in os.listdir(path) if
f.endswith('.jpg')]
# create labels
labels = [int(imfile.split('/')[-1][0]) for imfile in imlist]
# create features from the images
features = []
for imname in imlist:
im = array(Image.open(imname).convert('L'))
features.append(compute_feature(im))
return array(features),labels
```

The labels are extracted as the first character of the filename of each of the JPEG files and stored in the labels list as integers. The feature vectors are computed using the function above and stored in an array. With the training data in place, we are ready to learn a classifier. Here we'll use a multi-class support vector machine.

```
from svmutil import *
# TRAINING DATA
features,labels = load_ocr_data('training/')
# TESTING DATA
test_features,test_labels = load_ocr_data('testing/')
# train a linear SVM classifier
features = map(list,features)
test_features = map(list,test_features)
prob = svm_problem(labels,features)
param = svm_parameter('-t 0')
m = svm_train(prob,param)
# how did the training do?
res = svm_predict(labels,features,m)
# how does it perform on the test set?
```

```
res = svm_predict(test_labels,test_features,m)
```

This trains a linear SVM classifier and tests the performance on the unseen images in the test set. You should get the following printout from the last two svm_predict () calls.

Accuracy = 100% (1409/1409) (classification)

Accuracy = 99.2979% (990/997) (classification)

Extracting cells and recognizing characters

With a classifier that recognizes cell contents, the next step is to automatically find the cells. Once we solve that, we can crop them and pass the crops to the classifier. Let's for now assume that the image of the Sudoku is aligned so that the horizontal and vertical lines of the grid are parallel to the image sides (like the left image of Figure 8.8). Under these conditions, we can threshold the image and sum up the pixel values horizontally and vertically. Since the edges will have values of one and the other parts values of zeros, this should give strong response at the edges and tells us where to crop. The following function takes a grayscale image and a direction and returns the ten edges for that direction.

```
from scipy.ndimage import measurements
def find_sudoku_edges(im,axis=0):
""" Finds the cell edges for an aligned sudoku image. """
# threshold and sum rows and columns
trim = 1*(im<128)
s = trim.sum(axis=axis)
```

```
# find center of strongest lines
s_labels,s_nbr = measurements.label(s>(0.5*max(s)))
m = measurements.center_of_mass(s,s_labels,range(1,s_nbr+1))
x = [int(x[0]) for x in m]
# if only the strong lines are detected add lines in between
if len(x)==4:
dx = diff(x)
x = [x[0],x[0]+dx[0]/3,x[0]+2*dx[0]/3,
x[1],x[1]+dx[1]/3,x[1]+2*dx[1]/3,
x[2],x[2]+dx[2]/3,x[2]+2*dx[2]/3,x[3]]
if len(x)==10:
return x
else:
raise RuntimeError('Edges not detected.')
```

This is the code used to populate a matrix of the identified characters.

```
imname = 'sudokus/sudoku18.jpg'
vername = 'sudokus/sudoku18.sud'
 im = array(Image.open(imname).convert('L'))
# find the cell edges
x = find_sudoku_edges(im,axis=0)
y = find_sudoku_edges(im,axis=1)
# crop cells and classify
crops = []
for col in range(9):
for row in range(9):
crop = im[y[col]:y[col+1],x[row]:x[row+1]]
```

