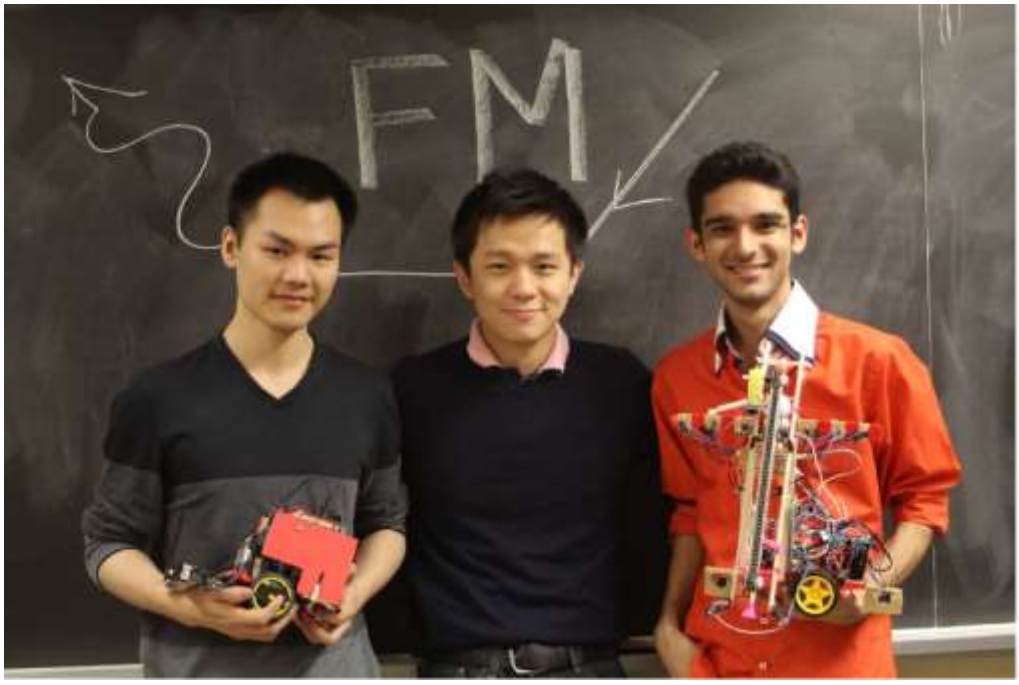
Team: Finemen (#14)

AER 201 Final Report

Teaching Assistant: Kaizad Raimalwala
Instructors: Victor Ragusila, Todd Reichert, Cameron Robertson, M. Reza Emami



Alyf Janmohamed, Howard Duan, Johnson Zhong
4/10/2015

A big thanks to our TA Kaizad for supporting our adventurous design of 2 independent robots. But much more gratitude for alerting us of things our whole team had overlooked. Without those gentle nudges in the right direction, our robots would not have worked as well as we hoped. Also Thanks to the whole floor 45-4 of New College III for enduring our all night jam sessions with grace and equanimity, as well as tolerating the mess we had left of the common study room. And finally, thank you to Todd, Victor, and Cameron for providing us with such an challenging yet rewarding AER201 design project.

Executive Summary

This report represent a half years worth of hard work and dedication to our team's collective design values. Our design is distinguished by distributing the game play to two robots, retriever-bot (Rbot) and gameplaying-bot (Gbot), and the sophisticated navigation system that links the two together. In this report, we present our design process and iterations that resulted in the refinements and details necessary for interacting two robots in addition to playing a game with randomized elements requiring robust navigation. Specific topics include the PID velocity controlled navigation system supplemented by line correct, the decision to use off the shelf chassis kits, and the results of extensive testing of sensors to detect the game state. In addition, we provide details about the materials, sensors, programming, and methods of construction used.

In the implementation section, we detail the logic behind our robot's behaviour, as well as the physical mechanisms of the electromechanical system. Topics in this section include the tests conducted to find the best sensors, a graphical map user interface for arbitrary waypoint setting via serial writing and debugging, and physical debugging frameworks, including a mockup of the game board and hoppers.

In project management, we highlight the difference in expected time to completion relative to the actual time taken for integration tasks. A table then summarizes the work completed by each role, as well as the crossover of responsibilities among the different roles. Then we justify our total cost of $239.23 with a bill of materials in appendix B.

To conclude, we reflect upon the lessons learned from undertaking a 2 robot design, and consider the unnecessary constraint we set on our robots by using a off the shelf chassis, and discuss methods to avoid such pitfalls in the future.

Table of Contents

Symbols and Abbreviations

R-bot: The ball **r**etrieving robot

G-bot: The **g**ame playing robot

LED: Light emitting diode

PID: Proportional, Integral, Derivative (a control technique)

Break beam is synonymous to photo-interrupter

1. Design Process

This section will provide background information regarding the project, by summarizing the task to be completed, highlighting the key components of our design, describing our design values and the performance metrics being used to evaluate our design. In addition, it will summarize the background research we conducted and our brainstorming process.

1.1 Gameplay Strategy

For the AER 201 competition this year, each team had to design and prototype a robot that could autonomously play a non-turn based version of Connect-4. The game balls (ping pong balls) must be collected either from corner hoppers at predetermined locations or from central hoppers in random locations. Each corner hopper holds 4 balls and each central hopper holds 7 balls, allowing each team to play a maximum of 22 balls.

The game field is split into two halves and the robots are confined to their respective halves, each of which contains 4 hoppers and 22 balls. A grid 20cm x 20cm grid is painted on the field. All of the grid lines are black, except for the red centre line that is perpendicular to the game board. An annotated diagram is shown below in Figure 1.

Lastly, a team would receive one point for every ball played, four points for each of their Connect-4s, and lose two points for each of their opponent's Connect-4s.

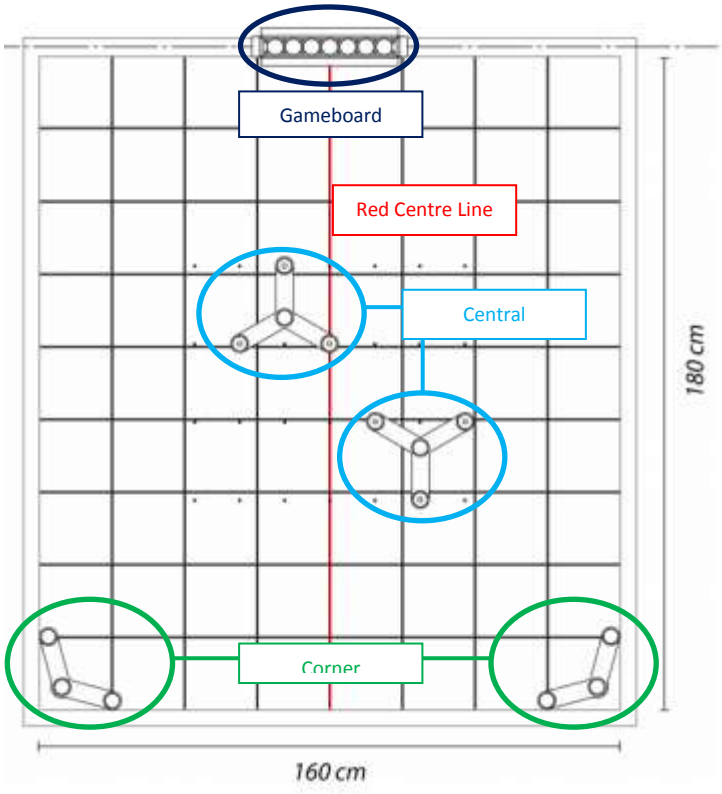


Figure 1 Annotated Gamefield

1.2 Our Design

To accomplish the task previously described, we decided to go with a two robot design. The first robot, R-Bot, was designed to retrieve balls from the hoppers and deliver them to the second robot, G-Bot, at a pre-determined location, known as the rendezvous point. At the beginning of the game, G-Bot would navigate from the starting position at the back, to the game board. Here it would play the balls retrieved by R-bot and monitor if the opposing team played any balls. From constant game board sensing, G-bot can keep track of game state and therefore play intelligently.

1.3 Design Values

At the beginning of our design process, we, as a team, identified the design values that we believed would be representative of a good robot design. They are listed and described below:

1. Simplicity - a modular design where each component has a specific task. Minimize the number of tasks per component and complexity of control required for successful execution. Ensure that all components can be tested independently to ease debugging.
2. Redundancy - the robot should have ways to verify and correct primary system. For example, odometry can specify a position and heading, but these should be updated by a secondary system such as line detection
3. Robustness and Tolerance - design should allow for error and the robot should still be able to complete task under uncertainty. For example, to collect the ball from the hopper, the robot should not have to be perfectly positioned
4. Speed - design should retrieve and play balls as fast as possible to maximize points scored.

1.4 Performance Metrics

Performance metrics were derived from the constraints identified in the course manual, the scoring system, and our design values. They are shown below in Table 1.

Table 1 Performance metrics to evaluate robot.

Objectives	Metrics	Constraints	Goal	Criteria
maximize balls retrieved (speed)	number of balls retrieved in 7 min	> 1	14 (1 every 30s)	more is better
consistent play (robustness)	percent of retrieved balls played (%)	N/A	100 %	more is better
minimize size (speed)	dimensions (cm x cm x cm)	< 40x40x40 cm each dimension independently	N/A	smaller is better
minimize weight (speed)	total weight (kg)	< 3 kg	N/A	lower is better
minimize cost	cost (CDN)	< $250	N/A	lower is better
minimize internal state modifiers (simplicity, robustness)	maximum number of functions in which the same internal state is modified	> 0	1	1 is best
maximize tolerance of position (robustness)	maximum error with correct play (cm)	> 0	5cm	higher is bettter
minimize number of resets per game (robustness)	number of resets	< 2	0	lower is better

1.5 Background Research

We began our design process by decomposing the overall task into five smaller tasks. The functional decomposition is listed below:

1. Grabbing the ball
2. Lifting and depositing the ball
3. Moving and navigating
4. Locating dispensers
5. Reading game state

For each of the five tasks listed above, reference designs were identified and are briefly described below in Tables 2, 3 and 4.

Table 2 Background research conducted for grabbing the ball and lifting and depositing it.

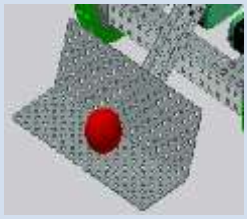
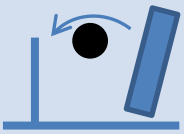
Function	Reference Design 1	Reference Design 2	Reference Design 3
Grabbing the Ball	**Scooping the ball**  Figure 2 Scoop manipulator from VEX robotics. Advantages: • Few moving parts • High tolerance Disadvantages: • Makes ball transfer difficult • Friction between ground and robot (external force could complicate navigation)	**Sweeping the ball**  Figure 3 Illustration of rotating arm to collect (sweep) ball Advantages: • Tolerance of position • Secures ball in a precise location Disadvantages: • Can be difficult to remove • Significantly increases turning radius of robot • Introduces asymmetry (rotating are on one side of robot)	**Grabbing the ball (claw)**  Figure 4 Claw manipulator from VEX robotics. Advantages: • Prebuilt system • Bidirectional • Secures ball in precise location Disadvantages: • Requires more moving parts • Requires precise positioning
Lifting and depositing the ball	**Vacuum** A fan could be used to create a pressure differential that would suck the ping pong ball up. It would then be ready to be played Advantages: • Quickly raise the ball Disadvantages: • Require significant power • Would still require another component to play the ball	**Conveyor** Continuous loop to move objects in one direction. Variation would use a 'paddle' to life the ball up an inclined plane Advantages: • Robust • 1-way looped operation Disadvantages: • Requires multiple moving parts • Structure may be heavy	**Elevator** Similar to conveyor method, special carriage would raise the ball to game board height. Actuated by motor winding a string via a pulley. Advantages: • Simple vertical motion • Multiple reference designs Disadvantages: • Many moving parts • Structure may be heavy • Less robust if ball misbehaves, no easy recovery

Table 3 Background research conducted for moving and navigation, and locating hoppers.

Function	Reference Design 1	Reference Design 2	Reference Design 3
Moving and navigating	**Line Following** Simple navigation system requiring only some type of photo sensor. Approach feasible because of grid on game field. Advantages: • Simple and proven implementation Disadvantages: • Resolution limited by width of line • Accurate movement in units of 20cm and 90^{o} • Likely to get completely lost if robot deviates from line	**Dead reckoning with odometry** Navigation system reliant on incremental updating position by integrating velocity over a time interval. Wheel encoders are used to measure velocity Advantages: • Allows arbitrary motion as opposed to line following, robot can take more direct paths • More robust, can still function without grid lines Disadvantages: • Small errors accumulate, causing large drifts and errors in heading angle • No way to correct for robot drifting • Performance dependent of resolution of encoders	**Wall Following** Robot maintains a pre-determined distance from the wall to navigate and drive straight Advantages: • Simplest navigation implementation • Very little processing required, therefore can probably drive faster • Automatically adjusts for external environment and can automatically recalibrate Disadvantages: • Cannot reliably access central hoppers • Must travel maximum distance
Locating hoppers	**Do not use central hoppers** Collect balls only from the corner hoppers whose positions can be hard coded. Advantages: • Simplest and most reliable solution Disadvantages: • Maximum 8 balls can be retrieved • Balls farthest from game board	**Keypad Input** Use a keypad to input locations of central hoppers Advantages: • Simple scheme for inputting central hopper location Disadvantages: • Limited information input • Requires keypad to be included in final design • May not be able to double check which buttons were pressed	**Serial Writing** Use Processing to provide information to robot. Does not require re-uploading code Advantages: • Large amounts of information can be provided to robot • Can visualize information provided (graphically) Disadvantages: • Requires laptop and more code to be developed and tested

Table 4 Background research conducted for sensing game state.

Function	Reference Design 1	Reference Design 2	Reference Design 3
Sensing game state	**No game sensing** Do not sense game state or game balls. Always deposit ball in the same column or a random column Advantages: • Simple design, no sensors required Disadvantages: • Cannot play intelligently • Cannot determine when column is full, would not get points for putting ball in column that is already full	**Sense only top row** Sense only the top row. Would always determine if there is space to play in a specific column Advantages: • Still mechanically simple • Avoids playing ball in column that is full Disadvantage: • Does not permit intelligent gameplay	**Full game state sensing** Sense the locations of all balls that have been played. Advantages: • Optimal gameplay, maximize points scored Disadvantages: • Most complex solution. If game board not being constantly monitored, either 42 sensors required or moving sensor bar

1.6 Conceptualization

This section will summarize our divergence, convergence, and decision making processes.

1.6.1 Divergence Process

Two techniques were used to expand our design space, 'Challenging Assumptions' and 'Abstracting Up'. In the first technique, 'Challenging Assumptions', specific constraints are ignored while brainstorming solutions. Later, the applicability of solutions to these 'simpler' design problems are evaluated, and necessary adaptations to the new ideas were made. The second technique, 'Abstracting Up', a more general version of the task was articulated and reference designs were researched.

1.6.2 Convergent and Decision Making Process

The divergence process described above creates multiple solutions that are unrealistic, or would be too difficult to implement with the given constraints. Therefore, the first step in our convergence process is to eliminate infeasible and unrealistic ideas. The remaining ideas are further explored through sketches, and low fidelity prototyping. After a better understanding of the potential solutions has been developed, a pugh chart is used to explore the advantages and disadvantages of our solutions. As a team, we used our design values and engineering judgement, informed by the pugh chart, to make our decision.

2. Technical Description

Technical description outlines in detail components used, programs deployed, and subsystem interactions within the robot.

2.1 System Level Description

When the game first starts, both robots are calibrated against the game field for five seconds. Then, human controllers plot the waypoints for navigating initially from the starting position, and the path for retrieving balls from the hoppers using the map user interface from Processing, communicated to the Arduinos via serial writing. Once this is completed, the USB serial cable is disconnected and the robots are set in their starting position (see figure 6) Rbot has a 5 second delay programmed into it to avoid collision with Gbot when both attemp to turn in place. Gbot's path is plotted in the same way as Rbot's, with the exemption from retrieving from hoppers. Gbot's path should go around the left side of the game board, hugging the wall but leaving enough room for turning in place. Near the gameboard, Gbot's automatic sonar correction activates to guide it to the desired distance away from the gameboard while its line correction brings it right past the center red line to the correct x coordinate. Rbot follows first the human plotted initial part to navigate away from the starting position to get into an advantageous location, then finds the nearest hopper and approaches it using the second path the human plotted. As Rbot approaches the hopper and the ball sits securely on the ramp, break beam sensors located on the side of the gate are broken, and signifying the secure possession of the ball, allowing the servoed gate to close. Rbot then backs up to a distance safe enough to turn in place, after which it heads towards the rendezvous point to deposit the ball to Gbot. Gbot detects the deposited ball via an IR break beam, navigates to the column giving the highest point if played, then lifts the ball up while watching for the ball to drop with its photoresistor and laser sensor bar. After it plays the ball, which it knows by the length of time since lifting and if it detects a ball drop in the appropriate column, Gbot returns to its rendezvous point to away more balls from Rbot. Figure 5 below show the graphical flow of each robots' specific and shared behaviours and interactions.

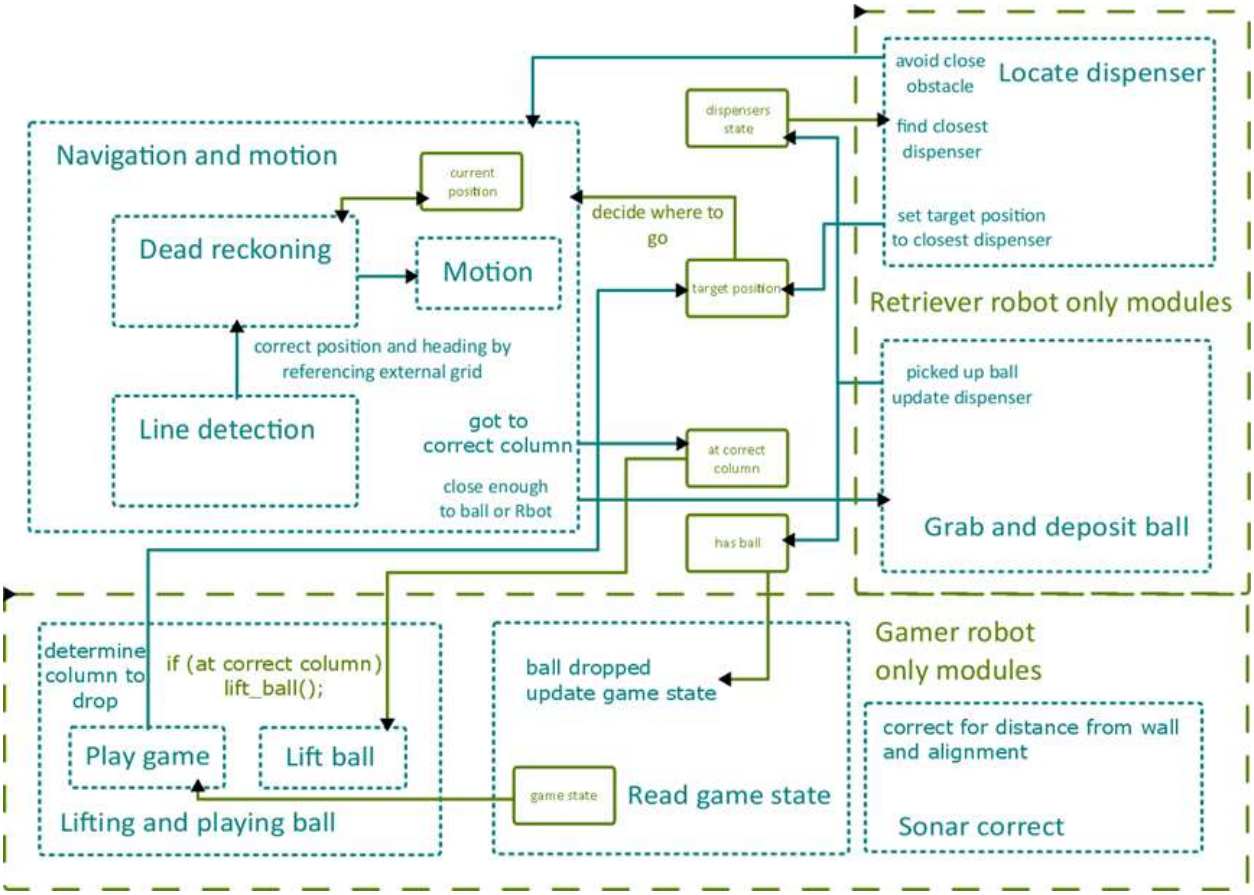


Figure 5. System level interaction of various modules.

The initial position of the two robots is shown below.

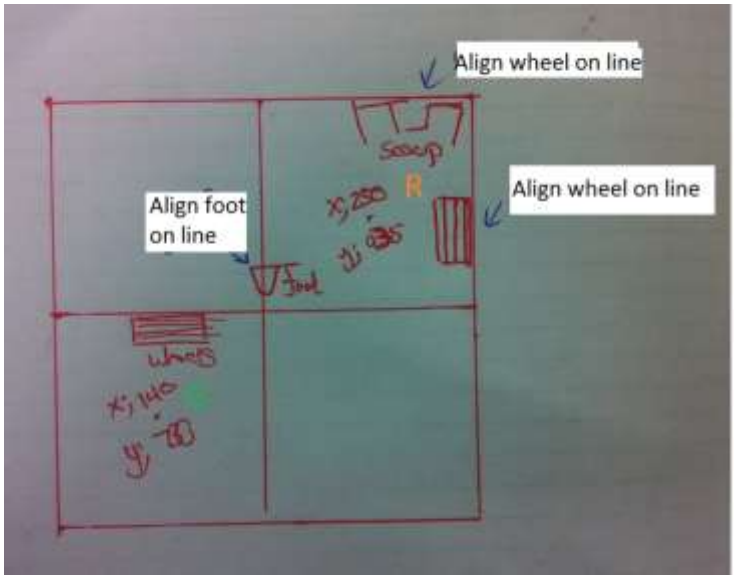


Figure 6 Map showing alignment of R-bot and G-bot, with coordinates.

2.2 Full Operating Procedure

The full operating procedure has been subdivided into 4 sections:

1. Before arriving at game field
2. First 5 minutes at game field
2 minutes setup time
3. After the game (Post-flight checklist)

2.2.1 Pre-flight checklists

The first three sections of the Full Operating Procedure are distinct Pre-flight checklists and are displayed below.

2.2.1.1 Before arriving at game field

- ✓ Check batteries: (R-bot and G-bot)
    - Replace 9V battery if measured voltage is below 8V
    - Replace 4 AA batteries if measured voltage is below 6V
- ✓ Test encoders: (R-bot and G-bot)
    - Run encoder program, ensure all 4 break beams are working
    - Push in all 4 wheels
- ✓ Check bottom sensors, reposition/realign if necessary (R-bot and G-bot)
- ✓ Check orientation of sensor bar incase it was nudged/displaced (G-bot)
- ✓ Check position of elevator and sonar (G-bot)

2.2.1.2 First 5 minutes at game field

- ✓ Test bottom sensors, ensure they can detect game lines (R-bot and G-bot)
- ✓ Upload code for G-bot
    - Give G-bot coordinates for navigation to game board
    - Double check that correct coordinates have been given to G-bot
    - Calibrate:
        - Plug in battery power to Arduino and remove USB serial port
        - Calibrate robot (3 second delay from uploading path to beginning of calibration)
    - G-bot is now game ready
- ✓ Upload code for R-bot
    - Test red line detection threshold
    - Have processing open and ready to communicate with R-bot

2.2.1.3 Three minutes setup

- ✓ Place G-bot in starting position
- ✓ Input path that R-bot should take via Processing
- ✓ Connect battery power to R-bot, then disconnect USB cable
- ✓ Calibrate R-bot

- ✓ Place R-bot in starting position, it is now game ready
- ✓ Wait for game to begin, start both robots

2.2.2 Post-Flight Checklist

- ✓ Check laser sensor bar (G-bot)
- ✓ Check bottom photo resistor (G-bot)
- ✓ Check bristly rigidity (G-bot)
- ✓ Check conveyor rigidity (G-bot)
- ✓ Check bottom sensor bar (R-bot)
- ✓ Check scoop position (R-bot)
- ✓ Check sword/gate rigidity (R-bot)

2.3 Sub-system Level Description

This section will provide full technical details for each of the three sub-systems of R-bot and G-bot.

2.3.1 Electro-mechanical Sub-system

Please refer to Appendix A for dimensions of parts refereed.

2.3.1.1 Rbot

Rbot uses an off the shelf chassis kit (the Magician Chassis by SparkFun), which includes two 3-6V motors, 2 wheels/hubs with fitting shafts holes, 4 mounting pieces to secure the motor, 1 castor wheel, 1 4xAA battery pack, and assorted nuts and bolts. Sharp Optical encoders are used with 3D printed encoder wheels (see Figure 7) and mounted with hot glue onto the pre-geared (before gearbox) motor shaft, corresponding break beam sensors are mounted on the underside - where the wheel encoders are - with hot glue as well.

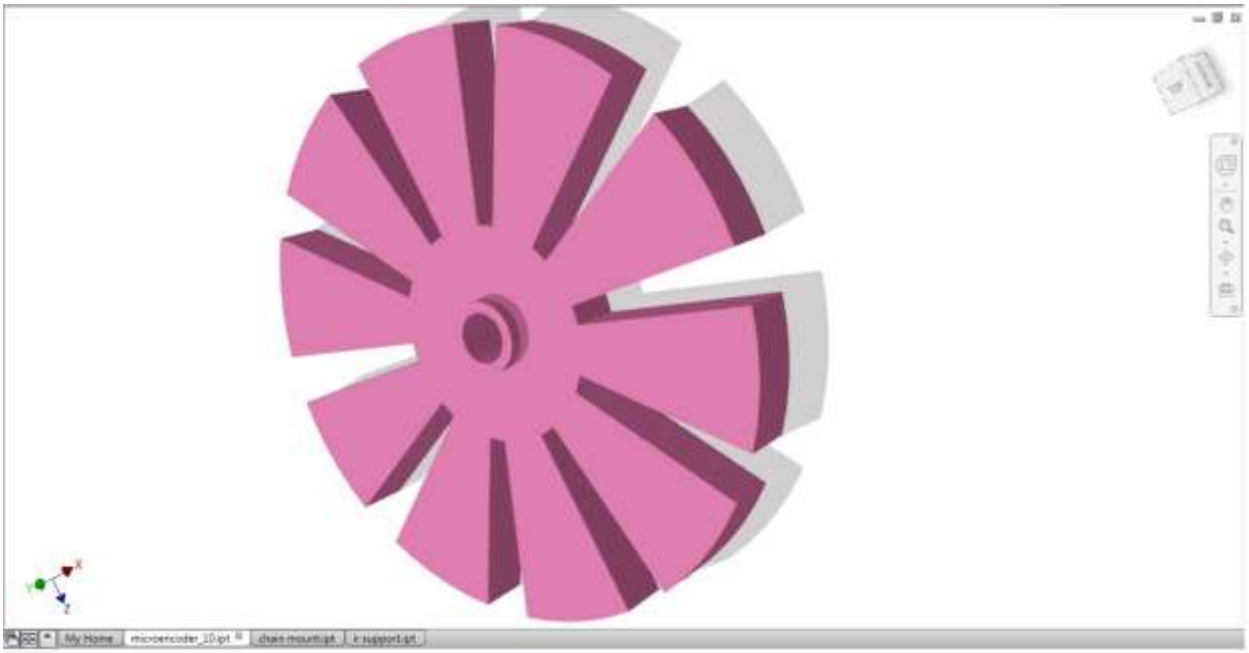


Figure 7 Raytracing rednering of encoder wheel, on Autodesk Inventor.

The arduino mega and protoboard, are attach vertically using taped elbow joints and a balsa wood block respectively.

Rbot's front scoop is shaped sheet aluminum, with recessed lip, and bent sides, creating a bowl cross-section as shown in Figure 8. This figure also shows the gate mounted on a servo motor which is sitting on a support arm that is secured by a screw and hot glue. The scoop is pressured fitted and hot glued into soft balsa wood mounted on the frame which is also mounted using the same method.

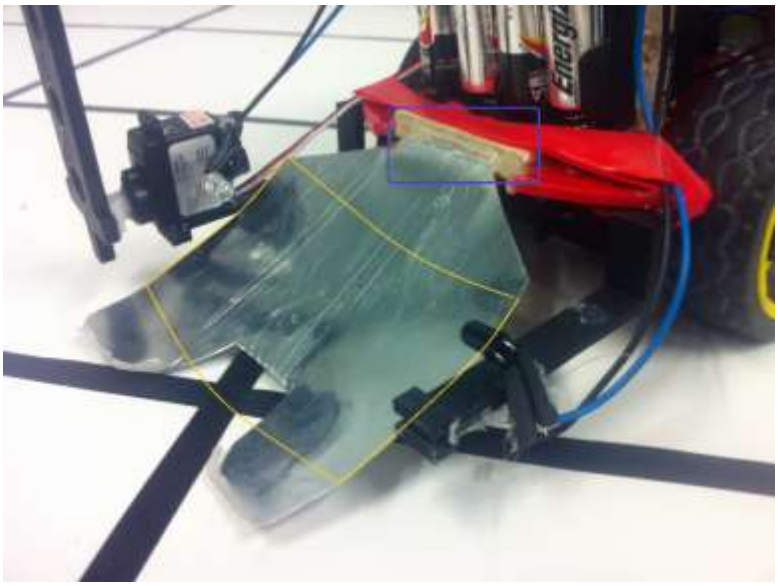


Figure 8 Photo of R-bot front section, scoop outlined in orange, and friction fitted to hot glued balsa wood in blue box.

Over the internal circuitry of Rbot sits a glued on foam covered particle board casing constructed using 2 self-threading screws, and 4 pairs of M10 bolt/nuts connected to a elbow joint acting as the bracing. This whole casing sits atop a velcro layer for ease of removal and attachment. It is shown in Figure 9.

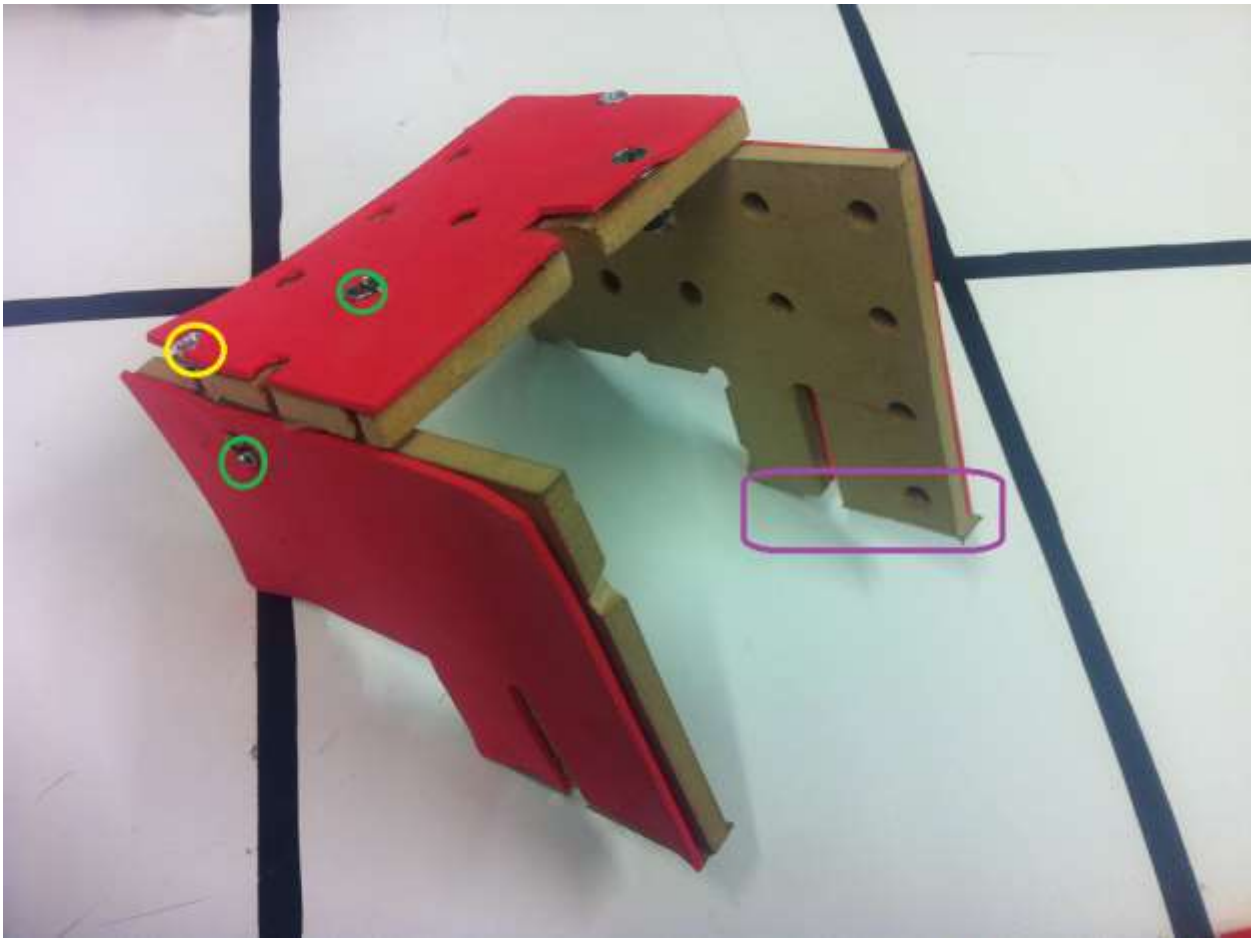


Figure 9 Foam covered particle board casing, comprised of 3 pieces held together by 2 bolts (green) and 1 screw (yellow on each side). Velcro is attached on the bottom (purple box)

Underneath Rbot, there sits a 3D printed sensor support bar, shown in Figure 10, meant to hold 3 IR modules, the 4 legs attach to the yellow gearboxes using hot glue. The sensors are attached to the bar using hot glue.

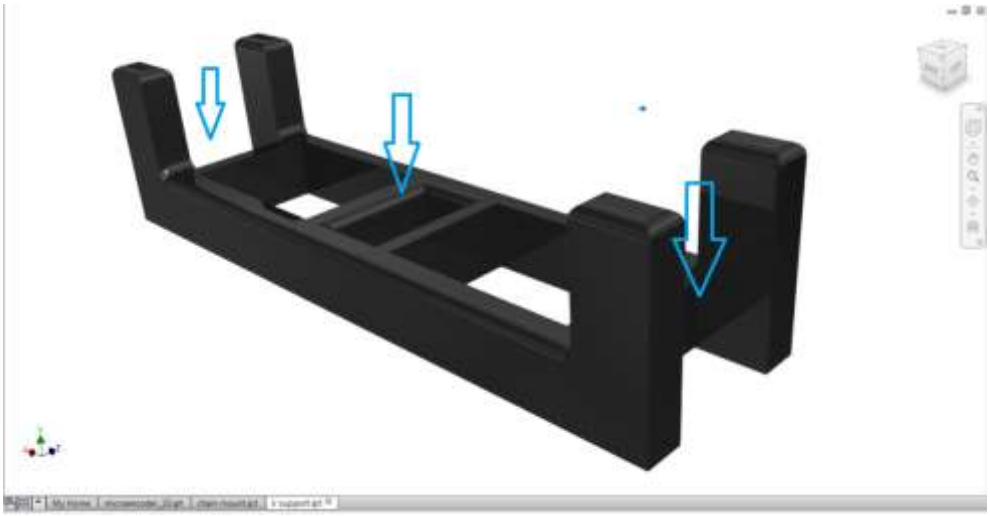


Figure 10 Raytracing rendering of IR bar. IR sensors hot glued in direction of the 3 blue arrows.

2.3.1.2 G-bot

Gbot uses the same chassis, motors, and wheels as the Rbot, save for a metal ball bearing castor. Below Gbot, there sits a balsa wood block functioning as the sensor bar, it has hole for the a 7mm photoresistor with a white LED. This block is attached to the yellow gearboxes using hot glue. In addition, Gbot has two particle board mounts flush with its long midplane, one at the back and one at the front of the robot. These are bolted to frame with a pair of M2 bolt/nuts. The particle board has a L shaped cross section, shown in Figure 11, and has a strip of velcro attached for securing the ultrasonic sensors.