```
crops.append(compute_feature(crop))
res = svm_predict(loadtxt(vername),map(list,crops),m)[0]
res_im = array(res).reshape(9,9)
print 'Result:'
print res_im
```

The results is as follows:

Accuracy = 100% (81/81) (classification)

Result:

[[0. 0. 0. 0. 1. 7. 0. 5. 0.]

[9. 0. 3. 0. 0. 5. 2. 0. 7.]

[0. 0. 0. 0. 0. 0. 4. 0. 0.]

[0. 1. 6. 0. 0. 4. 0. 0. 2.]

[0. 0. 0. 8. 0. 1. 0. 0. 0.]

[8. 0. 0. 5. 0. 0. 6. 4. 0.]

[0. 0. 9. 0. 0. 0. 0. 0. 0.]

[7. 0. 2. 1. 0. 0. 8. 0. 9.]

[0. 5. 0. 2. 3. 0. 0. 0. 0.]]

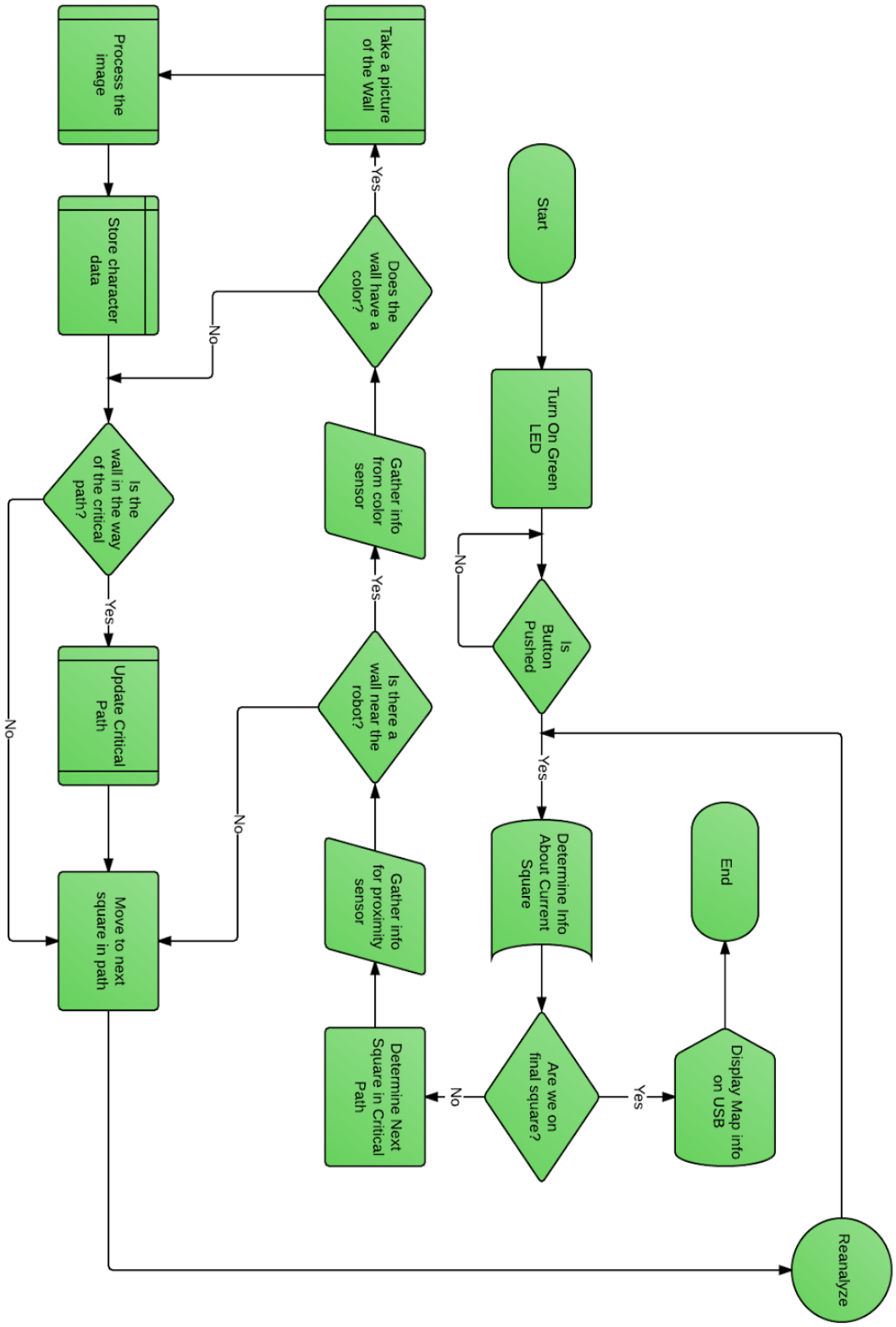
A.5: Propeller C3 Specifications Table

Module:	Propeller C3
Inputs:	Power: 7.5-12 VDC, 300+ mA VGA, RCA 2x16 Header
Outputs:	VGA, RCA 4 three wire servo header 2 12-bit A/D Converters 2x16 Header
Flash Memory:	1 MB
RAM:	32 KB x 8 cogs = 256 KB
ROM:	32 KB x 8 cogs = 256 KB
Clock Speed:	80 MHz
Dimensions:	75.9mm x 55.2mm
Price:	$89.99

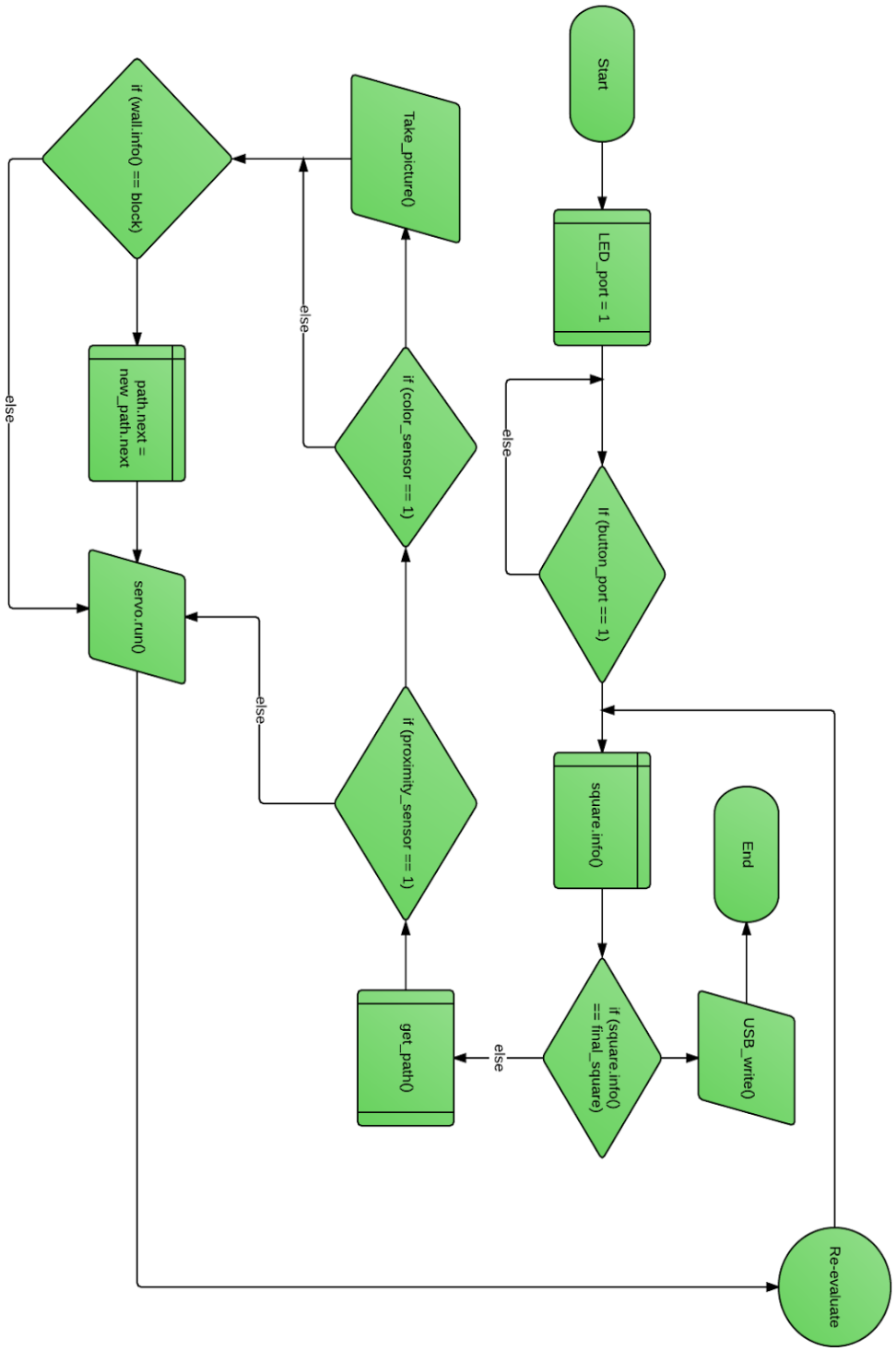
A.6: Arduino DUE Specifications Table

Module:	Arduino DUE
Inputs:	Voltage: Operating (3.3V), Vin Recommended (7-12V), Input Limit (6-16V) Digital I/O: 42 (12 PWM output) Analog: 12
Outputs:	Digital I/O: 54 (12 PWM output) Analog: 2 (DAC) DC output current: 130mA
Flash Memory:	512 KB
SRAM:	96 KB
Clock Speed:	84 MHz
Dimensions:	101.6mm x 53.3mm