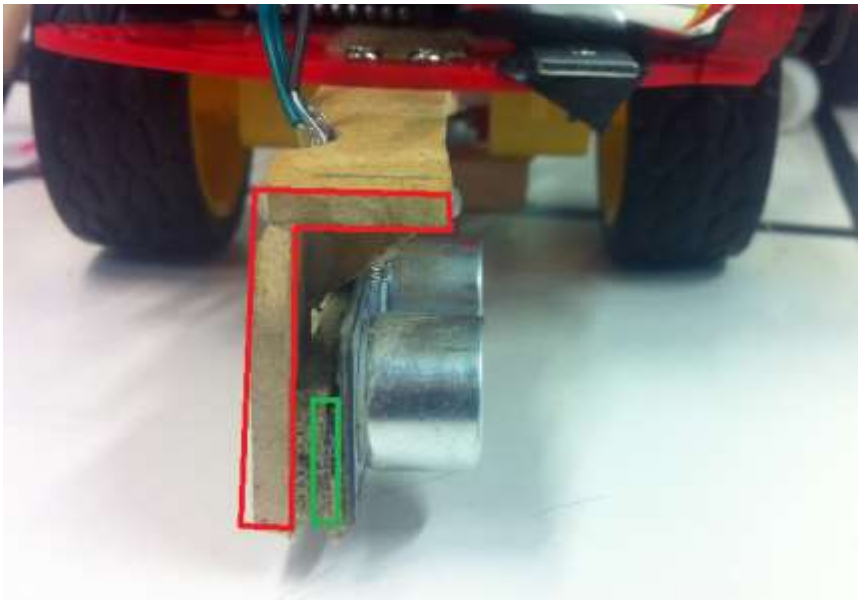


Figure 11 Photo of front sonar bar, with support highlighted in red, and velcro interface in green.

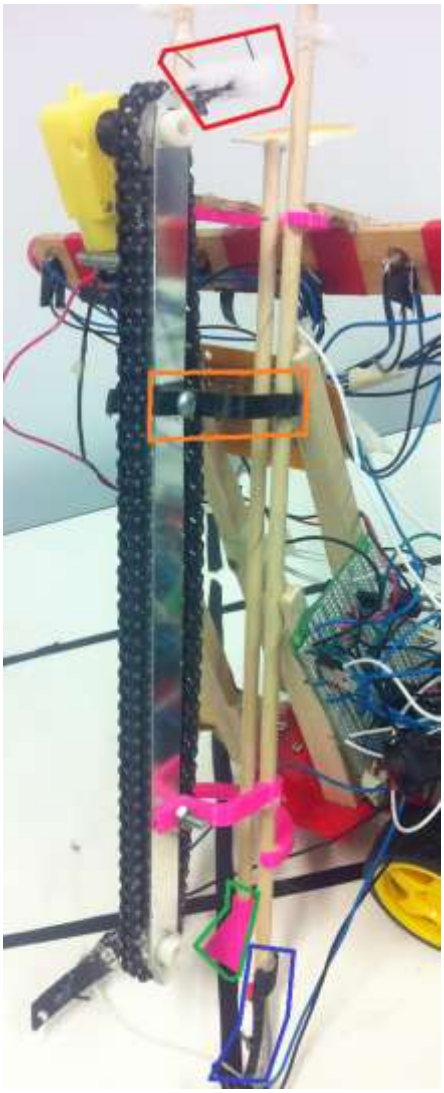


Figure 12 Ball lifting mechanism. Chain Support in orange, ball support arm in blue, foot in green, and chain mount/bristle assembly red.

The ball lifting mechanism (Figure 12) of the Gbot consists of 4 pillars supporting 2 of 3 sowel pieces forming a U shaped structure. This is supported by two 3D printed U-Brackets (pink parts). Above the U-tube structure sits the chain mechanism actuating the lifting. The chain is composed of a C shaped aluminum tube with a plastic sprocket mounted at each end to hold the metal chain. These are extended by 3D printed sprocket couplers to create a 2 point support, as shown in Figure 13. The top coupler is connected to a 3V-6V geared motor. Because this motor on rotates in only one direction, the torque produced is countered by a long M8 bolt cantilevered from the aluminum section (see Figure 13).

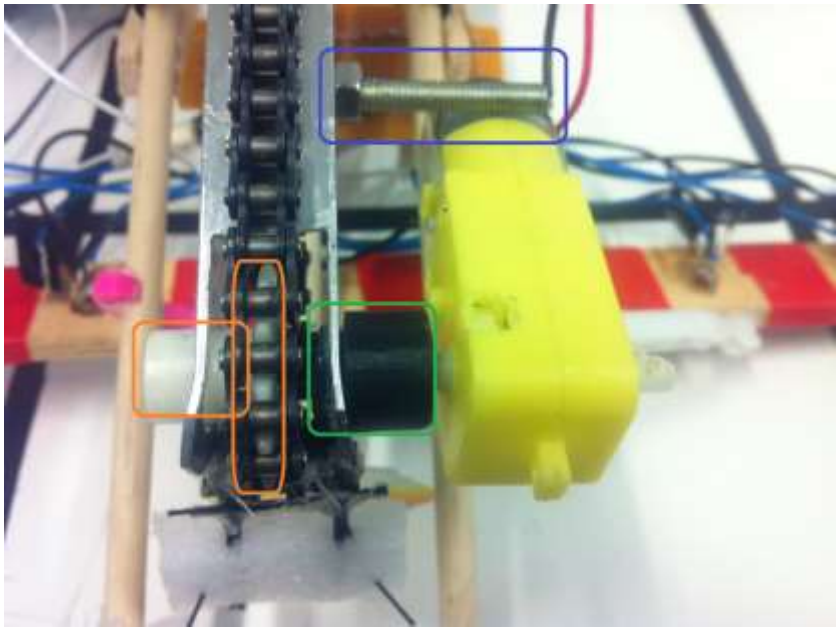


Figure 13 Top view of chain, showing sprocket (orange), sprocket coupler (green) and cantilevered bolt (blue) to counter motor rotation

On one chain link, the Chain mount (see cad) is hot glued to a scratch surface (for better adhesion) and 4-5 bristles are glued into two holes on the side of the chain mount. More hot glue is added at the base of the bristles to add support. White packing foam is cut into a rectangle and stuck through the bristles; additional bristles are stuck onto this platform at 70^{o} to prevent jamming of the ball.

Figure 14 Raytracing rendering of Chain mount, which attaches foam/bristle assembly to the chain.

The aluminum tube is supported by 3 3D printed arms mounted to 2 of the 3 dowel pieces forming the U-tube structure. The arms snap on to the round dowels, but are bolted by 2 M8 bolts to the aluminum section.

At the base of the U-tube structure is attached 2 3D printed curved arms to hold a IR break beam pair, a string to pen in the ball, and additional bristles to prevent the ball jamming.

The Gbot includes also a sensor to detect a ball dropping down a column, this is comprised of a 30cm long plywood piece with 7 holes, each hole accommodating 1 laser module and 1 10mm photoresistor covered by a red plastic filter to amplify signals (see Figure 15). The whole sensor bar is hot glued to the U-tube structure described above using 2 particle board tabs and 1 plastic arm.

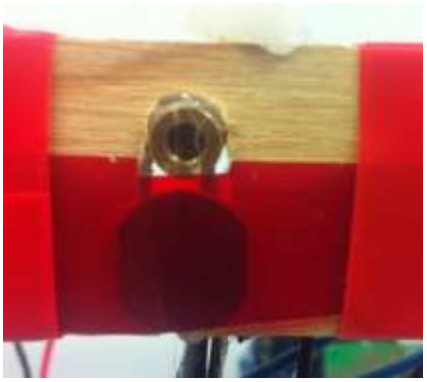


Figure 15 1 of 7 sensor pairs on the sensor bar

2.3.2 Circuitry Sub-system

This sub-section will provide the technical details for the circuitry subsystem for both R-bot and G-bot.

3.2.2.1 Circuity Sub-system R-bot

Figure 16 shows the circuitry and sensors in R-bot.

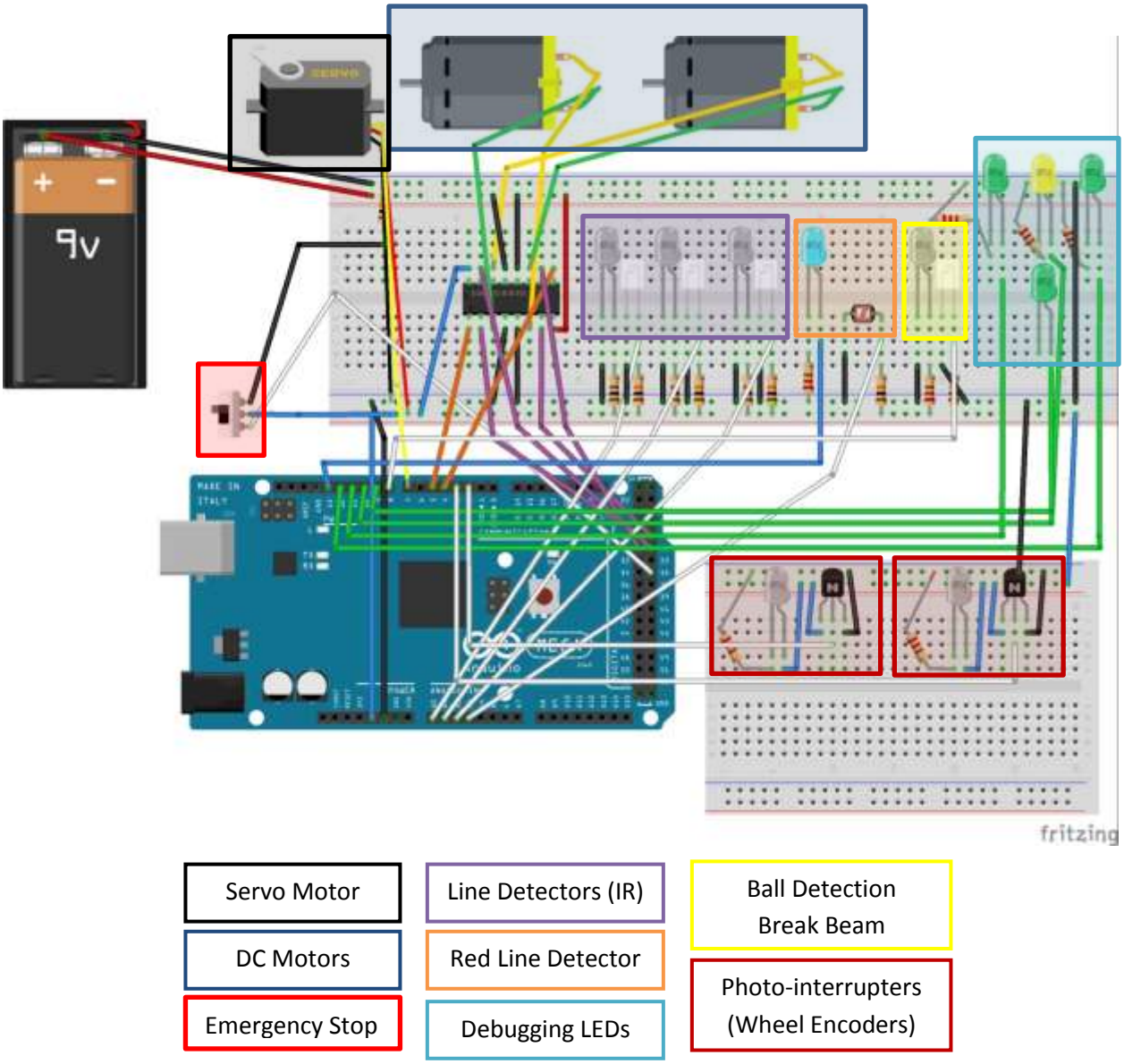


Figure 16 All circuitry, sensors and circuit components in R-bot.

Table 5 All circuit elements in R-bot that are powered by the Arduino and their power draw.

Circuit Component	Product Number	Quantity	Power Draw per Component	Total Power Draw
Servo Motor	SM-S2309S	1		0.00
Emergency Stop Switch		1	0.00	0.00
IR Detectors	TRCT5000	3	0.10	0.30 [1]
Visible Light LEDs		5	0.11	0.57
Photoresistor	OPTRE-000010	1	0.00	0.00
IR LED	LTR-4208	1	0.11	0.11
IR Reciever (Phototransistor)	LTR-3208E	1	0.00	0.00
Photo-interrupters	GP1A57HRJ00F	2	0.25	0.50 [2]
H-bridge	L293D	1	1.70	1.70 [3]

R-bot Arduino Power Calculations

The total power draw of all the components powered by the Arduino is 3.19 W. However, we also need to find the power draw of the Mega. When tested (by a third party), the Arduino Mega drew 35mA from a 9V power supply, therefore consuming 0.315 W [1]. Therefore the total power consumption from the Arduino, Sensors and Integrated Circuits is 3.5W. Over 7 minutes, a total of 1470 J of energy would need to be provided by the 4AA batteries. In addition, assuming that each battery provides a potential difference of approximately 1.5 V [2], the current draw from each battery would be approximately 583mA. Referencing the data sheet, at 500mA, the battery's capacity is approximately 1500mAh, or approximately 3 hours [2]. Therefore, the 4AA Energizer batteries should be able to provide the Arduino, sensors, and integrated circuits with sufficient power for multiple rounds.

R-bot Motor Power Calculations

The motors to drive R-bot were powered by a 9V Energizer Industrial D-cell battery. The maximum power draw from each motor was 1.125 W (250mA at 4.5V) [3]. Therefore, the maximum power draw from both the motors is 2.25W, or 250mA of current supplied from the battery. According to the data sheet, at a discharge of 300mA, the capacity is just under 400 mAh, equivalent to over 1 hour of run time [4]. Therefore, the 9V Energizer D-cell battery should be able to provide sufficient energy to the motors for multiple rounds.

3.2.2.2 Circuity Sub-system G-bot

Figure 17 shows the circuitry and sensors in G-bot. Only 3 of the 7 laser modules are shown to help simplify the diagram.

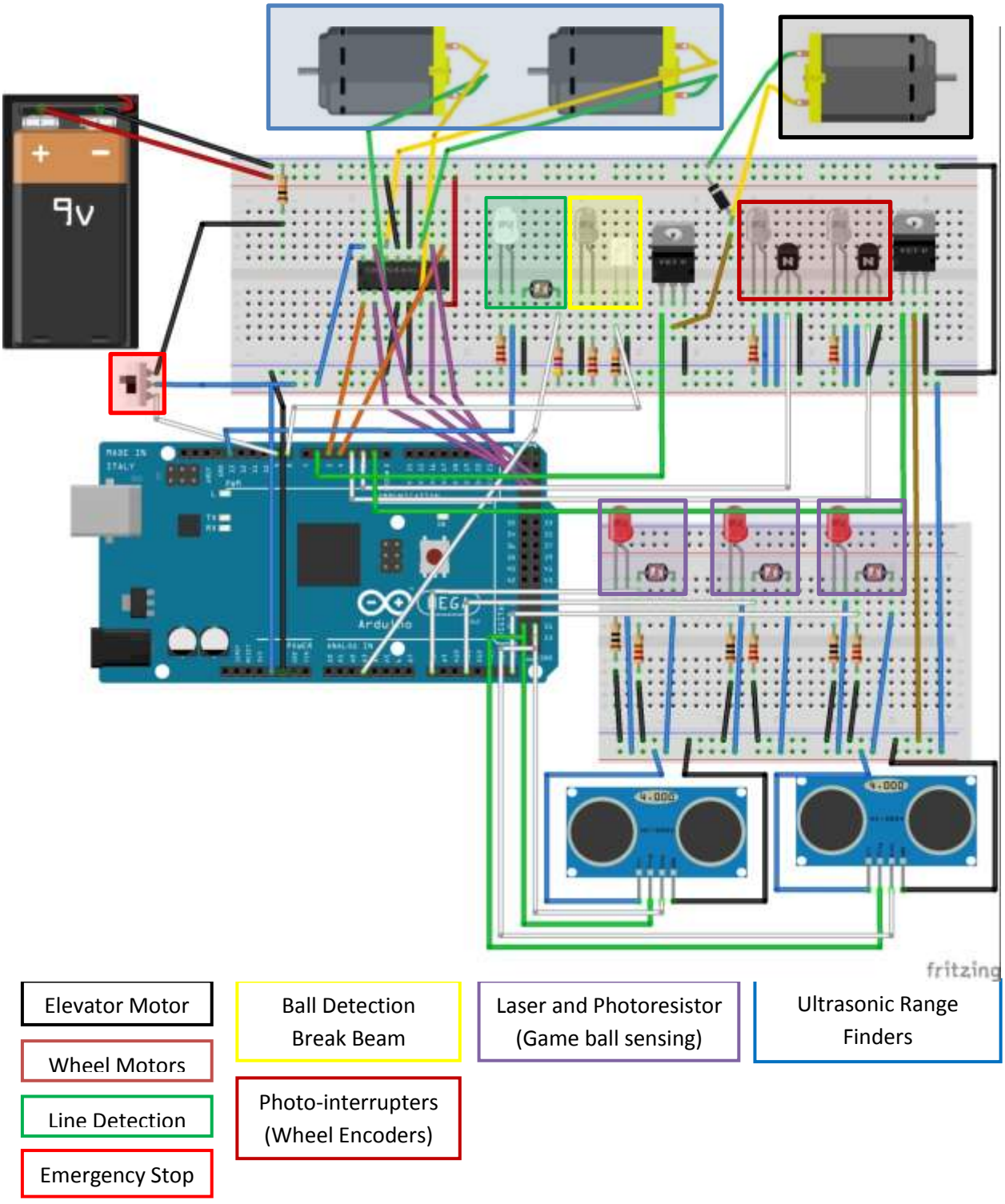


Figure 17 All circuitry, sensors and circuit components in G-bot. Only 3 of 7 identically connected laser modules are shown [1]

Table 6 All circuit elements in G-bot that are powered by the Arduino and their power draw.

Circuit Component	Product Number	Quantity	Power Draw per Component	Total Power Draw
Emergency Stop Switch	N/A	1	0.003	0.0025
Visible LED	N/A	1	0.114	0.114
Photoresistor		8	0.003	0.02
IR LED	LTR-4208	1	0.114	0.114
IR Reciever (Phototransistor)	LTR-3208E	1	0.003	0.0025
Photointerruptors	GP1A57HRJ00F	2	0.250	0.5 [2]
H-bridge	L293D	1	1.700	1.7 [3]
Lasers	N/A	7	0.005	0.035 [9]
Ultrasonic Range Finders	HC SR04	2	0.075	0.15 [10]
Mofset Transistor	BSH105	2	0.417	0.834 [11]

G-bot Arduino Power Calculations

The power draw of all G-bot components powered by the Arduino is 3.47 W. As discussed previously, the power draw of the Arduino Mega is 0.315 W [1]. Therefore the total power consumption from the Arduino, Sensors and Integrated Circuits is 3.79W. Over 7 minutes, a total of 1591.8 J of energy would need to be provided and each of the 4 AA Energizer batteries. Assuming that each provides a potential difference of 1.5 volts, each battery would have to provide approximately 631.7mA, and therefore should last just under 3 hours [2]. Therefore, the 4AA Energizer batteries should be able to provide sufficient power for the Arduino, sensors, and integrated circuits for multiple rounds.

G-bot Motor Power Calculations

The circuitry of G-bot included three DC motors, 2 for driving the wheels and one for the elevator. However, at one time, at most two motors were activated because the robot was either driving or lifting the ball, but not simultaneously. Therefore the calculations from the R-bot Motor Power Calculations can be applied, and we expect the battery to have about 1 hour of run time, sufficient for multiple runs.

Explanation of Power Calculations

This section will summarize how the values of the power draw per component were found or derived.

The power draw of the following components were found from their datasheets:

- Vishar IR Proximity Sensors [6]
- Sharp Photo-interrupters [7]
- Texas Instruments H-bridge [8]
- Mofset Transistor [9]

In addition, the power consumption for the lasers was provided on the web page that we used to order the lasers.

The power draw of the following components was derived from information from their datasheets. For example, if maximum current was provided, the power consumption was found using $P = V \times I_{max}$, where V was +5V if it was being powered by the Arduino.

- DC Motors
- Ultrasonic Range Finders (HC SR04) [10]

Lastly, the power drawn by the LEDs, photoresistors and IR phototransistors was calculated by assuming that all their resistances went to zero and using the power that would be dissipated across the resistor. The justification for this technique is provided in . For example, an LED is connected in series with a 220 ohm resistor and the power drawn by LED was assumed to be the power drawn by the 220 ohm resistor. These two circuits are shown below. The power is then calculated using $P = \frac{V^2}{R}$.

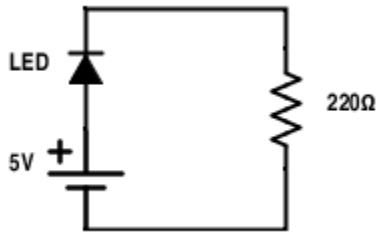


Figure 18 Circuit for connecting an LED to Arduino Power

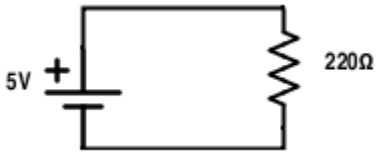


Figure 19 Circuit used to calculate power draw of an LED connected to Arduino Power

Using the method described above, the power draw of each LED was found to be 0.114 W and the power draw of each photoresistor and phototransistor was found to be 2.5 mW.

H-bridge circuitry

An H-bridge was used to control the two motors turning the wheels. For each motor, there was an enable pin and two control pins. The control pins were used to control whether the wheel rotated clockwise or counter clockwise. The enable pin was pulsed with modulation to control the total current provided to each wheel. This gave us more control, allowing us to use our PID controller and the various applications of it.

2.3.3 Microcontroller subsystem:

2.3.3.1 Parallel systems

At the highest level, the logic is split into a gameplay system and a position correction system for each robot, with gameplay run at 20Hz and correction run at 100Hz. The necessity of a dedicated position correction system was determined through testing without it and the observation of accumulated drift error from relying solely on odometry. This is due to imperfections in our measurement of wheel dimensions (relative sizes), momentum of DC motors, and the imperfect contact of the wheels to the game board.

The fundamental state of both robot includes its current position (x, y, theta) with x and y measured in mm inside an internal grid and theta in radians from [-π, π] where 0 is oriented facing the game board. Each robot also maintained a stack of targets, each with an absolute x, y, theta, and type. Its theta is the desired angle for the robot upon reaching the target and its type is the associated task to be performed

upon arriving at the target; for its full declaration, see Appendix A. Upon arriving at a waypoint, the top most target is popped and the next (if there exists any) target is automatically loaded, with potential changes in behaviour calculated by *waypoint* and *user_waypoint*.

The gameplay system directed all the actions of the robots, with the only communication with the correction system being a shared internal state, from which the position correction system would read and modify position and heading - (x, y, theta) only. See figure 20 for the control flow of the gameplay system architecture for both robots.

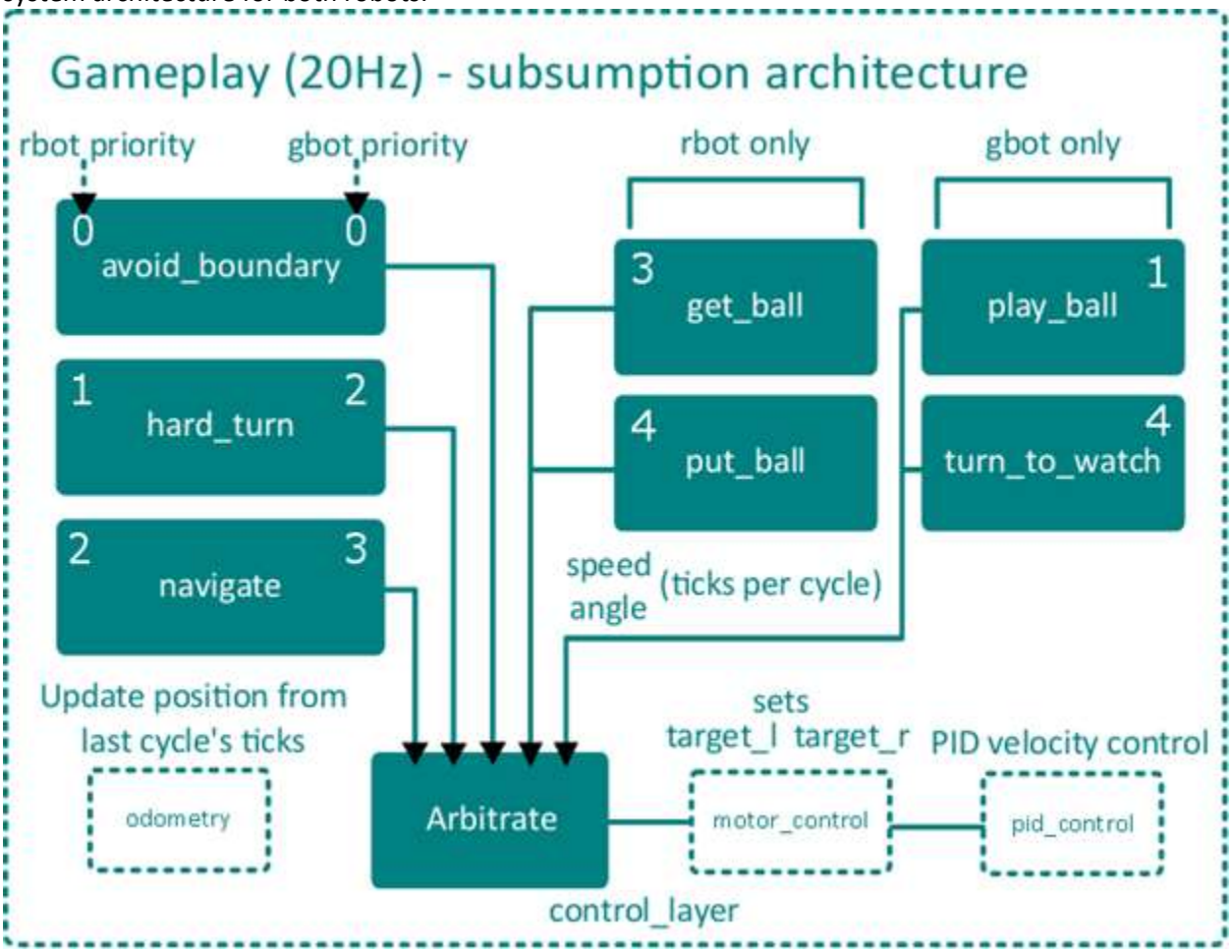


Figure 20 Gameplay systems architecture and behaviour layers with corresponding priorities for both Rbot and Gbot.

2.3.3.2 Odometry

At the start of each gameplay cycle, odometry is done to collect the ticks accumulated from the start of last cycle. Displacement and distance are found from that through the formulas below:

```
// displacements in mm
double displacement_l = dir_l * (double)instant_tick_l * MM_PER_TICK_L;
double displacement_r = dir_r * (double)instant_tick_r * MM_PER_TICK_R;
double displacement = (displacement_l + displacement_r) * 0.5;

// total distance is a scalar
if (displacement > 0) tot_distance += displacement;
else tot_distance -= displacement;

theta += atan2(displacement_l - displacement_r, BASE_WIDTH);
x += displacement * cos(theta);
y += displacement * sin(theta);
```

The MM_PER_TICK_L parameters were carefully calibrated through many trials of driving a known distance and measuring the ticks accumulated. A negative theta represents a counterclockwise turn from 0, which is pointed "up" towards the gameboard.

2.3.3.3 Gameplay layers

A subsumption architecture for the gameplay layers was used because it was best suited for the event driven environment the robot would be in, which would be much more robust than a finite-state machine approach where the robot is assumed to be in a set of defined states. In this architecture, each behaviour runs parallel to one another with a defined priority. Many behaviours can be active at once, such as *avoid_boundary* and *navigate*, but *avoid_boundary* would have control as long as it is active since it holds higher priority.

Each behaviour layer controls for speed and angle, measured in effective ticks per cycle (scaled by non-equal wheel size), and is either active or non-active. Their declaration can be seen below.

```
struct Layer {
	// speed and angle in units of ticks/cycle
	int speed, angle, active;
};
```

Layers can be activated inside itself, such as *avoid_boundary*, or inside other layers, such as *hard_turn* by *navigate* when the heading error becomes large enough.
The layers are complementary and lead to effective emergent behavior, such as a smooth turn around boundaries, as shown in figure 21 below.

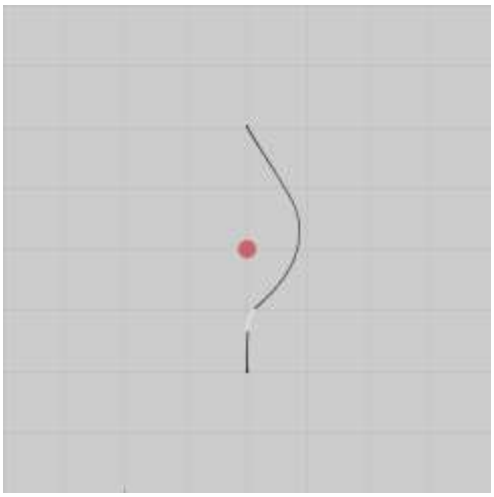


Figure 21 Navigating around a boundary; the red circle is the boundary with scaled radii, white path represents when avoid_boundary was the active layer while black path represents when navigate was the active layer.

2.3.3.4 Motor_control and PID - controls translated to motor outputs

The layers communicate only in terms of effective ticks per cycle, which as seen in Figure #### (systems flow chart) is selected by *arbitrate* to create the targeted left and right ticks for the next cycle according to the formula below:

```
target_l = control_layer.speed + control_layer.angle;
target_r = control_layer.speed - control_layer.angle;
```

For a differential drive system, the angle ticks added to the left target and subtracted from the right target completely controls navigation. A negative angle would slow the left wheel and speed up the right wheel, turning the robot right. A zero speed and non-zero angle would cause equal speed in opposite direction on the wheels, causing turning in place.
The targeted ticks are then converted into PWM output values by the PID controller using the proportional error, the integral of past error, and the rate of change of error to drive the H-bridge.

2.3.3.5 Avoid_boundary layer

This base layer is shared by both Rbot and Gbot as the highest priority layer. The reason being that colliding with a boundary would render the rest of the behaviours ineffective.
Each boundary in the array of known boundaries, which are added at run-time, is computed for their parameters as shown below:

```
Boundary& boundary = boundaries[b];
// check distance to boundary
double diff_x = boundary.x - x;
double diff_y = boundary.y - y;
// approximate each boundary as circle, from center to point - radius
boundary.distance = sqrt(sq(diff_x) + sq(diff_y)) - boundary.r - TURNING_RADIUS;
// compare this with theta to see if collision likely
boundary.theta = atan2(diff_y, diff_x) - theta;
// high threat comes from being closer and a straight hit
boundary.threat = (BOUNDARY_TOO_CLOSE - boundary.distance) *
             (BOUNDARY_TOLERANCE - abs(boundary.theta)) / BOUNDARY_TOLERANCE;
```

The threat of a boundary is negatively proportional to the distance from it as well as being negatively proportional to the size of the heading error. Therefore, heading square at a close target would produce the highest threat. When any boundary threat exceeds a threshold, *avoid_boundary* is activated and remains activated until that boundary is some distance away.

2.3.3.6 Navigate layer

Another base layer shared by both Rbot and Gbot responsible for steering and controlling the speed en route to most targets. It is activated whenever there is a valid target and the position is more than a immediate threshold (5mm) around it. See figure 22 below for a graphical representation of how a target is navigated to.

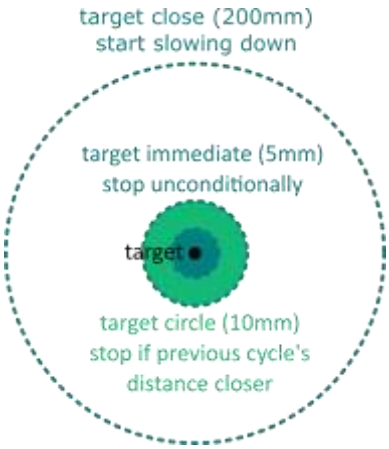


Figure 22 Navigate target rings corresponding different distances to responses.

It turns toward the target when the heading error exceeds a minimum tolerance (0.03 rads), and slows down the closer it gets to the target in order to prevent overshooting. The speed is clamped to both a minimum value to prevent stalling and a maximum value to prevent slipping.

2.3.3.7 Hard_turn layer

The last layer shared by both robots, this fundamental navigation layer turns in place by controlling speed to be 0 and angle to be non-zero. It is activated only inside the *navigate* layer when either 1. the heading error exceeds a minimum threshold (0.5 rads) when turning in place would lead to a better and faster path to target, or 2. when the target is reached but the current heading is not close enough to the desired heading at target.

The turn speed is a function of the proportion of the original turn completed, slowing down as the entire turn nears completion. This is seen in the code below:

```
to_turn = targets[target].theta - theta;
// would be faster to turn in the opposite direction
if (to_turn > PI) to_turn -= TWOPI;
else if (to_turn < -PI) to_turn += TWOPI;
// either to_turn close to turn size or even greater, kick start it (usually at start
of turn)
if (turn_size - abs(to_turn) < THETA_TOLERANCE) {
      if (to_turn < 0) turn.angle = -KICK_SPEED;
      else turn.angle = KICK_SPEED;
      return;
}
// compare against initial turn size
else if (abs(to_turn) < turn_size) {
      turn.angle = to_turn/turn_size * NAV_TURN * 3;
}
```

turn_size is defined every time the *hard_turn* layer is activated and is the initial and expected largest turn size. Small turns need to be kick started with a higher than normal target since DC motors take more voltage to start than to run.

2.3.3.8 Get_ball layer (R-bot)

This layer is activated when a target with a TARGET_GET type is reached and deactivated when the robot has sufficiently backed up enough distance (250mm after getting the ball) to clear the hoppers and allow turning in place. This mode is a closed behaviour in that it activates no other layers and is supposed to be activated only during a specific activity.

Its behaviour involves heading towards the active hopper, waiting for the ball to be in the scoop for enough cycles (5) before closing the gate for 15 more cycles, then backing out for 250mm.

2.3.3.9 Put_ball layer (R-bot)

Similar to *get_ball*, this layer is a closed behaviour activated when a target with a TARGET_PUT type is reached, which in gameplay should be at the rendezvous point after the ball is retrieved by *get_ball* and it has backed up enough to allow for turning in place.

This behaviour assumes its at the right location, so speed is 0, and turns slowly until the angle stabilizes for enough cycles at near 0 degrees (within 0.06 radians). It then opens the gate and drops off the ball, releasing control.

Watch layer (Gbot)

This layer, unlike the other layers, controls for more than one behaviour. Gbot needs to turn to face the wall perpendicularly as well as watch the game, which naturally splits into the two behaviours of *turn_to_watch* and *watch_balls_drop*. This layer is only active when not moving around, being activated after arriving at the rendezvous point or a column to drop off a ball.

turn_to_watch adjusts Gbot's theta by turning in place until it stabilizes to near 90 degrees (within 0.06 radians) for 6 cycles. When displaced from the alignment, such as being pushed by Rbot or pushed by the opponent robot, *turn_to_watch* activates again to realign itself. This behaviour is best supplemented

by the *touch_wall* correction from sonar, but does not require it as it operates only on internal heading. This behaviour is illustrated in the figure below:

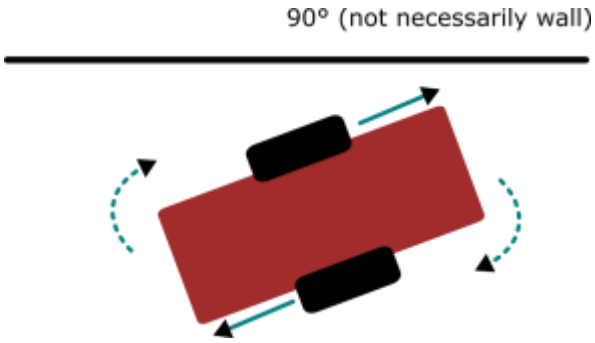


Figure 23 Turn to watch behaviour for G-bot alignment parallel to 90 degrees.

watch_balls_drop does not influence navigation at all, but simply updates Gbot's internal representation of the gameboard by detecting ball drops using the laser sensor bar. A ball drop is registered as a sufficient (> 30) sensor reading offset of only one sensor from its ambient reading for more than 3 consecutive cycles. Multiple sensors reading large offsets usually indicates reading the other robot while non-consecutive offsets likely indicate random fluctuations.

2.3.3.10 Play layer (Gbot)

This layer is activated after Gbot detects a ball in its collection area. It is responsible for navigating to the correct column (sometimes just the middle column at the rendezvous), lifting the ball after the watch layer has ensured Gbot is aligned and calibrated, and moving back to the rendezvous after the watch layer detects the ball has dropped. This is all done while keeping a constant angle parallel to the wall since the large sensor bar and close proximity to the gameboard prevents any turns.

The play layer only considers the y position (parallel to game board, perpendicular to side walls) when deciding when it has arrived at the target, since the x position fluctuates based on the sonar reading and adjusting for it would require large turns.