Price:	$45.84

A.7: Behavioral Model of the Robot



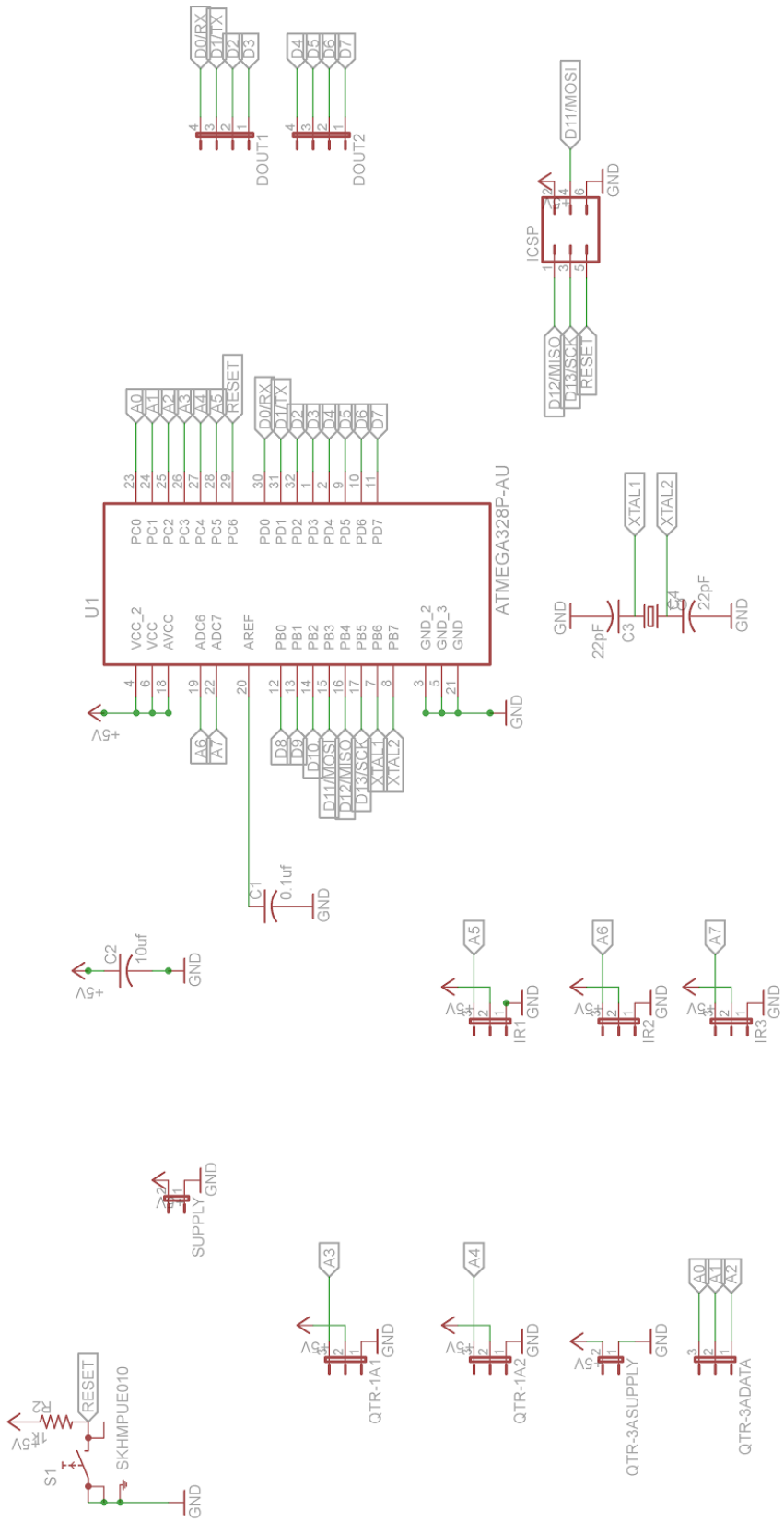
A.8: Behavioral Model of the Processor



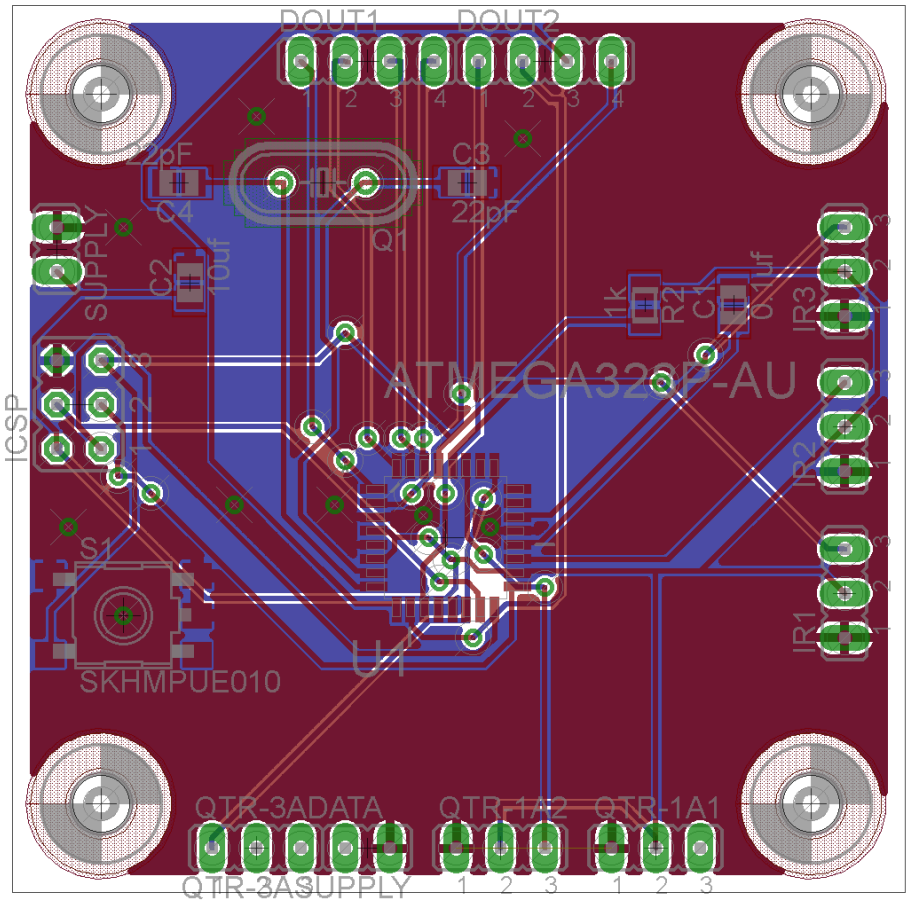
A.9: Functional Decomposition of the Processor

Module:	Processor
Inputs:	Power Button Control Images Proximity Sensor Reflective Sensor
Outputs:	Motor Signals Character (data) to USB
Functionality	Takes a button control to start or stop running. Takes a proximity signal and reflective signal to determine the walls around it and the placement on the maze cells respectively. It takes an image from a camera. It uses this data to output signals to motor and data to a USB.

A.10: Sensor Interface Board Circuit Diagram



Interface Board Layout



A.11: Calibration Plans for the Sensor Package

The sensor calibration test consists of wiring the sensors to an Arduino Nano with the following pin connections:

Left IR Sensor - A5

Right IR Sensor - A7

Front IR Sensor - A6

Left Ground Sensor - A3

Right Ground Sensor - A4

Front Ground Sensor - A0, A1, A2 (left to right)

The Arduino Nano is being used for ease of data acquisition through use of the built-in serial monitor within the Arduino IDE.

The Arduino Nano will then be connected to the PC and loaded with the following code:

```
long IRsensor1 = 0;
long IRsensor2 = 0;
long IRsensor3 = 0;

long Refsensor1 = 0;
long Refsensor2 = 0;
long Refsensor3 = 0;
```

```
long RefsensorL = 0;
long RefsensorR = 0;

void setup(){
  Serial.begin(9600);
}

void loop(){
  IRsensor1 = analogRead(A5);
  IRsensor2 = analogRead(A6);
  IRsensor3 = analogRead(A7);

  Refsensor1 = analogRead(A0);
  Refsensor2 = analogRead(A1);
  Refsensor3 = analogRead(A2);

  RefsensorL = analogRead(A3);
  RefsensorR = analogRead(A4);

  Serial.print("Left Wall:      ");
  Serial.println(IRsensor1);
  Serial.print("Front Wall:     ");
  Serial.println(IRsensor2);
  Serial.print("Right Wall:     ");
  Serial.println(IRsensor3);
  Serial.println();
  Serial.print("Left GSensor:   ");
  Serial.println(RefsensorL);
```

```
  Serial.print("Right GSensor:  ");
  Serial.println(RefsensorR);
  Serial.print("Front GSensorL: ");
  Serial.println(Refsensor1);
  Serial.print("Front GSensorM: ");
  Serial.println(Refsensor2);
  Serial.print("Front GSensorR: ");
  Serial.println(Refsensor3);
}
```

Next, each sensor will be tested under different conditions and the corresponding analog values under each condition will be recorded, and the calibration values will be used to determine the target ranges for each individual sensor.

IR Sensor Test Sheet

IR Sensor (Circle):	**Left**	**Front**	**Right**					
Distance From Object:			**Analog Value:**	**NOTES:**				
5cm								
6cm								
7cm								
8cm								
9cm								
10cm								
11cm								
12cm								
13cm								
14cm								
15cm								
16cm								
17cm								
18cm								
19cm								
20cm								
21cm								
22cm								
23cm								
24cm								
25cm								

Reflectance Sensor Test Sheet:

Ground Sensor (Circlce):	**Left**	**F-Left**	**F-Middle**	**F-Right**	**Right**			
Color of Object:			**Analog Value:**	**NOTES:**				

A.12: CMUcam4 Specifications Table

Pros	Cons
Arduino Shield Compatible	Cost $115-$130
Onboard Image Processing (QQVGA 160x120)	TTL UART (up to 250,000 baud – 19,200 baud by default)
µSD/µSDHC flash card slot with FAT16/32 full file system driver support	Power **250 mA** at between **4V to 9V DC**.
Two-port servo controller (pan and tilt w/ 1µs resolution at a 50 Hz refresh rate)	Diminutions 52.25mm*53.50mm*13mm
Monochrome baseband analog video output (NTSC/PAL) of 160x120 resolution for tracking visualization (segmented (threshold) image w/ color centroid and bounding box overlay at 30 FPS)	No USB connector