2.3.3.11 Path selection

Arriving at a target internally is equivalent to calling the *waypoint* function, which performs the job of loading up the next target if there are any, and deciding whether to add additional targets or not based on the previous target and the current position.

Gbot's game strategy involves only getting to the gameboard and staying there while Rbot's game strategy involves three kind of complicated paths: 1. navigating to somewhere safe from the initial starting position, 2. navigating to the hoppers from the rendezvous point, and 3. navigating back to the rendezvous point after backing out from a hopper.

2.3.3.12 Position correct system

Since each robot has a different array of sensors, their position correct systems are also going to be very different, as seen in Figure 24 below:

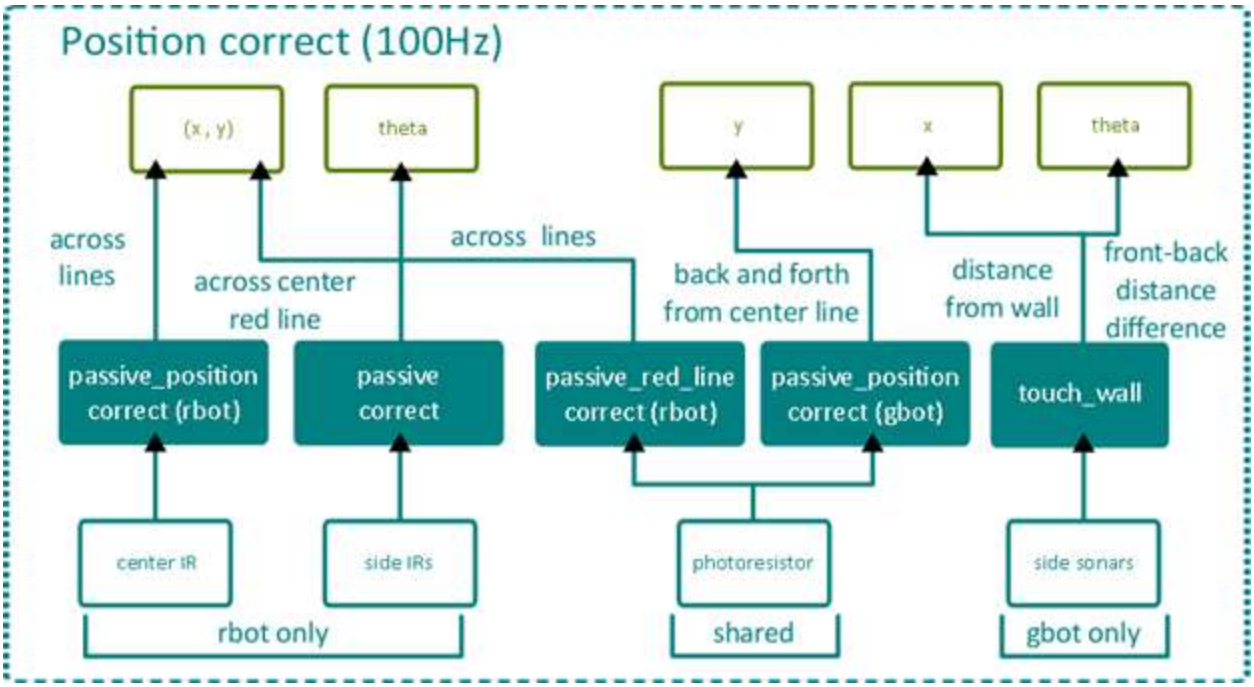


Figure 24 Position correct system architecture diagram for both Rbot and Gbot.

Both robots share a photoresistor near the center of the bottom, but use it for different purposes. As mentioned in the circuits section previous, the photoresistor coupled with an LED can detect both black and red lines without being able to distinguish between them. For Rbot, the IR sensors cannot read red lines, so when the photoresistor detects a line when the last black line was detected far enough away (>35mm), Rbot is likely over the center red line. For Gbot, its motion is simplistic enough that it does not need this extra information and can correct on every line.

All of the correction functions are passive in the sense that they do not directly control for speed or angle like a gameplay behaviour layer, only updating (x, y, theta). Separating correction and navigation allows for easier debugging and makes the robot faster since it does not interrupt the gameplay behaviour to correct.

One condition shared by all line correction functions is the need to avoid the ambiguity of intersections, as shown in figure 25 below:

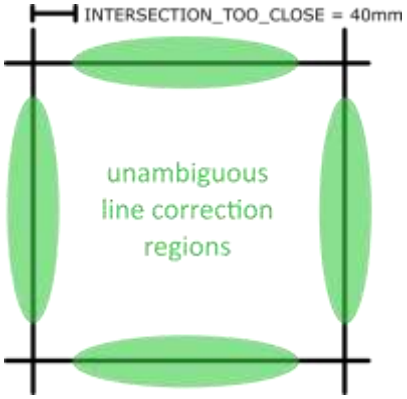


Figure 25 Unambiguous line correction regions of a grid; where the line correction functions are allowed to consider correction.

2.3.3.13 Passive_position_correct

This function is activated when a line is fully passed - when the center sensor (IR for Rbot and photoresistor for Gbot) reads no line after 3 consecutive cycles of reading a line. Leaving a line is used instead of hitting a line since it requires consecutive cycles of being on a line prior to leaving it, giving a point of checking for accidental readings.

The position closer to a grid line is rounded to that grid line, accounting for the line width by the direction of movement. This essentially splits each grid into four quadrants, as shown in figure 26 below:

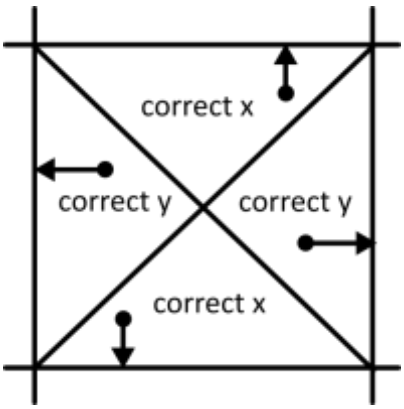


Figure 26 Passive position correct quadrants for which position to correct based on closest to grid.

2.3.3.14 Passive_theta_correct

Drift error mostly manifests in an inaccurate theta (heading), which can be corrected by assuming a straight path and considering the distance between each sensor along the wheel of Rbot first encountered a line, illustrated and explained in the figure below.

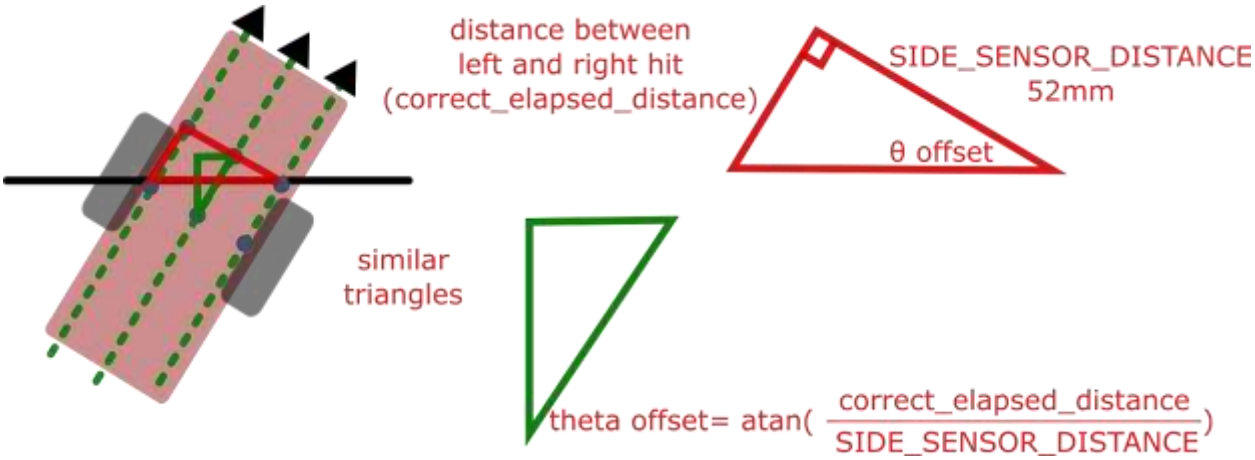


Figure 27 Passive theta correct based on the assumption of straight paths while crossing a line and using similar triangles.

The angle retrieved is a theta_offset from being normal to the line, which can be easily used to correct for theta according to the code described below:

```
float theta_candidate;
// assume whichever one passed first was the first to hit
if (passive_status & PASSED_LEFT) theta_candidate = (square_heading()*DEGS) +
theta_offset;
else if (passive_status & PASSED_RIGHT) theta_candidate = (square_heading()*DEGS) -
theta_offset;
else if (hit_first == LEFT) theta_candidate = (square_heading()*DEGS) + theta_offset;
else if (hit_first == RIGHT) theta_candidate = (square_heading()*DEGS) -
theta_offset;
// hit at the same time?
else theta_candidate = (square_heading()*DEGS);
```

2.3.3.15 Passive red line correct

As mentioned previous, a red line is indicated by the photoresistor reading a line and the last black line being read (> 35mm) away, assuming Rbot is not going backwards. Otherwise, red line correct always corrects the y coordinate to be close to rendezvous y (800mm) since that is the only place where a red line is.

2.3.3.16 Touch_wall

Two sonars at the front and back of Gbot gives absolute position of both ends. These values are averaged over 4 cycles and used to determine both the heading and the x coordinate of Gbot.

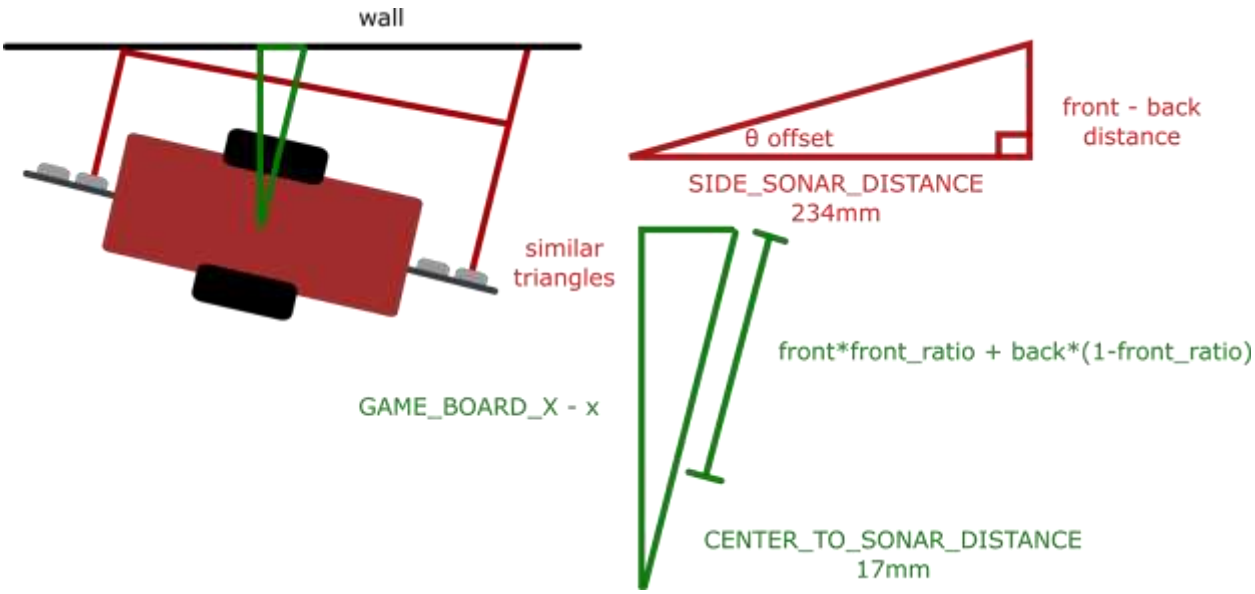


Figure 28 Sonar mechanism for correction for theta as well as x.

3. Implementation

This section will provide an overview of system level implementation before looking at each sub-system individually.

3.1 System Level

The gameplay system rests on a subsumption architecture, with a distinctive active layer each cycle that is the only point where speed and angle is controlled from. See figure (human plotted waypoints) below to see how the active layer can be easily visually represented. The entire system was designed towards ease of integrating various layers, as the core behaviours (navigation) function indepently of any potential new layers such as getting and putting the ball, which are closed behaviours.

The general strategy for integration is testing from the lowest level up, and testing each component individually with the assistance of visualization from the map user interface.

3.1.1 Integrating Get and Put ball for Rbot

Starting from the lowest abstraction level up, Rbot's ramp was tested first by manually placing a ball on the ramp and analyzing the direction of the roll and how fast it reached upon hitting the ground. The ramp's physical design was changed three times to optimizing for a small shooting angle and fast velocity at the foot of the ramp, as well as the height of the ramp such that instead of pushing the ball away from the hopper, simply slid underneath the ball to pick it up.

Next, the servo controlling the gate was tested to find the best placement and orientation. The orientation originally had the servo sideways, but that increased Rbot's turning radius by about 8mm and was later mounted sideways. The simplest code for driving a servo arm was used to test the circuit's validity, with later tests moving onto which PWM values corresponded to each orientation of the servo. The IR break beam to detect the presence of ball was tested next, particularly its placement on the servo sideways across the ramp. The sensor performed exceptionally and did not require further testing. The logic was tested last, after all the electromechanical and circuits potential problems have been eliminated. The robot's internal position can be arbitrarily set via the map user interface in Processing, and configuring Rbot's internal state to get the ball, the mock hopper was used to see how well Rbot navigated to the hopper from various starting locations and how often it failed to retrieve a ball. The number of cycles to close the gate before leaving the hopper was configured as tested to allow enough time for Rbot to move into position before backing up.

Put ball behaviour's logic was tested in a similar method, except the state is now configured to have the ball and near the rendezvous point where it would drop the ball. A frequent problem was not fully orienting 0 degrees to face Gbot perpendicularly and stopping before the red line was reached. This prompted an additionally turning behaviour inside the put layer and a seek red line behaviour implicit for navigation, which when Rbot thinks it is at the red line but have not detected one recently, it is internally placed further away from the center line to prompt it to keep going forward until it hits the line. Adding in this behaviour seems to have guaranteed red line arrival.

3.1.2 Integrating Navigation

The goal and innovation stated in the design proposal named Rbot's navigation system to be the foundational system that would enable all the other behaviours. Thus this was the system that the most time was spent refining and testing.

To test each part of navigation, a mock game field with mock hoppers, as seen in figure 29 (mock game field) were created. Walls were also used to test for Gbot's sonar.

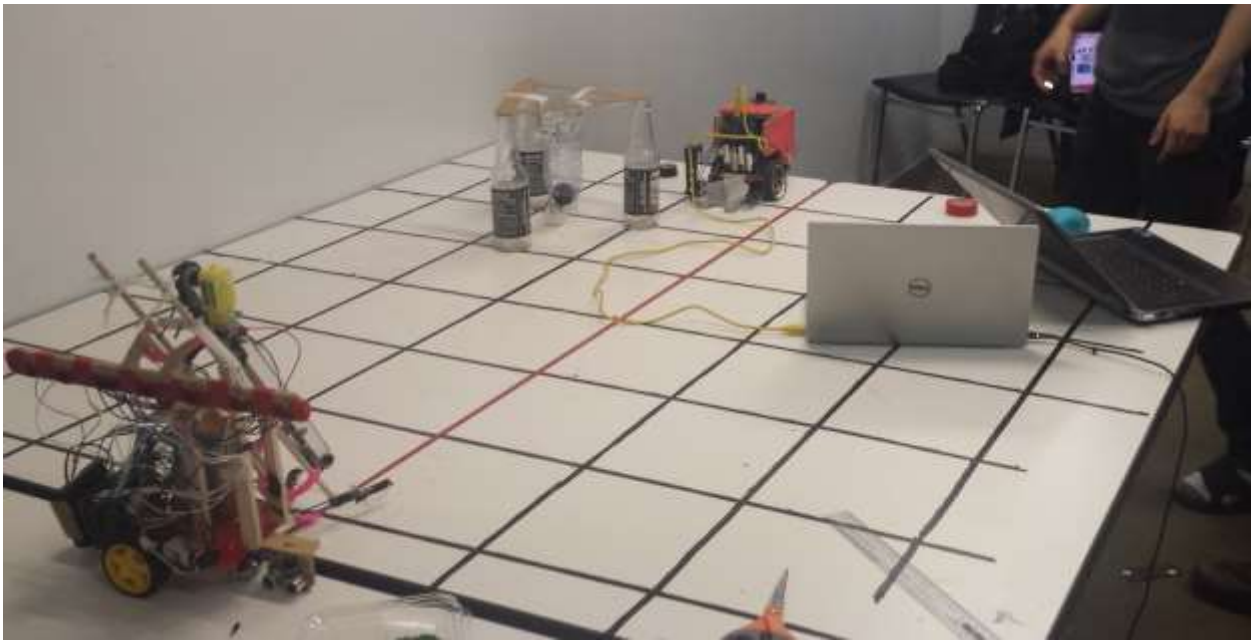


Figure 29 Mock game board and hopper for navigation and integration testing.

During the course of testing, Gbot's tolerance for turning radius was determined to be very wide - a consequence of its wide sensor bar. The solution was to set waypoints further away from the wall.

Navigation's integration came naturally with the way targets are designed - each with a heading and a type. The heading is the desired heading after arriving at the target, which were important to *get_ball*, *put_ball*, and the watch behaviours of Gbot. *get_ball* requires the robot to be pointed towards the hopper, the angle of which can be found by considering the vector from the hopper's position to the waypoint's position. *put_ball* requires the robot to be pointed towards Gbot, which would always be 0 degrees at the rendezvous point. The watch behaviours of Gbot requires theta to be 90 degrees pointed right so that Gbot was always kept parallel to the game board, even while driving backwards.

The type system gave information to *waypoint* upon arriving at a target to consider the next state. For example, if the previous target was of type TARGET_PUT, it implies that Rbot was just able to deposit a ball to Gbot and can go find the nearest hopper to retrieve more balls. The target type system provided some familiarity of a finite state machine which made it easier to debug.

During the course of testing, issues with Rbot's *passive_theta_correct* was found in being able to pick waypoints that would guarantee it many opportunities to correct drift error. This problem, as discussed later in the microcontroller section, was addressed by adding in the feature for a human to select the sequence of waypoints for each robot through the map user interface.