A.13: Comparison Matrix and Analysis of Maze Solving Algorithm

To solve the maze we decided on the recursively maze solving algorithm.

Algorithm	Solutions	Guarantee?	Focus	Human Doable?	Passage Free?	Memory Free?	Fast?
Random Mouse	1	no	You	Inside / Above	no	Yes	no
Wall Follower	1	no	You	Inside / Above	Yes	Yes	Yes
Pledge Algorithm	1	no	You	Inside / Above	Yes	Yes	Yes
Chain Algorithm	1	Yes	You +	no	Yes	no	Yes
Recursive Backtracker	1	Yes	You	no	Yes	no	Yes
Trémaux's Algorithm	1	Yes	You	Inside / Above	no	no	Yes
Dead End Filler	All +	no	Maze	Above	no	Yes	Yes
Cul-de-sac Filler	All +	no	Maze	Above	no	Yes	Yes
Blind Alley Sealer	All +	Yes	Maze	no	no	no	Yes
Blind Alley Filler	All	Yes	Maze	Above	no	Yes	no
Collision Solver	All Shortest	Yes	You +	no	no	no	Yes
Shortest Paths Finder	All Shortest	Yes	You +	no	Yes	no	Yes
Shortest Path Finder	1 Shortest	Yes	You +	no	Yes	no	Yes

This table summarizes the characteristics of the Maze solving algorithms above. Maze solving algorithms can be classified and judged by these criteria. Descriptions of the columns follow:

- **Solutions:** This describes the solutions the algorithm finds, and what the algorithm does when there's more than one. An algorithm can pick one solution, or leave multiple solutions. Also the solution(s) can be any path, or they can be the shortest path. The dead end and cul-de-sac fillers (and the blind alley sealer when considering its inaccessible sections) leave all solutions, however they may also leave passages that aren't on any solution path, so are marked "All +" above.

- **Guarantee:** This is whether the algorithm is guaranteed to find at least one solution. Random mouse is "no" because it isn't guaranteed to terminate, and wall follower and Pledge algorithm are "no" because they will fail to find a solution if the goal is within an island. The dead end and cul-de-sac fillers are "no" because they may not do anything to the Maze at all in purely braid Mazes.
- **Focus:** There are two general types of algorithms to solve a Maze: Focus on "you", or focus on the Maze. In a you-focuser, you have a single point ("You" above) or a set of points ("You +" above) and try to move them through the Maze from start to finish. In a Maze-focuser, you look at the Maze as a whole and invalidate useless passages.
- **Human Doable:** This refers to whether a person could readily use the algorithm to solve the Maze, either while inside a life sized version, or while looking at a map from above. Some you-focuser algorithms can be implemented by a person inside (or above) the Maze, while some Maze-focusers can be implemented by a person, but only from above. Other algorithms are complicated or intricate enough they can only reliably be done by a computer.
- **Passage Free:** This is whether the algorithm can be done anywhere. Some algorithms require the Maze to have obvious passages, or distinct edges between distinct vertices in graph terms, or one pixel wide passages when implemented on a computer. The wall follower, Pledge algorithm, and chain algorithm only require a wall on one side of you. The recursive backtracker and the shortest path(s) finders make their own paths through open spaces.
- **Memory Free:** This is whether no extra memory or stack is required to implement the algorithm. Efficient algorithms only require and look at the Maze

bitmap itself, and don't need to add markers to the Maze during the solving process.

- **Fast:** This is whether the solving process is considered fast. The most efficient algorithms only need to look at each cell in the Maze once, or can skip sections altogether. Running time should be proportional to the size of the Maze, or in Computer Science terms O(n^2) where n is the number of cells along one side. Random mouse is slow because it isn't guaranteed to terminate, while the blind alley filler potentially solves the Maze from each junction.

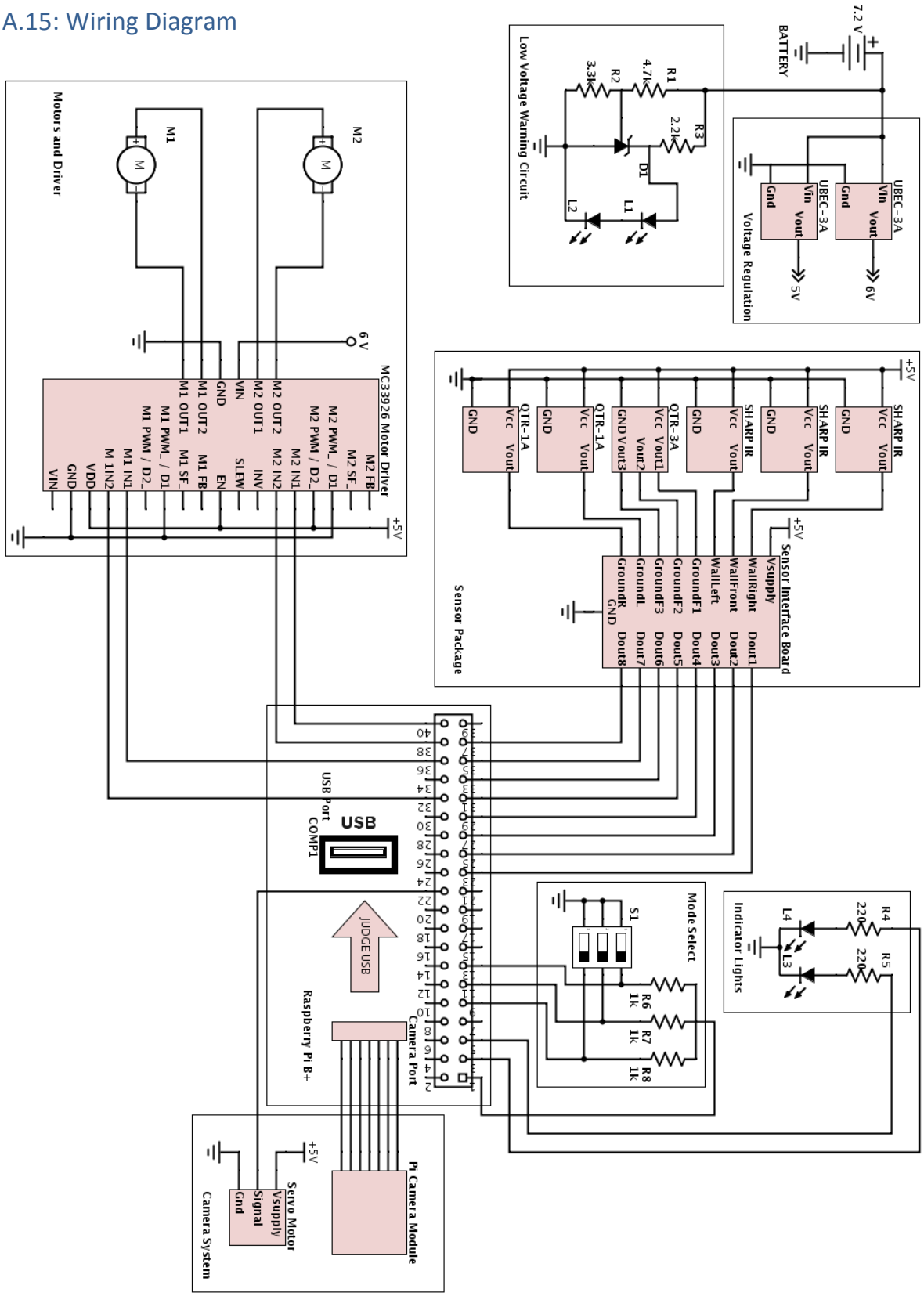
A.14: Full Bill of Materials

Part	Supplier	Cost	Qty	Ext Price
Raspberry Pi	Amazon	37.49	1	37.49
Camera Module	Amazon	27.21	1	27.21
Sensors				
QTR-3A	Pololu	4.95	1	4.95
QTR-1A	Pololu	4.25	2	8.50
Sharp IR Sensor	Digikey	7.40	3	22.20
Bracket Pair	Pololu	3.49	2	6.98
Crimp Connector	Pololu	0.79	2	1.58
3-Pin JST Cable	Pololu	1.25	3	3.75
M3 Machine Screws	Pololu	0.99	4	3.96
Hex Nuts M3	Pololu	0.99	4	3.96
6" F-F Servo Cables	Pololu	1.95	4	7.80
SubMicro Servo	Pololu	4.95	1	4.95
Test Maze				
4'x8'x3/4" MDF	Lowes	35.59	2	71.18
4'x8'x1/4" Ply	Lowes	21.92	2	43.84
Gal White	Lowes	16.98	1	16.98
Qt. Green	Lowes	20.44	2	40.88
Qt. Pink	Lowes	20.44	1	20.44
Qt. Blue	Lowes	20.44	1	20.44

Paint Marker	Amazon	10.25	1	10.25
Painter's Tape	Amazon	6.35	1	6.35
Sensor Board				
PCB Fab	SeeedStudio	9.90	1	9.90