3.1.3 Integrating Interaction

One of the key innovations noted earlier in the design proposal was the use of two robots. As such, interaction between the two is a key challenge. This challenge had implications on the methods Rbot could use to retrieve the ball and Gbot could use to lift the ball.
An earlier idea of a sweeper arm for Rbot to retrieve the ball was not pursued as it made navigation asymmetrical and required the ball to be on one side of the robot to be effect. It lengthed Rbot's turning radius, which made turning in place difficult, and had the fundamental challenge of needing to trade off

turning radius for increases in sweeping tolerance, as a longer arm could sweep better, but required to be extended from the center of the robot.

A one way-valve retrieving mechanism for Rbot was pursued, made of a one-directional buckling configuration such as a sideways U-shaped measuring tape. This allowed the ball to be retrieved without any actuators, but depositing the ball to Gbot proved difficult. There needed to be a resistive force from Gbot to remove the ball from Rbot, but providing any such force would also shift Gbot by its castor wheel, according to Newton's third law.

Potential energy was required to be stored so that no force would be required from Gbot to retrieve the ball. This led to the use of a ramp to pickup and drop the ball. After surveying the game fields the final competition would be held on, dropping off the ball from the ramp proved inconsistent. The fields were not level, with local divots that would attract the ball and prevent it from rolling into Gbot's receiver. Placing the rendezvous point closer provided other issues such as Rbot bumping into and getting caught in Gbot's retrieving spot, leading to irreparable states requiring resets. Ultimately, interaction between the two robots was not adequately solved and proved to be the largest point of failure as Rbot was able to retrieve all the middle hopper balls, but could only successfully transfer 1 to 2 balls per game and Gbot was able to play all the balls it successfully received.

Theoretically, when Rbot turned 0 degrees to face Gbot squarely, the ball should roll down straight into Gbot, but that did not happen during testing and actual game play.

With more time, a sweeper arm at Gbot's retrieving spot would have been explored, as well as a 45 degree coned IR sensor near that region to detect Rbot's presence. The sweeper arm would extend the retrieving area a great deal and would not increase the navigation difficulty by increasing turning radius since it could be at different heights from Gbot's other components, and be normally sheathed. Alternatively, the one-way-valve and a slope from Gbot could have been explored so that Rbot deposits the ball when it drivese into the slope, with the ball falling down the other side of the slope into Gbot.

3.2 Sub-system Level

Implementation of the Electro-mech subsystem will first be explained, followed by the circuitry sub-system, and lastly the microprocessor sub-system.

3.2.1 Electro-mech subsystem

The design of both robots were dimensionally constrained by the store bought chassis kits, as it was the very first components acquired. This resulted in a space conscious design of all subsequent parts, not limited to just electro/mechanical components. A figure of the chassis used for both robots is shown below.

Figure 30 contents of store bought "Magician" chassis kit. [12]

The chassis measure 13cm X 17cm. Components were constructed with this as a basis. Construction of the stock chassis presented no difficulty, as M2 nuts/bolts were provided as well as threaded spacers. This also made attachment of custom parts easier as there were many hole/gaps on the chassis itself.

For parts of a specific shape, one that cannot be store bought. Such as a U-Bracket, 3D printers at Gerstein Science MAD lab were used to facilitate construction. Dimensions were measured beforehand and designed on AutoDesk Inventor software. These files were easily exportable to .stl and fed into a Replicator 2 3D printer. Challenges arose when attaching these parts, but were easily mitigated through the use of nut/bolts in conjunction with hot glue.

Because of the accuracy with which the parts were printed (~1mm), many components were fabricated exactly, and those that did not work were sanded or drilled into to make attachment easier. This means most challenges were to find a way to attach components suitable to their function.

3.2.1.1 Attaching Components

While most component were able to attach through bolting it or it's mount onto the chassis, some parts required holes on the chassis where there were not any. The drilling of these holes is not difficult, the challenge arises in keeping the structural integrity of the frame intact. As we learned once that a power drill generates too much torque for the brittle frame to support, thereby snapping it. Holes were then drilled at low speed with the rotary tool from then on.

3.2.1.2 The ball retrieving mechanism

As shown in previous pictures, the Rbot uses a scoop to intake the ball, and a "sword" gate attached to a serval to pen the ball in. This system was easily constructed out of stock parts available from the machine shop, and quickly attached using balsa wood and hot glue. In manually driving the robot into a hopper we first found that the scoop was too sloped and pushed balls out, this was adjusted by bending

the sheet metal more. However, in ball transfer, the less sloped scoop gave the ball less potential energy, and made transfer from Rbot to Gbot inconsistent.

3.2.1.3 The ball transfer or "handshake"

To transfer the ball, we took upon our initial experience of using a circuit wire laid across the floor to act as a potential barrier to accept balls but keep them inside. This idea has worked well with the minor modification of the barrier size through using twine (see Figure 31) instead of wire. When Rbot open the gate, the ball rolls down the slope, over the potential barrier and is kept in place for Gbot to lift it. In testing manually this mechanism worked consistently, however upon playing on the gameboard, we noticed the slight slope of the gameboard counteracted the kinetic energy given to the ball by our sloped scoop, resulting in a few missed balls.

Figure 31 Picture of base of lift mechanism, foam/bristle "pusher" in white, anti-jamming bristle boxed in red, and usual cause of jamming marked by blue star.

3.2.1.4 The ball lift mechanism

The main actuator of the lift is a chain driven by 3V-6V motor. As the chain rotates, one link has attached a mount that connects to a broom bristle/foam assembly which pushes the ball up a U-tube structure comprised of 3 dowel pieces and 2 3D printed U-Brackets. Construction was facilitated by access to a a aluminum C cross sectioned tube. Main difficulty of this mechanism arises during activation of the lift, when the bristle/foam pusher is just about to reach the ball; the ball jams due to the angle of the force being applied (see figure 31). This was quickly resolved by attach more bristles to the foam at a perpendicular direction, thereby pushing the ball into a more favorable position before the pusher applies most of the force.

3.2.1.5 Possible improvements

From gameplay experience, our team has reflected that a critical point of failure was the "handshake" or ball transfer between Rbot and Gbot. The mechanisms described above is sensitive to errors in position and angle. A better transfer would have more tolerance of these almost certain errors. Additionally, instead of using the stored potential energy of the ball, to consistently transfer the ball regardless of the level of the board we would implement another servo in place of the scoop, and actively push the ball when desired. A more drastic change, would be to combine both robots into one, entirely nullifying the handshake issue.

In construction, we would first consider the necessary components that would be in the robot, then make an informed decision about the shape and size of the chassis. In retrospect, using store-bought kits limited our scope of investigation unnecessarily, as a custom frame can be built to support more components or mechanisms that were ruled out by the small frame.

3.2.2 Circuitry Sub-system

This section will summarize the component testing and justify why the sensors on the final robots were chosen. Some of the difficulties will be explored in the component selection section. Finally potential improvements to the circuitry will also be discussed.

3.2.2.1 Component Selection

This section will summarize the component selection process for the following sensing tasks:

- Sensing lines on the game field
- Sense if robot has gameball
- Sensing distance from game wall (G-bot)
- Sensing ball played in game board

Sensing lines on the game field

Two possible sensors were considered for use when detecting lines on the game field. The first was an IR Proximity Sensor (Vishay TCRT5000) and the second was a photoresistor with an LED. Their advantages and disadvantages are summarized in the table below.

Table 7 Advantages and disadvantages of the IR proximity sensor and Photoresistor for detecting game lines

Sensor	IR Proximity (TCRT5000)	Photoresistor
Advantages	Very large difference in reading between black lines and white board (~800-900 vs. ~200-300 for Photoresistor) Sensor contains both emitter and detector, easier to mount and position	Able to detect red line Slightly cheaper (LED + 7mm Photoresistor is about 15 cents cheaper than IR proximity module) Filters can be used to enhance detection of specific colors
Disadvantages	Unable to sense red line Filters cannot be used to adjust detection of specific colors	Less significant change in readings for black line and white board Very susceptible to changing ambient lighting conditions. Adding LED reduces but does not eliminate effect More difficult to mount (LEDs must be mounted next to Photoresistor)

R-bot's primary method of line detection uses three IR proximity sensors to detect the line and update our heading. However, it also has a Photoresistor (with an LED) to detect the central red line to update its position on the gameboard. We chose to us IR proximity sensors as the primary method of detecting

lines because they were more reliable than the photoresistors. They provided a larger change in readings and were not susceptible to external lighting conditions. Since R-bot was doing most of the driving and required the more accurate navigation system, we decided to use the more reliable sensors.

On the other hand, G-bot used one photoresistor and LED to detect game lines. Since G-bot would constantly be near the board, redline detection was extremely important for its navigation system. Therefore, the photoresistor was used. Only one photoresistor was used because the ultrasonic range finders were providing additional information regarding the position and heading of the robot.

Sense if robot has ball

Both R-bot and G-bot have to successfully complete this task. R-bot senses if it has collected the game ball from the hopper and G-bot senses if the ball has been successfully transferred. Three sensors were identified as candidates for this task: IR Emitter/Receiver pair (LTR-3208E/LTR-4208), IR Proximity Sensor (TCRT5000), and Laser-Photoresistor pair. The advantages and disadvantages of each sensor combination is described in the table below.

Table 8 Advantages and disadvantages of the IR Emitter/Receiver pair, IR Proximity Sensor, and Laser-Photoresistor combination for detecting if the robot has the ball.

Sensor	IR Emitter/Receiver pair (LTR-3208E/LTR-4208)	IR Proximity (TCRT5000)	Laser-Photoresistor
Advantages	Very large change readings even when emitter/receiver very far apart. Readings change by 1000 (for both black and white balls), separation was 5 inches	Easier to position, only one module with both emitter and receiver No concerns about misalignment / calibration	Very easy to reposition and detect if problem is misalignment Reliable readings, change of about 500 for black and white balls when separation is about 5 inches
Disadvantages	Difficulty to accurately position and recalibrate because light beam not visible Cannot easily tell if problem is misalignment	Ball must be very close to IR proximity for reliable readings (< 1 inch) Very small change in readings between ambient and black ball	Very small range where system works because light from laser is so focused, not robust

Both R-bot and G-bot used the IR Emitter/Receiver pair for detecting if the robot has the game ball. They were chosen because they were the most robust system that could reliably detect both black and white game balls.

Sensing distance from wall

This task is specific to G-bot and was used when navigating to the game board to ensure correct positioning. Two candidate sensors were considered for this task: bumper switches and ultrasonic range finders. The advantages and disadvantages of both sensors are summarized in the table below.

Table 9 Advantages and disadvantages of bumper switches and ultrasonic range finder to detect the distance from the wall

Sensor	Bumper Switch	Ultrasonic Range Finder
Advantages	Binary data from bumper switch is easier to process Most reliable data, no forms of interference	Continuous data from ultrasonic range provides more information (ex: heading) Nicely supplements navigation by providing information regarding position and heading to update internal position
Disadvantages	Does not provide information above is this part of the robot sufficiently close to the game board To integrate with navigation, would most likely require driving into gameboard, causing wheel to slip, making navigation much more unreliable	More complicated to use data input from Ultrasonic Range Finder Could give readings very different from internal position Susceptible to interference from other Ultrasonic Range Finders

G-bot used the ultrasonic range finders to determine its position from the game wall and gameboard because it was the best fit with our navigation system, and provided us the most information about our position.

Sense ball playing in gameboard

This was the most difficult task for component selection and finding adequate sensors became an obstacle. Ultimately, two sensors were given significant consideration for this task: IR Proximity sensors (TCRT5000) and Laser-Photoresistor combination. The advantages and disadvantages of both are described in the table below.

Table 10 Advantages and disadvantages of the IR Proximity sensors and Laser-Photoresistor (with filter) combination for sensing a ball being played in the game board.

Sensor	IR Proximity (TCRT5000)	Laser-photoresistor (with filter)
Advantages	One unit, mechanically simpler to implement Very clearly detects white balls falling	Can detect both white and black balls with a significant (difference ~100-200) Robots in the background should not be an issue (determined in a controlled testing environment)
Disadvantages	Can barely detect black balls falling, difference of approximately 30. A robot in the background would provide similar readings to a black ball falling, not robust/reliable	More expensive, approximately twice the price of IR Proximity sensors False positives on gameboard

G-bot used the Laser-photoresistor combination because it gave the more reliable results. However, outside of the controlled environment, on the gameboard, it did 'sense' game balls that were not being played.

3.2.2.2 Possible Improvements and Alternate Component Selection

For the first three sensing tasks described and analyzed previously, the sensors worked reliably and were able to be successfully integrated with the design. Therefore, alternate components would not be selected for these tasks. However, the fourth sensing task was not completed reliably by the sensors and therefore is an area for possible improvements where alternate components should be considered. One type of sensor that was not tested but would have a high likelihood of being successful would be mechanical switches that would activate when a fall ball hits them. This mechanical solution would be able to reliably sense a ball falling. However, it would also introduce additional complexity because it would prevent G-bot from being able to navigate between different columns on the gameboard. To solve this, a servo could be used to rotate the column. In one orientation, the mechanical switches would be pointed away from G-bot, towards the game board, while in the other the switches would be pointed downwards, allowing the robot to drive back and forth between columns.

3.2.2.3 Changes during implementation

This section will summarize the changes made to the circuitry during Implementation and briefly justify each change. The changes for R-bot are listed below:

- One of three DC motors was changed to a servo motor.
  Justification: change in design for ball collection
- Sensing lines on the gameboard: Photoresistors to IR Proximity Sensors
  Justification provided previously 3.2.2.1
- Reduced H-bridges from 3 to 1
  Justification: 2 DC motors can be controlled via 1 H-bridge to reduce costs (no tradeoff in performance), one DC motor was changed to servo motor which requires no H-bridge
- Added emergency stop button
  Justification: Safe practice, constraint

The changes to G-bot circuitry are listed below:

- IR Proximity Sensors changed to Laser and Photoresistors (with filter)
  Justification: Provided earlier 3.2.2.1
- Added IR emitter/receiver pair to detect when ball is transferred
  Justification: Provided earlier 3.2.2.1
- Reduced H-bridges from 3 to 1
  Justification: 2 DC motors can be controlled via 1 H-bridge to reduce costs (no tradeoff in performance), other DC motor controlled via mofset transistor (loose ability to reverse motor but not important for functionality)
- Added emergency stop button
  Justification: Safe practice, constraint

3.2.3 Microcontroller sub-system:

3.2.3.1 Debugging framework

Internal position was crucial to communicating between the gameplay and correction systems, as well as between behaviour layers inside the gameplay system. To effectively test out any behaviour, a

debugging framework allowing visualization of position was required. This challenge was met by using Processing, which allowed easy integration with Arduino as it communicated via COM ports (same as Arduino), and produced easily readable maps such as figure 32 below.
Each cycle is plotted as a dot with the colour representing the active layer, allowing for easy tracking of motion in conjunction with serial printing.

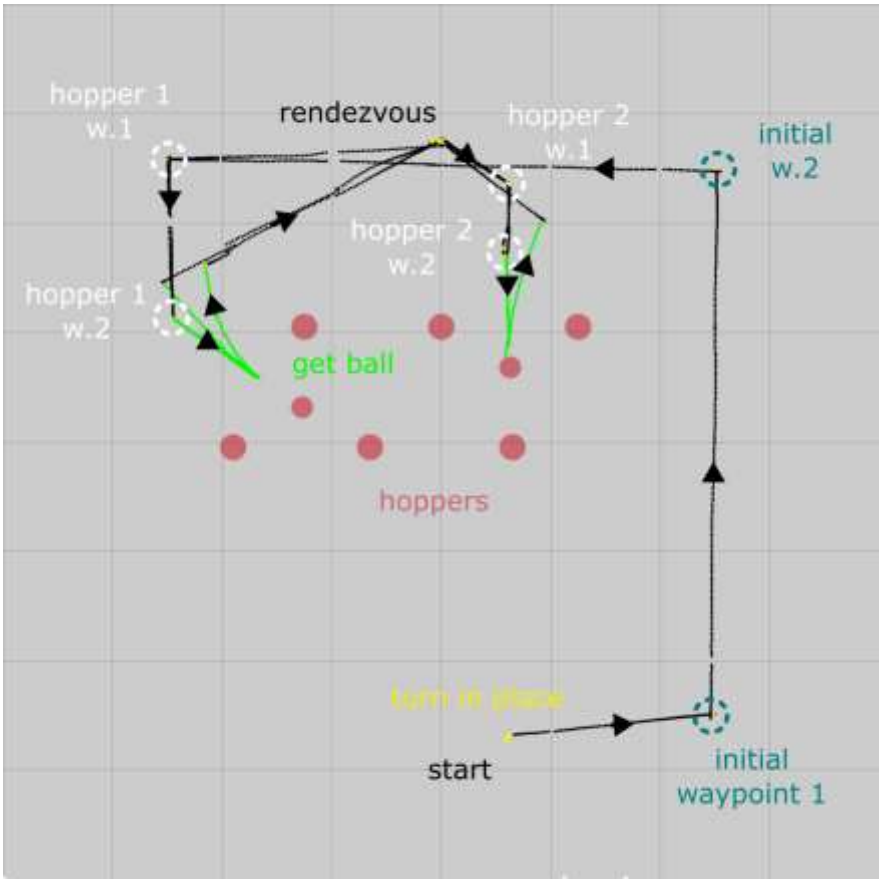


Figure 32 Debugging map of an experimentation run with the returning path (path 3) being the shortest path to rendezvous with various active layers labelled.

Another dimension of debugging is testing out each small component in isolation. Individual debugging programs for optical encoders, IR sensors, sonar, H-bridge drivers, PID controllers, gate opening, ball lifting, ball sensing, and boundary avoidance allow better tracking down of bugs. Each can be found in the debug directory under Arduino.

3.2.3.2 Changing Environment

The IR and photoresistor sensors are sensitive to environment factors such as lighting levels, which changes with both time and position. In addition, the power supply likely diminishes over time, lowering LED brightness. This is an especially large problem for Gbot's sensor bar. The solution was to calibrate when given the opportunity by measuring the highest and lowest values over a time period (by default 5s) and taking the threshold as low + (high-low)*THRESHOLD_TOLERANCE for each sensor, where a threshold tolerance of 0.5 would mean the average of high and low values.
Gbot calibrates its sensor bar whenever it is stationary and about to lift a ball, so as to calibrate as close to when an expected deviation was expected as possible.

3.2.3.3 Key Challenges

The largest challenge was correcting for drift error from wheel slippage and imperfect measurements, which created large errors in heading and led to internal position being off by one grid - at which point correction cannot bring the robot back to its actual position until the red line is hit.