Atmel 328p	Digikey	3.85	1	3.85
0.1 uF Cap	Digikey	0.09	10	0.89
22 pF cap	Digikey	0.29	10	2.94
10 uF	Digikey	0.50	10	5.00
Res 1k	Digikey	1.72	5	8.60
Osc 16Mhz	Digikey	0.36	5	1.80
Headers	Digikey	0.40	1	0.40
Body and Movement				
Pololu 30T track	Pololu	14.95	1	14.95
Pololu micrometal gearmotor	Pololu	15.95	2	31.90
1/8" Acrylic sheets (12"x12")	Pololu	5.64	2	11.28
2" standoffs (10 pack)	Pololu	7.95	1	7.95
Pololu gearmtor bracket pair (extended)	Pololu	3.25	1	3.25
Dual MC33926 Motor Driver	Pololu	29.95	1	29.95

Carrier				
Power				
Tenergy 7.2V	Pololu	27.00	1	27.00
NiMH Smart Charger	Pololu	18.00	1	18.00
Hobbywing Regulator	Pololu	9.00	2	18.00
			Total	**559.35**

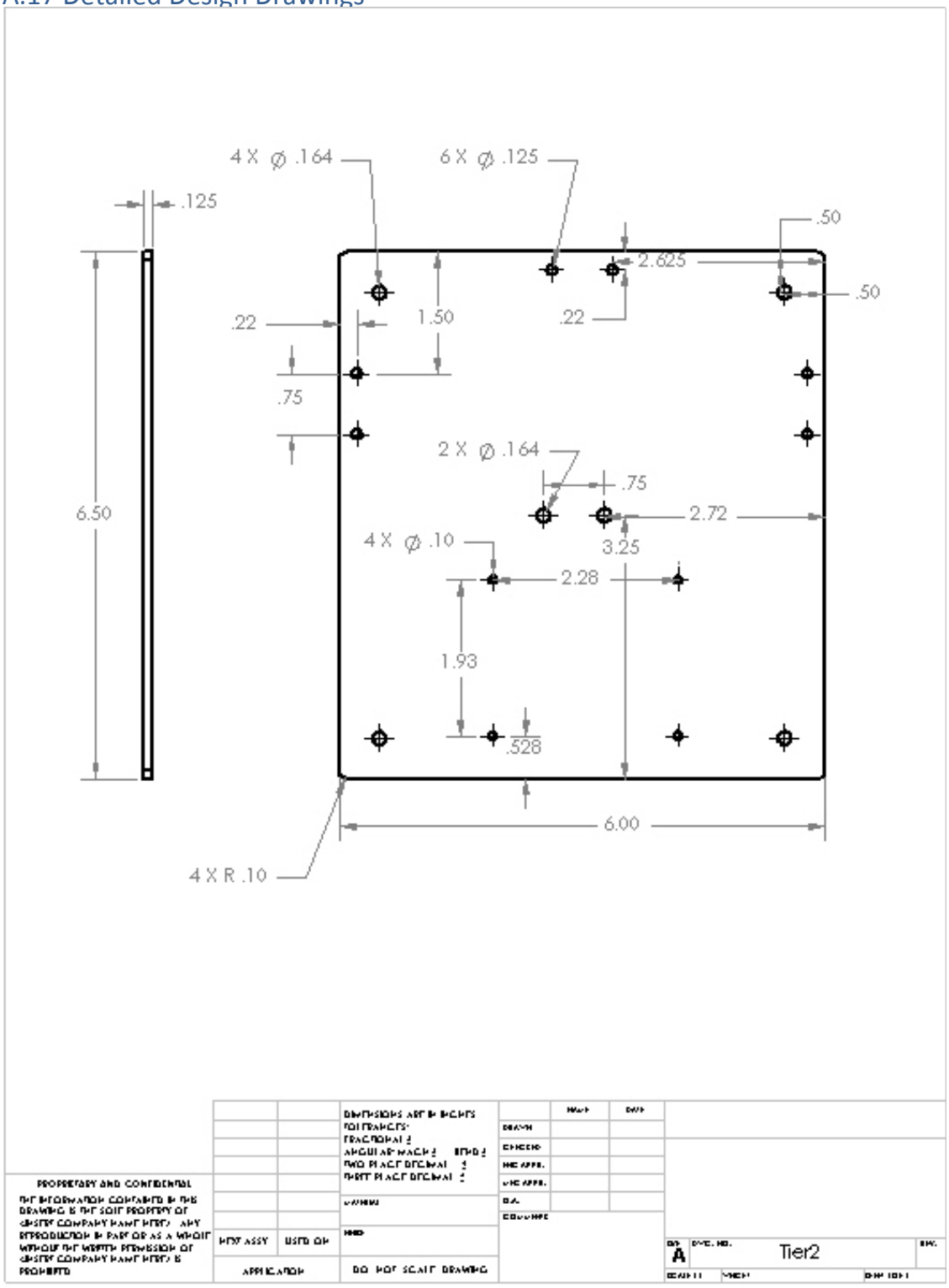
A.15: Wiring Diagram

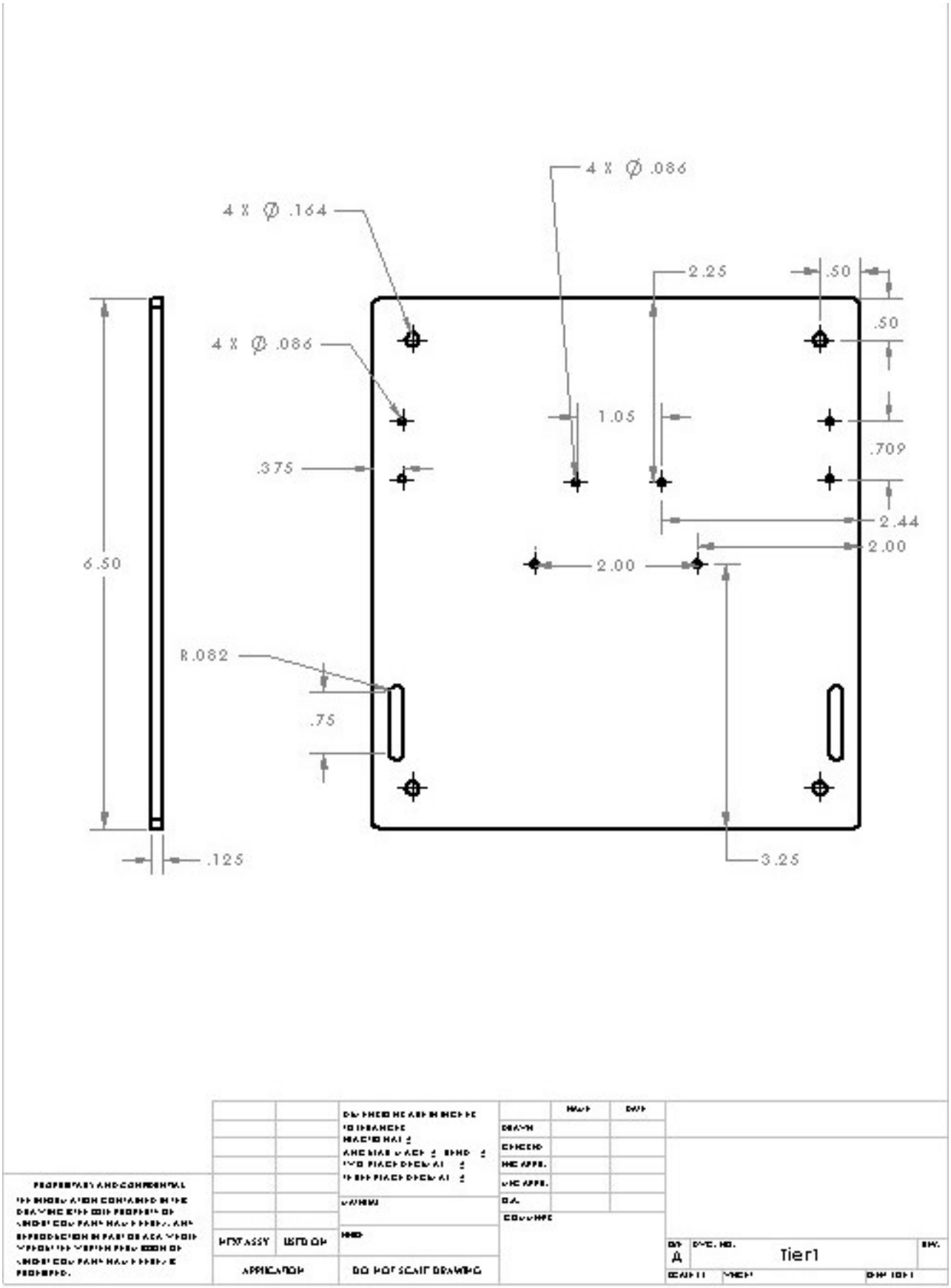


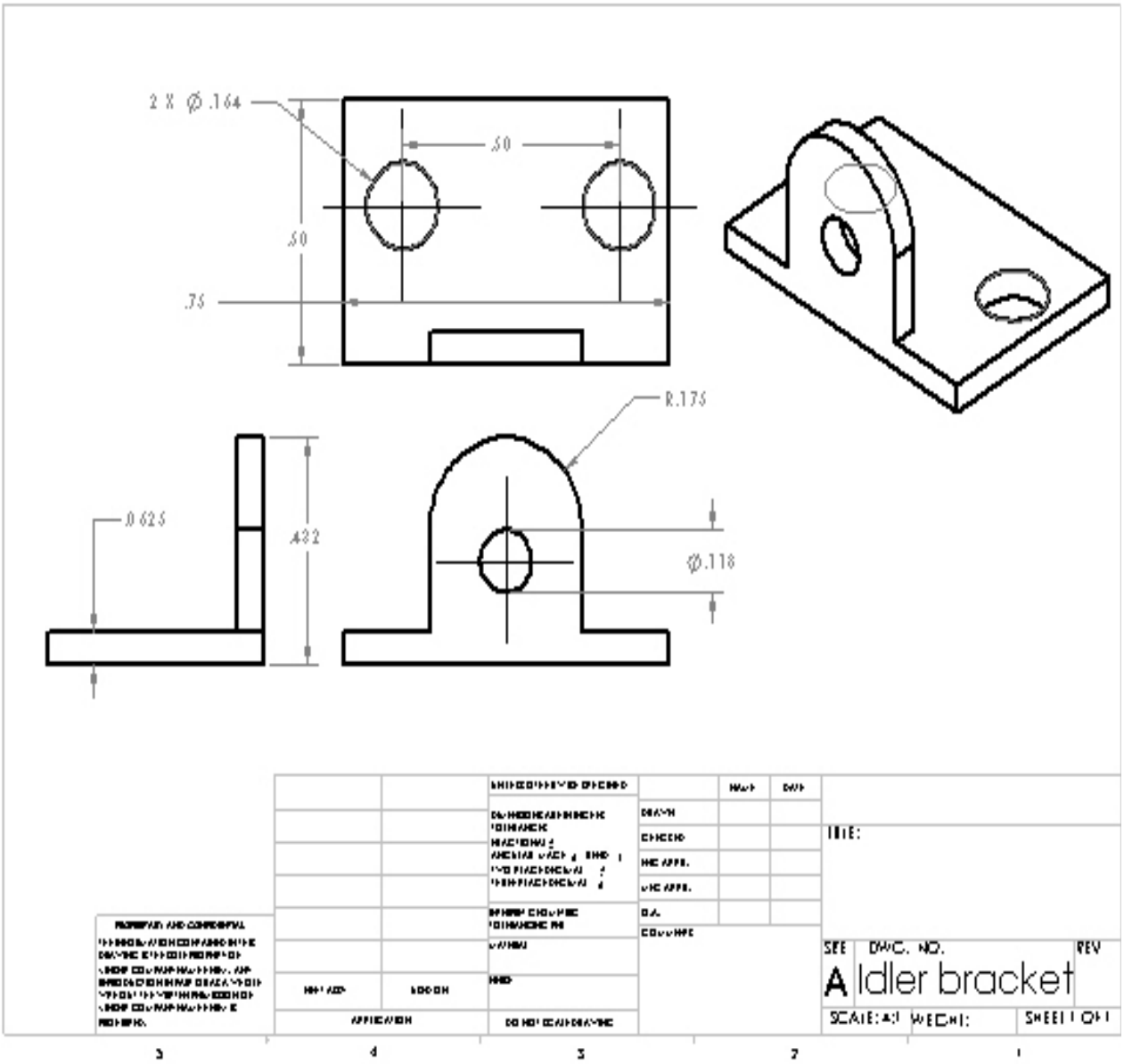
A.16: Detailed Character Recognition Testing Plan

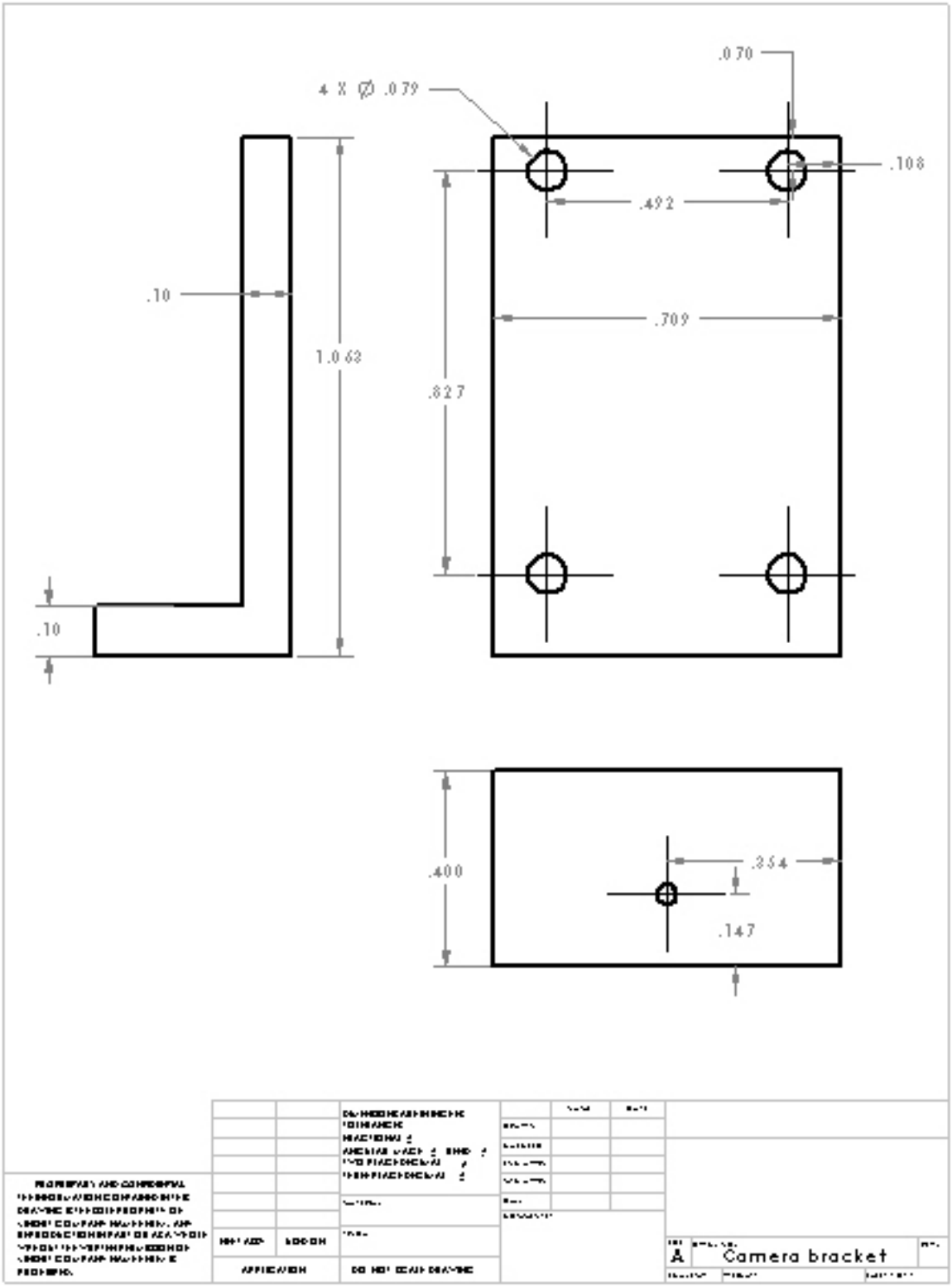
First we have to test basic functionality of the raspberry pi Camera and the raspberry Pi B+. The Pi has to be able to provide the camera with the sufficient power and read the right data sent from the camera to the GPU to for the camera to operate correctly and efficiently. Then we have to test if the camera can take pictures with the desired resolution and if the GPU can store them with relative square number using a predefined algorithm written by us that determines the location from the start position and the maze size. Then we have to check for that algorithm correctness. We also need to test the camera video capability as for how clear the taken videos, as that will help us to correctly determine the wall colors. The camera will also be calibrated with the other sensors after integration to see if they can response together to the robot position, (i.e. stopping at the intersections of the black lines in the middle of the squares) and the wall colors (i.e. taking picture) while the robot is not moving. Test also for picture storing in the Pi's memory and if it can be recalled. Another test will be for the character identification accuracy with noise and distortion be applying noise to some stored pictures and/or taking pictures of the maze walls while robot is moving and test for how much recognition accuracy we can get. Test the robot for the ability of storing the output file for the competition judges, and the correct configuration.

A.17 Detailed Design Drawings