By default, Rbot selects the shortest path between the waypoints, and while that provides the fastest way to play the game, it does not lead to good opportunities for position correction. The solution is to allow human input on the sequence of waypoints: all routes are plotted by a human controller at the start of the game during the 3 minutes after the hopper configuration has been shown by clicking on a map user interface in Processing, as shown in figure 32 above. The waypoints for paths 2 and 3 are the same, but are arrived at in reverse of each other, with some modification to target types of the last target.

A human can very easily select paths that would lead to easy passive line correction, staying away from intersections and sharp angles near lines. This solution is much easier and robust than programming in automated selection criteria.

On the other side of the abstraction spectrum, the other key challenge was having the desired speed and angle translate into physically convergent behaviour. Oscillations often occurred where the previous cycle's target angle and speed overturned.
This was solved by programming in dampening factors in simulation of a "leaky integrator" where each trial the factor would be multiplied by 0.9 such that the magnitude after each cycle converges to 0. Timed pauses in the form of *hard_break(activating_layer, cycles_to_break)* were also added before and after turning in place as that is the most likely failure point for creating drift error.

Another challenge was keeping track of all the behaviour layers, their interactions, and how the position correct system interacted with them. A total of 3207 lines of library code was written for Rbot and Gbot combined, the size of which by itself becomes a challenge to manage. The solution to this was to use version control in the form of git, with commits allowing flexible reverts in case any recent change made significant detriments to the system. Keeping access points in one place also lowered the complexity significantly - having the position correct system communicate only by correcting position and heading, leaving all the speed and angle control to behaviour layers, keeping all processing after arriving at a target to the *waypoint* function, and inside each behaviour dealing with how to control for speed and angle independently.

3.2.3.4 Possible improvements

Considering how much time the navigation system required, a simpler, less versatile navigation system could have been pursued. This would have led to earlier debugging of other components and integration. Line following with a simple sensor bar would have likely worked adequately, and getting the ball from the corner hoppers could have been explored.

Overall, too much complexity was attempted, which while successfully met, took too much time to implement and resulted in not enough time to test the integration.

4. Project Management

4.1 Project Schedule: Revised Gantt Chart

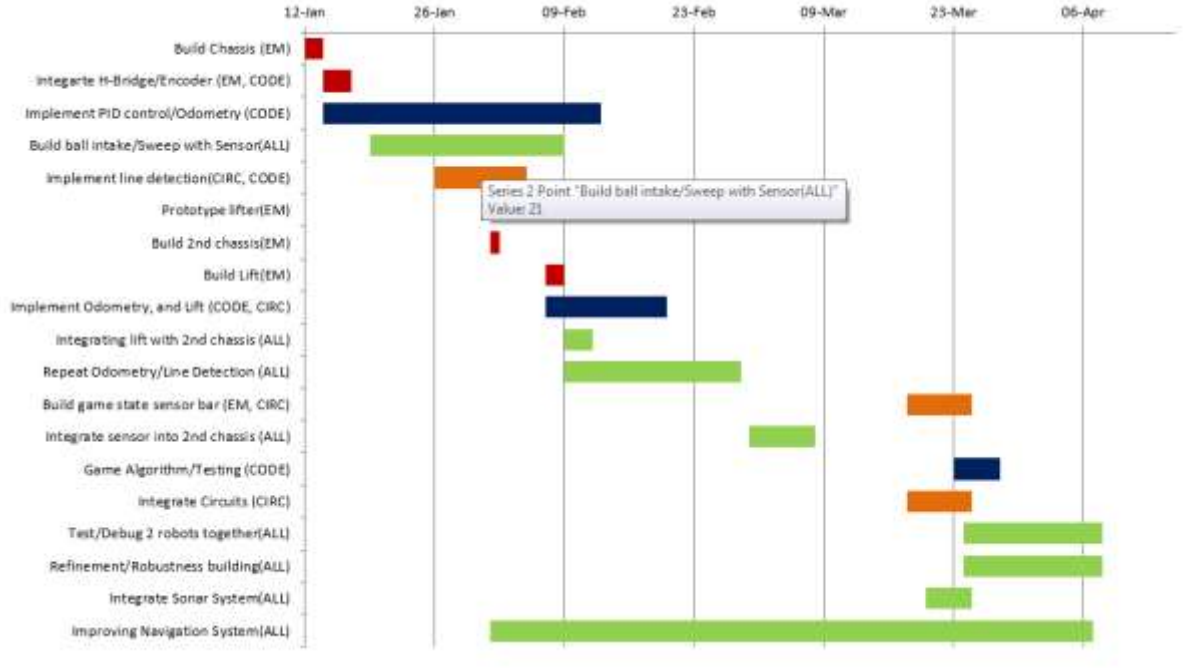


Figure 33 Actual Project Schedule.

Some things of note are the reduction in individual tasks such as creating electromechanical parts or soldering circuits onto PCB's. This is in contrast to the extension of integration tasks (green) which involved all 3 roles. A longer than expected task was creating a reliable navigation framework. This involved the design and execution of new mechanical parts, circuits, and programs. This is reflected by the long green bar at the very bottom. And, though not apparent in the chart, creating navigation forced some tasks, such as testing 2 robots, refining robustness, to be delayed until navigation was at a state where the robots could navigate consistently.

4.2 Division of Labour

Table 11 Division of Labour for AER 201.

Role	Electromech work done	Circuits work done	Microcontroller work done
Electromech	-Fabrication and assembly of components and robots -Mechanical design ideation to solve new problems -Implementing new mechanical solutions	-Soldering and debugging of some components	-Helping debug some programs
Circuits	-Mechanical design	-Planning and	-Improving robot behavior

	ideation to solve new problems	executing final PBC board scheme -Soldering circuits -Testing and selecting sensors to be used	with ideas for new programs
Microcontroller	-Mechanical design ideation to solve new problems	-Soldering and debugging of some components	-Design and execution of C++ programs -Improving robot behavior with new programs -Debugging programs using self created debugging frameworks

4.3 Budget

The final budget for the two robots is $239.23 dollars. Prices are summarized in Appendix B. Receipts included in external file folder "Receipts"

5. Conclusion

From the beginning, our team wanted to design a unique 2 robot solution to the problem. We hoped the final design would embody our values of simplicity, robustness, redundancy, and speed. And our designs of the ball intake, ball lift satisfied our desire for simplicity, while precise odometry supported by passive line detection and range sensors gave our robots redundant navigation systems. Having two robots working semi-parallel to each other also made the whole system work faster together as a whole, for they each worked independently.

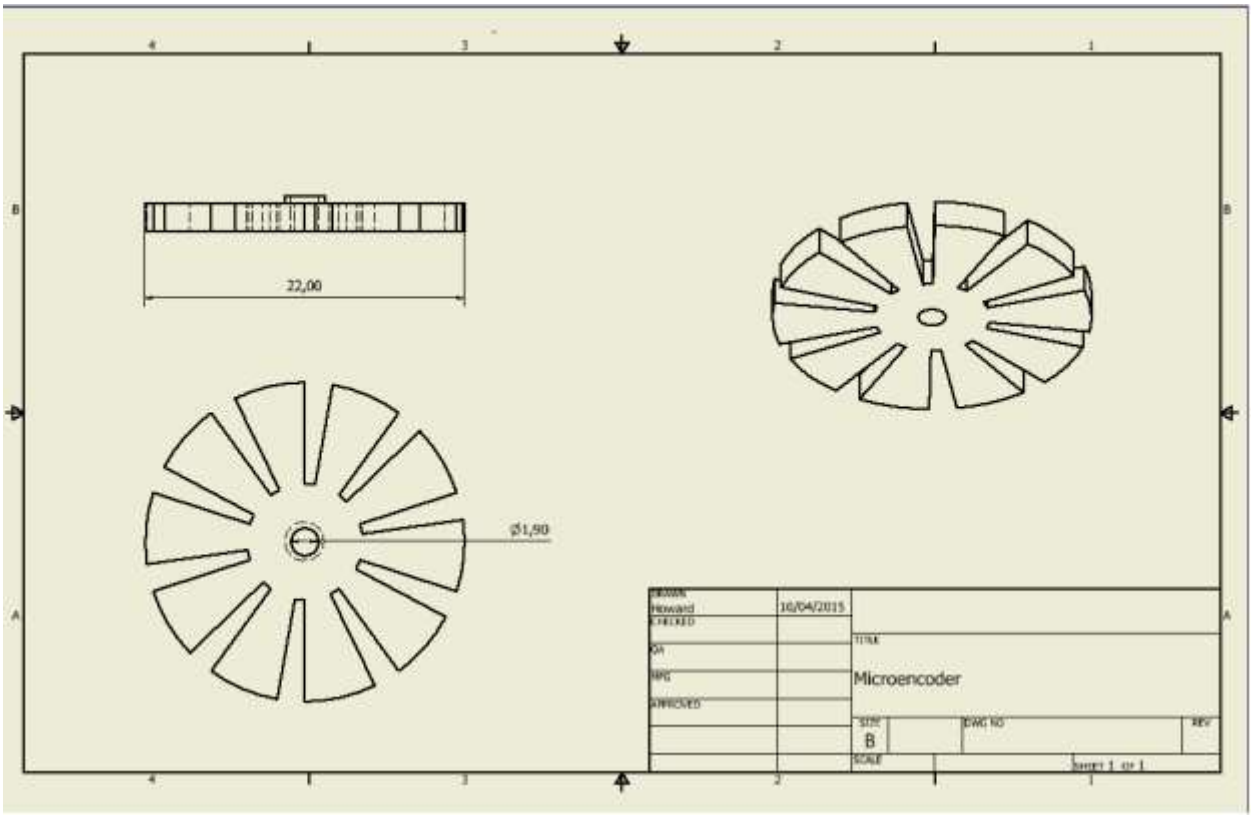
We've seen all of the above factors working in our team's favour during the competition, when Rbot managed to retrieve balls from the middle hoppers, which few teams aimed to do. The robustness of our navigation system meant our robots could get stuck and slip for a bit with no divergent consequence. In all our rounds until we were sudden-death eliminated, our two robots always played at least 1 ball.

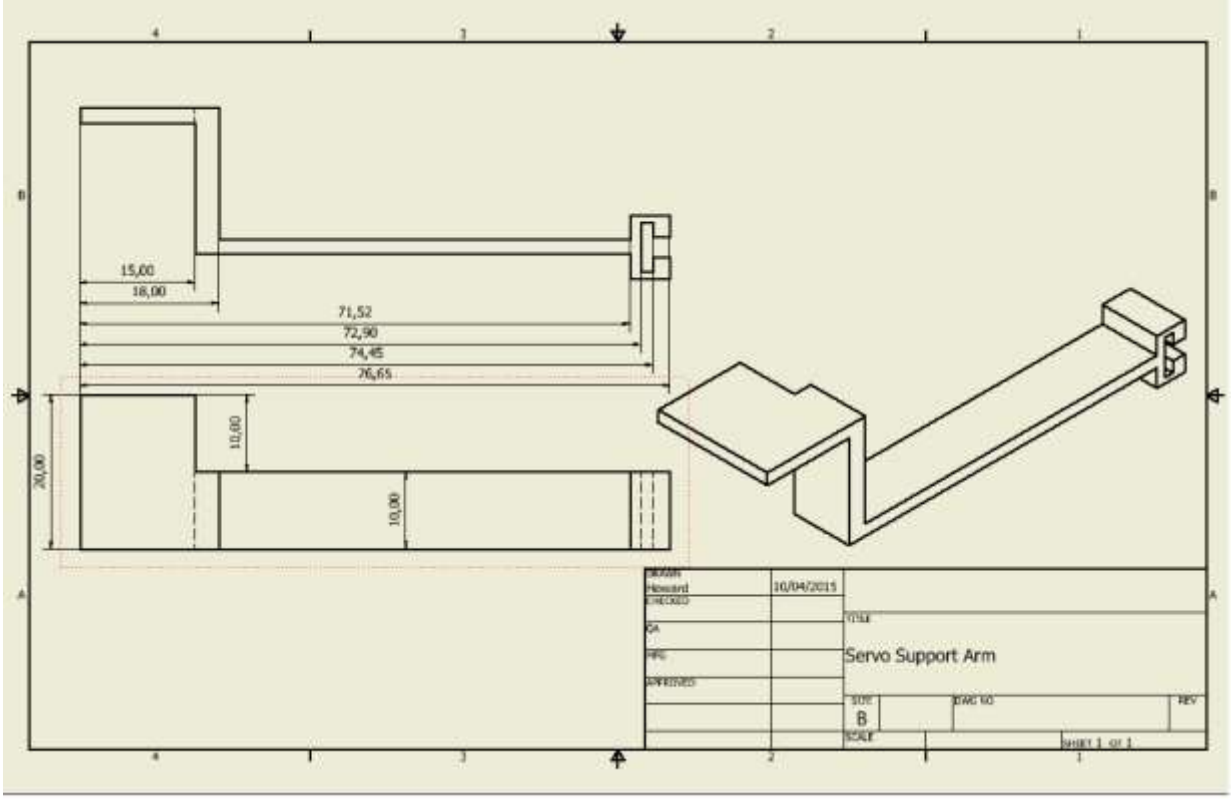
However, there are some crucial stages along the process without which our system would not work. We had not anticipated the inconsistency of the ball transfer or "handshake" between Rbot and Gbot, nor had we thought that we might play 2 rounds with the most challenging hopper configurations, where a 5mm error in position could mean a successful transfer or Rbot hitting a hopper pillar.

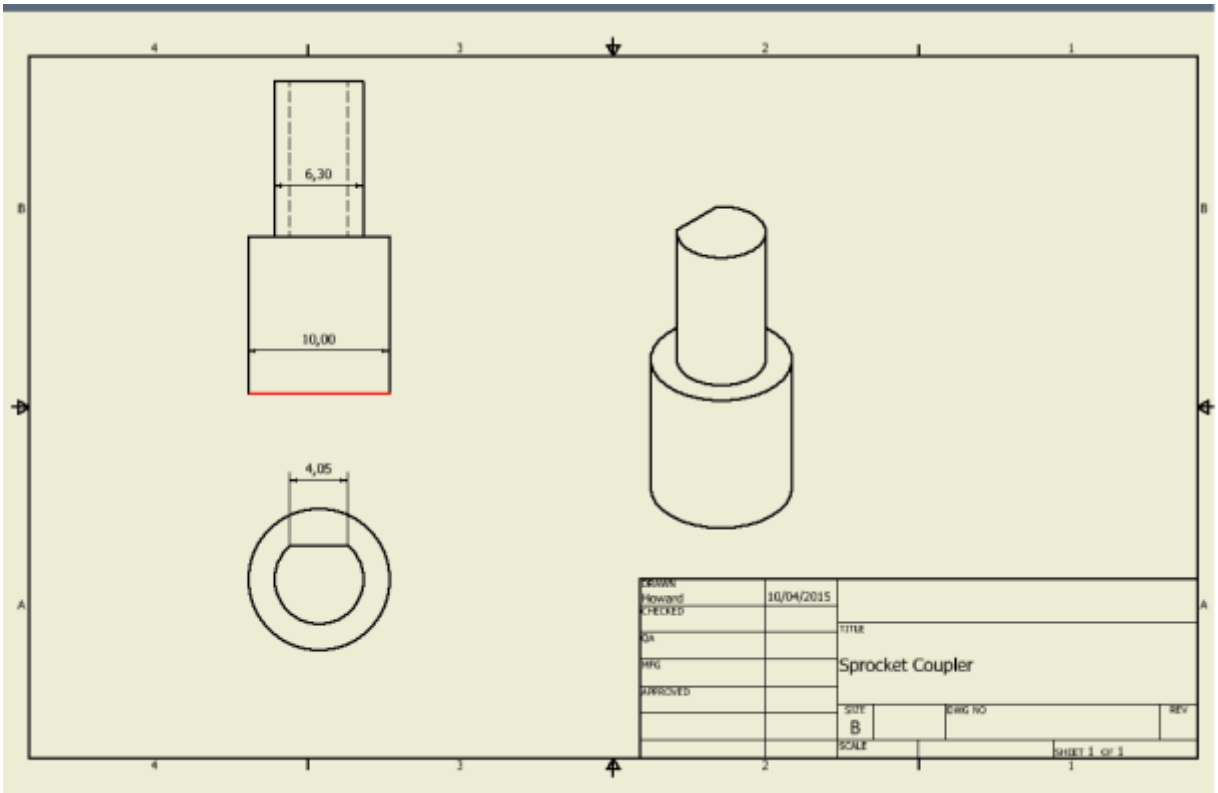
Nevertheless, our team was overjoyed at seeing our 2 robot system working together to place a ball into the game board. After the competition, we reflected upon all the design choices we should have considered in depth, such as the choice to use store bought chassis that is convenient but constrains the dimensions we could work with. Perhaps a better understanding of each other's roles and design ideals would help us recognize these type of mistakes. In the future, our team will try to better understand the problem more from the other members' points of view, and hopefully act and think in unison to prevent any error one member may commit.