4 X Ø .086
4 X Ø .164
2.25
.50
.50
4 X Ø .086
1.05
.709
.375
2.44
2.00
6.50
2.00
R.082
.75
.125
3.25
Tier1


2 X Ø .164
.50
.50
.75
R.175
0.625
.432
Ø.118
TITLE:
SIZE
DWG. NO.
REV
A
Idler bracket
SCALE: 4:1
WEIGHT:
SHEET 1 OF 1


4 X Ø .079
.070
.108
.492
.10
.709
1.063
.827
.10
.400
.254
.147
A
Camera bracket

Works Cited

Websites

"2015-IEEE-R5-Robotics Competition Rules - Aug Release." - *R5StudentCompetition*. N.p., n.d. Web. 1 Sept. 2014.

"Arduino - ArduinoBoardDue." *Arduino - ArduinoBoardDue*. N.p., n.d. Web. 10 Sept. 2014.

"CMUcam: Open Source Programmable Embedded Color Vision Sensors." *CMUcam: Open Source Programmable Embedded Color Vision Sensors*. N.p., n.d. Web. 20 Oct. 2014.

"Difference between Microprocessor and Microcontroller." *EngineersGarage*. N.p., n.d. Web. 20 Oct. 2014.

"How the Robot Sense?sensors!!!!!" *Pavanreddi*. N.p., n.d. Web. 15 Oct. 2014.

"How to Build a Robot Tutorials - Society of Robots." *How to Build a Robot Tutorials - Society of Robots*. N.p., n.d. Web. 10 Oct. 2014.

"Linux.com." *Linux.com*. N.p., n.d. Web. 18 Oct. 2014.

"MIPI Alliance." *Camera Interface Specifications*. N.p., n.d. Web. 20 Oct. 2014.

"Model B+." *Raspberry Pi Model B+ Comments*. N.p., n.d. Web. 15 Oct. 2014.

"OCRchie: Modular Optical Character Recognition Software." *OCRchie Project Spring 1996*. N.p., n.d. Web. 20 Oct. 2014.

"Propeller C3." *Parallax Inc*. N.p., n.d. Web. 16 Oct. 2014.

"Rpi Camera Module." *ELinux.org*. N.p., n.d. Web. 20 Oct. 2014.

"RS-232 vs. TTL Serial Communication - SparkFun Electronics." *RS-232 vs. TTL Serial Communication - SparkFun Electronics*. N.p., n.d. Web. 20 Oct. 2014.

"Top News." *IFR RSS*. N.p., n.d. Web. 15 Sept. 2014.

"USART (Universal Synchronous/Asynchronous Receiver/Transmitter)." *What Is ?* N.p., n.d. Web. 20 Oct. 2014.

http://blog.famzah.net/2010/07/01/cpp-vs-python-vs-perl-vs-php-performance-benchmark/

Books

Cook, David. *Intermediate Robot Building*. Berkeley, CA: Apress, 2010. Print.

Ford, Ralph M and Chris S Coulston. *Design for Electrical and Computer Engineers: Theory, Concepts, and Practice*. McGraw-Hill's Custom Publishing, 2005. Print.

McComb, Gordon. *Robot Builder's Bonanza*. New York: McGraw-Hill, 2011. Print.

Papers

Jung, Keechul, Kwang In Kim, and Anil K. Jain. "Text Information Extraction in Images and Video: A Survey." *Pattern Recognition* 37.5 (2004): 977-97. Web.