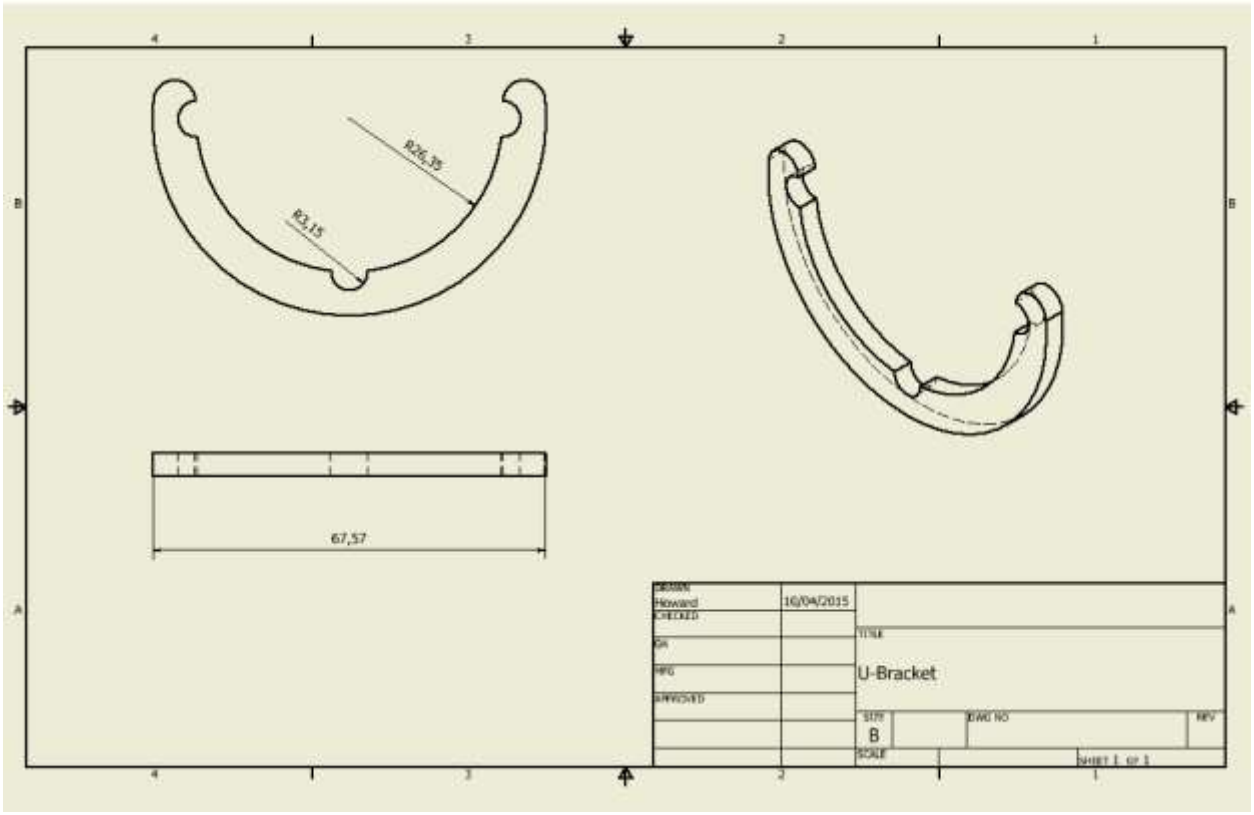
Appendices

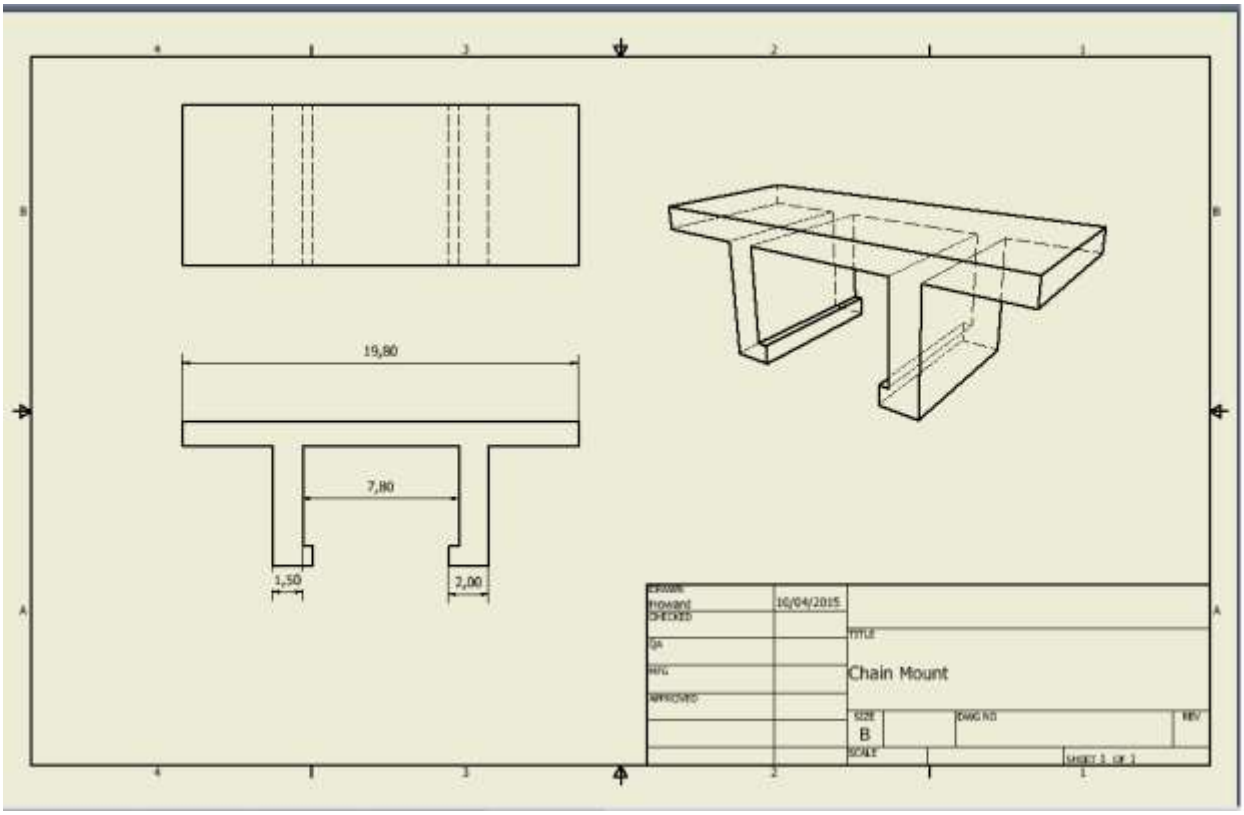
Appendix A: Engineering Drawings of 3D printed parts, name of parts in title block on lower left.

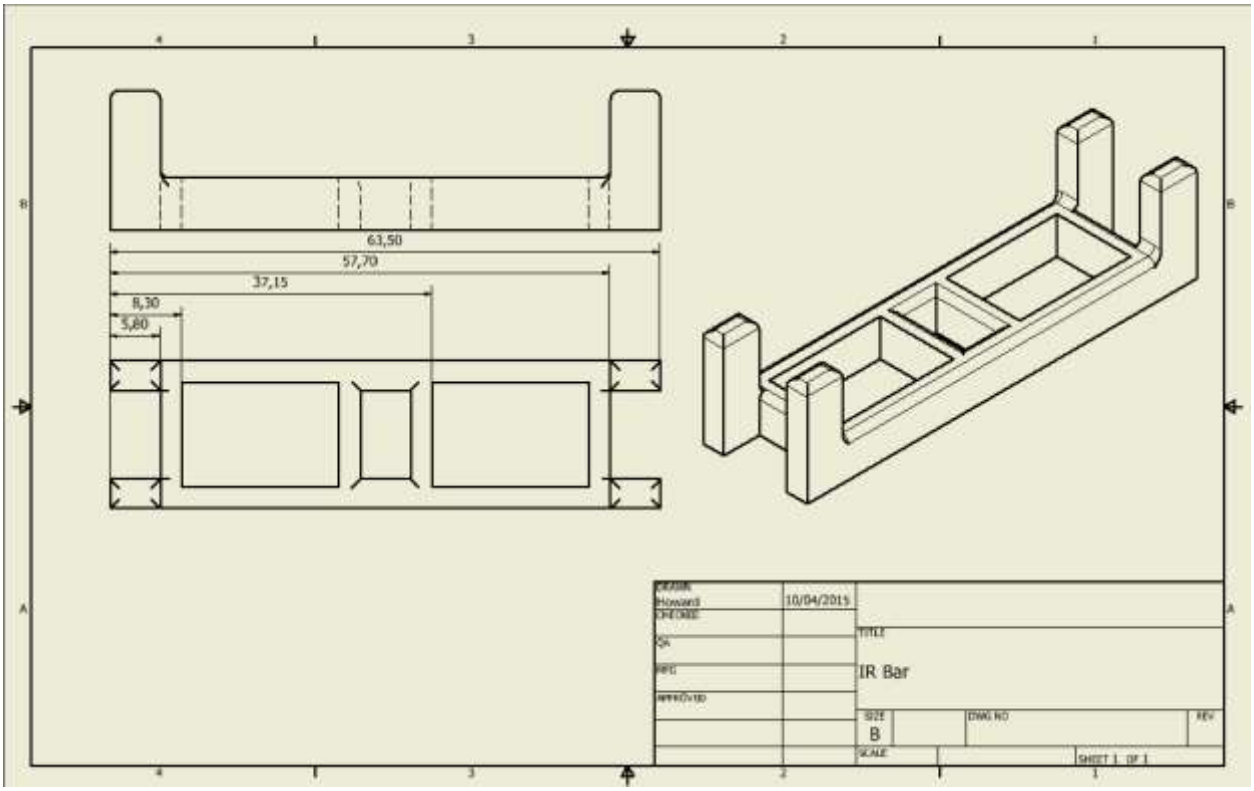


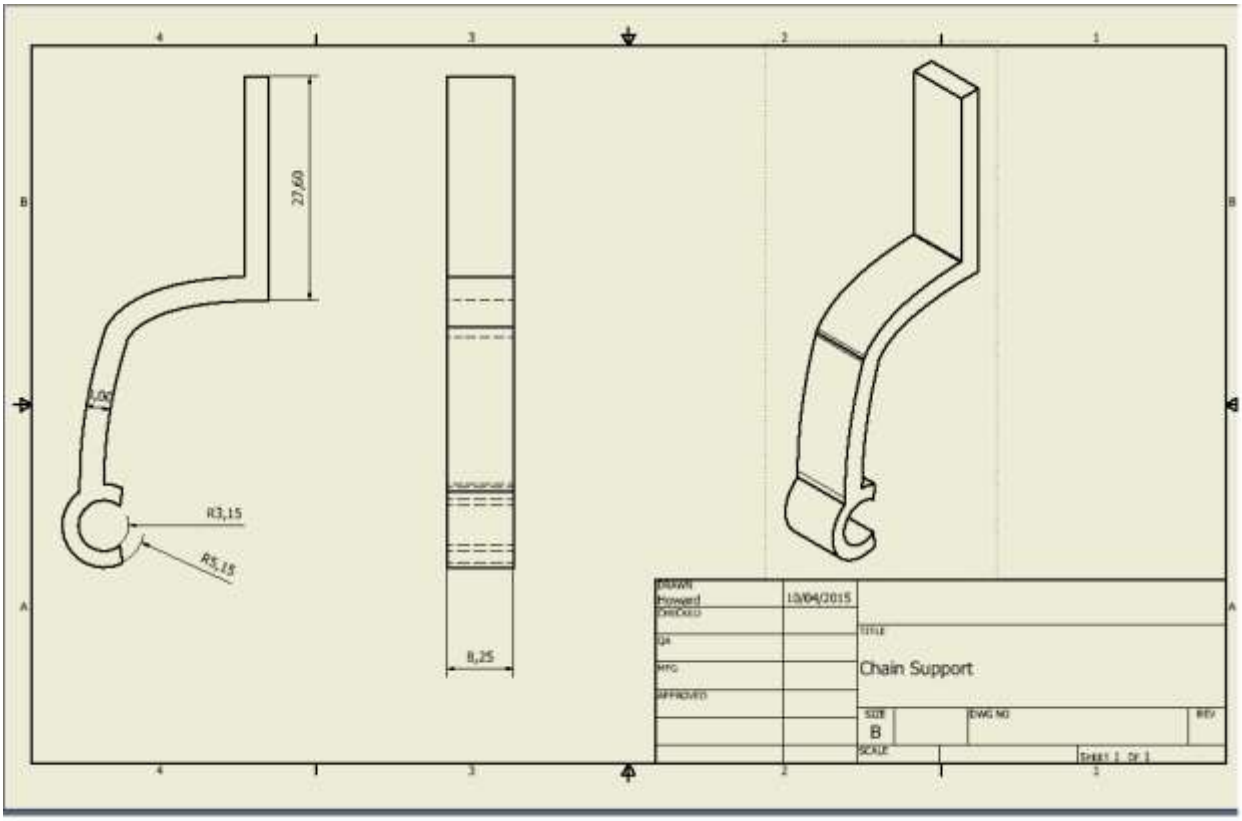

15,00
18,00
71,52
72,90
74,45
76,65
10,00
20,00
10,00
Howard
10/04/2015
Servo Support Arm
B

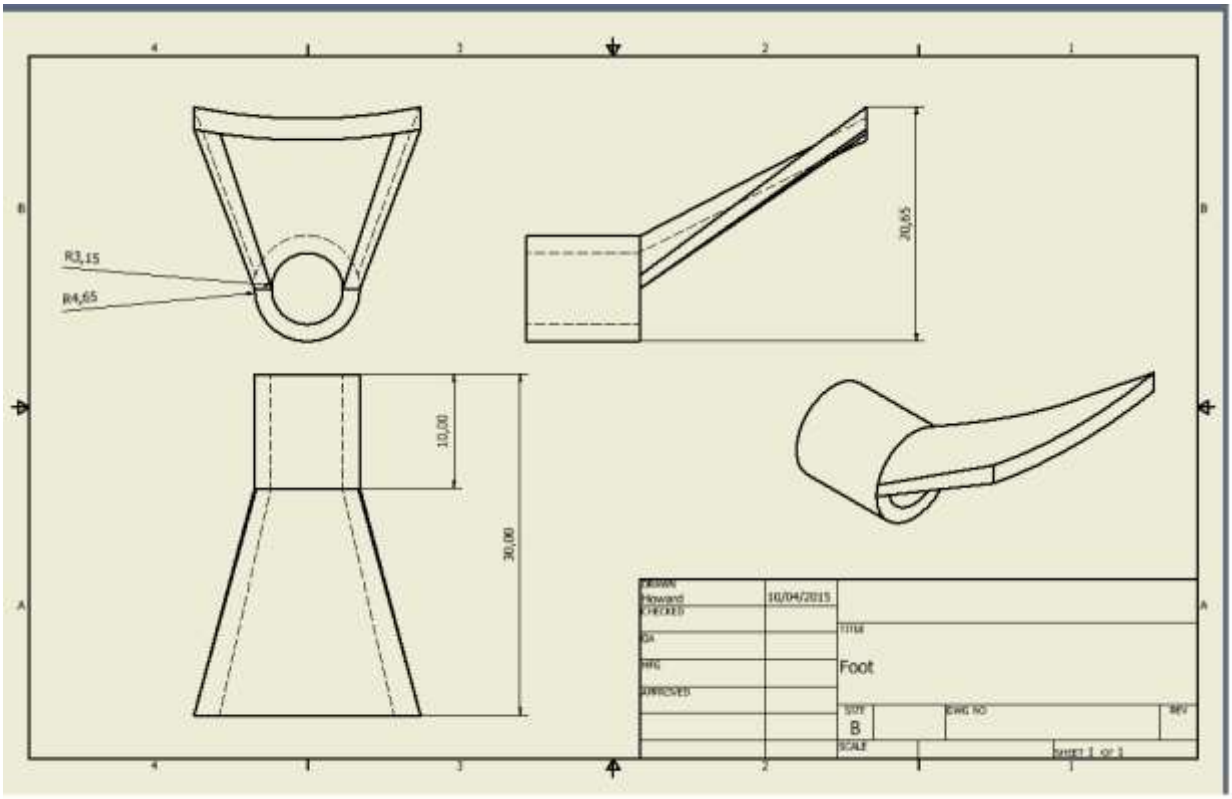

6,30
10,00
4,05
Howard
10/04/2015
Sprocket Coupler
B


R26,35
R3,15
67,57
Howard
16/04/2015
U-Bracket
B


19,80
7,80
1,50
2,00
Howard
16/04/2015
Chain Mount
B


63,50
57,70
37,15
8,30
5,80
10/04/2015
Howard
IR Bar
B


27,60
R3,15
R5,15
8,25
13/04/2015
Howard
Chain Support
B



Appendix B: Budget and Bill of Materials

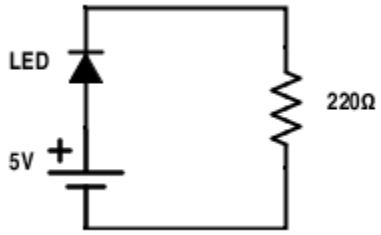
Table 12 Complete list of materials used

Item	Quantity	Cost	Source
Sauder	1	$2.00	Active Surplus
Tape	1	$1.00	Active Surplus
Filter	1	$0.50	Active Surplus
Sainsmart Mega (blue)	1	$32.00	Amazon
Sainsmart Mega (blue)	1	$28.00	Amazon
DC Motor	5 (4 from kits)	$3.58	Project Kit
IR Proximity Sensor (TCRT)	3	$3.23	Creatron (look for receipt with 10 TCRT sensors
Breakbeam Sensors	4	$7.25	Digikey
Photoresistor	9	$18.81	Creatron
H-Bridge	2	$9.00	Project Kit
Resistor	~34	$1.70	Project Kit
Pushbutton	2	$0.50	Project Kit
Red Buttons	2	$3.39	Active Surplus
PCB Protoboard	2	$8.02	Creatron

Laser Protoboard	1	$0.68	Home Hardware
Diode	1	$0.25	Project Kit
Mofset Transistor	2	$3.00	Project Kit
Battery Snap	2	$0.56	Creatron
Pin Holders	2	$0.32	Project Kit
3 Pin Holder	1	$0.44	Creatron
4 Pin Holders	2	$0.95	Creatron
Lasers	7	$8.40	Ebay
Ultrasonic Range Finders	1	$3.24	Amazon
Ultrasonic Range Finders	1	$6.00	Amazon
Servo Motor	1	$7.00	Project Kit
IR Sender/Reciever	2	$3.16	Creatron
LEDs	6	$3.00	Project Kit
Chasis Kit	2	$42.20	Canada Robotix
Wiring	1	$7.00	Home Hardware
Batteries (9V)	2	$5.27	Canadian Tire?
Batteries (AA)	8	$6.88	Canadian Tire?
Elevator (includes dowel piece, chain, and sprockets)	1	$8.48	Miscellaneous
3D Printing	1	$11.87	Gerstein Science
Sensor Bar (Materials)	1	$0.56	Active Surplus
Robot Total		$239.23	

Appendix C: Power Draw of LEDs, Photoresistors and Phototransistors.

The LEDs, Photoresistors and Phototransistors are all connected to +5V and ground via a resistor that is in series with it, as shown below.



Let the resistor in series have the value $R_{resistor}$, and let the LED, photoresistor or phototransistor have the value R_{load}. Since the resistors are in series, the overall resistance can be described by $R_{total} = R_{resistor} + R_{load}$. The current through the resistor and circuit element is given by $I = \frac{V}{R}$. The total Power Drawn can be found using $P = V \times I$.

$$P_{resistor+load} = V_{resistor+load} \times I_{resistor+load} = 5V \times \frac{5V}{R_{total}} = 5V \times \frac{5V}{R_{resistor} + R_{load}}$$

$$P_{resistor+load} < 5V \times \frac{5V}{R_{resistor}} = P_{resistor}$$

Therefore, we use an upper bound estimate of the power drawn by each component by using the power that would be dissipated by the resistor without the circuit element.